

# HTTP Traffic Analysis Using Wireshark

*(Portfolio Project – Network Traffic Analysis & Packet Inspection)*

## Overview

This project demonstrates a full HTTP packet analysis using Wireshark. I captured network traffic while visiting a webpage, then analyzed the HTTP GET request and the server's HTTP 200 OK response. This exercise highlights my ability to perform packet-level inspection, extract protocol metadata, measure response times, and interpret how multiple network layers interact.

## Tools Used

- Wireshark
- Windows 10
- Microsoft Edge Browser

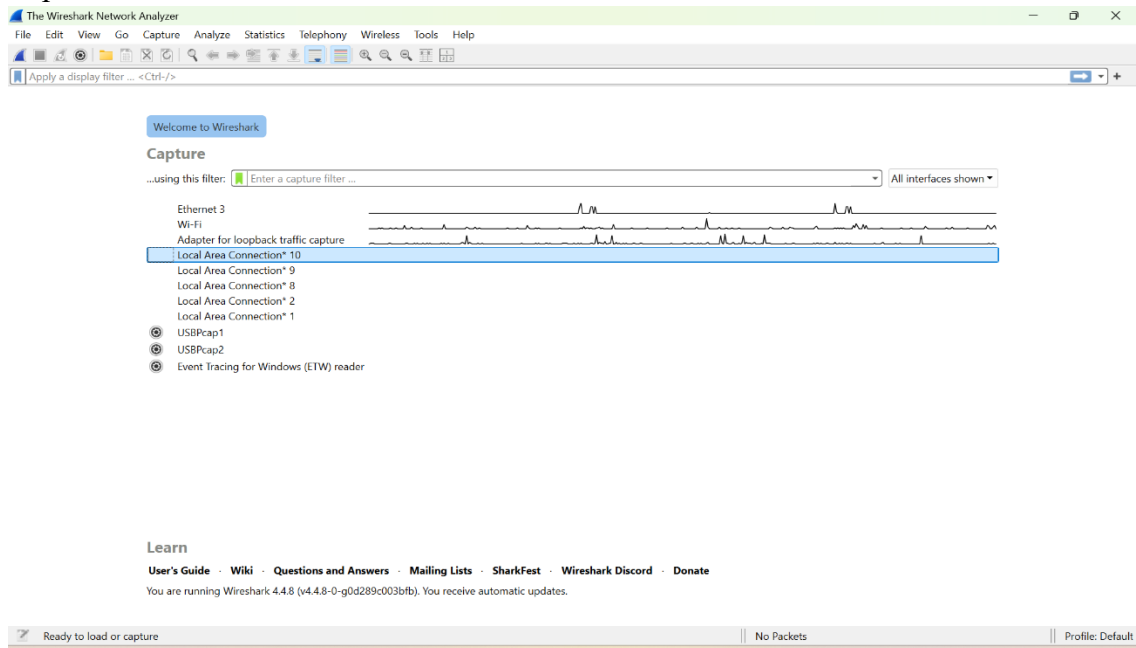
## Objectives

- Capture real network packets
- Filter and analyze HTTP traffic
- Examine protocol layers (Ethernet → IP → TCP → HTTP)
- Identify client/server IPs, ports, browser metadata
- Measure HTTP response time
- Produce clear evidence-based documentation

## Project Implementation

### 1. Launching Wireshark

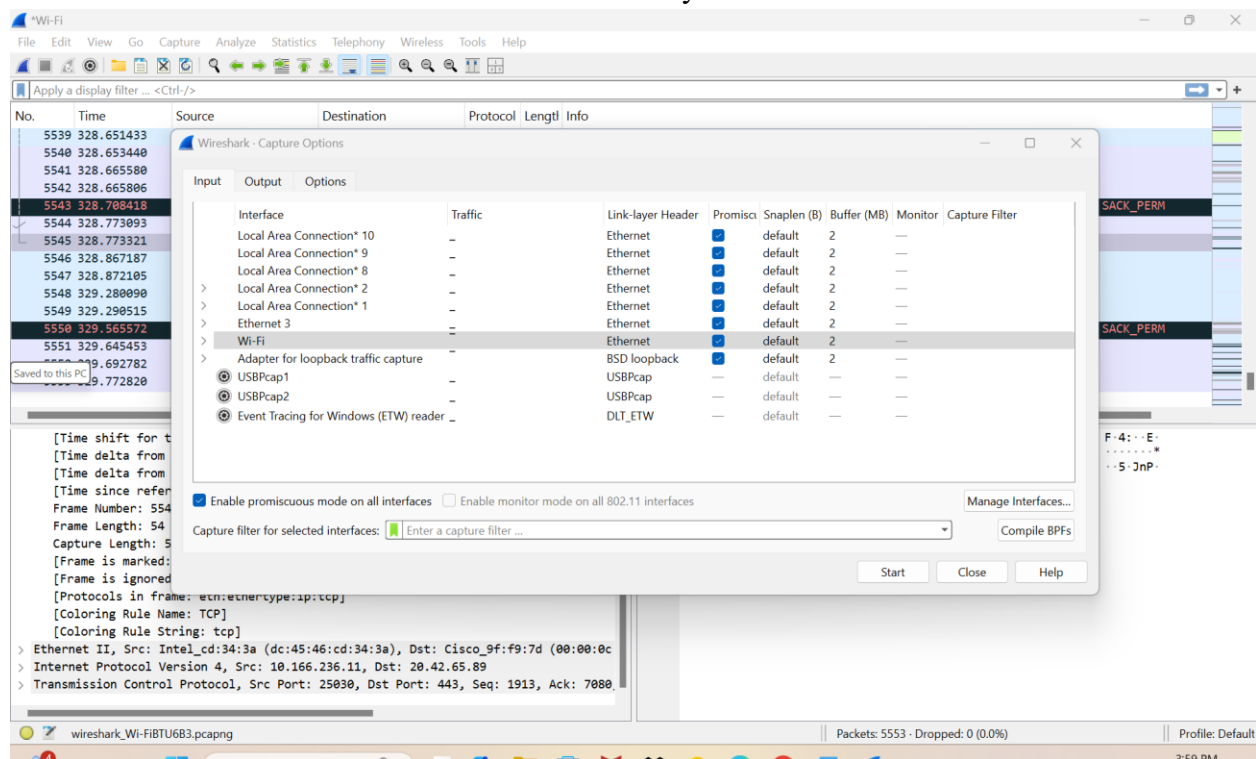
I opened Wireshark and viewed the list of available network interfaces.



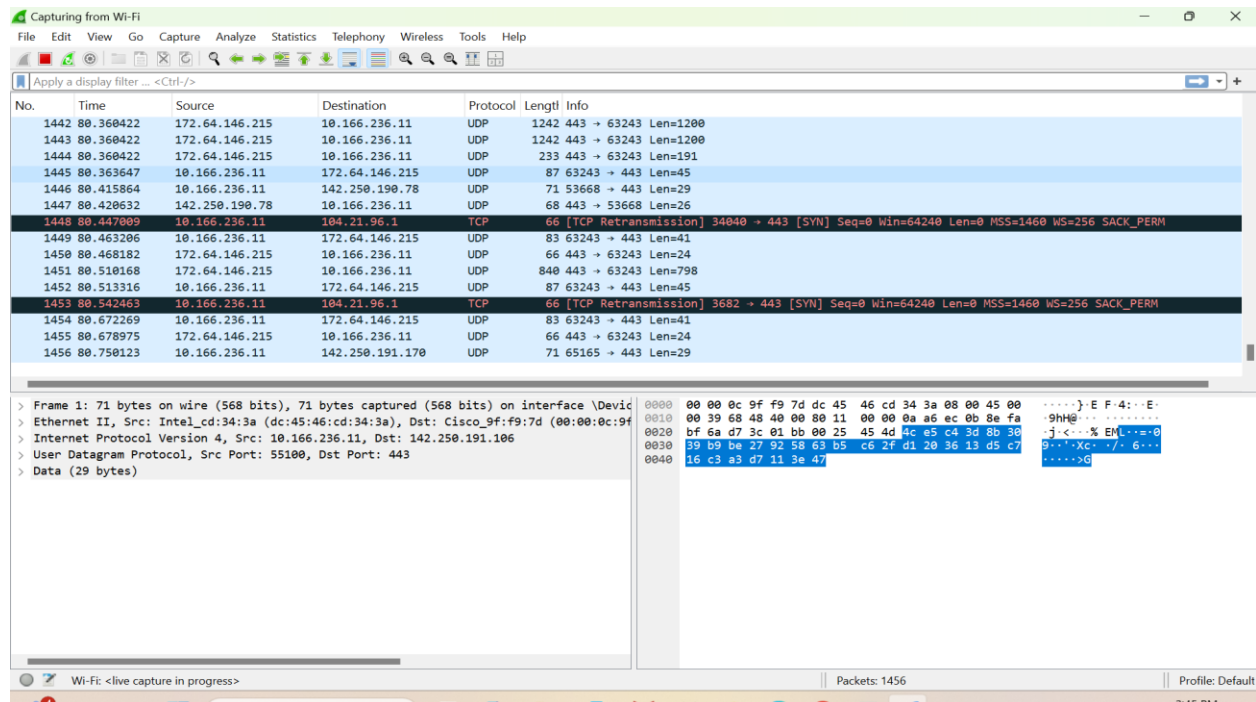
Screenshots/wireshark-startup.png

## 2. Selecting the Correct Interface

I selected the **Wi-Fi** interface where network activity would occur.



Screenshots/wireshark-interfaces1.png

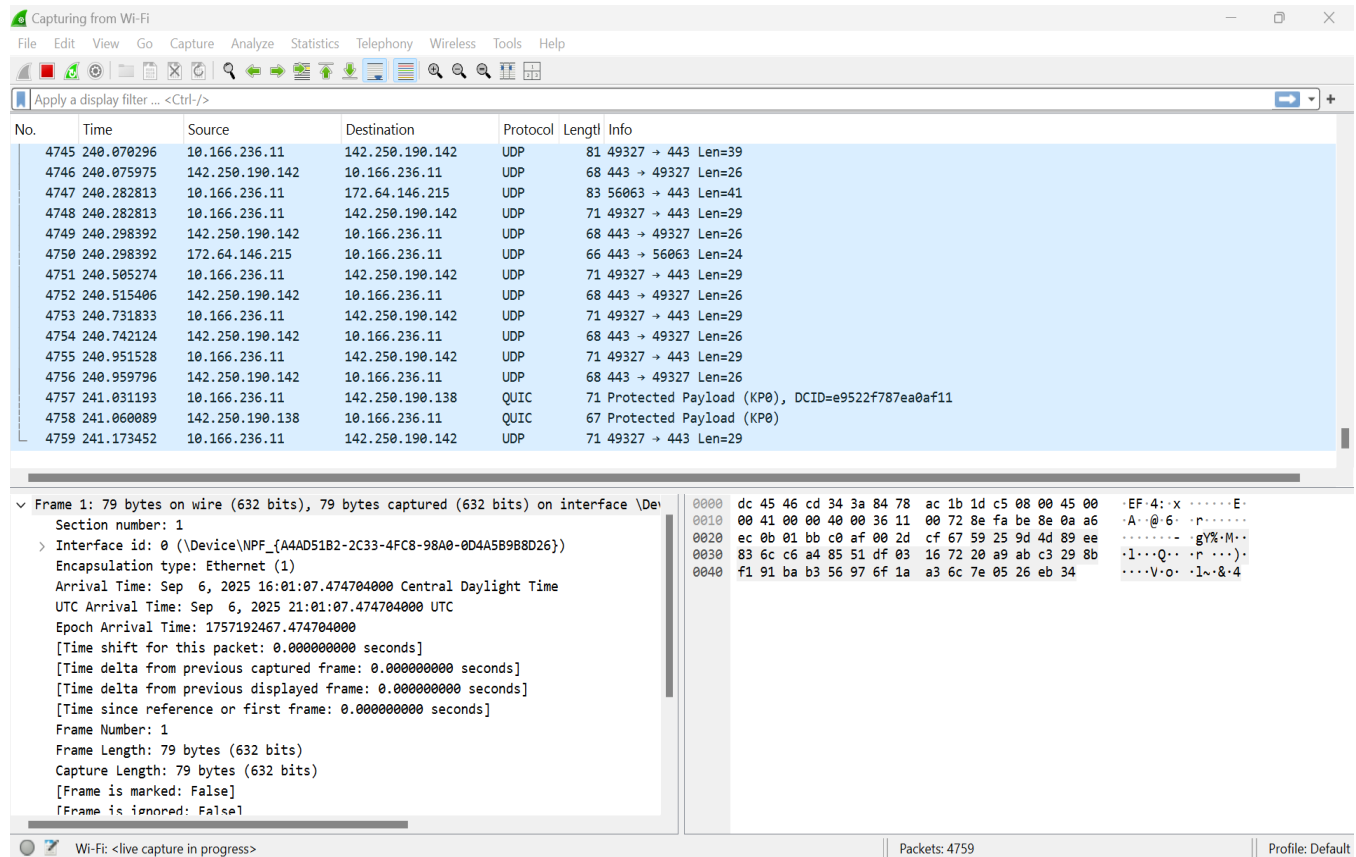


Screenshots/wireshark-interfaces2.png

### 3. Starting the Packet Capture

Once the interface was selected, I began capturing traffic.

Wireshark immediately displayed live packets in real time.



Screenshots/wireshark-live-capture.png

### 4. Generating HTTP Traffic

I navigated to:

<http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html>

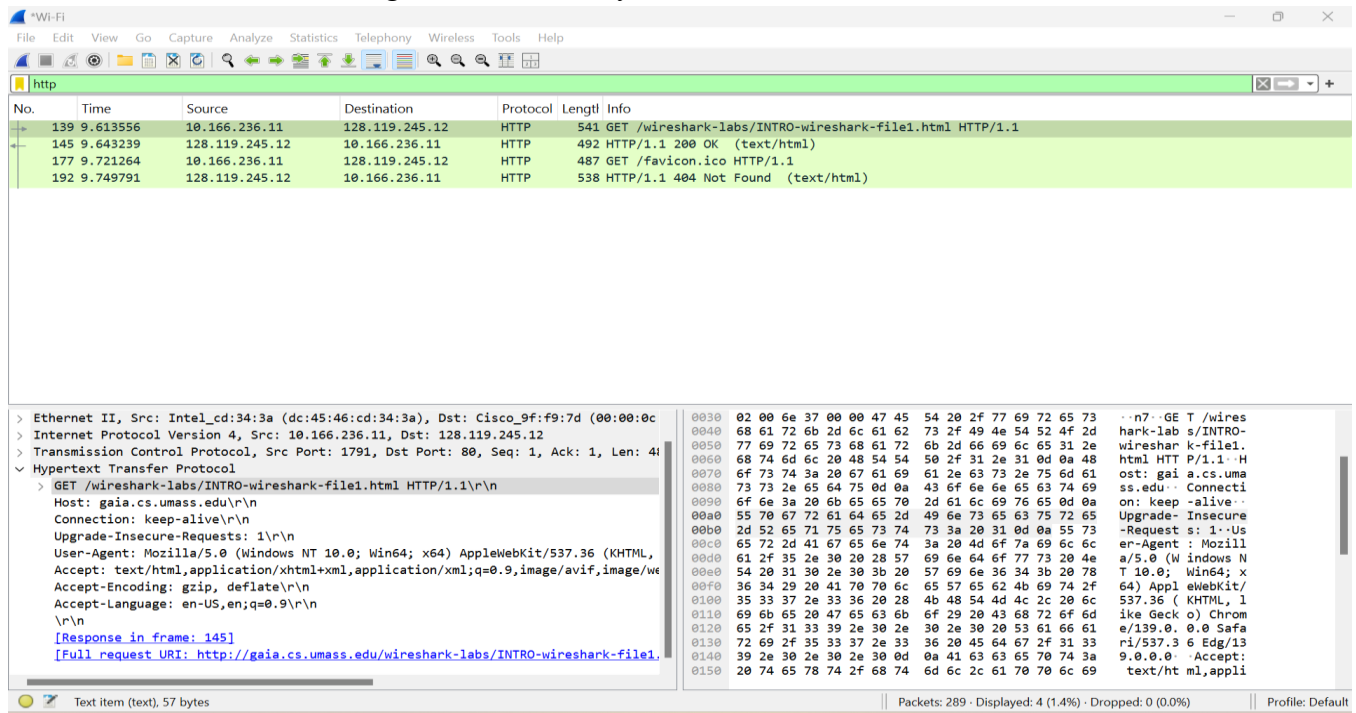
This produced a clear HTTP GET and HTTP OK exchange between my computer and the server.

## 5. Filtering HTTP Packets

I applied the filter:

http

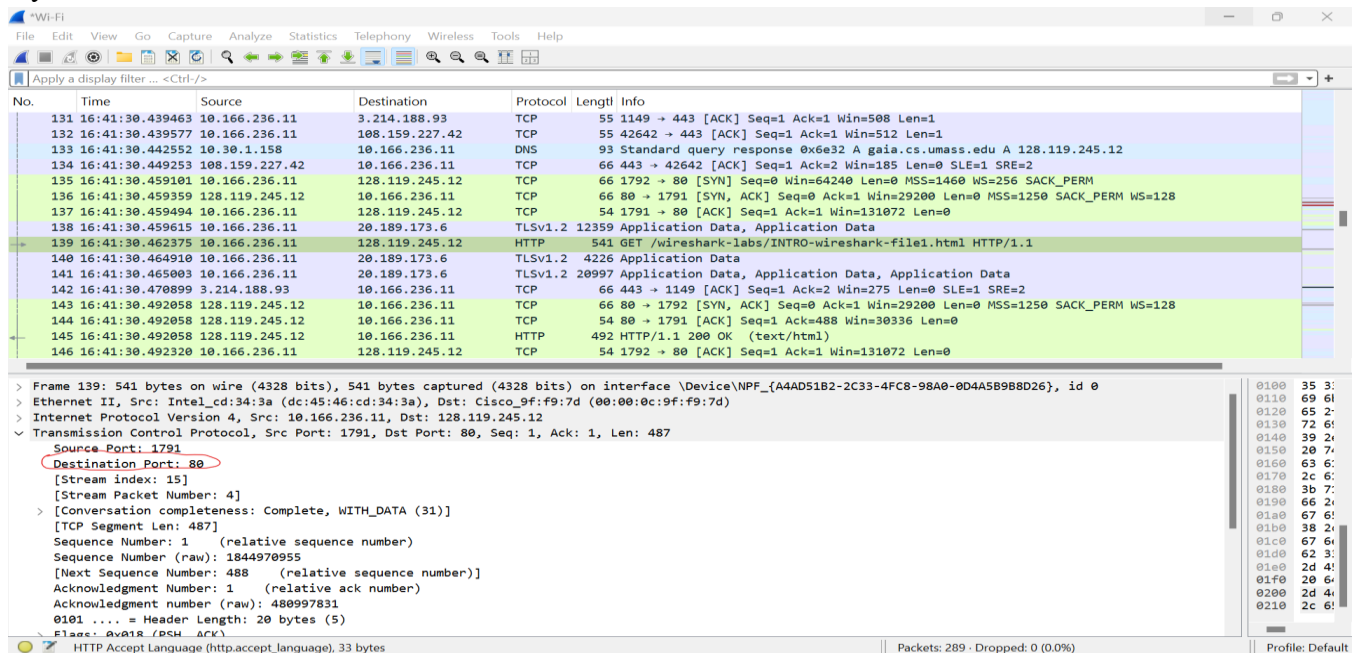
This isolated all HTTP messages for easier analysis.



Screenshots/http-filter.png

## 6. Identifying the HTTP GET Request

I located and expanded the GET request within Wireshark, reviewing its structure across the protocol layers.



Screenshots/http-get-details.png

## 7. Inspecting the HTTP 200 OK Response

I then examined the corresponding HTTP OK response returned by the server.

The image shows a Wireshark packet capture window. The top pane displays a list of captured packets. Packet 145 is highlighted, showing an HTTP 200 OK response from 128.119.245.12 to 10.166.236.11. The bottom pane shows the details of this packet, including the HTTP headers and the request URI. The status bar at the bottom indicates that 289 packets are displayed, with 4 (1.4%) dropped.

| No. | Time            | Source         | Destination    | Protocol | Length | Info  |
|-----|-----------------|----------------|----------------|----------|--------|---|
| 139 | 16:41:30.462375 | 10.166.236.11  | 128.119.245.12 | HTTP     | 541    | GET /wireshark-labs/INTRO-wireshark-file1.html HTTP/1.1 |
| 145 | 16:41:30.492058 | 128.119.245.12 | 10.166.236.11  | HTTP     | 492    | HTTP/1.1 200 OK (text/html)                             |
| 177 | 16:41:30.570083 | 10.166.236.11  | 128.119.245.12 | HTTP     | 487    | GET /favicon.ico HTTP/1.1                               |
| 192 | 16:41:30.598610 | 128.119.245.12 | 10.166.236.11  | HTTP     | 538    | HTTP/1.1 404 Not Found (text/html)                      |

```
> HTTP/1.1 200 OK\r\n
Date: Sat, 06 Sep 2025 21:41:30 GMT\r\n
Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/7.4.33 mod_perl/2.0.11 Perl/v5.16.3\r\n
Last-Modified: Sat, 06 Sep 2025 05:59:01 GMT\r\n
ETag: "51-63e1ba6a9b409"\r\n
Accept-Ranges: bytes\r\n
> Content-Length: 81\r\n
Keep-Alive: timeout=5, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=UTF-8\r\n
\r\n
[Request in frame: 139]
[Time since request: 0.029683000 seconds]
[Request URI: /wireshark-labs/INTRO-wireshark-file1.html]
[Full request URI: http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html]
File Data: 81 bytes
```

Screenshots/http-get-ok.png

## Findings

### 1. Protocols Observed

During the capture, the following protocols appeared:

- DNS
- TCP
- TLSv1.2
- HTTP

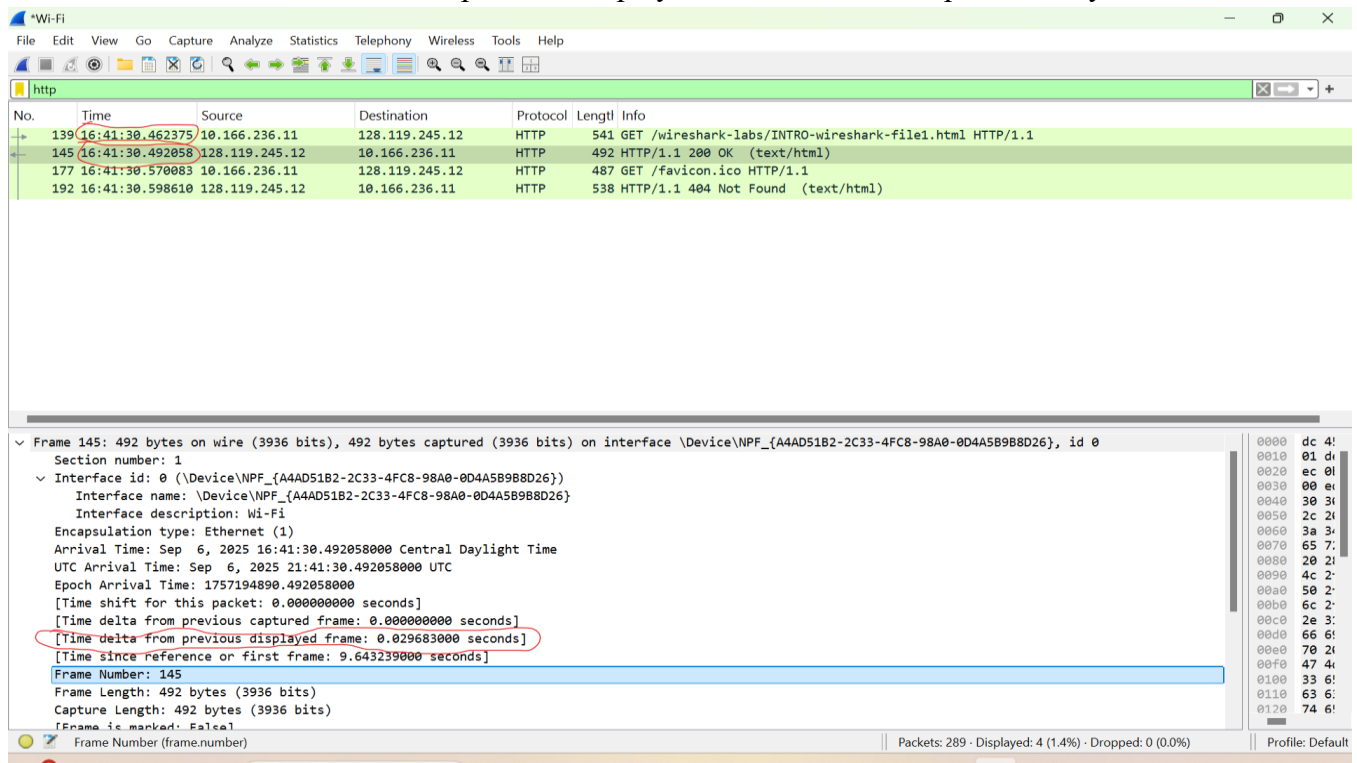
### 2. HTTP Response Time Measurement

Using Wireshark timestamps:

- **GET request sent:** 16:41:30.462375
- **HTTP OK received:** 16:41:30.492058

**Response Time** = 0.029683 seconds

This matches the “Time delta from previous displayed frame” within the packet analysis.

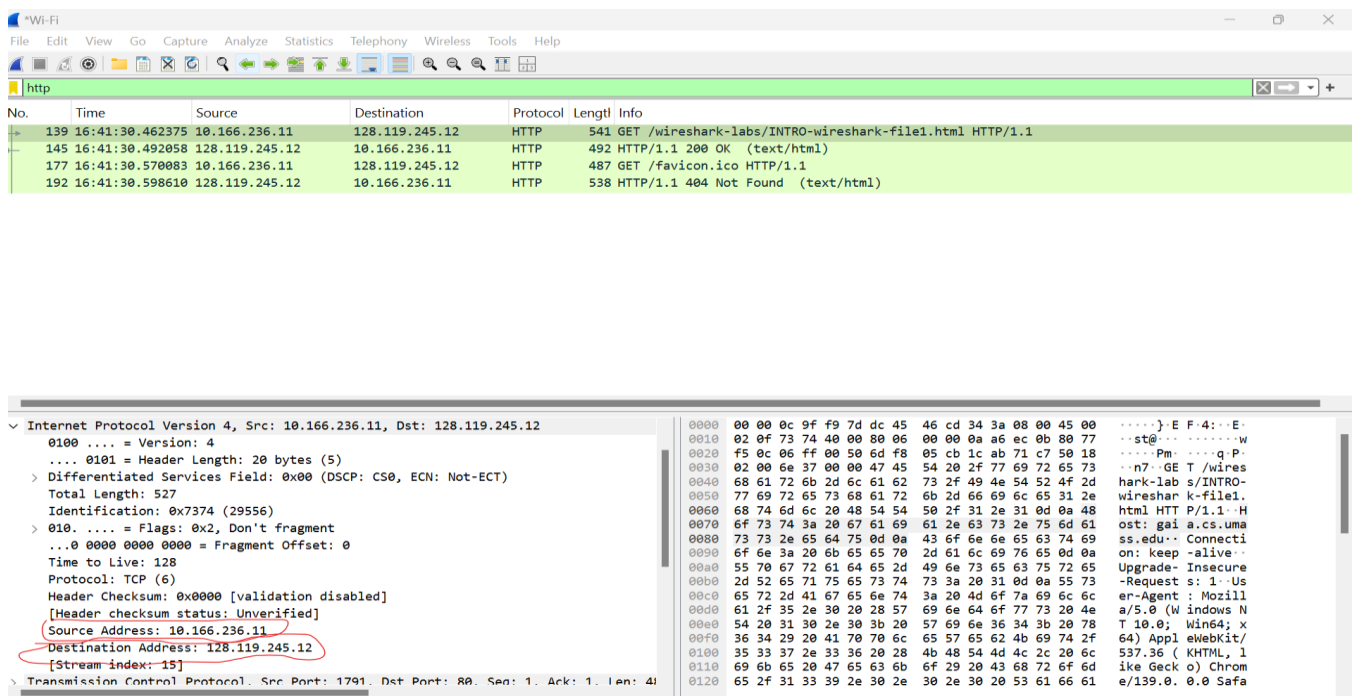


Screenshots/http-time-calc.png

### 3. IP Addresses Identified

- Client IP: 10.166.236.11
- Server IP (gaia.cs.umass.edu): 128.119.245.12

These were extracted from the IP header fields.



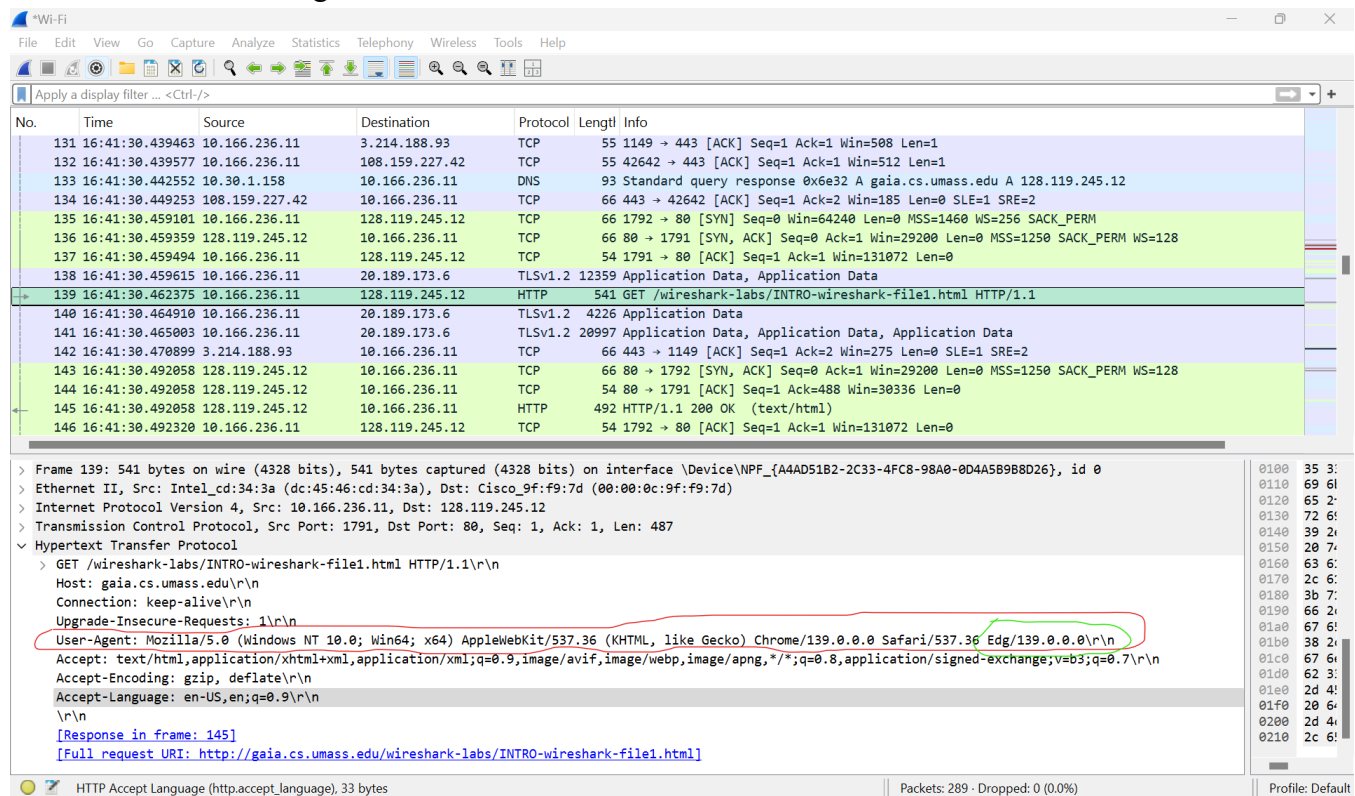
Screenshots/ip-identification.png



## 4. Browser Information (User-Agent)

From the HTTP GET request header, the User-Agent field showed:

**Browser:** Microsoft Edge



The screenshot shows a Wireshark packet capture of an HTTP GET request. The packet list on the left shows packet 139 selected, which is an HTTP GET request to /wireshark-labs/INTRO-wireshark-file1.html. The packet details pane on the right shows the request structure, with the User-Agent field highlighted in red. The User-Agent string is: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/139.0.0.0 Safari/537.36 Edg/139.0.0.0. The packet bytes pane at the bottom shows the raw data of the request.

Screenshots/browser-information.png

## 5. Destination TCP Port

The HTTP GET request was sent to:

**Port 80** (standard port for HTTP traffic)

## Skills Demonstrated

- Packet capture and protocol inspection
- HTTP GET/Response analysis
- Filtering and isolating traffic
- Response-time calculation
- Metadata extraction (IP, ports, browser)
- Understanding of TCP/IP and application-layer behavior
- Professional documentation and reporting

## Conclusion

This project provided practical experience in network traffic analysis using Wireshark. By capturing and analyzing a real HTTP exchange, I demonstrated an understanding of protocol interactions across multiple layers, the structure of HTTP communication, and essential skills used in SOC investigations, blue-team network monitoring, and cybersecurity analysis.