Lightweight Custom Discovery Protocol (LCDP) Implementation Report

Introduction

This report details the implementation of a Custom Lightweight Discovery Protocol (CLDP) using raw sockets in C according to the requirements specified in Assignment 7 of CS39006: Networks Laboratory. The protocol is designed for a closed network environment where nodes can announce their presence and query other nodes for specific application-level metadata.

Protocol Specification

Message Types

The protocol supports three message types as required:

- **0x01 (1)**: HELLO - Used to announce a node is active
- **0x02 (2)**: QUERY - Request for metadata from other nodes
- **0x03 (3)**: RESPONSE - Response with the requested metadata

Packet Structure

The CLDP packet structure consists of:

1. **IP Header**: Standard IPv4 header (20 bytes)
2. **CLDP Header** (9 bytes):
   - Message Type (1 byte): Indicates the type of message (HELLO, QUERY, or RESPONSE)
   - Payload Length (4 bytes): Length of the payload in bytes
   - Transaction ID 1 (2 bytes): Primary transaction identifier, used as PID by the server
   - Transaction ID 2 (2 bytes): Secondary transaction identifier, used as PID by the client
3. **Payload**: Optional data field based on message type

Protocol Format

The protocol uses IP protocol number 253 (custom) and raw sockets for communication. The packets are manually crafted at the IP level, without relying on transport-layer protocols like TCP or UDP.

Implementation Details

Components

The implementation consists of two main components:

1. **Server (`server.c`)**: Listens for incoming CLDP packets, processes requests, and sends responses

2. **Client (`client.c`)**: Constructs CLDP packets and sends HELLO and QUERY messages

Server Implementation

The server performs the following functions:

- Creates a raw socket with protocol number 253
- Runs an announcer thread that broadcasts HELLO messages every 5 seconds
- Listens for incoming QUERY messages
- Processes queries for three types of metadata:
    1. CPU LOAD: System information including uptime, load averages, RAM, and process count
    2. SYSTEM TIME: Current local time
    3. HOSTNAME: System hostname
- Sends RESPONSE messages with requested metadata

Client Implementation

The client performs the following functions:

- Creates a raw socket with protocol number 253
- Listens for incoming HELLO messages from servers
- Sends QUERY messages to active servers based on command-line arguments
- Processes RESPONSE messages and displays the received metadata

Packet Crafting

Both server and client manually craft IP packets:

- Set appropriate fields in the IP header (version, IHL, TTL, protocol)
- Include proper source and destination addresses
- Calculate checksums for the IP header
- Append the CLDP header and payload

Checksum Calculation

A standard IP checksum function is implemented:

c Copy

```c
unsigned short checksum(void *b, int len) {
    unsigned short *buf = b;
    unsigned int sum = 0;
    for (; len > 1; len -= 2) sum += *buf++;
    if (len == 1) sum += *(unsigned char *)buf;
    sum = (sum >> 16) + (sum & 0xFFFF);
    sum += (sum >> 16);
    return ~sum;
}
```

Metadata Support

The implementation supports three types of metadata queries:

1. **CPU LOAD**: Returns detailed system information including:
   - System uptime (days, hours, minutes, seconds)
   - Load averages (1, 5, and 15 minutes)
   - Memory usage (total, free, shared, and buffered RAM)
   - Swap space usage
   - Number of active processes
2. **SYSTEM TIME**: Returns the current local time in the format "YYYY-MM-DD HH:MM
   "
3. **HOSTNAME**: Returns the system hostname

Communication Flow

1. **Server Announcement**:
   - Server creates a HELLO message (type 0x01)
   - Sets Transaction ID 1 to its PID
   - Broadcasts the message every 5 seconds
2. **Client Query**:
   - Client receives HELLO messages
   - Client constructs a QUERY message (type 0x02)
   - Sets Transaction ID 2 to its PID
   - Sends the query with a specific metadata request
3. **Server Response**:
   - Server receives QUERY message

- Verifies Transaction ID 1 matches its PID
- Processes the request and prepares metadata
- Constructs RESPONSE message (type 0x03)
- Sends the response back to the client

4. **Client Processing**:
   - Client receives RESPONSE message
   - Verifies Transaction ID 2 matches its PID
   - Displays the received metadata

Technical Challenges and Solutions

1. **Raw Socket Configuration**:
   - Used `setsockopt()` with `IP_HDRINCL` to manually craft IP headers
   - Set `SO_BROADCAST` option for broadcasting HELLO messages
2. **Transaction Tracking**:
   - Used process IDs (PIDs) as transaction identifiers
   - Server uses Transaction ID 1 to identify itself
   - Client uses Transaction ID 2 to track its requests
3. **Broadcasting**:
   - Server sends HELLO messages to the broadcast address (255.255.255.255)
   - Enables discovery without knowing specific IP addresses

Testing and Validation

The implementation can be validated using:

- Direct observation of server and client output
- Network packet capture using Wireshark or tcpdump
- Multiple instances to verify discovery and query functionality

Limitations and Assumptions

1. **IP Address Configuration**:
   - The server uses a hardcoded IP address (192.168.0.3)
   - This may need to be changed based on the network configuration
2. **Security Considerations**:
   - The protocol does not implement authentication or encryption
   - Suitable only for trusted network environments

3. **Raw Socket Privileges**:
    - Requires root privileges (sudo) for raw socket operations

Build and Run Instructions

Building the Code

Compile the server and client programs:

bash Copy

```
gcc -o cldp_server server.c -lpthread
gcc -o cldp_client client.c
```

Running the Server

Run the server with root privileges:

bash Copy

```
sudo ./cldp_server
```

Running the Client

Run the client with root privileges, optionally specifying a query type:

bash Copy

```
sudo ./cldp_client [query_number]
```

Where `query_number` is:

- 0: CPU LOAD (default)
- 1: SYSTEM TIME
- 2: HOSTNAME

Conclusion

The implemented Custom Lightweight Discovery Protocol successfully meets the requirements specified in the assignment. It provides a functional discovery mechanism and metadata query system using raw sockets with a custom protocol number. The implementation demonstrates key concepts in network programming, including:

- Raw socket creation and configuration
- Manual IP packet crafting

- Custom protocol header design
- System information retrieval
- Network discovery mechanisms

The protocol is suitable for closed network environments where nodes need to discover each other and exchange basic system metadata.