CLDP (Custom Lightweight Discovery Protocol)

Chandransh Singh
22CS30017

1 Introduction

The Custom Lightweight Discovery Protocol (CLDP) is a simple protocol designed for node discovery and metadata exchange in a network. It operates over raw sockets using a custom IP protocol number and provides lightweight discovery capabilities via HELLO announcements and QUERY/RESPONSE mechanisms.

2 Assumptions and Limitations

- The protocol operates on IPv4.
- It uses a custom IP protocol number (253) for communication.
- The protocol does not implement authentication or encryption.
- The system must allow the creation of raw sockets (requires root privileges in most systems).
- The server sends HELLO announcements every 10 seconds.
- The client listens for HELLO from server(s) for 10 seconds.
- Then the client sends a custom query to the active servers.
- Query responses provide metadata such as hostname, system time, and CPU load.
- The maximum payload size is 1024 bytes.

3 Message Types

Message Type	Code	Description
HELLO	0x01	Announced by servers periodically for discovery.
QUERY	0x02	Sent by clients to request metadata.
RESPONSE	0x03	Sent by the server in response to a QUERY message.

4 Message Structure

Each CLDP packet consists of an IP header followed by a CLDP header and an optional payload.

4.1 CLDP Header (8 bytes)

Field	Size (bytes)	Description
msg_type	1	Type of message (HELLO, QUERY, RESPONSE)
payload_len	1	Length of payload in bytes
trans_id	2	Unique transaction identifier
reserved	4	Reserved for future use

4.2 Payload

- **HELLO:** No payload.
- **QUERY:** A 1-byte bitmask indicating requested metadata.
- **RESPONSE:** Variable-length text containing requested metadata.

5 Metadata Flags

Flag Name	Value	Description
META_HOSTNAME	0x01	Request hostname
META_TIME	0x02	Request system time
META_CPULOAD	0x04	Request CPU load

6 Packet Exchange

6.1 HELLO Announcement

The server broadcasts a HELLO message every 10 seconds to announce its presence.

6.2 Query-Response Mechanism

1. **Client Sends Query:**
   - Constructs a QUERY message with `msg_type` set to 0x02.
   - Assigns a unique `trans_id` for tracking.
   - Sets `payload_len` to 1 byte, containing a metadata bitmask.
   - Sends the QUERY message to the CLDP server using raw sockets.
2. **Server Processes Query:**
   - Extracts the `trans_id` and metadata bitmask.
   - Retrieves the requested metadata fields.
   - Formats the retrieved metadata into a response payload.
3. **Server Sends Response:**
   - Constructs a RESPONSE message with `msg_type` set to 0x03.
   - Copies the `trans_id` from the QUERY message.
   - Sends the RESPONSE message back to the client.
4. **Client Receives Response:**

- Extracts the `trans_id` and matches it with the QUERY request.
- Parses and processes the metadata payload.

7 Build and Run Instructions

7.1 Makefile Explanation

The provided Makefile automates the compilation and execution of the CLDP server and client.

```
CC = gcc
CFLAGS = -Wall

all: cldp_server cldp_client

cldp_server: cldp_server.c
	$(CC) $(CFLAGS) -o s cldp_server.c

cldp_client: cldp_client.c
	$(CC) $(CFLAGS) -o c cldp_client.c

rs: cldp_server
	sudo ./s

rc: cldp_client
	sudo ./c

clean:
	rm -f s c
```

7.2 Running the Code

1. To compile both the server and client, run:

   ```
   make
   ```

2. To run the CLDP server, use:

   ```
   make rs
   ```

3. To run the CLDP client in another terminal, use:

   ```
   make rc
   ```

4. To clean up compiled binaries, run:

   ```
   make clean
   ```

8 Communication Flow

The following steps describe the sequence of communication between the client and server(s):

8.1 Step 1: Communication with Server on the Same Machine

1. The CLDP server starts on the local machine.
2. The client sends a QUERY message to the local server.
3. The server processes the request and responds with metadata (e.g., hostname, system time, CPU load).
4. The client receives and parses the response.

8.2 Step 2: Another Machine Starts Server and Broadcasts HELLO

1. A second server starts on a different machine in the network.
2. The new server periodically broadcasts HELLO messages.
3. The client listens for HELLO messages to discover available servers.

8.3 Step 3: Communication with Both Servers

1. The client has now discovered both servers.
2. The client sends QUERY messages to both servers.
3. Each server processes the request and responds with metadata.
4. The client receives and parses responses from both servers.

9 Sample Output

9.1 Server Output 1(192.168.64.8)

```
> sudo ./s
+++ CLDP Server running...
<== Broadcast HELLO sent.
<-- Sent RESPONSE to 192.168.64.8 (trans_id 34683)
<== Broadcast HELLO sent.
<== Broadcast HELLO sent.
<-- Sent RESPONSE to 192.168.64.8 (trans_id 17901)
<== Broadcast HELLO sent.
```

9.2 Server Output 2(192.168.132.4)

```
> sudo ./s
+++ CLDP Server running...
<== Broadcast HELLO sent.
<-- Sent RESPONSE to 192.168.64.8 (trans_id 53624)
<== Broadcast HELLO sent.
<== Broadcast HELLO sent.
```

9.3 Client Output

```
> sudo ./c
+++ CLDP Client running...
~~~ Listening for HELLO messages: 7 seconds remaining....
==> Received HELLO from 192.168.64.8
+++ Added new server: 192.168.64.8
~~~ Listening for HELLO messages: 1 seconds remaining...
+++ Found 1 new servers during HELLO listening.
Querying 1 active servers...
>>> Select metadata to request (enter y/n for each option):
Request hostname? (y/n): y
Request system time? (y/n): y
Request CPU load? (y/n): n
<-- Sent QUERY (trans_id 34683) to 192.168.64.8

--> Received RESPONSE from 192.168.64.8:
Hostname: moonserver
Time: 2025-03-31 17:49:53

:D Query complete. [1/1] servers responded.

Press Enter to repeat the process or type 'exit' to quit:
~~~ Listening for HELLO messages: 10 seconds remaining...
==> Received HELLO from 192.168.64.8
~~~ Listening for HELLO messages: 3 seconds remaining....
==> Received HELLO from 192.168.64.8
==> Received HELLO from 192.168.132.4
+++ Added new server: 192.168.64.8
~~~ Listening for HELLO messages: 1 seconds remaining...
+++ Found 1 new servers during HELLO listening.
Querying 2 active servers...
>>> Select metadata to request (enter y/n for each option):
Request hostname? (y/n): y
Request system time? (y/n): y
Request CPU load? (y/n): y
<-- Sent QUERY (trans_id 17901) to 192.168.64.8

<-- Sent QUERY (trans_id 53624) to 192.168.132.4

--> Received RESPONSE from 192.168.64.8:
Hostname: moonserver
Time: 2025-03-31 17:50:08
CPU Load: 0.24

--> Received RESPONSE from 192.168.132.4:
Hostname: gaurav-roy-HP-Laptop-15-da0xxx
Time: 2025-03-31 17:50:08
CPU Load: 0.80

:D Query complete. [2/2] servers responded.
```

```
Press Enter to repeat the process or type 'exit' to quit: exit
~/Desktop >
```

10 CLDP Validation

10.1 Using `tcpdump`

`tcpdump` is a command-line packet analyzer that can be used to capture raw socket traffic.

10.1.1 Start Capturing Traffic

Run the following command **before** starting your CLDP server and client:

```
sudo tcpdump -i any proto 253 -vv
```

- `-i any` → Captures packets on all network interfaces. - `proto 253` → Filters packets using your custom protocol number (253). - `-vv` → Displays verbose output with packet details.

To save the captured packets for later analysis:

```
sudo tcpdump -i any proto 253 -w cldp_traffic.pcap
```

Press **Ctrl+C** to stop 'tcpdump'.

10.2 Using Wireshark

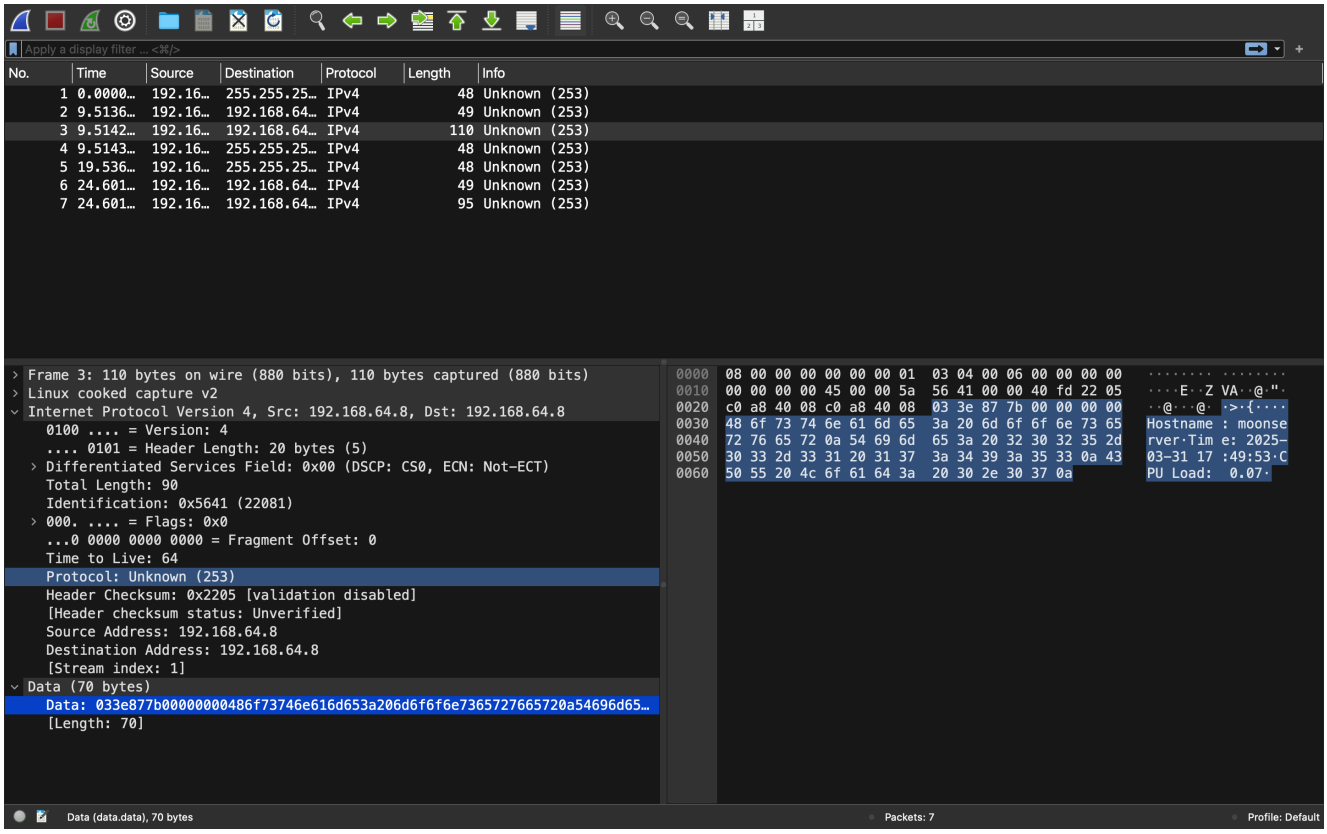
Wireshark is a GUI-based packet analyzer that provides detailed visualization.

10.2.1 Start Capturing

1. Open Wireshark.
2. Select the appropriate network interface (e.g., `eth0`, `wlan0`, `lo0`).
3. Apply a capture filter:
   ```
   ip proto 253
   ```
4. Click **Start** to begin capturing packets.

10.2.2 Analyze the Packets

- Verify **Protocol: IPv4** and check for protocol 253.
- Expand the IP header to verify:
  - **Protocol** → 253 (custom protocol)
  - **Source/Destination IPs**
  - **Total length**
  - **Checksum**
- Expand the payload section to analyze CLDP messages.


Apply a display filter ... <⌘/>
No. Time Source Destination Protocol Length Info
1 0.0000… 192.16… 255.255.25… IPv4 48 Unknown (253)
2 9.5136… 192.16… 192.168.64… IPv4 49 Unknown (253)
3 9.5142… 192.16… 192.168.64… IPv4 110 Unknown (253)
4 9.5143… 192.16… 255.255.25… IPv4 48 Unknown (253)
5 19.536… 192.16… 255.255.25… IPv4 48 Unknown (253)
6 24.601… 192.16… 192.168.64… IPv4 49 Unknown (253)
7 24.601… 192.16… 192.168.64… IPv4 95 Unknown (253)
Frame 3: 110 bytes on wire (880 bits), 110 bytes captured (880 bits)
Linux cooked capture v2
Internet Protocol Version 4, Src: 192.168.64.8, Dst: 192.168.64.8
0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 90
Identification: 0x5641 (22081)
000. .... = Flags: 0x0
...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 64
Protocol: Unknown (253)
Header Checksum: 0x2205 [validation disabled]
[Header checksum status: Unverified]
Source Address: 192.168.64.8
Destination Address: 192.168.64.8
[Stream index: 1]
Data (70 bytes)
Data: 033e877b00000000486f73746e616d653a206d6f6f6e7365727665720a54696d65…
[Length: 70]
0000 08 00 00 00 00 00 00 01 03 04 00 06 00 00 00 00
0010 00 00 00 00 45 00 00 5a 56 41 00 00 40 fd 22 05
0020 c0 a8 40 08 c0 a8 40 08 03 3e 87 7b 00 00 00 00
0030 48 6f 73 74 6e 61 6d 65 3a 20 6d 6f 6f 6e 73 65
0040 72 76 65 72 0a 54 69 6d 65 3a 20 32 30 32 35 2d
0050 30 33 2d 33 31 20 31 37 3a 34 39 3a 35 33 0a 43
0060 50 55 20 4c 6f 61 64 3a 20 30 2e 30 37 0a
Data (data.data), 70 bytes
Packets: 7
Profile: Default