

CLDP (Custom Lightweight Discovery Protocol)

Specification

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 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

The CLDP protocol defines the following 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:**
 - The client constructs a QUERY message, setting the `msg_type` field to `0x02`.
 - A unique `trans_id` is assigned to the request for tracking purposes.
 - The `payload_len` field is set to 1 byte, containing a bitmask representing the requested metadata (e.g., hostname, system time, CPU load).
 - The QUERY message is then sent to the CLDP server using raw sockets.
2. **Server Processes Query:**

- The server receives the QUERY message and extracts the `trans_id` and metadata bitmask from the payload.
- It checks the bitmask to determine which metadata fields are requested.
- The server retrieves the requested metadata:
 - If the `META_HOSTNAME` flag is set, it retrieves the system hostname.
 - If the `META_TIME` flag is set, it fetches the current system time.
 - If the `META_CPULOAD` flag is set, it computes the CPU load.
- The retrieved metadata is formatted into a response payload.

3. Server Sends Response:

- The server constructs a RESPONSE message, setting `msg_type` to `0x03`.
- The `trans_id` from the QUERY message is copied to the RESPONSE message for correlation.
- The payload contains the formatted metadata values.
- The RESPONSE message is sent back to the client using raw sockets.

4. Client Receives Response:

- The client listens for a RESPONSE message from the server.
- Upon receiving the RESPONSE, it extracts the `trans_id` to match it with the original QUERY request.
- The client parses the metadata from the payload and processes it accordingly.

7. Error Handling

- If a received message has an invalid structure, it is ignored.
- If the requested metadata is unavailable, the server returns an empty RESPONSE.
- Packets from unknown protocols are ignored.

Build and run instructions

1. run `cldp_server` in a terminal

```
make rs
```

2. run `cldp_client` in other terminal

```
make rc
```

We are using raw sockets and need root(`sudo`) privileges

CLDP Validation

1. Using tcpdump

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

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
```

This saves the packets in a `.pcap` file, which can be analyzed later using Wireshark.

- To stop `tcpdump`, press **Ctrl+C**.

2. Using Wireshark

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

Start Capturing

1. Open Wireshark.
2. Select the appropriate network interface (e.g., `eth0`, `wlan0`, `lo0`).
3. Apply a capture filter for your custom protocol:

```
ip proto 253
```

4. Click **Start** to begin capturing packets.

After testing, stop the capture and save the file:

- **File** → **Save As** → **cldp_traffic.pcapng**

Analyze the Packets

- You should see raw packets under **Protocol: IPv4** (since your protocol is at the IP level).
- Expand the IP header section to verify:
 - **Protocol** → 253 (custom protocol)
 - **Source/Destination IPs**
 - **Total length**
 - **Checksum**
- Expand the packet payload section to check your **CLDP message types and payload**.

The image shows a Wireshark packet capture of an ICMP Echo (ping) request. The packet list at the top shows 7 packets, with packet 4 selected. The packet details pane on the left shows the structure of the selected packet, including the Ethernet II header, Internet Protocol Version 4 header, and ICMP Echo request. The packet bytes pane on the right shows the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000...	127.0.0.1	127.0.0.1	IPv4	48	Unknown (253)
2	10.043...	127.0.0.1	127.0.0.1	IPv4	48	Unknown (253)
3	10.636...	127.0.0.1	127.0.0.1	IPv4	49	Unknown (253)
4	10.637...	127.0.0.1	127.0.0.1	IPv4	110	Unknown (253)
5	19.672...	127.0.0.1	127.0.0.1	IPv4	48	Unknown (253)
6	24.712...	127.0.0.1	127.0.0.1	IPv4	49	Unknown (253)
7	24.713...	127.0.0.1	127.0.0.1	IPv4	95	Unknown (253)

Frame 4: 110 bytes on wire (880 bits), 110 bytes captured (880 bits) on Linux cooked capture v2

Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

0100 = Version: 4

.... 0101 = Header Length: 20 bytes (5)

Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)

Total Length: 90

Identification: 0xfca5 (64677)

0000 = Flags: 0x0

...0 0000 0000 0000 = Fragment Offset: 0

Time to Live: 64

Protocol: Unknown (253)

Header Checksum: 0x7eff [validation disabled]

[Header checksum status: Unverified]

Source Address: 127.0.0.1

Destination Address: 127.0.0.1

[Stream index: 0]

Data (70 bytes)

Data: 033e775900000000486f73746e616d653a206d66f6f67365727665720a54696d65...

[Length: 70]

0000 08 00 00 00 00 00 00 01 03 04 00 06 00 00 00 00E...Z...@...
0010 00 00 00 00 45 00 00 5a fc a5 00 00 40 fd 7e ffE...Z...@...
0020 7f 00 00 01 7f 00 00 01 03 3e 77 59 00 00 00 00wY...
0030 48 6f 73 74 6e 61 6d 65 3a 20 6d 6f 6f 6e 73 65 Hostname : moonse
0040 72 76 65 72 0a 54 69 6d 65 3a 20 32 30 32 35 2d rver:Tim e: 2025-
0050 30 33 2d 32 39 20 31 35 3a 31 31 3a 33 38 0a 43 03-29 15 :11:38: C
0060 50 55 20 4c 6f 61 64 3a 20 30 2e 30 30 0a PU Load: 0.00.

Demo output:

1. CLDP server

```
gcc -Wall -o s cldp_server.c
sudo ./s
+++ CLDP Server running...
<== Broadcast HELLO sent.
<-- Sent RESPONSE to 127.0.0.1 (trans_id 2824)
<== Broadcast HELLO sent.
<== Broadcast HELLO sent.
<-- Sent RESPONSE to 127.0.0.1 (trans_id 18260)
<== Broadcast HELLO sent.
```

2. CLDP client

```
gcc -Wall -o c cldp_client.c
sudo ./c
+++ CLDP Client running...
~~~ Listening for HELLO messages: 6 seconds remaining....
==> Received HELLO from 127.0.0.1
+++ Added new server: 127.0.0.1
~~~ 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): y
<-- Sent QUERY (trans_id 2824) to 127.0.0.1

--> Received RESPONSE from 127.0.0.1:
Hostname: moonserver
Time: 2025-03-29 15:05:48
CPU Load: 0.43

: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 127.0.0.1
~~~ Listening for HELLO messages: 4 seconds remaining....
==> Received HELLO from 127.0.0.1
~~~ Listening for HELLO messages: 1 seconds remaining...
`` `No new servers found 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 18260) to 127.0.0.1

--> Received RESPONSE from 127.0.0.1:
Hostname: moonserver
Time: 2025-03-29 15:06:08

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

Press Enter to repeat the process or type 'exit' to quit: exit
```

reference: [A Guide to Using Raw Sockets](#)