**Academic Session 2024-2025**
**Spring Semester**

# Computer Networks Lab

## Assignment 7: Design and Implement a Lightweight Custom Discovery Protocol (LCDP) using Raw Sockets in POSIX C

**More Aayush Babasaheb (22CS30063)**

Server side-

1. When server starts, an initial HELLO packet is broadcasted
2. A select call is started with timeout of 10 seconds, and for checking if anything is available to read (HELLO and QUERY packets received) on our raw socket
3. CASE 1: TIMEOUT
   A HELLO packet is broadcasted, along with resetting the timeout for select call.
4. CASE 2: HELLO packet received
   Print who sent the HELLO packet and the transaction id
5. CASE 3: QUERY packet received
   Determine what is being queried by the sender, determine what should be the response, create the RESPONSE packet, and send it to the client who had queried

Client side -

1. Wait until a HELLO packet is received
2. Randomly decide what to query the server from available queries { HOSTNAME, CPULOAD, SYSTIME}
3. Create a query packet using the decided query and send it to the server from whom HELLO was received.
4. Print the RESPONSE received (no blocking for RESPONSE packet to be received)

IP Headers-

- Version
- IHL
- TTL
- PROTOCOL
- SADDR
- DADDR
- TOT_LEN

Custom Header-

- Message type - 32 bit integer storing what type of message this is
  - HELLO (0th bit is set) - Set for HELLO packets
  - QUERY (1st bit is set) - Set for QUERY packets
  - RESPONSE (2nd bit is set) - Set for RESPONSE packets
  - CPULOAD (3rd bit is set) - Only if also a QUERY or RESPONSE message, for CPULOAD query/response.
  - SYSTIME (4th bit is set) - Only if also a QUERY or RESPONSE message, for SYSTIME query/response
  - HOSTNAME (5th bit is set) - Only if also a QUERY or RESPONSE message, for HOSTNAME query/response
  - The message type is created by combining appropriate MACROS with the same name. However only one QUERY can be requested and replied at a time.
- Payload length - 32 bit integer storing length of message after Custom Header

- Transaction id - 32 bit integer, starting from 0, storing the transaction id. The transaction id is maintained by the server, which increments it for every HELLO message sent, and is same for QUERY and RESPONSE packets triggered by this HELLO message
- Reserved - 32 bits, currently unused
- All of the above integers are converted to network byte order before storing in packet, and need to be converted to the host byte order before being used

Payload - It is used in response packets to store the response of the server to the query message sent by the client. Its length is stored in Payload length in the custom header section.

Example of an HELLO packet created-

```
struct iphdr *send_ip_packet = (struct iphdr*)send_buf;
memset(send_buf, 0, BUF_SIZE);
send_ip_packet->version = 4;
send_ip_packet->ihl = 5;
send_ip_packet->ttl = 64;
send_ip_packet->protocol = 253;
send_ip_packet->saddr = my_ip;
send_ip_packet->daddr = inet_addr("255.255.255.255");
send_ip_packet->tot_len = htons(send_ip_packet->ihl*4 + 16);

char *send_custom_header = send_buf + send_ip_packet->ihl*4;

int send_msg_type = htonl(HELLO);
int send_payload_len = htonl(0);
int send_tran_id = htonl(cur_transaction_id);
int send_reserved = htonl(0);

memcpy(send_custom_header, &send_msg_type, 4);
memcpy(send_custom_header+4, &send_payload_len, 4);
memcpy(send_custom_header+8, &send_tran_id, 4);
memcpy(send_custom_header+12, &send_reserved, 4);
```

Example of an QUERY packet created-

```
struct iphdr *send_ip_packet = (struct iphdr*)send_buf;
memset(send_buf, 0, BUF_SIZE);
send_ip_packet->version = 4;
send_ip_packet->ihl = 5;
send_ip_packet->ttl = 64;
send_ip_packet->protocol = 253;
send_ip_packet->saddr = my_ip;
send_ip_packet->daddr = src.sin_addr.s_addr;
```

```
        send_ip_packet->tot_len = htons(send_ip_packet->ihl*4 + 16);

        char *send_custom_header = send_buf + send_ip_packet->ihl*4;


        int send_msg_type = htonl(QUERY | CPULOAD);
        int send_payload_len = htonl(0);
        int send_tran_id = htonl(recv_tran_id);
        int send_reserved = htonl(0);

        memcpy(send_custom_header, &send_msg_type, 4);
        memcpy(send_custom_header+4, &send_payload_len, 4);
        memcpy(send_custom_header+8, &send_tran_id, 4);
        memcpy(send_custom_header+12, &send_reserved, 4);
```

Example of an RESPONSE packet created-

```
        struct iphdr *send_ip_packet = (struct iphdr*)send_buf;
        memset(send_buf, 0, BUF_SIZE);

        send_ip_packet->version = 4;
        send_ip_packet->ihl = 5;
        send_ip_packet->ttl = 64;
        send_ip_packet->protocol = 253;
        send_ip_packet->saddr = my_ip;
        send_ip_packet->daddr = src.sin_addr.s_addr;

        char *send_custom_header = send_buf + send_ip_packet->ihl*4;
        char *send_payload = send_custom_header + 16;
        switch (recv_query_type){
        case CPULOAD:
            printf(" CPULOAD\n");
            printf("Sent CPULOAD in response to %s\n",
            inet_ntoa(src.sin_addr));

            get_sysinfo(send_payload,
            BUF_SIZE-(send_ip_packet->ihl*4 + 16)-1);

            break;
        case SYSTIME:
            printf(" SYSTIME\n");
            printf("Sent SYSTIME in response to %s\n",
            inet_ntoa(src.sin_addr));
```

```
            get_time(send_payload, BUF_SIZE-(send_ip_packet->ihl*4 +
            16)-1);

            break;
        case HOSTNAME:
            printf(" HOSTNAME\n");
            printf("Sent HOSTNAME in response to %s\n",
            inet_ntoa(src.sin_addr));

            get_hostname(send_payload,
            BUF_SIZE-(send_ip_packet->ihl*4 + 16)-1);

            break;
        }

        int send_msg_type = htonl(RESPONSE | recv_query_type);
        int send_payload_len = htonl(strlen(send_payload));
        int send_tran_id = htonl(recv_tran_id);
        int send_reserved = htonl(0);

        memcpy(send_custom_header, &send_msg_type, 4);
        memcpy(send_custom_header+4, &send_payload_len, 4);
        memcpy(send_custom_header+8, &send_tran_id, 4);
        memcpy(send_custom_header+12, &send_reserved, 4);

        send_ip_packet->tot_len = htons(send_ip_packet->ihl*4 + 16 +
        ntohl(send_payload_len));
```