ECS 152A – Project 1 Report

Students: Aktan Azat (924035096), Gezheng Kang (923288158)

Submitted PCAP Files

- `part-1-google-ping.pcap`
- `part-2-example-com.pcap`
- `part-3-http-forever-com.pcap`
- `part-4-tmz-com.pcap`
- `part-5-ftp-anonymous-login.pcap`
- `part-6-ssh.pcap`

Submitted Python Scripts

- `part1a12_[Aktan_Azat]_[924035096]_[Gezheng_Kang]_[923288158].py`
- `part1a34_[Aktan_Azat]_[924035096]_[Gezheng_Kang]_[923288158].py`
- `part1b1_[Aktan_Azat]_[924035096]_[Gezheng_Kang]_[923288158].py`
- `udp_server_[Aktan_Azat]_[924035096]_[Gezheng_Kang]_[923288158].py`
- `udp_client_[Aktan_Azat]_[924035096]_[Gezheng_Kang]_[923288158].py`
- `proxy_server_[Aktan_Azat]_[924035096]_[Gezheng_Kang]_[923288158].py`
- `client_[Aktan_Azat]_[924035096]_[Gezheng_Kang]_[923288158].py`
- `server_[Aktan_Azat]_[924035096]_[Gezheng_Kang]_[923288158].py`

Part 1(a)

Q1: Application Layer Protocols and Their Counts

Part	File Name	Application Layer Protocols	Count	How Protocol Was Determined
1	part-1-google-ping.pcap	DNS mDNS HTTPS	2 7 11	Identified using Wireshark's "Protocol" column or via the Python parser output that inspects packet layers and counts by protocol name.
2	part-2-example-com.pcap	DNS HTTPS	14 31	Protocols determined based on the parsed packet summary from the `.pcap` file using filtering by protocol field.
3	part-3-http-forever-com.pcap	HTTP DNS HTTPS	8 15 35	HTTP and HTTPS detected via standard TCP port numbers (80 and 443). DNS detected via port 53.
4	part-4-tmz-com.pcap	DNS HTTPS	120 761	Counted by analyzing packets labeled under these protocols in the `.pcap` capture.
5	part-5-ftp-anonymous-login.pcap	FTP	12	FTP protocol identified via port 21 traffic.
6	part-6-ssh.pcap	NBNS HTTPS	1 1	NBNS (NetBIOS Name Service) and HTTPS found through packet headers and protocol labels.

Q2: HTTP and HTTPS Packet Counts for Activities 2 and 3

Activity	File Name	HTTP Packets	HTTPS Packets	Total (HTTP + HTTPS)
2	part-2-example-com.pcap	0	31	31
3	part-3-http-forever-com.pcap	8	35	43

Q3. Destination IP addresses and first packets

Activity	Destination IP	First packet (UTC)
Ping google.com	142.250.189.206	2025-10-21T21:23:23.159701Z
Visit example.com	23.220.75.232	2025-10-21T22:08:26.486174Z
Visit httpforever.com	146.190.62.39	2025-10-21T22:09:02.141485Z
Visit tmz.com	3.169.183.125	2025-10-21T22:09:33.736467Z
FTP anonymous login	209.51.188.20	2025-10-21T21:28:20.825205Z
SSH session	169.237.240.10	2025-10-21T21:30:36.139693Z

Assumption: I list only the first external IPv4 destination that actually carried each activity's payload; supporting infrastructure (campus DNS, local gateways, multicast chatter) appears in the captures but is excluded because the prompt asks for the activity destinations.

Q4. Browser identification from captures

- Activity 2 (example.com, HTTPS): TLS hides the headers, so the browser cannot be identified.
- Activity 3 (httpforever.com, HTTP): `User-Agent` shows Chrome 141 on macOS.
- Activity 4 (tmz.com, HTTPS): TLS hides the headers, so the browser cannot be identified.

Assumption: Browser identification depends entirely on clear-text `User-Agent` strings; for encrypted TLS traffic I assume no session keys are available, so the browser stays unknown.

Part 1(b) – PCAP1_1 Analysis

Using `dpkt` to scan `PCAP1_1.pcap` for clear-text application data shows one HTTP GET request carrying obvious secrets:

```
GET /?secret=secret1 HTTP/1.1
Host: example.com
User-Agent: test-client/2
MY-SECRET: Zubair Rocks!!
```

Key findings:

- The client exposed the query parameter `secret=secret1` directly in the URL.
- A custom header `MY-SECRET` leaked the phrase **“Zubair Rocks!!”**, constituting another secret sent to the server.
- Request originated from the client to `example.com` over HTTP (unencrypted), so anyone capturing the traffic can read these secrets.

Sample output:

```
----- Secret-bearing HTTP request -----
Timestamp    : 2025-10-15T01:00:58.200875+00:00
Client -> Server: 2601:204:c200:a930:4d70:f5b6:e2e8:ac72 -> 2600:1408:ec00:36::1736:7f31
Request line : GET /?secret=secret1
Header       : my-secret: Zubair Rocks!!
```

Part 1(b) – PCAP1_2.pcap Analysis

Going through the packets in PCAP1_2.pcap, I noticed a mix of local network discovery traffic (like mDNS and ICMPv6) along with some DNS and TCP/TLS activity that stood out. The main activity I saw was that the computer was trying to connect to `google.com`.

Here’s what I found and how I figured it out:

1. Normally, when opening a website, the sequence follows this order: DNS resolution, to, TCP handshake, to, TLS handshake, to, encrypted HTTP request and response.
2. In this capture, however, I didn’t see the full handshake sequence because the recording started mid-session. That’s why some TCP packets (port 443) appear before the DNS queries — those were already-active HTTPS connections.
3. After that, there were several DNS requests where the device tried to resolve google.com.
4. At first, I saw a few “No such name” responses for domains like google.com.hsd1.ca.comcast.net, which are DNS “NXDOMAIN” (non-existent domain) replies. This means extra lookups occurred before the request finally resolved to the actual google.com domain. This usually happens because the ISP automatically appends its own domain suffixes, based on how the DNS resolver’s search list is configured.
5. Eventually, the device successfully resolved the actual google.com domain, and the client continued sending or receiving encrypted TLS “Application Data,” which represents the actual web content.

Conclusion

Based on the DNS and HTTPS patterns, the specific activity captured in PCAP1_2.pcap was **the user opening Google.com in a web browser**. The other packets are just regular background noise from the local network.

Part 1(b) – PCAP1_3.pcap Analysis

Going through the packets in PCAP1_3.pcap, I found that the main activity captured here was a traceroute to Google’s IPv6 address. The packets show a clear pattern of ICMPv6 echo requests and replies that help map the route from Alex's computer to the destination.

Here’s what I found and how I figured it out:

1. The capture includes multiple **ICMPv6 Echo Request** packets with gradually increasing hop limits. This pattern is typical of a traceroute, which tests each hop along the path by incrementing the hop limit (similar to TTL in IPv4). For example, the packets with **hop limit = 13** were the final probes, meaning the traceroute was testing the thirteenth hop.
2. The destination address seen in the packets (`2607:f8b0:4005:80e::200e`) belongs to Google’s IPv6 network. The replies to Alex's Echo Requests are **ICMPv6 Echo Replies**, not “Time Exceeded” messages. This confirms that the probe packets actually reached the final destination—Google’s server—rather than stopping at an intermediate router (like, drop the packet before rreaching the server), in other words, the traceroute successfully reached Google after about **13 hops**.
3. The replies have a **hop limit of 115**, which indicates Google’s server started with a default hop limit (likely 128) and about 13 hops were used to reach back to Alex's computer.
4. Additionally, the reason multiple request–reply pairs share the same hop limit is because traceroute sends **several probes per hop** (often three) to measure latency consistency and packet loss.

Conclusion

The capture represents an **IPv6 traceroute to Google**, where Alex's computer sent multiple ICMPv6 Echo Requests with increasing hop limits to map the path. The destination was reached successfully at **hop 13**, meaning there were approximately 13 hops between Alex's device and Google’s IPv6 server.

Best of luck

Submission Page

Include this signed page with your submission

I certify that all submitted work is my own work. I have completed all of the assignments on my own without assistance from others except as indicated by appropriate citation. I have read and understand the university policy on plagiarism and academic dishonesty. I further understand that official sanctions will be imposed if there is any evidence of academic dishonesty in this work. I certify that the above statements are true.

Team Member 1:

Gezheng Kang	Gezheng Kang	10/30/2025
Full Name (Printed)	Signature	Date

Team Member 2:

Aktan Azat	Aktan Azat	10/30/2025
Full Name (Printed)	Signature	Date