Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа компьютерных технологий и информационных систем

**Отчёт по лабораторным работам**

Дисциплина: Технологии компьютерных сетей.

Выполнил студент гр. 5130901/10101 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.Л. Симоновский (подпись)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н.В. Богач (подпись)

“20” октября 2024 г.

Санкт-Петербург

2024

Оглавление

[1. Лабораторная работа 1. Wireshark: Введение. 2](#_Toc180336977)

[1.1. Цель работы: 2](#_Toc180336978)

[1.2. Ход работы: 2](#_Toc180336979)

[1.3. Вывод: 5](#_Toc180336980)

[2. Лабораторная работа 2. Wireshark: DNS. 6](#_Toc180336981)

[2.1. Ход работы: 6](#_Toc180336982)

[2.2. Вывод: 11](#_Toc180336983)

[3. Лабораторная работа 2. Wireshark: HTTP. 11](#_Toc180336984)

[3.1. Ход работы: 11](#_Toc180336985)

[3.2. Вывод: 16](#_Toc180336986)

[4. Лабораторная работа 2. Сокет 1. Веб-сервер. 16](#_Toc180336987)

[4.1. Задание: 16](#_Toc180336988)

[4.2. Ход работы: 16](#_Toc180336989)

[4.3. Вывод: 18](#_Toc180336990)

[5. Лабораторная работа 2. Сокет 2. UDPPinger. 19](#_Toc180336991)

[5.1. Задание: 19](#_Toc180336992)

[5.2. Ход работы: 19](#_Toc180336993)

[5.3. Вывод: 21](#_Toc180336994)

[6. Лабораторная работа 2. Сокет 3. SMTP. 22](#_Toc180336995)

[6.1. Задание: 22](#_Toc180336996)

[6.2. Ход работы: 22](#_Toc180336997)

[6.3. Вывод: 23](#_Toc180336998)

[7. Лабораторная работа 2. Сокет 4. Прокси-сервер. 23](#_Toc180336999)

[7.1. Задание: 23](#_Toc180337000)

[7.2. Ход работы: 24](#_Toc180337001)

[7.3. Вывод: 26](#_Toc180337002)

[8. Приложение: 26](#_Toc180337003)

# Лабораторная работа 1. Wireshark: Введение.

## Цель работы:

В этой лабораторной работе мы познакомимся с программой Wireshark, которая используется для анализа сетевого трафика путем перехвата пакетов данных и изучения их структуры. Wireshark является мощным инструментом для мониторинга сетевой активности, позволяющим наблюдать за обменом сообщениями между протоколами, такими как HTTP, FTP, TCP, UDP, DNS, и IP, на различных уровнях сетевой архитектуры.

Цель данной работы — углубить понимание работы сетевых протоколов, увидеть их в действии, анализируя последовательности пакетов, передаваемых между устройствами в сети. Мы будем наблюдать, как протоколы, используемые нашим компьютером, обмениваются данными с удаленными узлами сети Интернет, что позволит лучше понять работу сетевых приложений.

В ходе лабораторной работы мы научимся использовать Wireshark для захвата сетевых пакетов, интерпретации их структуры и анализа различных полей протокольных сообщений. Особое внимание будет уделено тому, как сообщения верхних уровней, такие как HTTP-запросы, инкапсулируются в кадры канального уровня и передаются по сети. Это позволит понять, как данные путешествуют через сетевые уровни от приложения до физического канала связи.

Таким образом, мы увидим на практике, как работают сетевые протоколы, и получим ценные навыки работы с инструментами анализа сетевого трафика, что является важной частью изучения современных компьютерных сетей.

## Ход работы:

Перейдем непосредственно в Wireshark и посмотрим на главный экран:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рис. 1.1. Главный экран Wireshark.

Здесь мы видим все подключенные интернет адаптеры к компьютеру, в том числе различные VPN подключения. Ethernet 2 является основным адаптером, поэтому выберем именно его. Тогда окно изменится следующим образом:

Изображение выглядит как текст, электроника, снимок экрана, программное обеспечение

Автоматически созданное описание

Рис. 1.2. Wireshark. Захват из Ethernet 2.

На рис. 1.2. мы видим следующие поля в интерфейсе:

* Командные меню — меню для сохранения, открытия, захвата данных и др.
* Поле фильтра — фильтрация пакетов по протоколам и критериям.
* Окно списка пакетов — список перехваченных пакетов с фильтрацией.
* Окно деталей заголовка — детальная информация по выбранному пакету.
* Окно содержимого пакета — данные пакета в шестнадцатеричном и ASCII формате.

Как мы видим по рисунку 1.2. наш компьютер непрерывно обменивается множеством различных пакетов с ресурсами, которые мы даже не запускали. Попробуем отследить конкретно какой-то пакет. Например, по протоколу http, для этого в поле фильтров напишем http. Теперь нам будут отображаться только http пакеты. Попробуем посмотреть на них, для этого перейдем в браузер и зайдем на сайт <https://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html>. В браузере отобразится следующее окно:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 1.3. Результат перехода на сайт.

Однако для нас представляет интерес, что мы увидим в wireshark:

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рис. 1.4. Окно Wireshark после открытия страницы.

Можно увидеть 2 GET запроса и 2 ответа. Первый – запрос непосредственно веб-страницы, по протоколу HTTP, а второй запрос какого-то файла иконки, который не был успешно получен. Рассмотрим первый запрос подробнее:

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рис. 1.5. Пакет запроса в Wireshark.

Как мы видим, здесь есть вся информация о нашем запросе, в том числе метод (GET) и другая служебная информация, такая как User-Agent и др.

Далее рассмотрим ответ, который, как видно по рисунку, идет следующим:

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рис. 1.6. Пакет ответа в Wireshark.

Как видим, тут есть как тело ответа с веб-страницей, так и описание самого ответа, в том числе его код и другая служебная информация.

Также используя Wireshark можно обнаружить ip адрес своего компьютера, он находится в столбце Source (Отправитель) у отправляемого пакета и наоборот в столбце назначение (Destination) у принимаемого, в моем случае это 192.168.1.120, это ip адрес внутри моей локальной подсети. Также можно посмотреть на ip адрес сервера, куда отправляется запрос, в столбце назначение (Destination) у отправляемого пакета и наоборот в столбце назначение Source (Отправитель) у принимаемого, в данном случае сервер имеет ip 128.119.245.12.

Еще можно посмотреть время между принятием и отправкой пакета, для этого обратимся к столбцу time, по которому видно, что между запросом и ответом прошло коло 120 ms (0.12 сек), что достаточно быстро т. к. принимаемый пакет достаточно маленький.

Сохраним пакеты, их можно будет найти в репозитории лабораторной или по следующей ссылке: [github.com/DafterT/TKS\_Labs/tree/main/Лабораторная 1](https://github.com/DafterT/TKS_Labs/tree/main/%D0%9B%D0%B0%D0%B1%D0%B0%201)

Wireshark позволяет анализировать не только http трафик, но и множество других, что видно на следующем рисунке:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рис. 1.7. Окно Wireshark с различными пакетами.

Здесь можно заметить такие протоколы, как UPD, ARP, TCP и некоторые другие, этот список далеко не полный и Wireshark позволяет работать со множеством других интерфейсов, что будет рассмотрено в последующих лабораторных.

## Вывод:

В ходе лабораторной работы была успешно изучена программа Wireshark, которая используется для анализа сетевого трафика. Мы научились захватывать пакеты данных, анализировать их структуру и извлекать полезную информацию о работе сетевых протоколов. На практическом примере с HTTP-запросом и ответом мы рассмотрели, как сетевые протоколы передают данные между устройствами. Было продемонстрировано, как данные инкапсулируются и передаются через различные уровни сетевой архитектуры, от прикладного до канального.

Мы также научились использовать фильтры Wireshark для отслеживания конкретных протоколов, что упрощает анализ большого объема трафика. Определение IP-адресов отправителей и получателей, а также анализ времени между запросами и ответами помогли глубже понять процессы взаимодействия в сети.

Таким образом, данная работа позволила на практике увидеть, как функционируют сетевые протоколы и как передаются данные в сети. Мы освоили базовые навыки работы с инструментами анализа сетевого трафика, что является важным шагом в изучении принципов работы современных компьютерных сетей. Полученные знания помогут лучше понимать сетевые процессы и взаимодействие между устройствами в сети.

# Лабораторная работа 2. Wireshark: DNS.

## Ход работы:

1. Выполните команду nslookup, чтобы получить IP-адрес веб-сервера в Азии. Какой адрес вы получили?

Ответ: 2.16.56.149; 2.16.56.156.



Рис. 2.1. Ответ DNS.

1. Выполните nslookup, чтобы определить авторитетные DNS-серверы университета в Европе. Какие у них адреса?

Ответ: 195.130.120.120

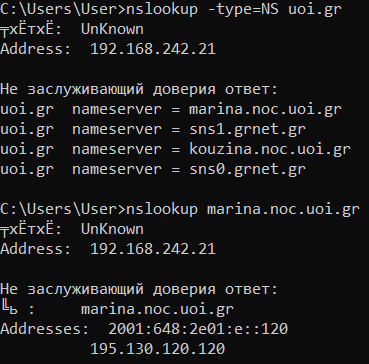


Рис. 2.2. Ответ DNS.

1. Выполните nslookup таким образом, чтобы произвести запрос адреса почтового сервера Yahoo! одному из DNS-серверов, полученных в ответе на вопрос 2. Какой адрес вы получили?

Ответ: DNS не знает об этом адресе.

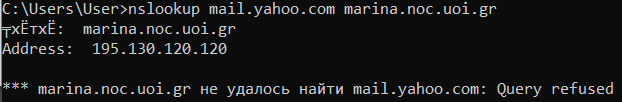


Рис. 2.3. Ответ DNS.

1. Найдите DNS-запрос и ответ на него. С использованием UDP или TCP они отправлены?

Ответ: UDP

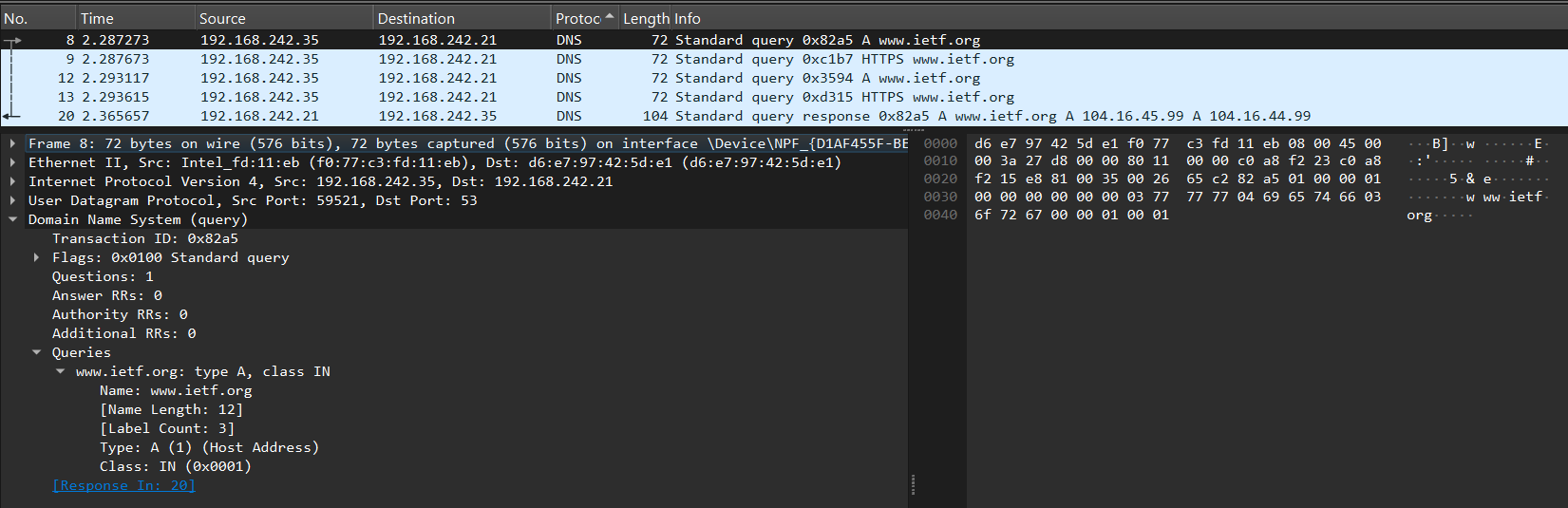


Рис. 2.4. Запрос DNS.



Рис. 2.5. Ответ DNS.

1. Какой порт назначения у запроса DNS? Каков исходящий порт у DNS-ответа?

Ответ: Порт назначения запроса DNS – 53, исходящий порт DNS ответа такой же.

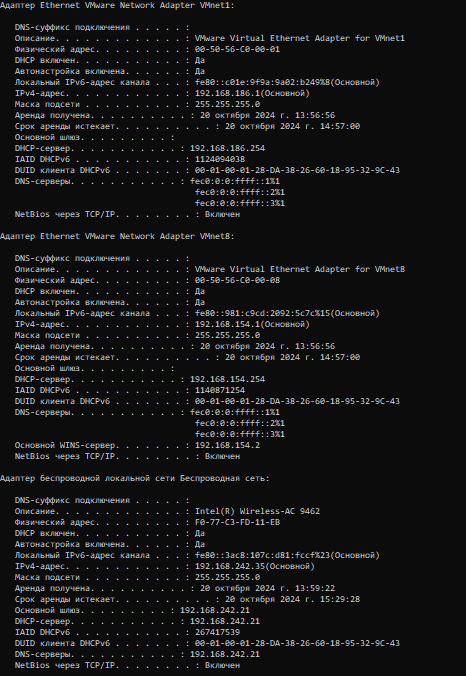
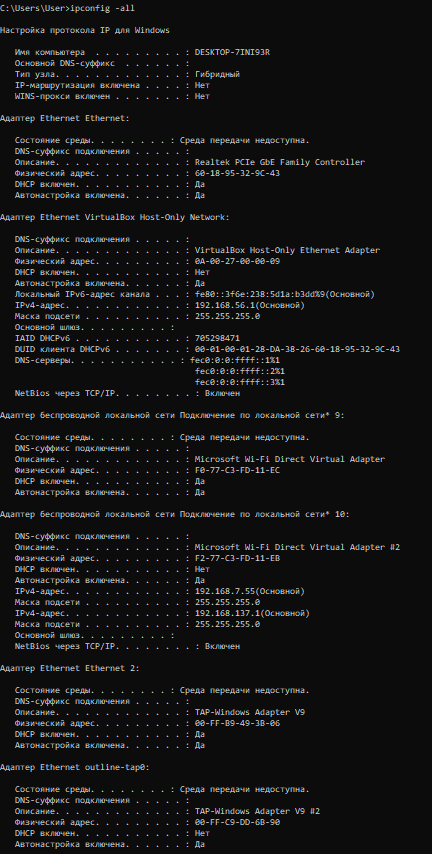


Рис. 2.6. ipconfig -all.

1. На какой IP-адрес отправлен DNS-запрос? Используйте nslookup для определения IP-адреса вашего локального DNS-сервера. Одинаковы ли эти два адреса?

Ответ: Запрос отправлен на адрес 192.168.1.1, который принадлежит локальному серверу DNS.

1. Проанализируйте сообщение-запрос DNS. Запись какого типа запрашивается? Содержатся ли в запросе какие-нибудь «ответы»?

Ответ: это стандартный запрос записи типа A, и он не содержит никаких ответов.

1. Проанализируйте ответное сообщение DNS. Сколько в нем «ответов»? Что содержится в каждом?

Ответ: было два ответа, в которых содержится информация об имени хоста, классе (типе сети), типе ресурсной записи, времени ее жизни в кэше (TTL), размере данных и IP-адресе.

Answers

www.ietf.org: type A, class IN, addr 104.16.45.99

Name: www.ietf.org

Type: A (1) (Host Address)

Class: IN (0x0001)

Time to live: 14 (14 seconds)

Data length: 4

Address: 104.16.45.99

www.ietf.org: type A, class IN, addr 104.16.44.99

Name: www.ietf.org

Type: A (1) (Host Address)

Class: IN (0x0001)

Time to live: 14 (14 seconds)

Data length: 4

Address: 104.16.44.99

1. Посмотрите на последующий TCP-пакет с флагом SYN, отправленный вашим компьютером. Соответствует ли IP-адрес назначения пакета с SYN одному из адресов, приведенных в ответном сообщении DNS?

Ответ: первый пакет с флагом SYN отправлен на адрес 104.16.45.99 – первый IP-адрес, полученный в ответном сообщении DNS.

1. Веб-страница содержит изображения. Выполняет ли хост новые запросы DNS перед загрузкой этих изображений?

Ответ: Нет.

1. Каков порт назначения в запросе DNS? Какой порт источника в DNS-ответе?

Ответ: Порт назначения запроса DNS – 53 , исходящий порт DNS ответа такой же.

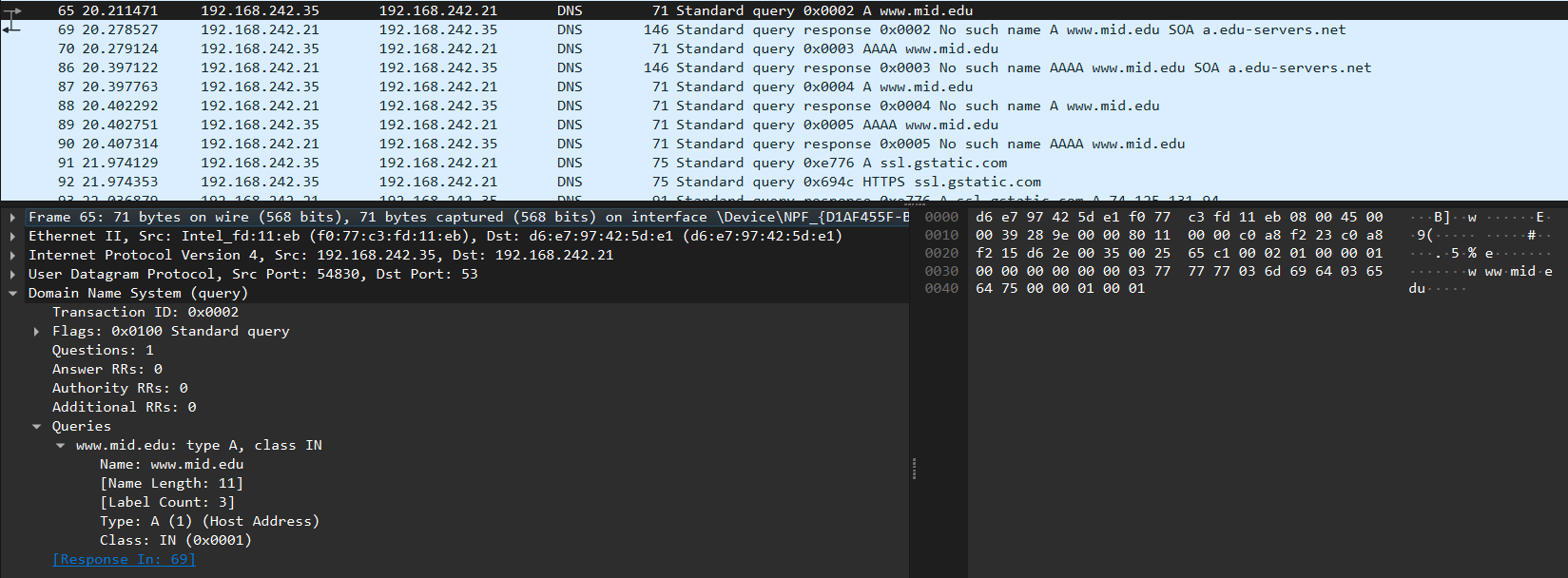


Рис. 2.7. nslookup запрос.

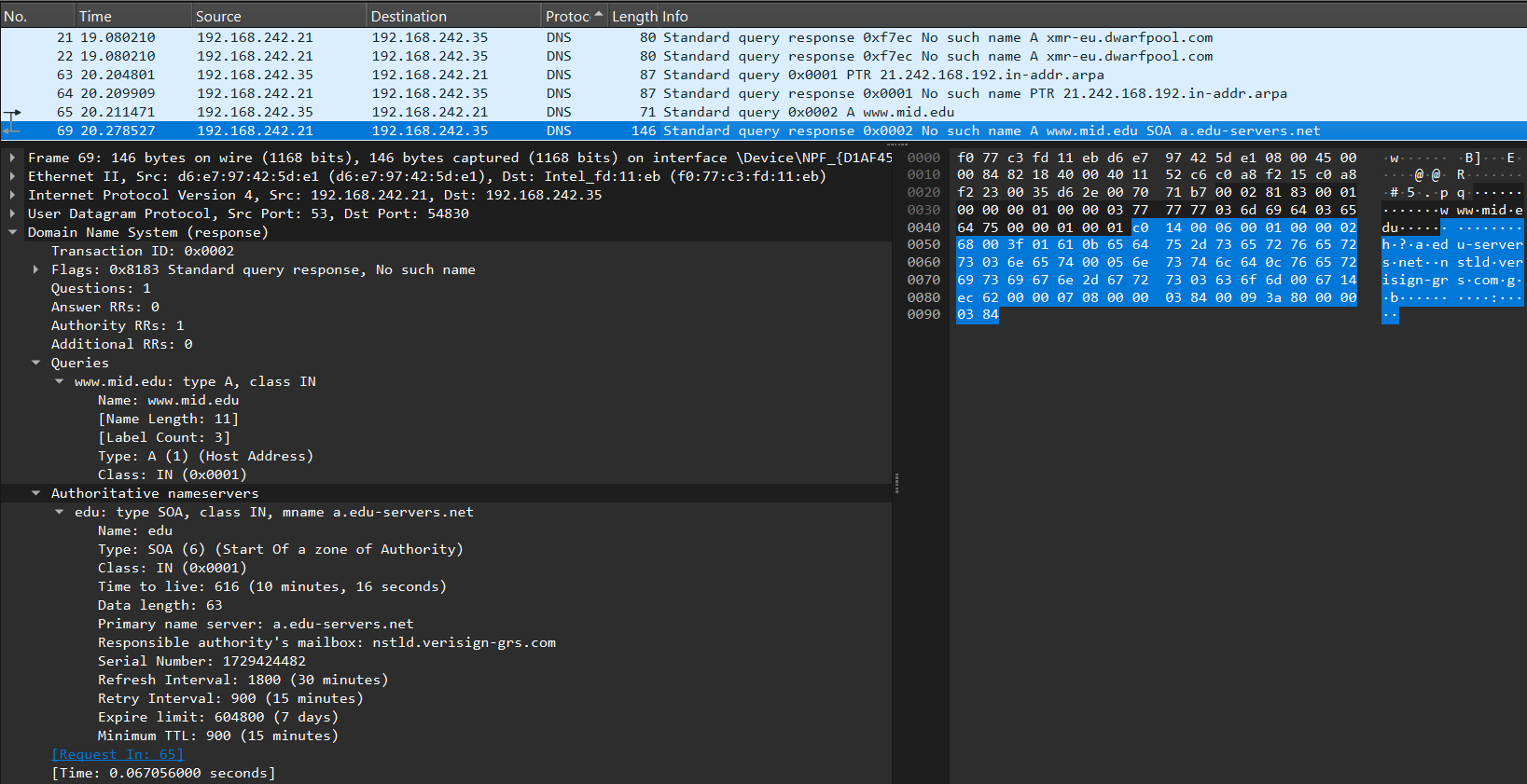


Рис. 2.8. nslookup ответ.

1. На какой IP-адрес отправлен DNS-запрос? Совпадает ли он с адресом локального DNS-сервера, установленного по умолчанию?

Ответ: Запрос отправлен на адрес 192.168.242.21, который, как можно видеть

1. Проанализируйте сообщение-запрос DNS. Запись какого типа запрашивается? Содержатся ли в запросе какие-нибудь «ответы»?

Ответ: это запрос записи типа A, и он не содержит никаких ответов.

1. Проанализируйте ответное сообщение DNS. Сколько в нем «ответов»? Что содержится в каждом?

Ответ: В сообщении один ответ, в котором содержится информация об имени хоста, классе (типе сети), типе ресурсной записи, времени ее жизни, размере данных и IP-адресе.

1. Сделайте снимок

Queries

www.mid.edu: type A, class IN

Name: www.mid.edu

[Name Length: 11]

[Label Count: 3]

Type: A (1) (Host Address)

Class: IN (0x0001)

1. На какой IP-адрес отправлен DNS-запрос? Совпадает ли он с адресом локального DNS-сервера, установленного по умолчанию?

Ответ: да, запрос отправлен локальному DNS-серверу по умолчанию, имеющему адрес 192.168.242.21.

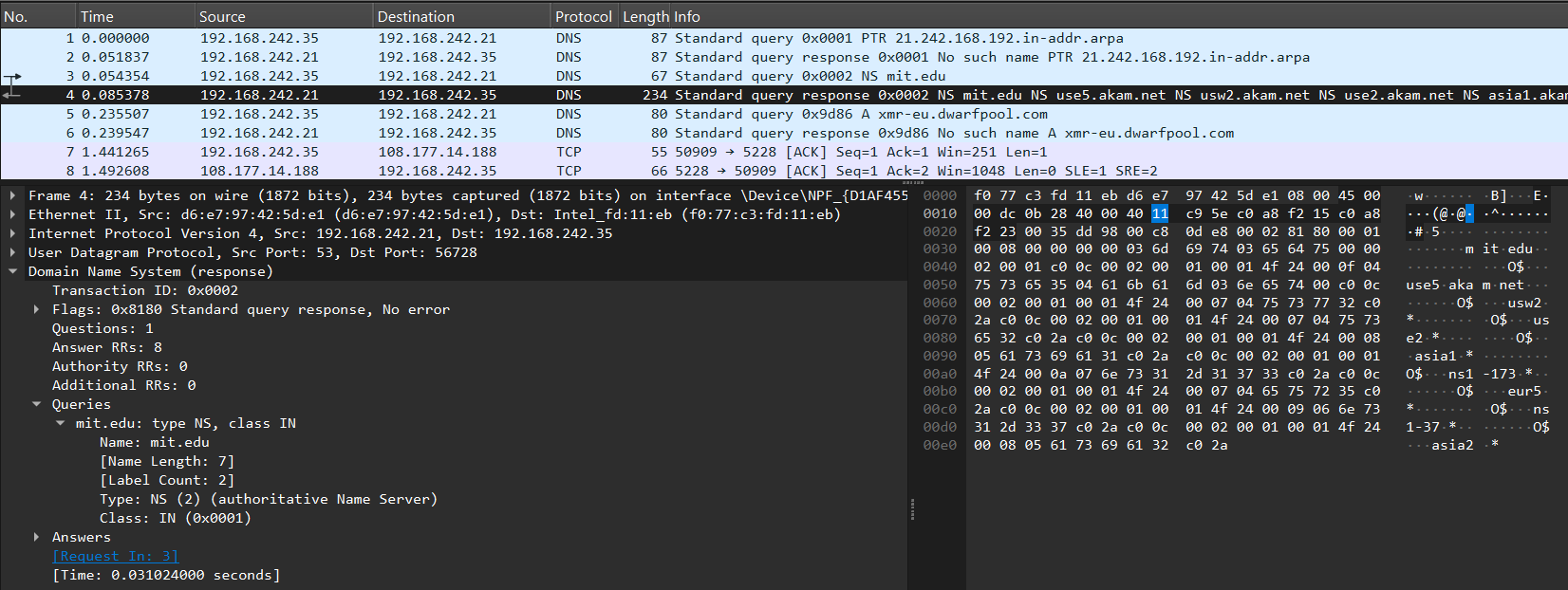


Рис. 2.9. nslookup ответ.

1. Проанализируйте сообщение-запрос DNS. Запись какого типа запрашивается? Содержатся ли в запросе какие-нибудь «ответы»?

Ответ: это запрос записи типа NS, и он не содержит никаких ответов

1. Проанализируйте ответное сообщение DNS. Имена каких DNS-серверов Массачусетского института в нем содержатся? Присутствуют ли их адреса в этом ответе?

Имена DNS-серверов –bitsy, strawb и w20ns. Их адреса можно увидеть, если раскрыть строчки с дополнительной информацией (additional records)

1. Сделайте снимок

Answers

mit.edu: type NS, class IN, ns use5.akam.net

mit.edu: type NS, class IN, ns usw2.akam.net

mit.edu: type NS, class IN, ns use2.akam.net

mit.edu: type NS, class IN, ns asia1.akam.net

mit.edu: type NS, class IN, ns ns1-173.akam.net

mit.edu: type NS, class IN, ns eur5.akam.net

mit.edu: type NS, class IN, ns ns1-37.akam.net

mit.edu: type NS, class IN, ns asia2.akam.net

1. На какой IP-адрес отправлен DNS-запрос? Совпадает ли он с адресом локального DNS-сервера, установленного по умолчанию? Если нет, то какому хосту он принадлежит?

Ответ: Запрос отправлен на адрес 18.0.72.3, который принадлежит серверу bitsy.mit.edu.

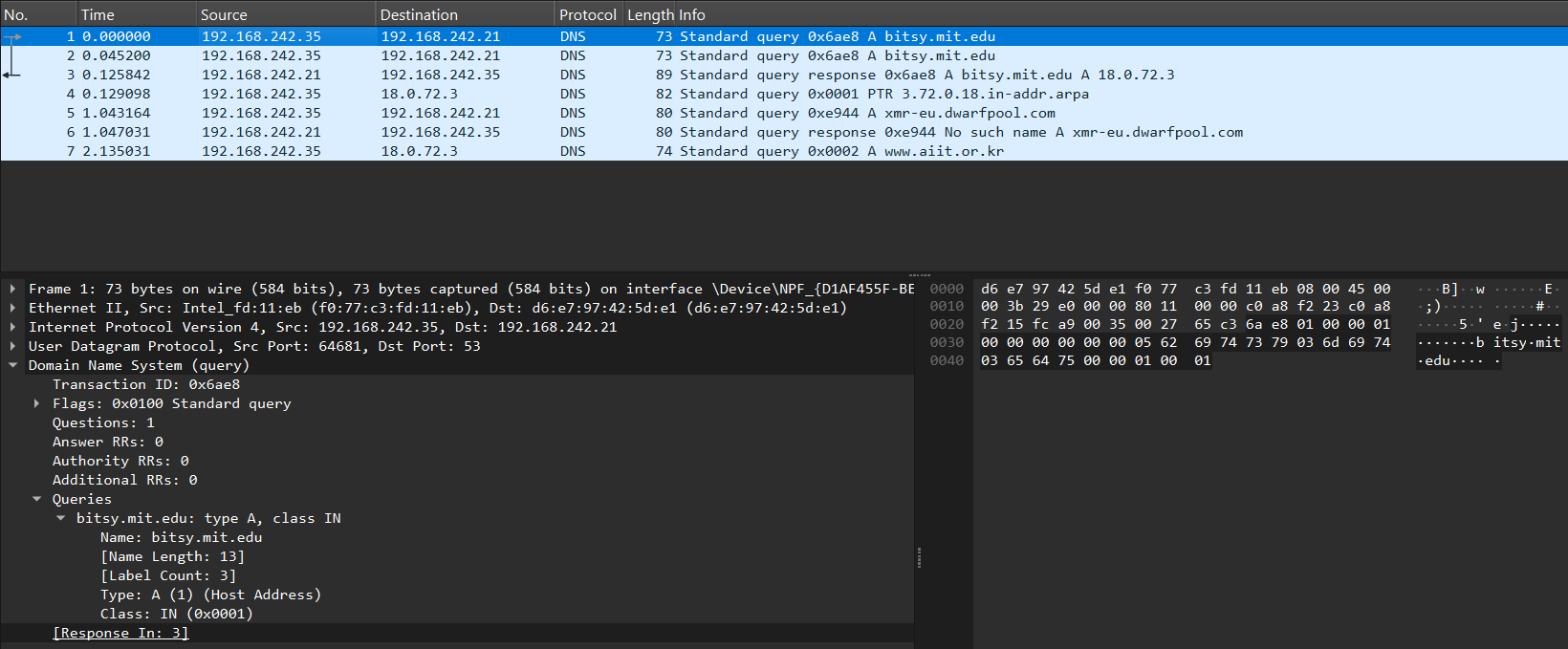


Рис. 2.10. DNS запрос.

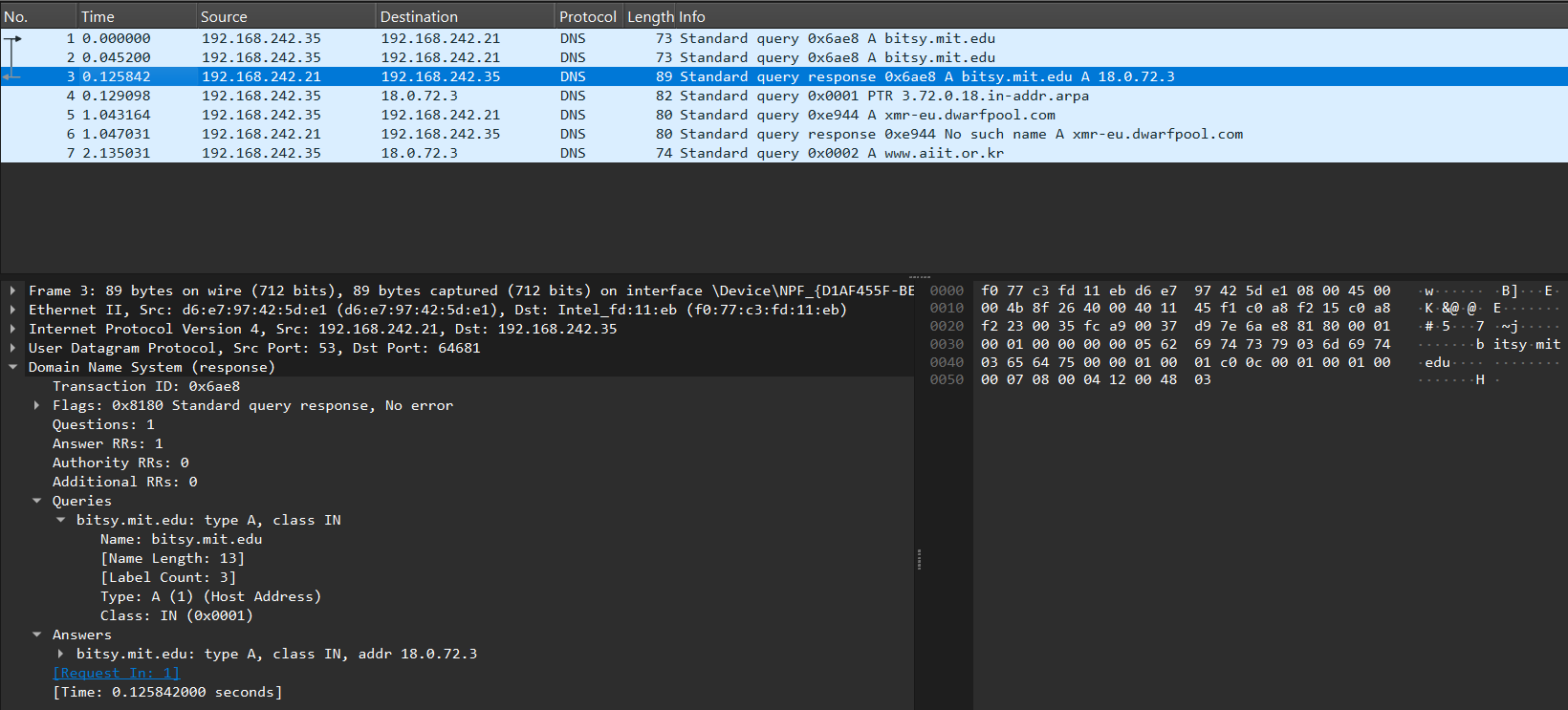


Рис. 2.11. DNS ответ.

1. Проанализируйте сообщение-запрос DNS. Запись какого типа запрашивается? Содержатся ли в запросе какие-нибудь «ответы»?

Ответ: это стандартный запрос записи типа A, и он не содержит никаких ответов

1. Проанализируйте ответное сообщение DNS. Сколько в нем «ответов»? Что содержится в каждом?

Ответ: В сообщении один ответ, в котором содержится следующее:

Answers

bitsy.mit.edu: type A, class IN, addr 18.0.72.3

Name: bitsy.mit.edu

Type: A (1) (Host Address)

Class: IN (0x0001)

Time to live: 1800 (30 minutes)

Data length: 4

Address: 18.0.72.3

## Вывод:

В ходе лабораторной работы была успешно изучена работа протокола DNS и утилиты Wireshark, которая используется для захвата и анализа сетевых пакетов. Мы научились захватывать DNS-запросы и ответы, анализировать их структуру, а также извлекать полезную информацию о взаимодействии клиента и сервера DNS. На практическом примере с использованием команд nslookup и анализа сетевых запросов мы рассмотрели, как протокол DNS преобразует имена доменов в IP-адреса.

Были продемонстрированы принципы работы DNS, включая использование различных типов запросов и ответов. Мы научились применять фильтры Wireshark для выделения только тех пакетов, которые связаны с нашим устройством, что значительно упрощает анализ. В результате эксперимента с очисткой кэша и последующей отправкой DNS-запросов мы смогли проследить полный цикл работы DNS, включая использование итеративных и рекурсивных запросов.

Таким образом, данная работа позволила глубже понять принципы работы системы DNS и механизмов разрешения доменных имен. Освоенные навыки анализа сетевого трафика и работы с Wireshark являются важным шагом в изучении сетевых технологий и помогут лучше разбираться в работе современных компьютерных сетей.

# Лабораторная работа 2. Wireshark: HTTP.

## Ход работы:

Выполним запрос [gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file1.html](https://vk.com/away.php?utf=1&to=http%3A%2F%2Fgaia.cs.umass.edu%2Fwireshark-labs%2FHTTP-wireshark-file1.html)

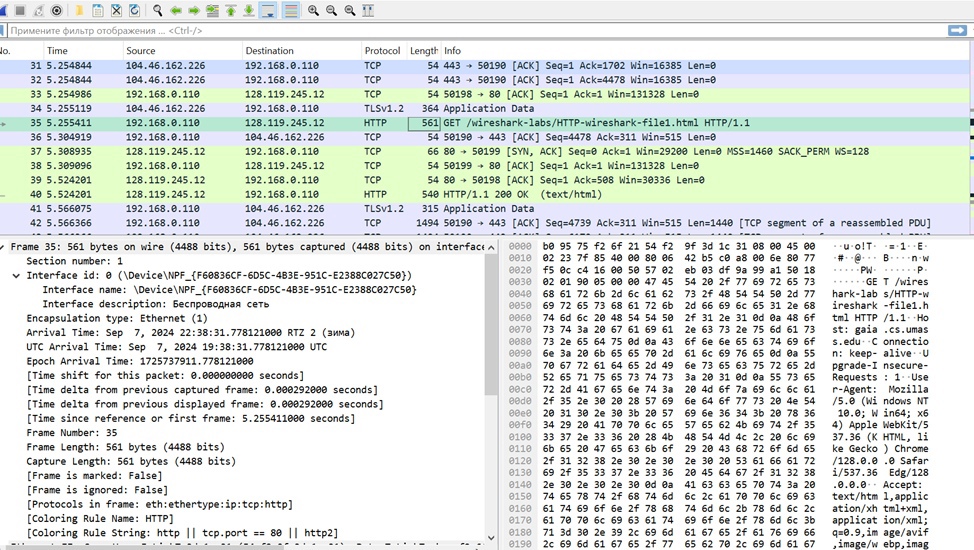


Рис. 3.1. HTTP запрос.

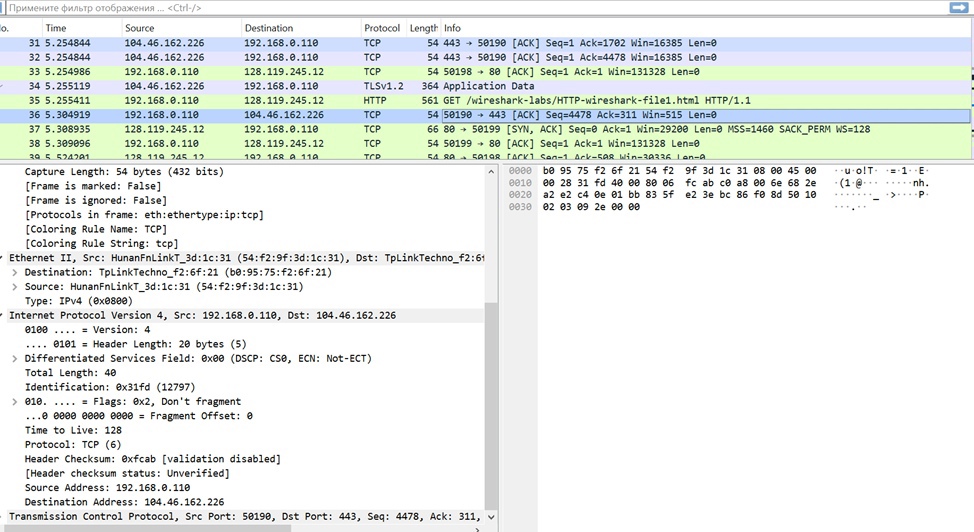


Рис. 3.2. HTTP ответ.

1. Какую версию HTTP использует ваш браузер –1.0 или 1.1? А какую – сервер?

Ответ: браузер и сервер используют версию HTTP/1.1

1. Что указывает браузер серверу относительно поддерживаемых языков?

Ответ: поддерживаются русский и английский языки. На это указывает строка Accept-language.

1. Какой IP-адрес у сервера [gaia.cs.umass.edu?](https://vk.com/away.php?utf=1&to=http%3A%2F%2Fgaia.cs.umass.edu%3F) Каков адрес вашего компьютера?

Ответ: у сервера – 128.119.245.12, у моего компьютера – 192.168.0.110.

1. Какой код состояния возвратил сервер браузеру?

Ответ: Код состояния – 200.

1. Какова дата последнего изменения на сервере HTML-файла, который вы запрашиваете?

Ответ: 07.09.2024 22:48:06 GMT

1. Каков размер содержимого, которое возвратил сервер браузеру?  
   Ответ: Размер контента – 128
2. Проанализировав исходные данные в окне содержимого пакетов, видите ли вы какие-либо заголовки, не отображенные в окне списка пакетов? Если да, то какие?  
   Ответ: Заголовков нет.

Запрос <http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file2.html>

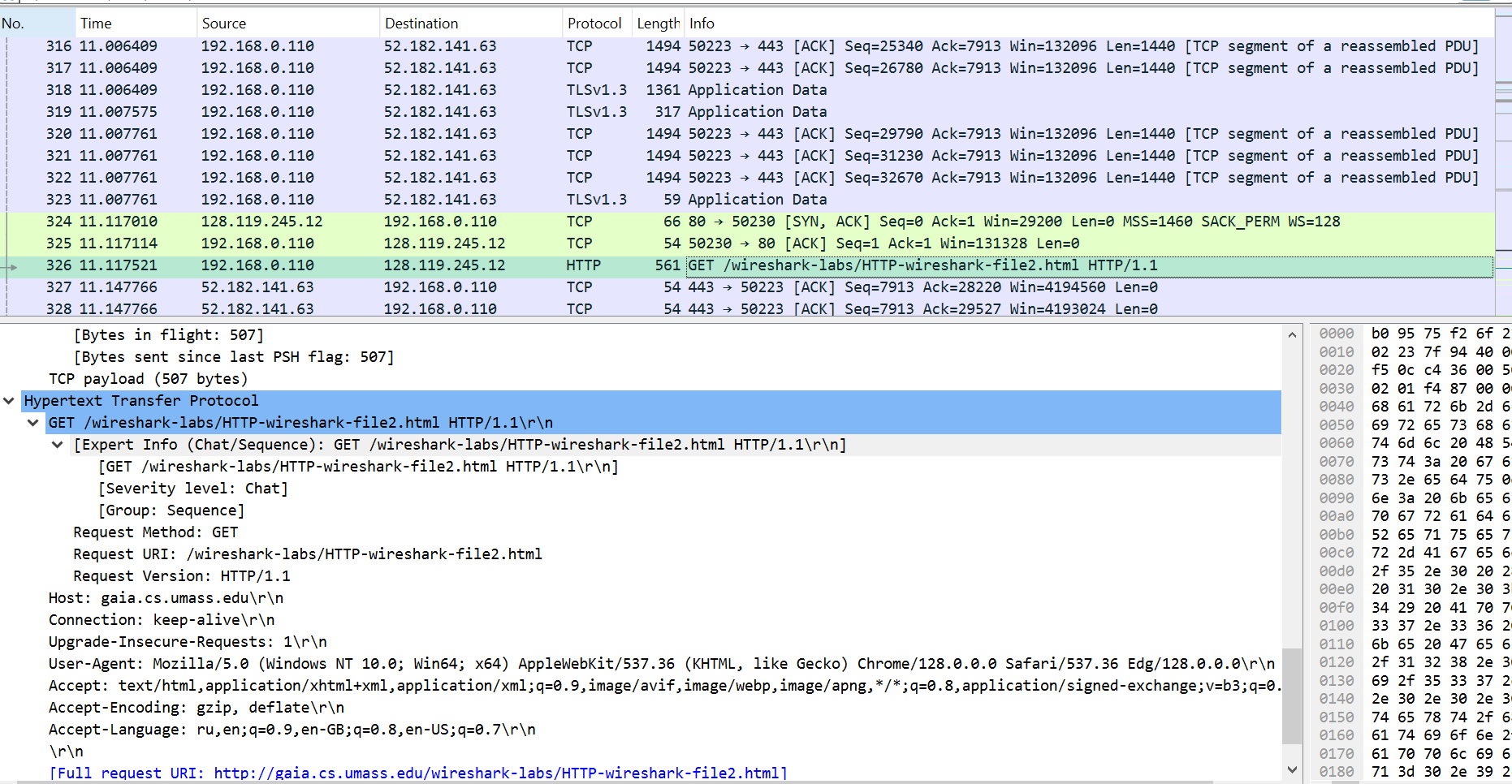


Рис. 3.3. HTTP запрос.

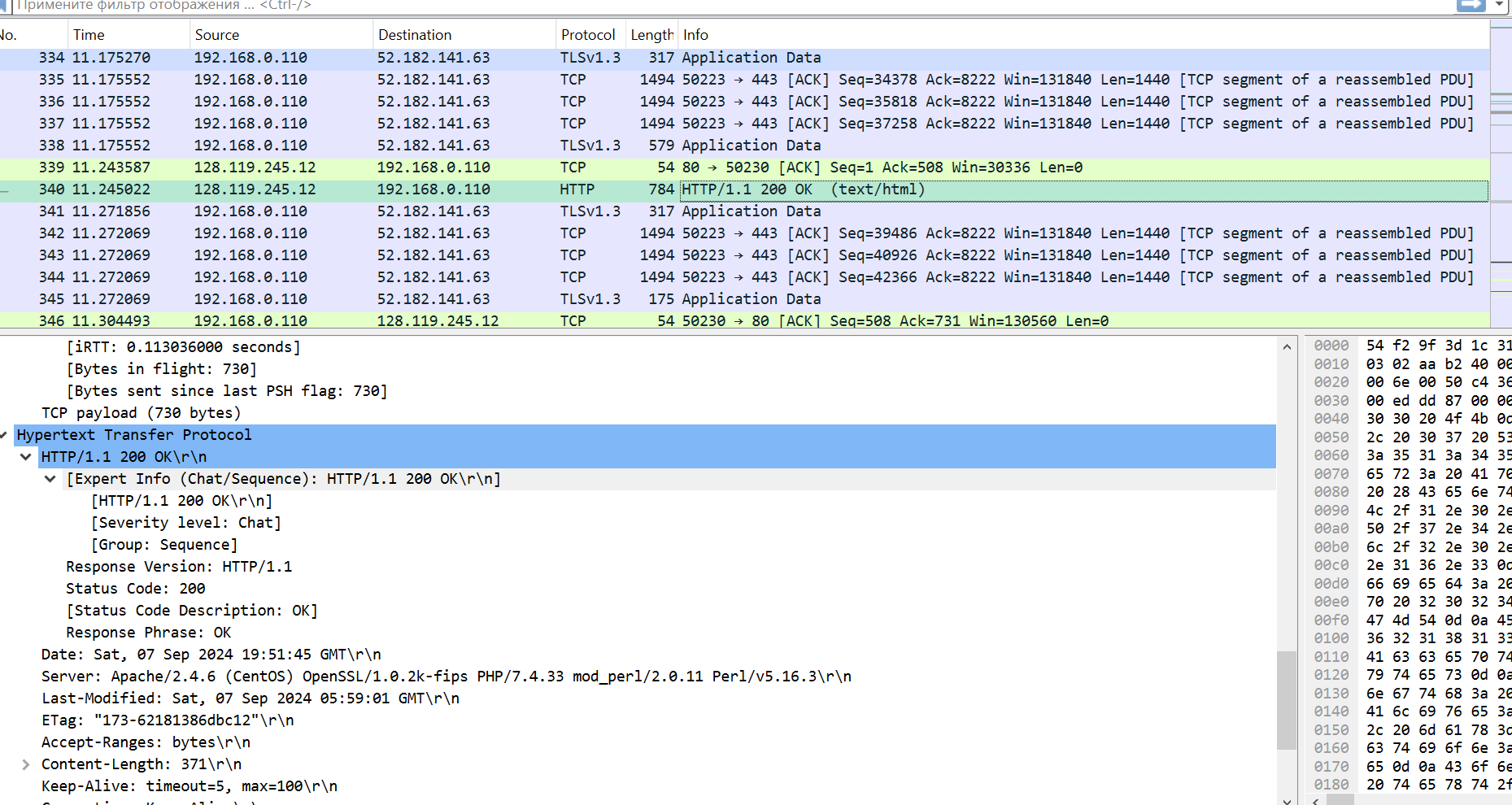


Рис. 3.4. HTTP ответ.

Запрос обновления страницы <http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file2.html>

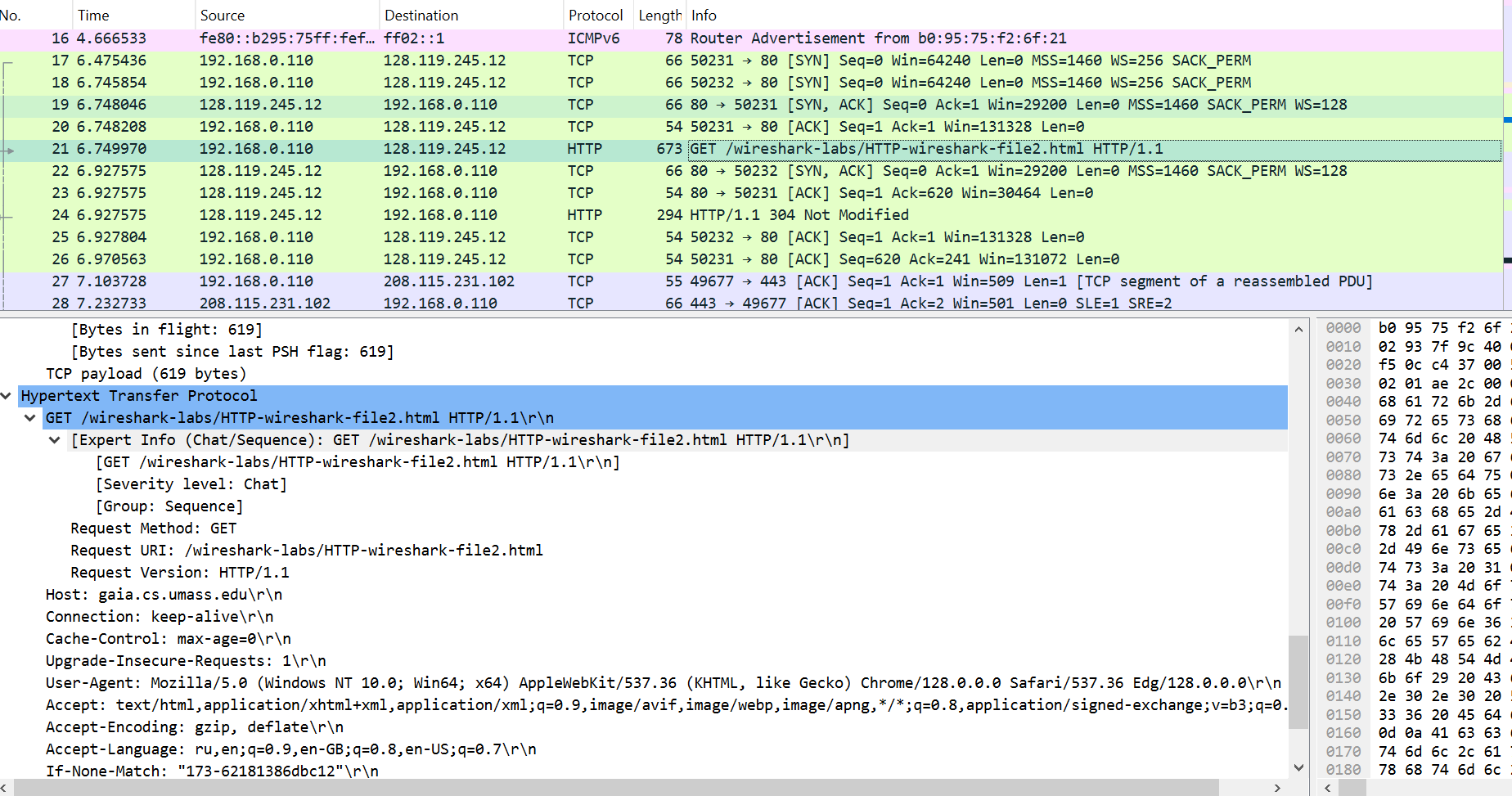


Рис. 3.5. HTTP запрос.

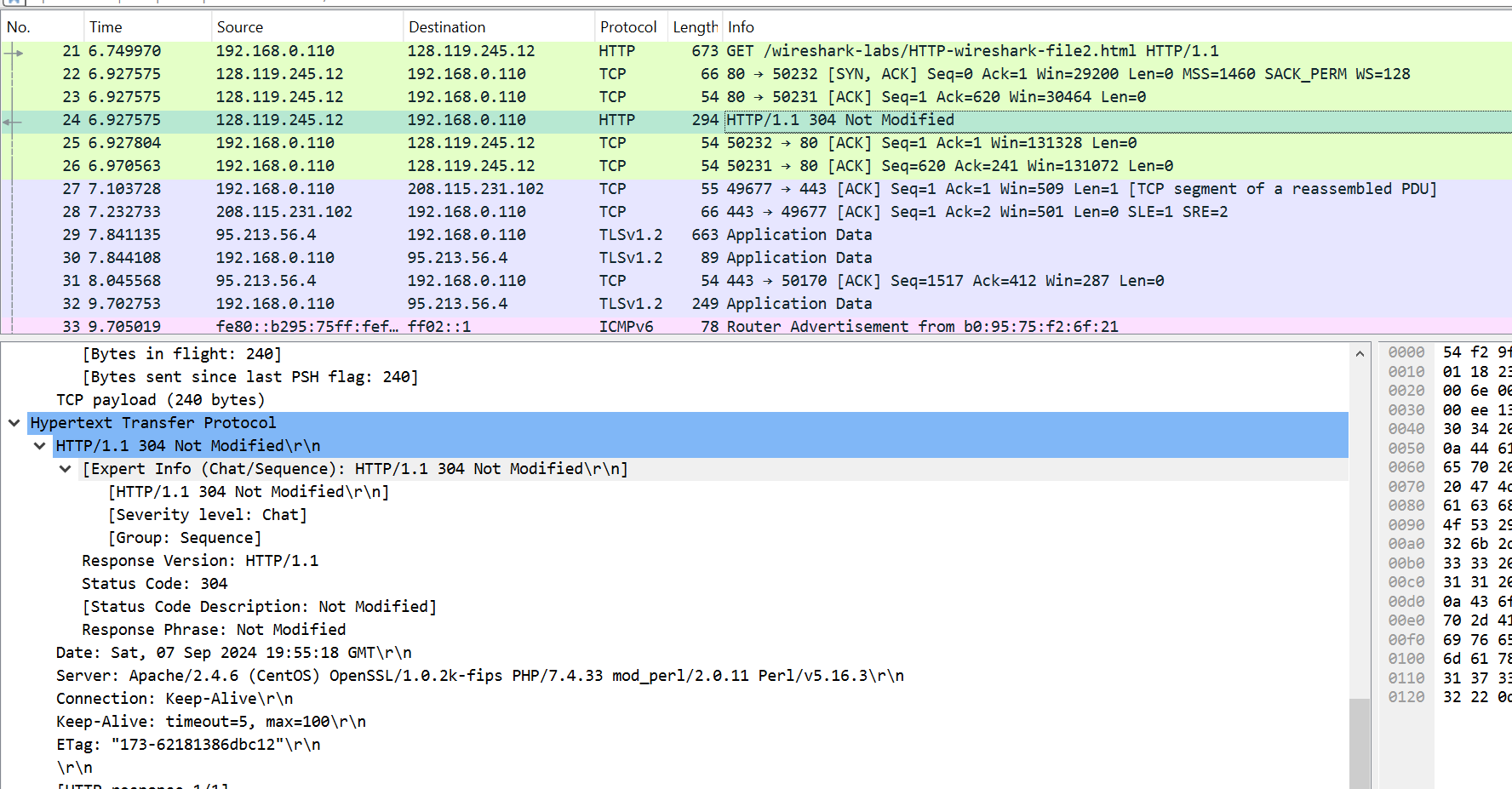


Рис. 3.6. HTTP ответ.

1. Изучите содержимое первого GET-запроса от вашего браузера серверу. Видите ли вы строку IF-MODIFIED-SINCE в запросе?

Ответ: Нет.

1. Проверьте ответ сервера. Возвращает ли он содержимое файла?

Ответ: Да.

1. Теперь изучите содержимое второго GET-запроса серверу. Видите ли вы теперь строку IF-MODIFIED-SINCE в запросе? Если да, то какая информация идет после заголовка IF-MODIFIED-SINCE?

Ответ: Строка есть, в ней написана дата обновления страницы

1. Что возвращает сервер в ответ на второй запрос (код состояния и фраза)? Возвращает ли он содержимое файла? Почему?

Ответ: возвращается код состояния – 304, и фраза not modified. Содержимое файла не возвращает, так как оно не изменилось.

Запрос <http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file3.html>

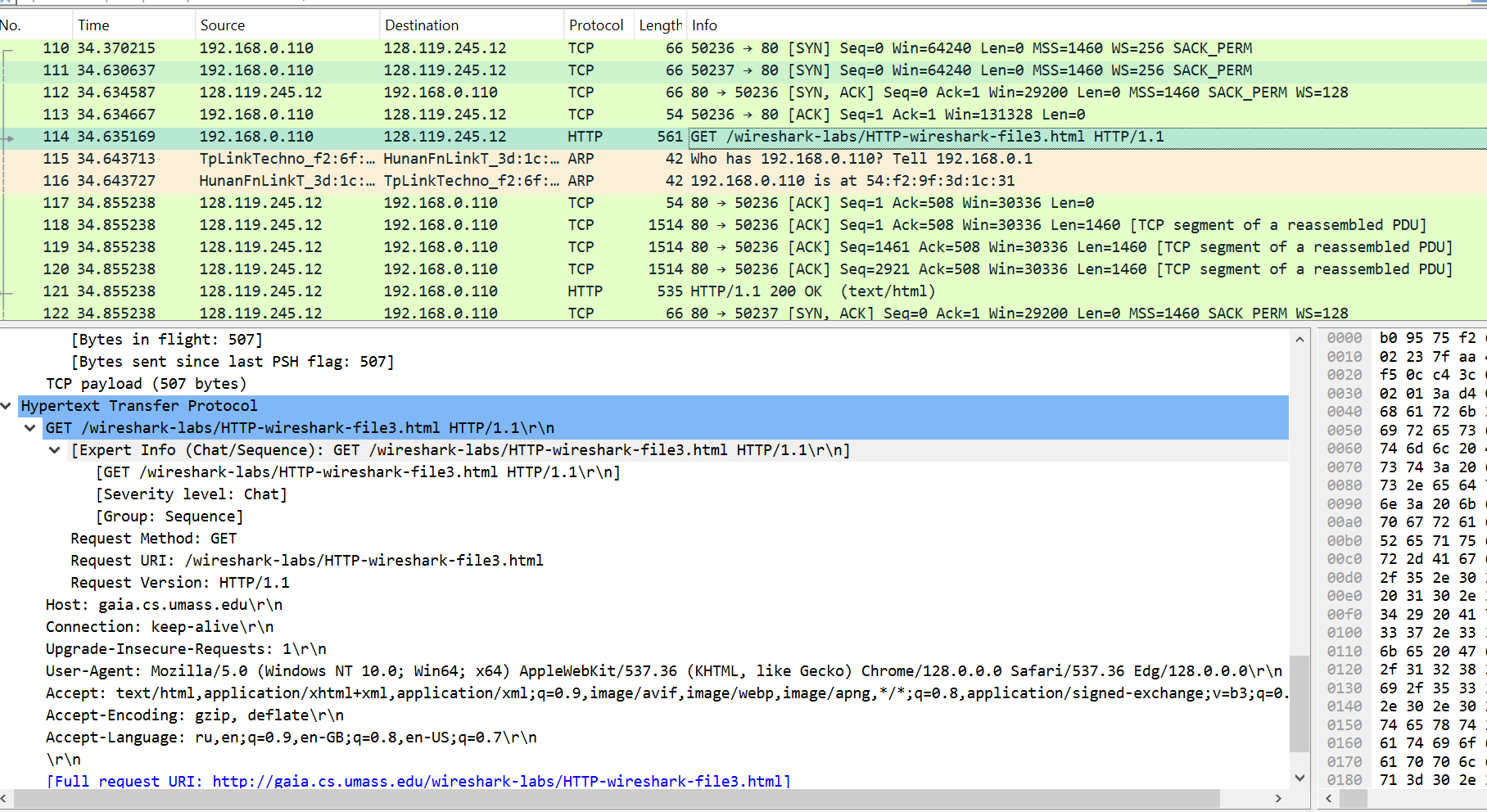


Рис. 3.7. HTTP запрос.

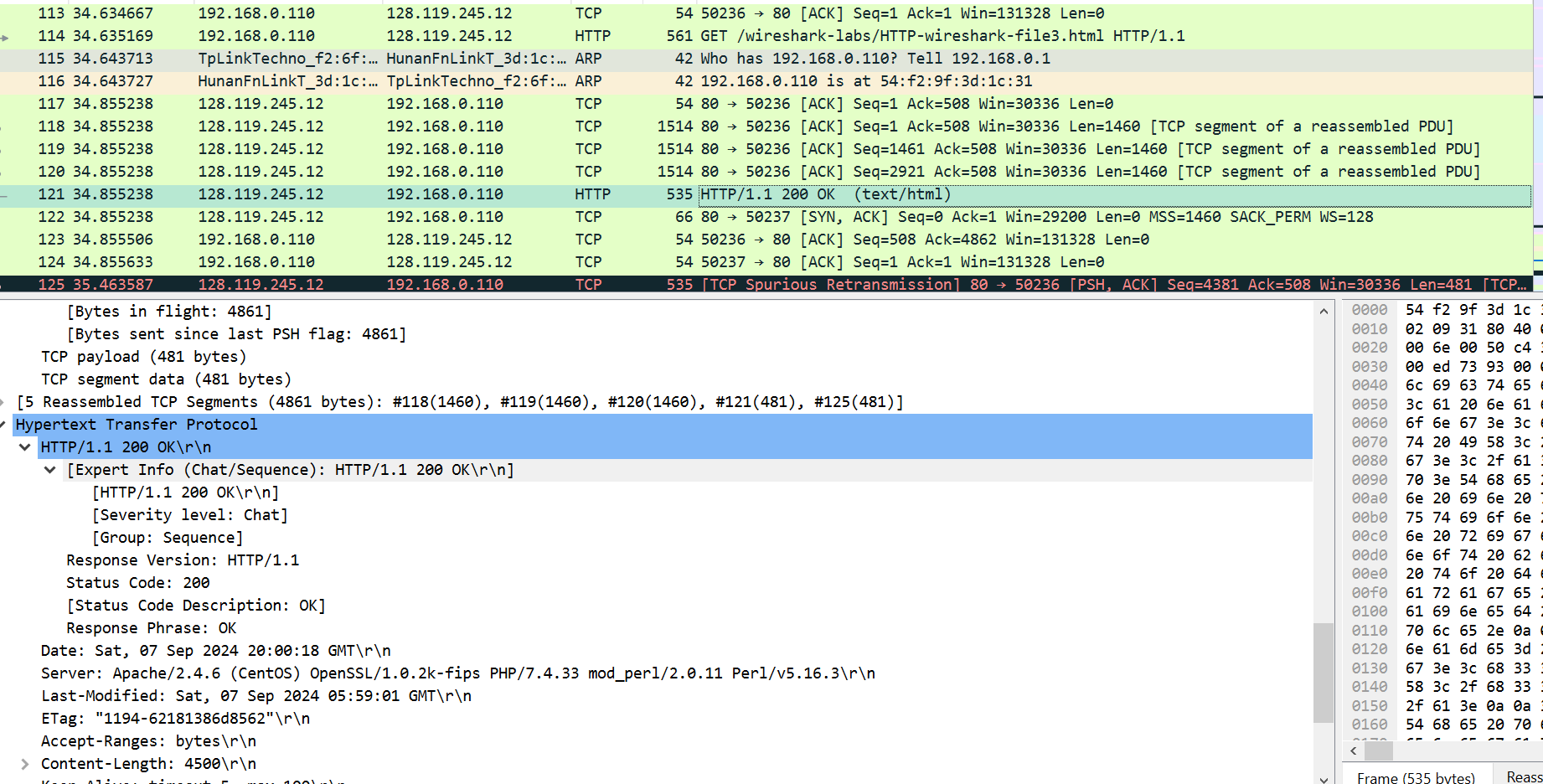


Рис. 3.8. HTTP ответ.

1. Сколько GET-запросов отправил ваш браузер? В пакете с каким номером содержится запрос Билля о правах в файле результатов?

Ответ: Браузер отправил 1 GET-запрос в пакете 342.

1. Какой пакет в результатах трассировки содержит код состояния и фразу, связанные с GET-запросом?

Ответ: Пакет 346.

1. Какой код состояния и фраза в ответном сообщении?

Ответ: 200 (OK)

1. Сколько необходимо сегментов TCP для передачи одного HTTP-ответа и текста Билля о правах?

Ответ: нужно 3 пакета (346, 347, 348 в файле трассировки)

1. Сколько GET-запросов отправил ваш браузер? На какие IP-адреса в Интернете были отправлены эти запросы?

Ответ: было отправлено 3 GET-запроса: пакет 121 (получение основного файла), пакет 126 (получение логотипа Pearson) и пакет 127 (получение изображения обложки книги). Запросы были отправлены на разные IP-адреса: Пакет 121 и пакет 126 были отправлены на 128.119.245.12, а пакет 128 – на адрес 173.194.73.188.

1. Можете ли вы сказать, каким способом ваш браузер загрузил изображения с двух веб-сайтов – параллельно или один за другим? Объясните.

Ответ: Загрузки происходили параллельно. Два GET-запроса изображений – это пакеты 114 и 118. Ответные пакеты, содержащие запрошенные изображения – это пакеты с номерами 121 и 127.

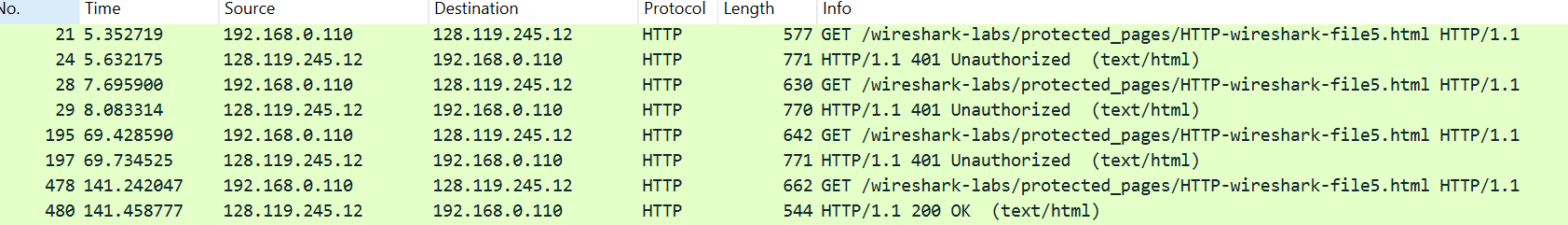


Рис. 3.9. HTTP запросы.

1. Каков первоначальный ответ сервера (код состояния и фраза) на первый GET-запрос вашего браузера?

Ответ: В пакете 21 содержится первый GET-запрос, а пакет 24 – ответ на него сервера. В ответе содержится: 544 Authorization Required

1. Какие новые поля добавляются в GET-сообщение при втором запросе браузера?

Ответ: Новое поле в запросе – Authorization: Basic: \*ключ\*, а также введенная пара логин-пароль

## Вывод:

В ходе лабораторной работы была успешно изучена работа протокола HTTP с использованием программы Wireshark. Мы научились захватывать HTTP-запросы и ответы, анализировать их структуру и извлекать полезную информацию о передаче данных. На примере GET-запросов мы рассмотрели, как веб-браузер взаимодействует с сервером для загрузки HTML-документов.

В процессе выполнения работы было продемонстрировано, как Wireshark помогает исследовать сетевые пакеты на всех уровнях сетевой архитектуры. Мы также научились использовать фильтры для отображения только HTTP-трафика, что упростило анализ захваченных данных. Дополнительно мы изучили ответы сервера, включая коды состояния, а также особенности передачи больших HTML-документов.

Таким образом, данная лабораторная работа позволила глубже понять процесс передачи данных по HTTP-протоколу, а также закрепить навыки работы с Wireshark для анализа сетевого трафика. Полученные знания помогут в дальнейшем изучении сетевых взаимодействий и обеспечении безопасности при работе с веб-ресурсами.

# Лабораторная работа 2. Сокет 1. Веб-сервер.

## Задание:

Создать веб-сервер, который будет обрабатывать один HTTP-запрос.

Веб-сервер должен будет принять и проанализировать HTTP-запрос, получить требуемый файл из файловой системы сервера, создать ответное HTTP-сообщение, состоящее из запрошенного файла и предваряющих его строк заголовка, а затем отправить ответ непосредственно клиенту. Если требуемый файл на сервере отсутствует, клиенту должно вернуться HTTP-сообщение 404 Not Found.

Разработанный вами веб-сервер обрабатывает только один HTTP-запрос. Реализуйте многопоточный сервер, который мог бы обслуживать несколько запросов одновременно. Сначала создайте основной поток (процесс), в котором ваш модифицированный сервер ожидает клиентов на определенном фиксированном порту. При получении запроса на TCP-соединение от клиента он будет устанавливать это соединение через другой порт и обслуживать запрос клиента в отдельном процессе. Таким образом, для каждой пары запрос-ответ будет создаваться отдельное TCP-соединение в отдельном процессе.

Вместо использования браузера напишите собственный HTTP-клиент для тестирования вашего веб-сервера. Ваш клиент будет подключаться к серверу с помощью TCP-соединения, отправлять ему HTTP-запрос с помощью метода GET и отображать ответ сервера в качестве результата.

Клиент должен будет в качестве входных параметров принимать аргументы командной строки, определяющие IP-адрес или имя сервера, порт сервера и полный путь хранения файла на сервере. Формат команды для запуска клиента следующий:

client.py хост\_сервера порт\_сервера имя\_файла

## Ход работы:

Дополним шаблон кода из пособия и выполним дополнительные задания. Реализуем многопоточность с помощью библиотеки threading. Каждый поток обрабатывает свой запрос, реализуя TCP-соединение в отдельном процессе. Напишем собственный HTTP-клиент, который будет устанавливать TCP-соединение с веб-сервером, отправлять GET-запросы и отображать результат.

Код веб-сервера:



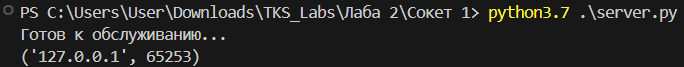
Код клиента:



Код http:



Запустим веб сервер:



При переходе на несуществующую страницу видим:

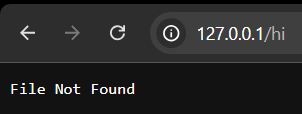


Рис. 4.1. Запрос к странице, которой нет.

Далее запустим созданный нами html файл:

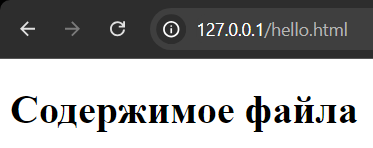


Рис. 4.2. Запрос к странице, которая существует.

Выполним запрос к серверу через клиента:

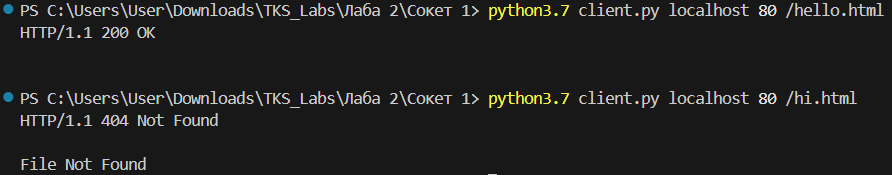


Рис. 4.3. Запрос через клиента.

## Вывод:

В ходе выполнения лабораторной работы был разработан веб-сервер, корректно отвечающий на HTTP-запросы. На первом этапе был реализован базовый сервер, который мог принимать один запрос, возвращая соответствующий файл или сообщение о том, что файл не найден (ошибка 404).

Дальнейшая работа позволила улучшить сервер. В первом дополнительном задании был создан многопоточный сервер, который может обрабатывать несколько запросов одновременно. Это обеспечивает параллельную работу с разными клиентами, создавая для каждого независимое TCP-соединение в отдельном процессе.

Во втором дополнительном задании был разработан HTTP-клиент для тестирования работы сервера. Этот клиент позволяет устанавливать соединение с сервером, отправлять GET-запросы и отображать полученные ответы. Клиент поддерживает параметры командной строки для указания IP-адреса сервера, порта и пути к файлу на сервере.

Таким образом, в ходе лабораторной работы был реализован веб-сервер с расширенными возможностями обработки запросов и собственным HTTP-клиентом для его тестирования.

# Лабораторная работа 2. Сокет 2. UDPPinger.

## Задание:

Необходимо изучить работу простого пинг-сервера, написанного на языке Python, и реализовать клиентскую программу для взаимодействия с ним. Программы будут использовать протокол UDP для обмена данными вместо стандартного протокола ICMP, который применяется в обычных утилитах для пингования. Функционал должен позволять клиенту отправлять пакеты данных на удаленный хост и получать их обратно без изменений (эхо-запрос и эхо-ответ), а также измерять время оборота пакета (RTT), то есть время, за которое данные проходят от клиента до сервера и обратно.

Требуется изменить вывод программы так, чтобы он был аналогичен стандартной утилите ping, с отображением минимального, максимального и среднего времени RTT. Также необходимо вычислить процент потерь пакетов. В дополнение, нужно разработать приложение UDP Heartbeat, которое будет мониторить доступность сервера. Клиент отправляет порядковый номер и временную метку в пакете UDP на сервер, который вычисляет время оборота пакета и отслеживает потери. В случае, если пакеты не поступают в течение определенного времени, предполагается, что клиентское приложение остановлено.

## Ход работы:

Напишем клиентскую часть приложения. Реализуем следующие функции:

* Отправьте эхо-запрос, используя UDP. Примечание: в отличие от TCP, вам не нужно сначала устанавливать соединение, так как UDP является протоколом без установления соединения.
* Распечатайте ответное сообщение от сервера, если таковое имеется.
* Вычислите и выведите на печать время оборота (RTT) в секундах для каждого пакета при ответе сервера.
* В противном случае, выведите сообщение «Request timed out» (Время запроса истекло).

Код клиентской части для основного задания:



Запустим сервер и клиент и посмотрим на вывод в консоль:

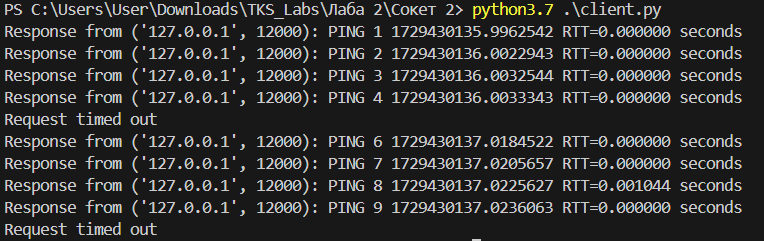


Рис. 5.1. Запрос.

Из рисунка выше можно увидеть, что требования к клиентской части выполнены.

Выполним дополнительное задание 1. Изменим вывод и вычислим коэффициент потери пакетов.

Код клиентской части для дополнительного задания 1:



Запустим сервер и клиент и посмотрим на вывод в консоль:

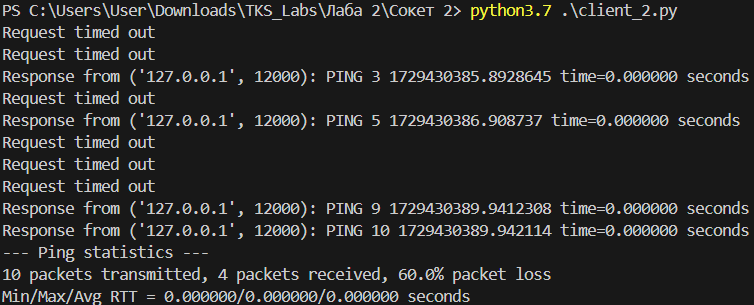


Рис. 5.2. Запрос.

Из рисунка выше можно увидеть, что требования из дополнительного задания 1 выполнены.

Выполним дополнительное задание 2. Реализуем UDP Heartbeat. Доработаем серверную часть и перепишем клиентскую.

Код сервера для дополнительного задания 2:



Код клиента для дополнительного задания 2:



Запустим сервер и клиент и посмотрим на вывод в консоль.

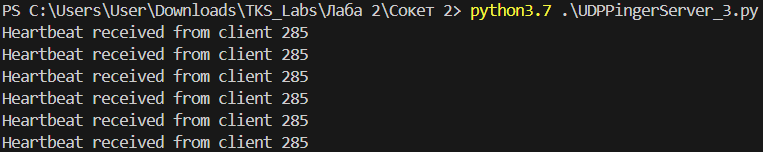


Рис. 5.3. Консоль в серверной части.

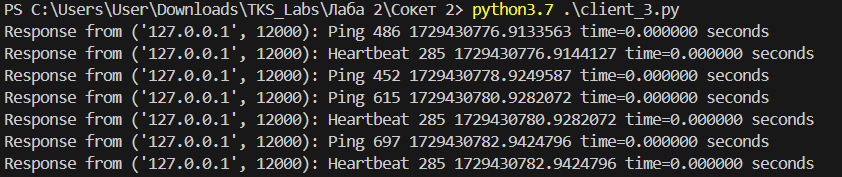


Рис. 5.4. Консоль в клиентской части.

Из рисунка выше можно увидеть, что требования из дополнительного задания 2 выполнены.

## Вывод:

В ходе выполнения лабораторной работы удалось углубиться в принципы работы пинг-сервера, реализованного на Python, и создать клиентское приложение для взаимодействия с ним. Главная задача состояла в симуляции функциональности утилиты "ping", но с использованием UDP-протокола вместо ICMP, что упростило архитектуру.

В процессе выполнения дополнительной части работы было разработано вычисление времени задержки (RTT) для каждого отправленного пакета, а также модифицирован формат вывода для максимального сходства с системной утилитой ping. Был произведен расчет минимального, максимального и среднего RTT, а также процента потерянных пакетов, что сделало вывод более информативным.

Отдельным этапом стало добавление функционала для реализации механизма UDP Heartbeat, что позволило обеспечить мониторинг работоспособности клиентской части и детектировать потерю пакетов в одну сторону. Реализация обеих частей программы помогла лучше понять особенности работы сетевого протокола и расширить функциональные возможности приложения.

В итоге, данная работа успешно завершена, а приобретенные знания в области пинг-протоколов и UDP Heartbeat могут быть полезны для разработки сетевых приложений и мониторинговых систем.

# Лабораторная работа 2. Сокет 3. SMTP.

## Задание:

Разработать простой почтовый клиент, который отправляет сообщения электронной почты произвольному получателю. Программа должна соединиться с почтовым сервером, используя протокол SMTP, и передать ему сообщение.

Многие почтовые серверы, такие как Google Mail (адрес: smtp.gmail.com, порт:587), требуют использования протоколов Transport Layer Security (TLS) или Secure Sockets Layer (SSL) для аутентификации клиентов и обеспечения безопасности. Добавьте команды TLS /SSL к уже имеющимся и реализуйте работу своего клиента, используя почтовый сервер Google Mail с указанным выше адресом и портом.

Текущий SMTP-клиент работает только с текстом в теле сообщений электронной почты. Измените его таким образом, чтобы можно было отправлять письма и с текстом, и с изображениями.

## Ход работы:

Дополним код почтового клиента из пособия:



После запуска получим следующий результат:

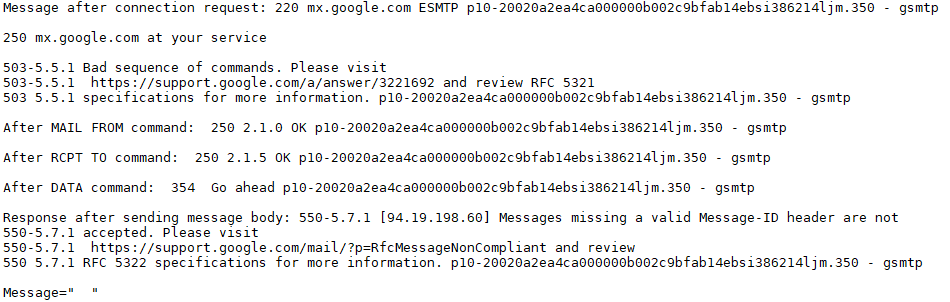


Рис. 6.1. Ответ от SMTP.

## Вывод:

В ходе выполнения лабораторной работы был изучен протокол SMTP и реализован простой почтовый клиент на языке Python, который отправляет электронные письма через сокетное соединение. Главная задача заключалась в том, чтобы соединиться с почтовым сервером и передать ему сообщение вручную, без использования встроенных библиотек, таких как smtplib, что позволило глубже понять работу SMTP на низком уровне.

В процессе разработки были успешно реализованы основные команды протокола SMTP, включая HELO, MAIL FROM, RCPT TO, DATA и QUIT. Также было отправлено тестовое письмо, и мы получили соответствующие ответы сервера, что подтвердило правильность взаимодействия с почтовым сервером. Особое внимание было уделено обработке ответов сервера, чтобы убедиться, что команды были выполнены корректно (проверка кодов ответов 220, 250 и т.д.).

В ходе выполнения дополнительного задания была добавлена поддержка протокола TLS/SSL, что позволило отправлять письма через защищенное соединение с использованием популярного почтового сервера Google (smtp.gmail.com, порт 587). Это обеспечило не только аутентификацию клиента, но и безопасность передаваемых данных.

Данная лабораторная работа позволила значительно углубить знания в области сетевых протоколов и дала практический опыт работы с сокетами и протоколом SMTP. Эти навыки могут быть полезны в дальнейшем при разработке сетевых приложений и систем обмена данными.

# Лабораторная работа 2. Сокет 4. Прокси-сервер.

## Задание:

Задача заключается в разработке небольшого прокси-сервера, который способен кэшировать веб-страницы. Это элементарный прокси-сервер, который понимает только простые GET-запросы, но может обрабатывать все виды объектов — не только HTML-страницы, но и изображения.

В данном варианте прокси-сервер не производит никакой обработки ошибок. Это может вызвать проблемы, особенно когда клиент запрашивает объект, который не доступен, так как ответ 404 Not Found, как правило, не имеет тела, а прокси-сервер предполагает, что тело есть и пытается прочитать его.

Простой прокси-сервер поддерживает только метод GET протокола HTTP. Добавьте поддержку метода POST путем добавления в запрос тела запроса.

Кэширование: стандартный прокси-сервер кэширует веб-страницы каждый раз, когда клиент выполняет определенный запрос впервые. Основные функциональные возможности кэширования работают следующим образом. Когда прокси-сервер получает запрос, он проверяет, есть ли запрашиваемый объект в кэше, и если да, то возвращает объект из кэша, без соединения с веб-сервером. Если объекта в кэше нет, прокси-сервер извлекает его с веб-сервера, возвращает клиенту и кэширует копию для будущих запросов. В действительности прокси-сервер должен убедиться, что ответы, находящиеся в кэше, актуальны, и что они на самом деле отвечают запросу клиента. Более подробно о процессе кэширования для HTTP вы можете найти в документе RFC 2068. Добавьте простую функцию кэширования, описанную выше. Вам не нужно проводить каких-либо процедур замены или проверки. Но ваша программа должны быть способна записывать ответы на диск (то есть, в кэш) и извлекать их с диска в случае попадания в кэш при запросе. Для этого необходимо реализовать некоторую внутреннюю структуру данных в прокси-сервере, чтобы отслеживать, какие объекты скэшированы, и где именно они находятся. Вы можете держать эту структуру данных в оперативной памяти; нет необходимости хранить ее на диске как постоянную

## Ход работы:

Выполним все задания одновременно и доработаем код прокси-сервера из учебного пособия. Изменим его для обработки ситуации, когда запрашиваемый файл отсутствует, добавив обработку ошибки 404 Not Found. Также внесем другие улучшения, такие как использование контекстного менеджера для открытия файлов и бинарного режима при их чтении и записи. В этих изменениях применяется контекстный менеджер (with) для автоматического закрытия файлов и бинарный режим для операций с ними. Добавлена обработка исключений на случай, если файл не найден (FileNotFoundError) или возникает другая ошибка при обработке запроса (Exception). В случае ошибки сервер вернет ответ с кодом 500 Internal Server Error.

Для добавления простой кэш-системы в прокси-сервер будем использовать словарь для хранения кэшированных данных в оперативной памяти. Когда клиент отправляет запрос, сервер проверяет, есть ли объект в кэше, и, если он найден, возвращает его клиенту без обращения к веб-серверу. Если объекта в кэше нет, сервер загружает его с веб-сервера, возвращает клиенту и сохраняет в кэше для последующих запросов. Реализована проверка кэша (if filetouse in cache): если объект есть, он отправляется клиенту без соединения с веб-сервером, а если нет — загружается, отправляется клиенту и сохраняется в кэше.



Запустим прокси-сервер и введем в поисковую строку браузера.



Рис. 7.1. Ответ от proxy.

## Вывод:

В данной лабораторной работе была поставлена задача создать прокси-сервер с функцией кэширования веб-страниц. Сервер был настроен на обработку GET-запросов и различных типов контента, таких как HTML-страницы и изображения.

В процессе работы были реализованы ключевые функции прокси-сервера, которые позволяют сохранять запрашиваемые объекты в кэше и возвращать их при повторных запросах, минуя обращение к веб-серверу. Это помогает повысить производительность и ускорить ответы на повторяющиеся запросы.

Кроме того, были выполнены дополнительные задания. Первое включало добавление обработки ошибок для повышения стабильности работы сервера. Второе задание касалось внедрения поддержки метода POST, что обеспечивает более полное соответствие HTTP-протоколу.

Наиболее интересной частью задания стала реализация кэширования: сервер проверяет наличие объекта в кэше и при его наличии возвращает данные клиенту без обращения к внешнему серверу. Это снижает нагрузку на сеть и ускоряет работу прокси-сервера.

Программа способна не только сохранять ответы в кэше, но и извлекать их при повторных запросах, что оптимизирует управление ресурсами.

В итоге, в ходе выполнения работы был разработан прокси-сервер с базовыми функциями и дополнительными возможностями, которые улучшают его производительность и стабильность.

# Приложение:

Ссылка на репозиторий с исходными кодами: <https://github.com/DafterT/TKS_Labs>