Bialystok University of Technology
Faculty of Electrical Engineering

LABORATORY REPORT

Computer Networks
IS-FEE-10082S

Subject:

Analysis of the operation of TCP/IP family protocols

Instructor: Andrzej Zankiewicz, PhD

Students:
Burak Düzenli
Bahadır Emre Yıldırım

Białystok, April 2025

Contents

1 Objective **3**

2 Equipment and Software Used **3**

3 Experiment 1 - IP configuration **3**
3.1 Setup Parameters 3

4 Experiment 2 - ICMP Packet Analysis with Ping **3**
4.1 Setup Parameters 3
4.2 Wireshark Screenshots 4
4.3 Results 4
4.4 Discussion 4

5 Experiment 3 - Traceroute Analysis **4**
5.1 Setup Parameters 4
5.2 Wireshark Screenshots 4
5.3 Results 5
5.4 Discussion 5

6 Experiment 4 - IP Fragmentation Analysis **5**
6.1 Setup Parameters 5
6.2 First Packet 5
6.3 IP Header 5
6.4 ICMP Header 5
6.5 Fragmentated Packet 6
6.6 Key Observations 6
6.7 Discussion 6

7 Experiment 5 - DF Flag and MTU Exceeded **6**
7.1 Setup Parameters 6
7.2 CMD Output 6
7.3 Discussion 7

8 Experiment 6 - ARP Message Analysis in LAN Communication **7**
8.1 Setup Parameters 7
8.2 Same Network Ping 7
8.3 Different Network Ping 7
8.4 Wireshark Screenshots 8
8.5 Discussion 8

9 Experiment 7 - DNS Communication using UDP **9**
9.1 Setup Parameters 9
9.2 CMD Output 9

9.3 Wireshark Analysis 9
9.4 Discussion 10
9.5 Wireshark Analysis 14

10 References **14**

1 Objective

The objective of this lab was to analyze network protocols using Wireshark, focusing on IP fragmentation, ICMP (ping) exchanges, and packet structure interpretation. By sending large ICMP packets and observing their fragmentation behavior, we aimed to understand how networks handle data exceeding the MTU size, particularly when the "Don't Fragment" (DF) flag is set. Additionally, we examined ARP and DNS operations to explore IP-to-MAC resolution and domain name queries in a controlled environment.

2 Equipment and Software Used

- Microsoft Windows 11
- Wireshark (Latest version)
- Access to LAN/Internet
- Command-line interface (cmd / terminal)

3 Experiment 1 - IP configuration

3.1 Setup Parameters

The command `ipconfig /all` was used to retrieve the IP configuration.

```
ipconfig /all
```

Listing 1: Windows IP Configuration

Result:

Discussion: This command displays details such as IP address, MAC address, subnet mask, gateway, and DHCP status. It is crucial for troubleshooting IP-related issues.

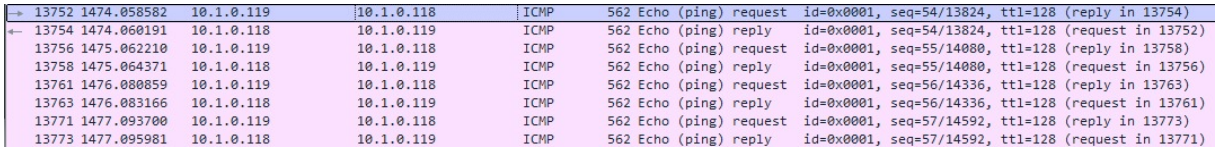
4 Experiment 2 - ICMP Packet Analysis with Ping

4.1 Setup Parameters

- Source IP: 10.1.0.119

- Destination IP: 10.1.0.118
- Wireshark filter: ICMP

4.2 Wireshark Screenshots



→	13752 1474.058582	10.1.0.119	10.1.0.118	ICMP	562 Echo (ping) request	id=0x0001, seq=54/13824, ttl=128 (reply in 13754)
←	13754 1474.060191	10.1.0.118	10.1.0.119	ICMP	562 Echo (ping) reply	id=0x0001, seq=54/13824, ttl=128 (request in 13752)
	13756 1475.062210	10.1.0.119	10.1.0.118	ICMP	562 Echo (ping) request	id=0x0001, seq=55/14080, ttl=128 (reply in 13758)
	13758 1475.064371	10.1.0.118	10.1.0.119	ICMP	562 Echo (ping) reply	id=0x0001, seq=55/14080, ttl=128 (request in 13756)
	13761 1476.080859	10.1.0.119	10.1.0.118	ICMP	562 Echo (ping) request	id=0x0001, seq=56/14336, ttl=128 (reply in 13763)
	13763 1476.083166	10.1.0.118	10.1.0.119	ICMP	562 Echo (ping) reply	id=0x0001, seq=56/14336, ttl=128 (request in 13761)
	13771 1477.093700	10.1.0.119	10.1.0.118	ICMP	562 Echo (ping) request	id=0x0001, seq=57/14592, ttl=128 (reply in 13773)
	13773 1477.095981	10.1.0.118	10.1.0.119	ICMP	562 Echo (ping) reply	id=0x0001, seq=57/14592, ttl=128 (request in 13771)

4.3 Results

- Observed ICMP Types: 8 (Request), 0 (Reply)
- TTL value: 128

4.4 Discussion

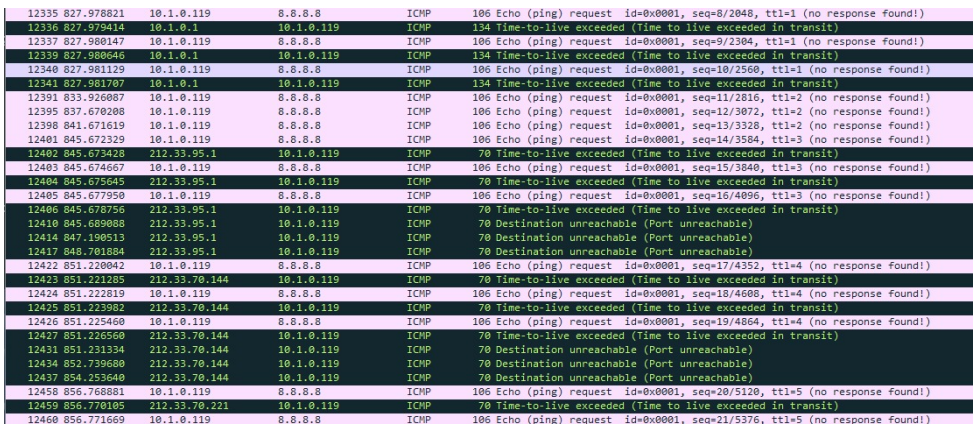
- The ICMP exchange confirms network connectivity.
- No packet loss indicates a stable connection.
- TTL decrement confirms proper routing (if passing through a gateway).

5 Experiment 3 - Traceroute Analysis

5.1 Setup Parameters

- Source IP: 10.1.0.119
- Destination: 8.8.8.8
- Wireshark filter: ICMP

5.2 Wireshark Screenshots



12335 827.978821	10.1.0.119	8.8.8.8	ICMP	106 Echo (ping) request	id=0x0001, seq=8/2048, ttl=1 (no response found!)
12336 827.979414	10.1.0.1	10.1.0.119	ICMP	134 Time-to-live exceeded (Time to live exceeded in transit)	
12337 827.980147	10.1.0.119	8.8.8.8	ICMP	106 Echo (ping) request	id=0x0001, seq=9/2304, ttl=1 (no response found!)
12339 827.980646	10.1.0.1	10.1.0.119	ICMP	134 Time-to-live exceeded (Time to live exceeded in transit)	
12340 827.981129	10.1.0.119	8.8.8.8	ICMP	106 Echo (ping) request	id=0x0001, seq=10/2560, ttl=1 (no response found!)
12341 827.981707	10.1.0.1	10.1.0.119	ICMP	134 Time-to-live exceeded (Time to live exceeded in transit)	
12391 833.926087	10.1.0.119	8.8.8.8	ICMP	106 Echo (ping) request	id=0x0001, seq=11/2816, ttl=2 (no response found!)
12395 837.670208	10.1.0.119	8.8.8.8	ICMP	106 Echo (ping) request	id=0x0001, seq=12/3072, ttl=2 (no response found!)
12398 841.671619	10.1.0.119	8.8.8.8	ICMP	106 Echo (ping) request	id=0x0001, seq=13/3328, ttl=2 (no response found!)
12401 845.672329	10.1.0.119	8.8.8.8	ICMP	106 Echo (ping) request	id=0x0001, seq=14/3584, ttl=3 (no response found!)
12402 845.673428	212.33.95.1	10.1.0.119	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)	
12403 845.674667	10.1.0.119	8.8.8.8	ICMP	106 Echo (ping) request	id=0x0001, seq=15/3840, ttl=3 (no response found!)
12404 845.675645	212.33.95.1	10.1.0.119	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)	
12405 845.677950	10.1.0.119	8.8.8.8	ICMP	106 Echo (ping) request	id=0x0001, seq=16/4096, ttl=3 (no response found!)
12406 845.678756	212.33.95.1	10.1.0.119	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)	
12410 845.689088	212.33.95.1	10.1.0.119	ICMP	70 Destination unreachable (Port unreachable)	
12414 847.190513	212.33.95.1	10.1.0.119	ICMP	70 Destination unreachable (Port unreachable)	
12417 848.701884	212.33.95.1	10.1.0.119	ICMP	70 Destination unreachable (Port unreachable)	
12422 851.220042	10.1.0.119	8.8.8.8	ICMP	106 Echo (ping) request	id=0x0001, seq=17/4352, ttl=4 (no response found!)
12423 851.221285	212.33.70.144	10.1.0.119	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)	
12424 851.222819	10.1.0.119	8.8.8.8	ICMP	106 Echo (ping) request	id=0x0001, seq=18/4608, ttl=4 (no response found!)
12425 851.223982	212.33.70.144	10.1.0.119	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)	
12426 851.225460	10.1.0.119	8.8.8.8	ICMP	106 Echo (ping) request	id=0x0001, seq=19/4864, ttl=4 (no response found!)
12427 851.226560	212.33.70.144	10.1.0.119	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)	
12431 851.231334	212.33.70.144	10.1.0.119	ICMP	70 Destination unreachable (Port unreachable)	
12434 852.739680	212.33.70.144	10.1.0.119	ICMP	70 Destination unreachable (Port unreachable)	
12437 854.253640	212.33.70.144	10.1.0.119	ICMP	70 Destination unreachable (Port unreachable)	
12458 856.768881	10.1.0.119	8.8.8.8	ICMP	106 Echo (ping) request	id=0x0001, seq=20/5120, ttl=5 (no response found!)
12459 856.770105	212.33.70.221	10.1.0.119	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)	
12460 856.771669	10.1.0.119	8.8.8.8	ICMP	106 Echo (ping) request	id=0x0001, seq=21/5376, ttl=5 (no response found!)

5.3 Results

- Number of hops: 5 (actually 10 but we couldn't take a screenshot of it.)
- Final destination reached? Yes.

5.4 Discussion

- Observed ICMP Time Exceeded messages from routers.
- Some hops may not respond (firewalls).
- Final hop responded with ICMP Echo Reply. (although there isn't final result in the screenshot, we received reply on 10th hop.)

6 Experiment 4 - IP Fragmentation Analysis

6.1 Setup Parameters

- Ping command: `ping -l 2000 10.1.0.118`
- Expected MTU: 1500 bytes
- Wireshark filter: `icmp`

6.2 First Packet

```
0000   44 45 6f 12 62 cd 44 45 6f 12 63 6d 08 00 45 00
0010   05 dc f1 f9 20 00 80 01 00 00 0a 01 00 78 0a 01
0020   00 79 08 00 46 31 00 01 00 27 61 62 63 64...
```

Listing 2: First packet

6.3 IP Header

- Total Length: 05 dc (1500 bytes) - This is a large packet
- Identification: f1 f9
- TTL: 80 (128 hops)
- Protocol: 01 (ICMP)

6.4 ICMP Header

- Type: 08 (Echo Request)
- Identifier: 00 01

6.5 Fragmentated Packet

```
0000   44 45 6f 12 62 cd 44 45 6f 12 63 6d 08 00 45 00
0010   05 dc f1 f9 20 00 80 01 00 00 0a 01 00 78 0a 01
0020   00 79 08 00 46 31 00 01 00 27 61 62 63 64...
```

Listing 3: Fragmented Packets (ICMP Echo Request)

6.6 Key Observations

- Total Length: 02 54 (596 bytes)
- Identification: f1 f9 (same as original packet)
- Flags/Offset: 00 b9 (Fragment offset of 185 * 8 = 1480 bytes)
- More Fragments flag is set (indicated by 00 in flags)

6.7 Discussion

- Original packet is 1500 bytes (05 dc in hex)
- This exceeds typical MTU (1500 bytes) when including headers.
- All fragments share same Identification field (f1 f9).
- Echo Request (Type 8) → Echo Reply (Type 0)

7 Experiment 5 - DF Flag and MTU Exceeded

7.1 Setup Parameters

- Ping command: `ping -n 1 -l 2048 -f 10.1.0.118`
- Expected MTU: 1500 bytes

7.2 CMD Output

```
C:\Users\Student>ping -n 1 -l 2048 -f 10.1.0.118

Pinging 10.1.0.118 with 2048 bytes of data:
Packet needs to be fragmented but DF set.

Ping statistics for 10.1.0.118:
    Packets: Sent = 1, Received = 0, Lost = 1 (100% loss),

//nothing is shown in Wireshark
```

Listing 4: DF ping output

7.3 Discussion

- Since DF flag forced the router to reject the oversized packet. That's why we can't see it on wireshark.

8 Experiment 6 - ARP Message Analysis in LAN Communication

8.1 Setup Parameters

- ARP cache cleared using: `arp -d *`
- Wireshark filter used: `arp`

8.2 Same Network Ping

```
C:\Users\Student>arp -d *
C:\Users\Student>ping 10.1.0.118

Pinging 10.1.0.118 with 32 bytes of data:
Reply from 10.1.0.118: bytes=32 time<1ms TTL=128

Ping statistics for 10.1.0.118:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
```

Listing 5: Same network ping with ARP resolution

ARP Cache Before/After Clearing: Before clearing, the ARP cache may contain entries for nearby hosts. After `arp -d *`, the cache is empty, forcing the system to resolve MAC addresses anew.

8.3 Different Network Ping

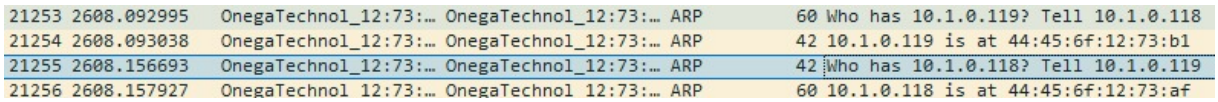
```
C:\Users\Student>arp -d *
C:\Users\Student>ping 8.8.8.8

Pinging 8.8.8.8 with 32 bytes of data:
Reply from 8.8.8.8: bytes=32 time=20ms TTL=120

Ping statistics for 8.8.8.8:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
```

Listing 6: Different network ping with gateway ARP

8.4 Wireshark Screenshots



8.5 Discussion

- In the same network, ARP is used to resolve the MAC address of the destination IP.
- ARP resolved the MAC of 10.1.0.118 directly.
- ARP requests are broadcast and answered by the target host.
- In different networks, ARP resolves the MAC address of the default gateway.
- ARP resolved the MAC of the default gateway (not 8.8.8.8), as the gateway handles forwarding to external networks.
- For external IPs (e.g., 8.8.8.8), ARP resolves the gateway's MAC, not the destination's. The gateway then handles further routing using its own ARP tables.
- The Ethernet frame's `Type` field was `0x0806` for ARP messages, confirming they operate at Layer 2. For IP traffic, this field would be `0x0800`.

9 Experiment 7 - DNS Communication using UDP

9.1 Setup Parameters

- Command used: `nslookup www.google.com`
- Tool: Wireshark with filter `udp.port == 53`

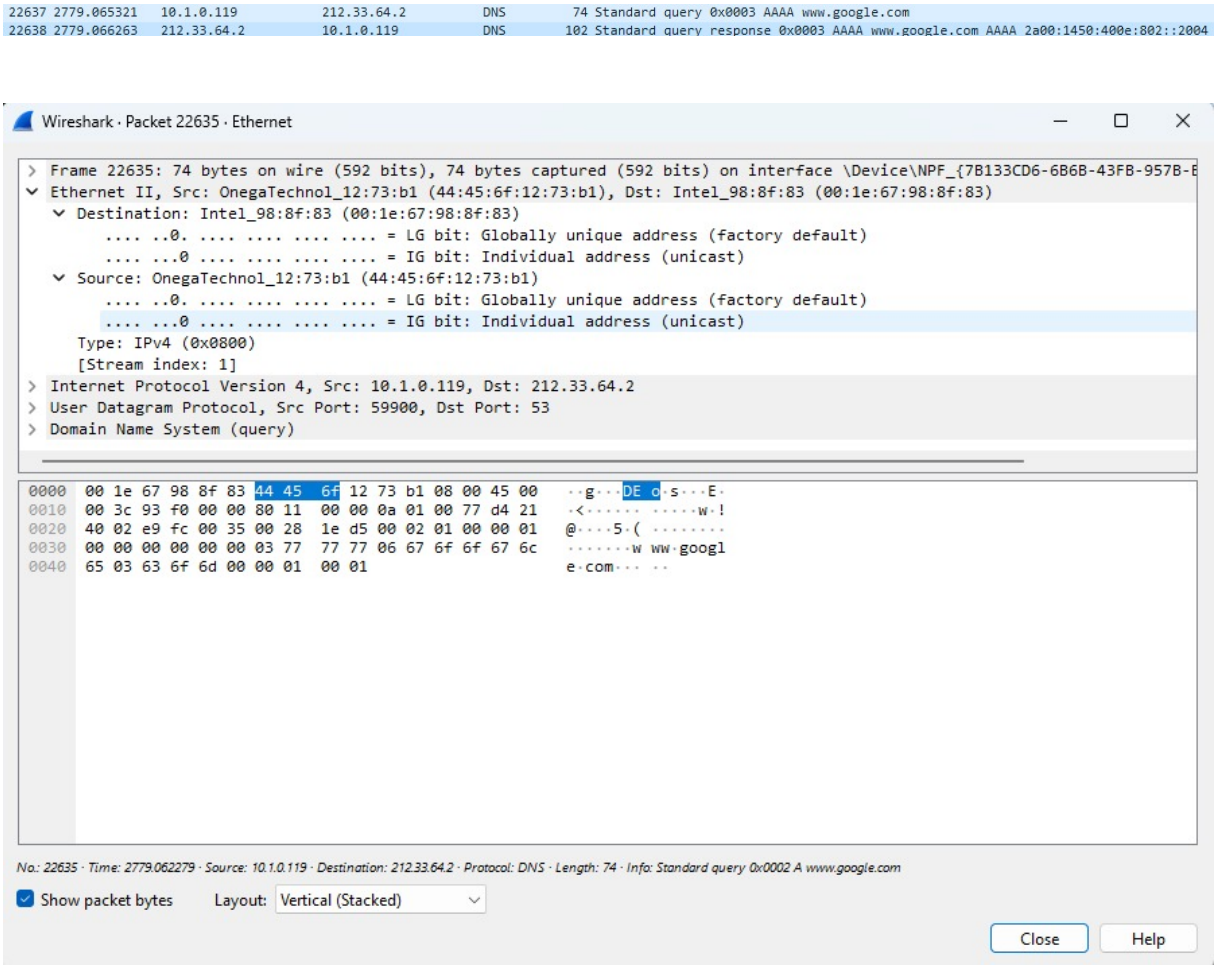
9.2 CMD Output

```
C:\Users\Student>nslookup www.google.com
Server:  dns.local
Address:  192.168.1.1

Non-authoritative answer:
Name:    www.google.com
Addresses:  142.250.190.14
```

Listing 7: DNS query using nslookup

9.3 Wireshark Analysis



- A DNS query was sent to the local DNS server on port 53 using UDP.

- The DNS request included a query for the A record of `google.com`.
- The response contained the resolved IP address `142.250.190.14`.
- All communication occurred over UDP protocol.

UDP Header Details:

- Source Port: Ephemeral port (e.g., 59900), chosen randomly by the client.
- Destination Port: 53 (standard DNS port).
- Length: Total segment size (header + payload).
- Checksum: Validates data integrity (optional in IPv4 but used here).

9.4 Discussion

- DNS uses UDP for quick transmission of queries and responses.
- Port 53 is the standard port for DNS queries.
- Wireshark allows inspection of each field within the UDP and DNS headers.
- The transaction includes a query ID, flags, question section, and answer section.
- The DNS query ID (`0x2022` in the screenshot) ensures responses match requests. The `Flags` field indicated a standard query (`RD=1`, recursion desired).
- The `Answer` section contained the A record (IPv4) for `www.google.com`, even though the query included AAAA (IPv6), suggesting the server prioritized IPv4.

Experiment 8: Recording a TCP Session Using the WWW Service

Objective

The objective of this experiment is to capture and analyze a TCP connection session using the WWW service. This includes understanding the TCP three-way handshake, data transfer, and connection termination phases using Wireshark.

Setup

- Use a computer with an IP address configured as `10.1.0.119`.
- The default gateway address is required and denoted as `gateway`.
- The destination web page used is `http://10.1.0.1/test.asp`, assuming `10.1.0.1` is the default gateway IP.

Procedure

1. Start Wireshark on the station with IP `10.1.0.119`.
2. Set a capture filter in Wireshark:

```
tcp and ip.addr==10.1.0.1 and ip.addr==10.1.0.119
```

3. Open a web browser and visit the URL: `http://10.1.0.1/test.asp`.
4. Observe and stop the Wireshark capture once the page is fully loaded.

Analysis

The TCP communication consists of three main phases:

1. Connection Establishment (Three-Way Handshake)

- Client (`10.1.0.119`) sends a `SYN` packet with an initial sequence number (e.g., `Seq = 0`).
- Server (`10.1.0.1`) replies with `SYN-ACK` (`Seq = 0, Ack = 1`).
- Client responds with `ACK` (`Ack = 1`), completing the handshake.

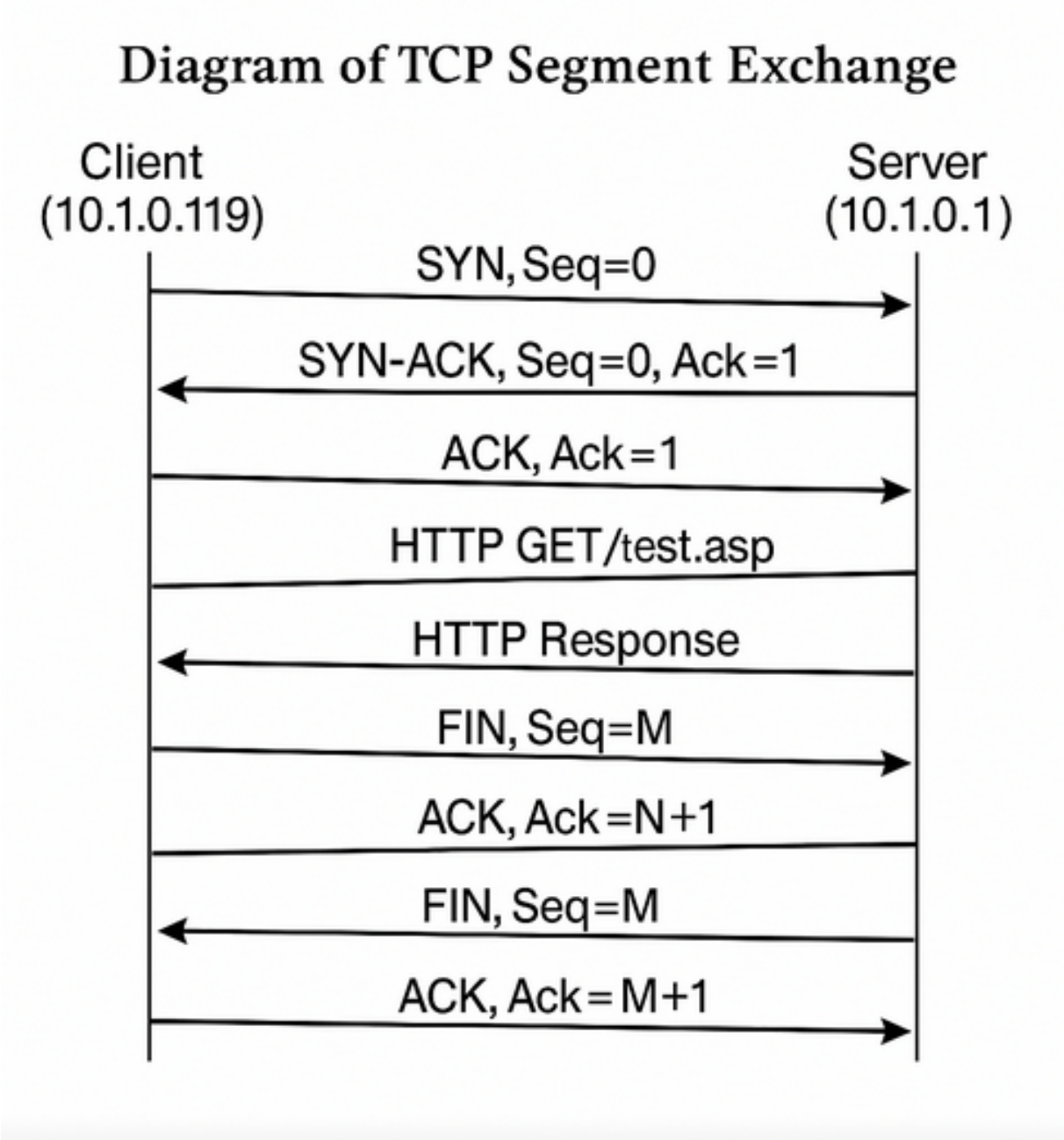
2. Data Transfer

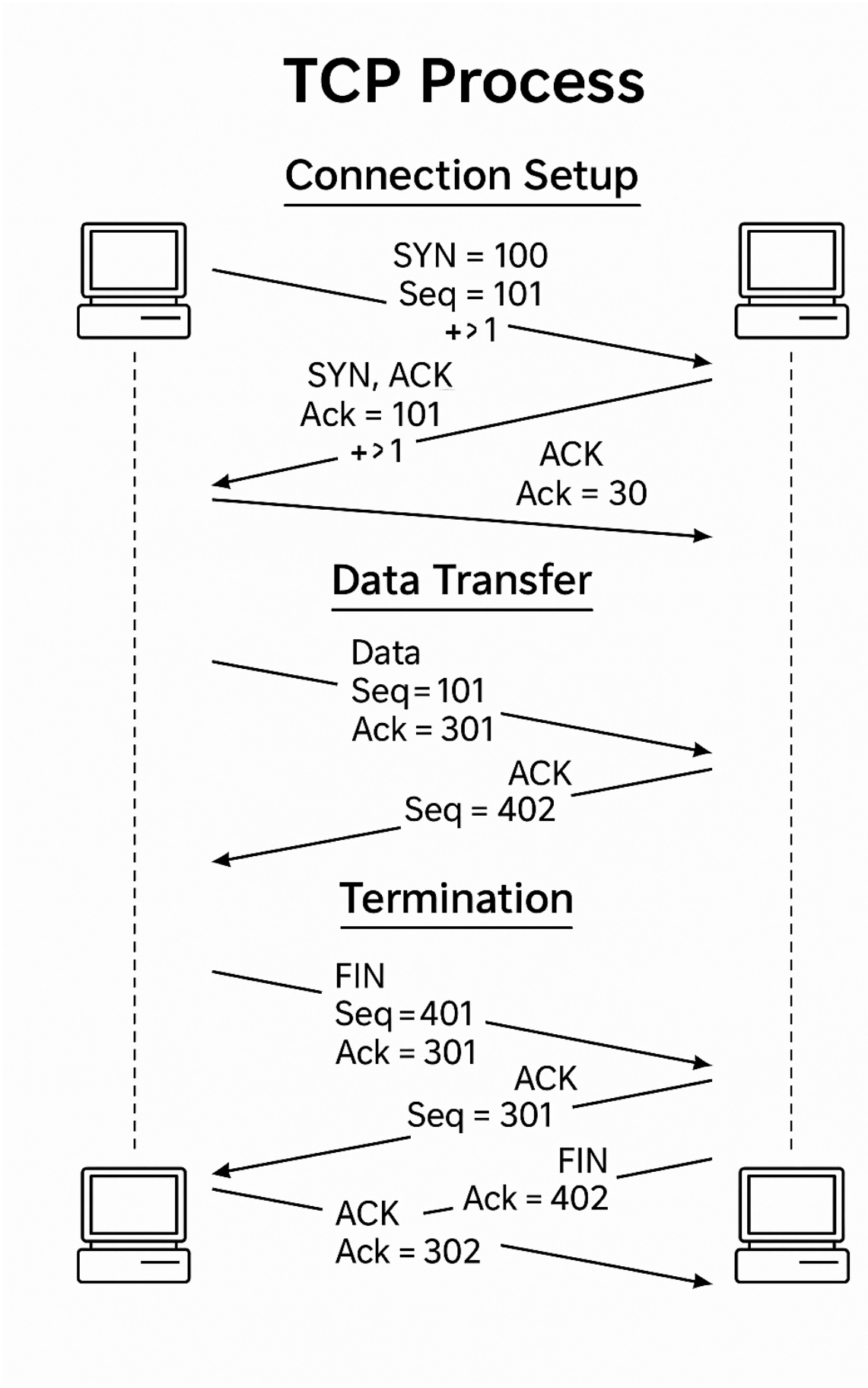
- HTTP request is sent from the client to the server.
- The server sends HTTP response data (HTML content).
- Wireshark captures the segment sequence and acknowledgment numbers.

3. Connection Termination

- Either party sends a `FIN` packet to initiate termination.
- The other party acknowledges and may also send its own `FIN`.
- The connection is closed after the final `ACK`.

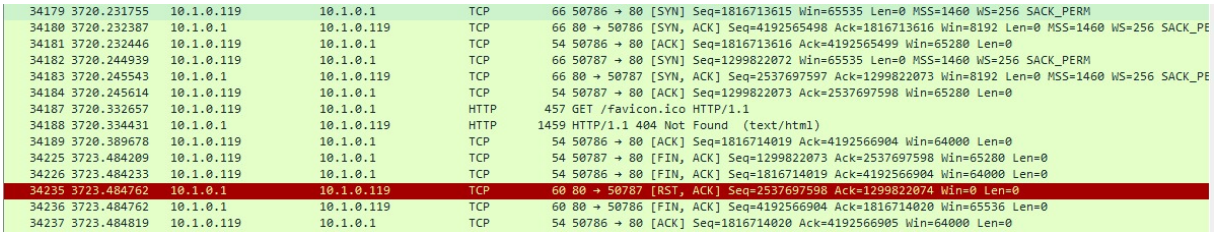
Diagram of TCP Segment Exchange




TCP Process
Connection Setup
SYN = 100
Seq = 101
+>1
SYN, ACK
Ack = 101
+>1
ACK
Ack = 30
Data Transfer
Data
Seq=101
Ack = 301
ACK
Seq = 402
Termination
FIN
Seq=401
Ack = 301
ACK
Seq = 301
FIN
Ack = 402
ACK
Ack = 302

Phase	Flags	Description
Handshake	SYN → SYN-ACK → ACK	Client Seq=0, Server Ack=1
Data Transfer	PSH, ACK	HTTP GET /test.asp
Termination	FIN → ACK → FIN → ACK	Graceful closure

9.5 Wireshark Analysis



Conclusion

This experiment successfully demonstrated the TCP connection lifecycle between a client and server over HTTP. Using Wireshark, the key stages of the TCP protocol were visualized, including the sequence numbers, acknowledgments, and control flags (SYN, ACK, FIN).

10 References

1. Sloan, J.D. (2001). Network Troubleshooting Tools. O'Reilly Media, Inc.
2. Wireshark Documentation: `https://www.wireshark.org/docs/`
3. Lab Manual by Andrzej Zankiewicz, PhD