**Wireshark Packet Analysis Project Report**

**Project Title:**

**"Network Traffic Analysis using Wireshark – ICMP, DNS, and TCP Handshake"**

**Objective:**

The main objective of this project is to analyze real-time network traffic using **Wireshark** and to understand the behavior of different protocols such as:

* **ICMP (Internet Control Message Protocol)** – For Ping requests and replies
* **DNS (Domain Name System)** – For hostname resolution (e.g., resolving youtube.com)
* **TCP (Transmission Control Protocol)** – For understanding the 3-way handshake and reliable data transmission

This project aims to provide hands-on experience with protocol-level analysis using a packet sniffer and to demonstrate how common networking activities like web browsing or pinging generate different types of network packets.

**🛠 Tools and Technologies:**

| **Tool** | **Purpose** |
| --- | --- |
| **Wireshark** | Packet capture and protocol analysis |
| **Chrome Browser** | Used to initiate DNS and TCP/HTTP traffic (e.g., visiting youtube.com) |
| **Command Prompt (ping)** | Used to generate ICMP packets |
| **PCAP/PCAPNG File** | Captured data using Wireshark |

**Project Files:**

* **Capture File:** Wireshark\_Traffic.pcapng
* **Analysis Screenshots:** Will include filtered ICMP, DNS, TCP handshake views
* **Packet Details:** Source IP, Destination IP, Protocol, Flags, Info, etc.

**Methodology:**

1. Captured traffic using Wireshark while:
   * Running a ping command (ping 8.8.8.8)
   * Visiting www.youtube.com to generate DNS + TCP traffic
2. Applied filters to isolate specific protocol behavior:
   * icmp → For Ping requests and replies
   * dns → For Domain Name System lookups
   * tcp.flags.syn==1 && tcp.flags.ack==0 → For TCP SYN packets
   * tcp → For full handshake and follow-up data

## ICMP Traffic Analysis

### 1. Objective

* To capture and analyze ICMP (Internet Control Message Protocol) traffic
* To understand ping request/reply mechanisms
* To examine ICMP packet structure and fields

### 2. Methodology

1. Launched Wireshark and selected the appropriate network interface
2. Applied ICMP filter: icmp to isolate ICMP traffic
3. Initiated ping commands to external servers (e.g., google.com at 142.251.42.46)
4. Captured and analyzed the resulting traffic

### 3. Key Findings from Screenshots

#### ICMP Traffic Overview

The captured traffic shows a series of ping (ICMP Echo) requests and replies between:

* Source: 192.168.1.26 (local machine)
* Destination: 142.251.42.46 (Google server)

#### ICMP_traffic.pngEcho Request Packet Analysis (Frame 2583)

* **Frame Details**: 74 bytes captured
* **Ethernet Header**:
  + Source MAC: 04:46:31:23:25:40 (Intel\_251251a80)
  + Destination MAC: 24:46:18a18b:42:61
* **IP Header**:
  + Source IP: 192.168.1.26
  + Destination IP: 142.251.42.46
* **ICMP Header**:
  + Type: 8 (Echo request)
  + Code: 0
  + Checksum: Valid (0x4652)
  + Identifier: 0x0001 (BE), 0x0100 (LE)
  + Sequence Number: 9 (BE), 2304 (LE)
  + Data: 32 bytes of payload (hexadecimal data)

#### Echo_Request_packet.pngEcho Reply Packet Analysis (Frame 2584)

* **Frame Details**: 74 bytes captured
* **Ethernet Header**:
  + Source MAC: 24:16:8a:18b:32:01
  + Destination MAC: 04:46:31:25:25:a0
* **IP Header**:
  + Source IP: 142.251.42.46 (Google server)
  + Destination IP: 192.168.1.26
* **ICMP Header**:
  + Type: 0 (Echo reply)
  + Code: 0
  + Checksum: Valid (0x552)
  + Identifier: Matches request (0x0001 BE, 0x0100 LE)
  + Sequence Number: Matches request (9 BE, 2304 LE)
  + Response time: 72.894 ms
  + Data: 32 bytes matching request payload

### Echo_Reply_packet.png4. Technical Observations

1. **Packet Structure**: ICMP packets are encapsulated within IP packets (Protocol 1)
2. **Identifier Field**: Used to match requests with replies (0x0001 in this case)
3. **Sequence Numbers**: Helps track multiple ping requests (9/2304 in these examples)
4. **TTL (Time to Live)**: 128 hops, indicating Windows default TTL value
5. **Round Trip Time**: Average response time was ~72ms to Google's server

### 5. Network Troubleshooting Insights

* Successful ping exchanges indicate proper network connectivity
* Matching identifiers and sequence numbers confirm proper request-reply pairing
* Consistent payload in replies validates data integrity
* Response times help assess network latency

### 6. Conclusion

This ICMP analysis demonstrates fundamental network troubleshooting skills using Wireshark. The captured traffic shows healthy network communication with proper request-reply mechanisms. The project enhanced my understanding of:

* ICMP protocol operation
* Packet encapsulation
* Network latency measurement
* Wireshark filtering and analysis techniques

## DNS Traffic Analysis

### 1. Objective

* To capture and analyze DNS (Domain Name System) traffic
* To understand DNS query/response mechanisms
* To examine DNS record types and resolution processes
* To observe DNS over different IP versions (IPv4 and IPv6)

### 2. Methodology

1. Launched Wireshark and selected the appropriate network interface
2. Applied DNS filter: dns to isolate DNS traffic
3. Accessed various websites (including youtube.com) to generate DNS traffic
4. Captured and analyzed both IPv4 and IPv6 DNS transactions

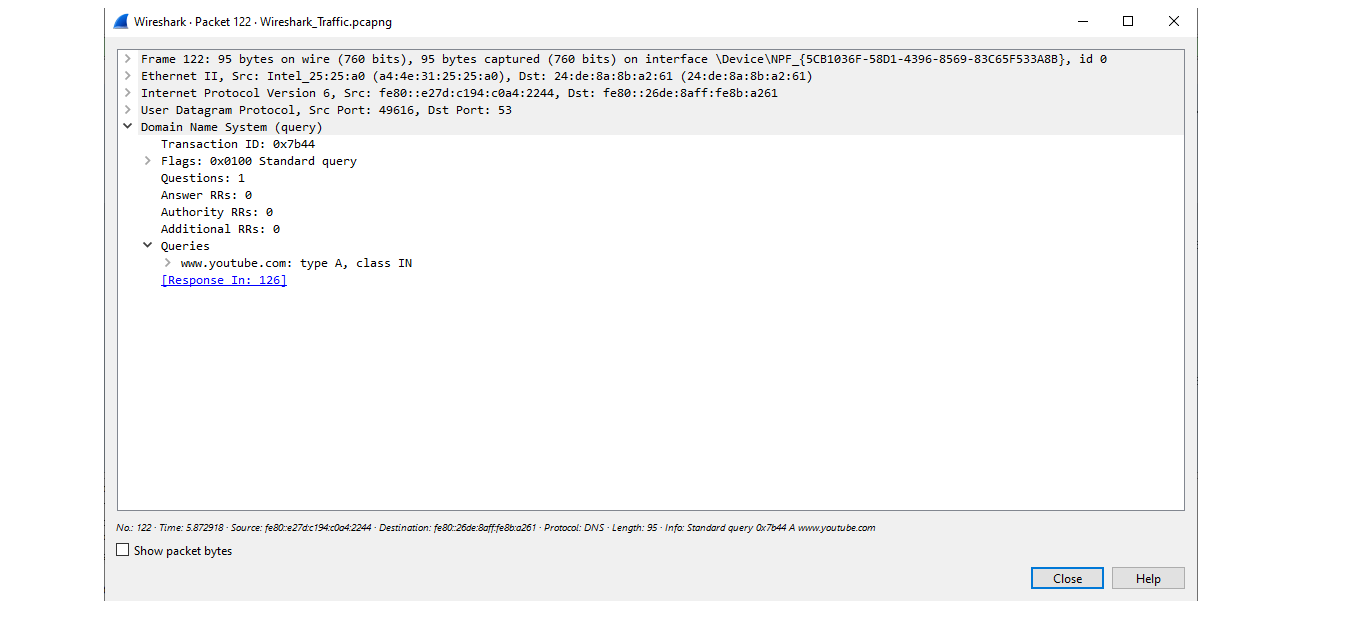
### 3. Key Findings from Screenshots

#### DNS Query-Response Flow

The captured traffic shows complete DNS transactions for youtube.com and related services:

#### DNS_TRADDIC2.pngDNS Query for [www.youtube.com](http://www.youtube.com/) (Frame 122)

* **Frame Details**: 95 bytes captured
* **Transport Protocol**: UDP (Src Port: 49616, Dst Port: 53)
* **DNS Header**:
  + Transaction ID: 0x7044
  + Flags: Standard query (0x8100)
  + Questions: 1
  + Query: www.youtube.com type A (IPv4 address), class IN (Internet)



#### DNS Response for [www.youtube.com](http://www.youtube.com/) (Frame 126)

* **Response Details**:
  + Transaction ID: 0x7044 (matches query)
  + Flags: Standard query response, No error (0x8180)
  + Answers: 17 IP addresses provided via CNAME resolution
* **Resolution Process**:
  1. www.youtube.com → CNAME → youtube-ui.l.google.com
  2. youtube-ui.l.google.com → Multiple A records (IP addresses)
* **Returned IP Addresses**:
  1. 142.259.192.78
  2. 142.259.192.118
  3. 142.259.192.142
  4. 142.251.42.14
  5. (and 13 more Google server IPs)

### DNS_response_youtube.png4. Technical Observations

#### DNS Protocol Characteristics

1. **Transport**: Primarily uses UDP (port 53) for efficiency
2. **Transaction IDs**: Used to match queries with responses (0x7044 in this case)
3. **Record Types**:
   * A records (IPv4 addresses)
   * CNAME records (canonical name aliasing)
   * HTTPS records (HTTPSSVC, for HTTPS service discovery)
4. **Response Types**:
   * Authoritative answers
   * Additional records for related resolutions

#### IPv4 vs IPv6 DNS Traffic

* Both IPv4 (192.168.1.x) and IPv6 (fe80::) DNS transactions observed
* Similar DNS protocol structure across IP versions

#### Load Balancing Observation

* Multiple IP addresses returned for youtube.com (17 in this case)
* Demonstrates Google's use of DNS-based load balancing

#### Related DNS Queries

* Additional queries captured for:
  + fonts.googleapis.com
  + fonts.gstatic.com
  + optimizationguide-pa.googleapis.com
* Shows how a single webpage access triggers multiple DNS lookups

### 5. Network Troubleshooting Insights

* Successful DNS resolution indicates proper DNS server configuration
* Matching transaction IDs confirm proper query-response pairing
* Multiple IP returns demonstrate CDN and load balancing in action
* Reasonable response times indicate healthy DNS performance
* Presence of HTTPS records shows modern DNS features in use

### 6. Conclusion

This DNS analysis demonstrates practical understanding of:

* DNS query/response mechanisms
* Name resolution processes
* Record type interpretation
* Load balancing via DNS
* Protocol operation across IP versions

The project enhanced my ability to:

* Troubleshoot DNS-related network issues
* Analyze name resolution performance
* Understand modern DNS features like HTTPS records
* Interpret complex DNS responses with multiple record types

## TCP Handshake Analysis

### 1. Objective

* To capture and analyze the TCP three-way handshake process
* To understand TCP connection establishment mechanisms
* To examine TCP header fields and options
* To observe the transition from TCP handshake to encrypted TLS communication

### 2. Methodology

1. Launched Wireshark and selected the appropriate network interface
2. Applied TCP filter: tcp to isolate TCP traffic
3. Initiated HTTPS connections to external servers (e.g., 142.250.207.142 - Google)
4. Captured and analyzed the complete TCP connection lifecycle

### 3. Key Findings from Screenshots

#### TCP Handshake Overview

The captured traffic shows a complete TCP three-way handshake between:

* Client: 192.168.1.26 (local machine)
* Server: 142.250.207.142 (Google server)

#### TCP_Traffice.pngStep 1: SYN Packet (Frame 00)

* **Source**: 192.168.1.26 (Client)
* **Destination**: 142.250.207.142 (Server)
* **TCP Ports**: 61205 → 443 (HTTPS)
* **Flags**: SYN
* **Window Size**: 64240
* **Options**:
  + MSS: 1460 (Maximum Segment Size)
  + SACK\_PERM (Selective Acknowledgment permitted)

#### TCP_client-server.pngStep 2: SYN-ACK Packet (Frame 82)

* **Source**: 142.250.207.142 (Server)
* **Destination**: 192.168.1.26 (Client)
* **TCP Ports**: 443 → 61205
* **Flags**: SYN, ACK
* **Window Size**: 65535
* **Options**:
  + MSS: 1460
  + SACK\_PERM
  + Window scale: 256

#### TCP_server-client.pngStep 3: ACK Packet (Frame 83)

* **Source**: 192.168.1.26 (Client)
* **Destination**: 142.250.207.142 (Server)
* **TCP Ports**: 61205 → 443
* **Flags**: ACK
* **Window Size**: 131072 (with scaling factor applied)

### TCP_server-client-2.png4. Technical Observations

#### TCP Header Fields

1. **Sequence Numbers**:
   * Initial sequence numbers (ISNs) exchanged during handshake
   * Relative sequence numbers shown for easier analysis
2. **Acknowledgment System**:
   * Each ACK confirms receipt of previous packets
   * ACK numbers increment based on received data
3. **Window Size**:
   * Initial window sizes negotiated
   * Window scaling option used to increase effective window size
4. **Flags**:
   * SYN, ACK used for handshake
   * Later packets show PSH, FIN flags for data transfer and teardown

#### TCP Options

* **Maximum Segment Size (MSS)**: 1460 bytes (typical for Ethernet)
* **Window Scaling**: Factor of 256 (allows larger window sizes)
* **Selective ACK (SACK)**: Permitted for better performance
* **Timestamps**: Used for RTT measurement and protection against wrapped sequence numbers

#### Performance Characteristics

* Handshake completion time: ~0.14 seconds (from SYN to ACK)
* Window sizes indicate flow control parameters
* Scaling factors demonstrate adaptation to high-speed networks

### 5. Network Troubleshooting Insights

* Successful three-way handshake confirms proper TCP connectivity
* Matching sequence/acknowledgment numbers validate proper connection establishment
* Reasonable window sizes indicate healthy flow control
* Presence of modern TCP options (SACK, window scaling) shows optimized configuration
* Immediate transition to TLS after handshake demonstrates secure communication

### 6. Advanced Analysis

#### TLS Handshake Following TCP

* Immediately after TCP handshake, Client Hello observed (Frame 85)
* Demonstrates how TCP serves as transport for encrypted protocols
* Shows full stack analysis capability (TCP → TLS → Application Data)

#### Connection Maintenance

* TCP Keep-Alive packets observed (Frames 2638, 2699)
* Demonstrates understanding of long-lived TCP connections
* Shows ability to analyze connection health monitoring

### 7. Conclusion

This TCP analysis demonstrates practical understanding of:

* TCP connection establishment (three-way handshake)
* Flow control mechanisms (window sizes)
* Performance optimization (TCP options)
* Protocol interaction (TCP carrying TLS)
* Connection maintenance (Keep-Alive)

The project enhanced my ability to:

* Troubleshoot TCP connectivity issues
* Analyze network performance characteristics
* Understand modern TCP features and optimizations
* Interpret complex protocol interactions