Subject: DHCP Server Implementation

Abstract

動態主機設定協定（Dynamic Host Configuration Protocol，縮寫為DHCP）是一種網路協定，用於自動分配 IP 位址和其他網路配置參數給連接到網路的設備。DHCP的目的是簡化網路管理的工作，讓網路設備能夠更容易地獲取有效的IP位址和相關的網路設定。

本次 Project 實作了 DHCP Server ，其執行於一台 Host 上，可使其成為一台 DHCP Server，為與該 Host 位於相同 LAN 之 Hosts 動態配置 IP 位置和其他網路配置參數。以此了解 DHCP 運作原理、方式，並深入了解封包格式、建立、傳送方式。最後發現了 DHCP 的 Reply 要自己去填 Dst MAC ，不能透過 ARP 來幫忙填，因其尚未獲取 IP 位置，尚不能響應 ARP，因此 DHCP 的 Request 中包含 MAC 位置的資訊提供給 Server。

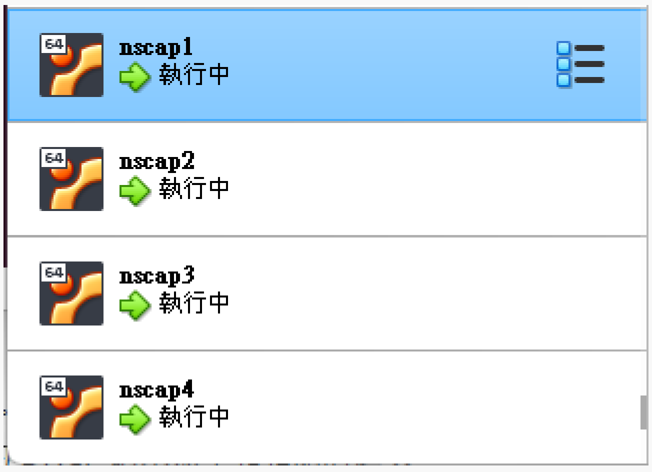
Motivation

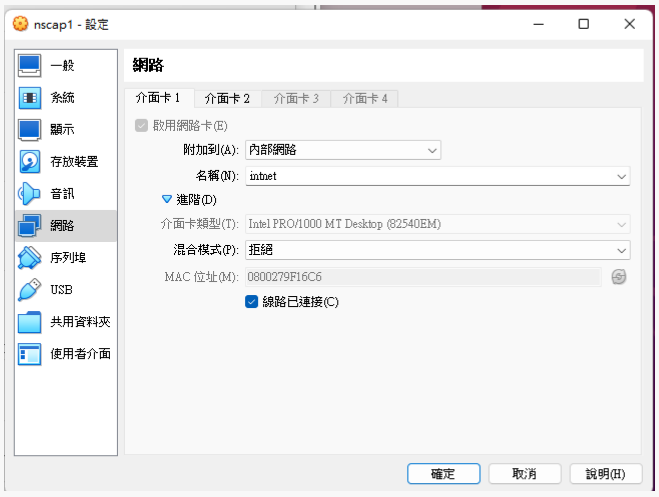
於課中實作過 HTTP Server，另外於其他課程實做過 Chat Server、DNS Server、Proxy Server，於實作過程可以更詳細了解與之相關的 Protocol (HTTP、IRC、DNS、SOCKS4/4A)，因此於這次 Final Project 就決定再選一個 Protocol 進行實作，於是選中了 DHCP Server。

Results

實驗環境：

於 VirtualBox 中開啟四台虛擬機 (HW1 的虛擬機)。將網路介面卡設定更改為 Internal Network (內部網路)，可以使四台虛擬機處於相同的 LAN，且沒有 VirtualBox 預設提供的 DHCP Server ，另外要注意其 MAC 位址是否一樣，若一樣要記得更換 MAC 位址，防止其 MAC 位址重複。


nscap1 執行中
nscap2 執行中
nscap3 執行中
nscap4 執行中

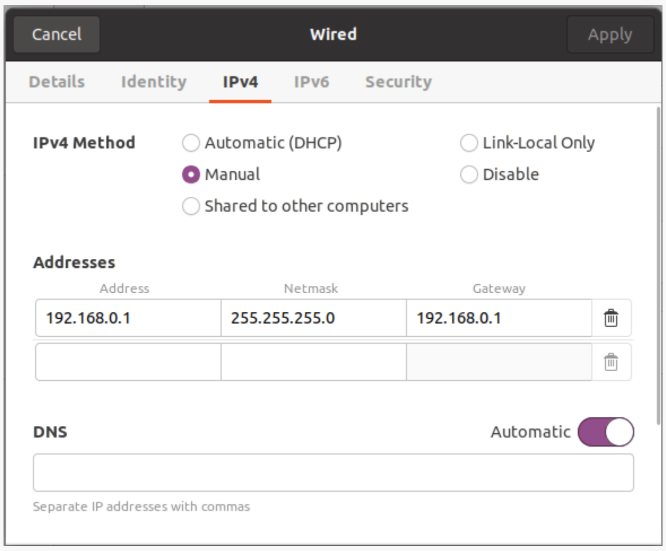

nscap1 - 設定

網路

介面卡 1

啟用網路卡(E)
附加到(A): 內部網路
名稱(N): intnet
進階(D)
介面卡類型(T): Intel PRO/1000 MT Desktop (82540EM)
混合模式(P): 拒絕
MAC 位址(M): 0800279F16C6
☑ 線路已連接(C)

將第一台選為我們的 DHCP Server，並為其設定 Static IP。其它台則都設為 Dynamic IP，用以作為 Client 端對 Server 進行測試，可根據 Client 是否正確取得 IP 位址來判斷實作是否正確。



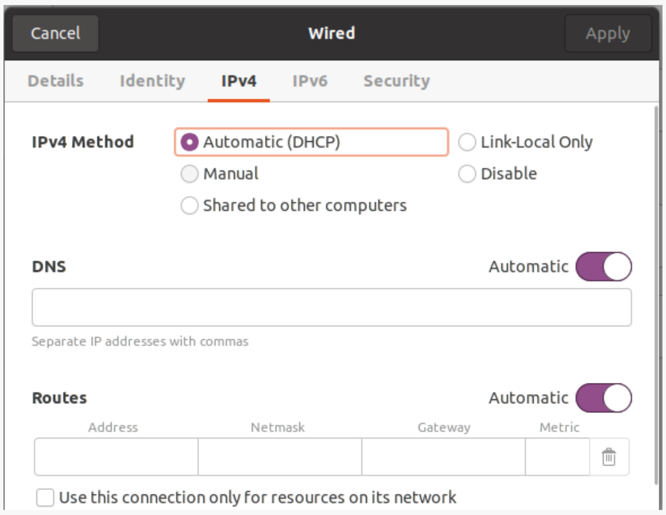
IPv4 Method: Automatic (DHCP) / Link-Local Only / [✓ Manual] / Disable / Shared to other computers

Addresses

Address	Netmask	Gateway
192.168.0.1	255.255.255.0	192.168.0.1

DNS — Automatic

Separate IP addresses with commas



IPv4 Method: [✓ Automatic (DHCP)] / Link-Local Only / Manual / Disable / Shared to other computers

DNS — Automatic

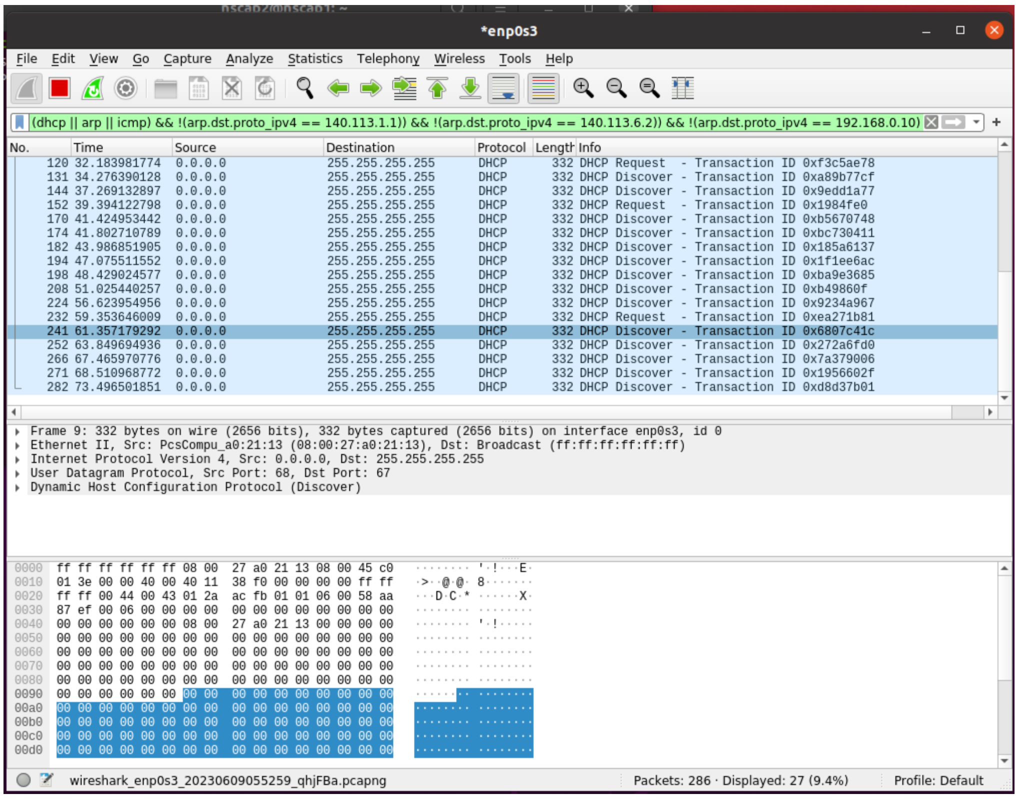
Separate IP addresses with commas

Routes — Automatic

Address	Netmask	Gateway	Metric

Use this connection only for resources on its network

設置完成開啟 Wireshark 檢視封包狀況會發現 Client 正在嘗試尋找 DHCP Server 以獲得 IP 位置。


*enp0s3
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
(dhcp || arp || icmp) && !(arp.dst.proto_ipv4 == 140.113.1.1)) && !(arp.dst.proto_ipv4 == 140.113.6.2)) && !(arp.dst.proto_ipv4 == 192.168.0.10)
```

```
No.   Time         Source     Destination      Protocol Length Info
  120 32.183981774 0.0.0.0    255.255.255.255  DHCP     332 DHCP Request  - Transaction ID 0xf3c5ae78
  131 34.276390128 0.0.0.0    255.255.255.255  DHCP     332 DHCP Discover - Transaction ID 0xa89b77cf
  144 37.269132897 0.0.0.0    255.255.255.255  DHCP     332 DHCP Discover - Transaction ID 0x9edd1a77
  152 39.394122798 0.0.0.0    255.255.255.255  DHCP     332 DHCP Request  - Transaction ID 0x1984fe0
  170 41.424953442 0.0.0.0    255.255.255.255  DHCP     332 DHCP Discover - Transaction ID 0xb5670748
  174 41.802710789 0.0.0.0    255.255.255.255  DHCP     332 DHCP Discover - Transaction ID 0xbc730411
  182 43.986851905 0.0.0.0    255.255.255.255  DHCP     332 DHCP Discover - Transaction ID 0x185a6137
  194 47.075511552 0.0.0.0    255.255.255.255  DHCP     332 DHCP Discover - Transaction ID 0x1f1ee6ac
  198 48.429024577 0.0.0.0    255.255.255.255  DHCP     332 DHCP Discover - Transaction ID 0xba9e3685
  208 51.025440257 0.0.0.0    255.255.255.255  DHCP     332 DHCP Discover - Transaction ID 0xb49860f
  224 56.623954956 0.0.0.0    255.255.255.255  DHCP     332 DHCP Discover - Transaction ID 0x9234a967
  232 59.353646009 0.0.0.0    255.255.255.255  DHCP     332 DHCP Request  - Transaction ID 0xea271b81
  241 61.357179292 0.0.0.0    255.255.255.255  DHCP     332 DHCP Discover - Transaction ID 0x6807c41c
  252 63.849694936 0.0.0.0    255.255.255.255  DHCP     332 DHCP Discover - Transaction ID 0x272a6fd0
  266 67.465970776 0.0.0.0    255.255.255.255  DHCP     332 DHCP Discover - Transaction ID 0x7a379006
  271 68.510968772 0.0.0.0    255.255.255.255  DHCP     332 DHCP Discover - Transaction ID 0x1956602f
  282 73.496501851 0.0.0.0    255.255.255.255  DHCP     332 DHCP Discover - Transaction ID 0xd8d37b01
```

```
Frame 9: 332 bytes on wire (2656 bits), 332 bytes captured (2656 bits) on interface enp0s3, id 0
Ethernet II, Src: PcsCompu_a0:21:13 (08:00:27:a0:21:13), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol Version 4, Src: 0.0.0.0, Dst: 255.255.255.255
User Datagram Protocol, Src Port: 68, Dst Port: 67
Dynamic Host Configuration Protocol (Discover)
```

```
0000  ff ff ff ff ff ff 08 00  27 a0 21 13 08 00 45 c0   ········ '·!···E·
0010  01 3e 00 00 40 00 40 11  38 f0 00 00 00 00 ff ff   ·>··@·@· 8·······
0020  ff ff 00 44 00 43 01 2a  ac fb 01 01 06 00 58 aa   ···D·C·* ······X·
0030  87 ef 00 06 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0040  00 00 00 00 00 00 08 00  27 a0 21 13 00 00 00 00   ········ '·!·····
0050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
```

wireshark_enp0s3_20230609055259_qhjFBa.pcapng — Packets: 286 · Displayed: 27 (9.4%) — Profile: Default

於此可以看到 DHCP 是 Application Layer 的 Protocol 使用的 Transport Layer Protocol 為 UDP，且 Server 固定使用 67 Port，Client 固定使用 68 Port。首先我們先寫一個 UDP Server 開在 67 Port 持續接收封包。

```
int serverSocket = socket(AF_INET, SOCK_DGRAM, 0);
if (serverSocket < 0) {
    cerr << "Error creating server socket" << endl;
    return 1;
}

int optval = 1;
setsockopt(serverSocket, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(optval));

struct sockaddr_in serverAddress;
memset(&serverAddress, 0, sizeof(serverAddress));
serverAddress.sin_family = AF_INET;
serverAddress.sin_addr.s_addr = INADDR_ANY;
serverAddress.sin_port = htons(SERVER_PORT);
```

```
if (bind(serverSocket, (struct sockaddr *)&serverAddress, sizeof(serverAddress)) < 0) {
    cerr << "Error binding server socket" << endl;
    close(serverSocket);
    return 1;
}

cout << "DHCP server is running on port " << SERVER_PORT << endl;

while (true) {
    handleDHCPRequest(serverSocket);
}

close(serverSocket);
```

再來根據 [1] 所提供的 DHCP 格式，去解析封包，並根據其意義輸出出來。

DHCP packet format:

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |     op (1)    |   htype (1)   |   hlen (1)    |   hops (1)    |
    +---------------+---------------+---------------+---------------+
    |                            xid (4)                            |
    +-------------------------------+-------------------------------+
    |           secs (2)            |           flags (2)           |
    +-------------------------------+-------------------------------+
    |                          ciaddr  (4)                          |
    +---------------------------------------------------------------+
    |                          yiaddr  (4)                          |
    +---------------------------------------------------------------+
    |                          siaddr  (4)                          |
    +---------------------------------------------------------------+
    |                          giaddr  (4)                          |
    +---------------------------------------------------------------+
    |                                                               |
    |                          chaddr  (16)                         |
    |                                                               |
    |                                                               |
    +---------------------------------------------------------------+
    |                                                               |
    |                          sname   (64)                         |
    +---------------------------------------------------------------+
    |                                                               |
    |                          file    (128)                        |
    +---------------------------------------------------------------+
    |                                                               |
    |                          options (variable)                   |
    +---------------------------------------------------------------+
```

```
struct DHCPMessage {
    uint8_t op;
    uint8_t htype;
    uint8_t hlen;
    uint8_t hops;
    uint32_t xid;
    uint16_t secs;
    uint16_t flags;
    uint32_t ciaddr;
    uint32_t yiaddr;
    uint32_t siaddr;
    uint32_t giaddr;
    uint8_t chaddr[16];
    uint8_t padding[192];
    uint32_t magic_cookie;
    uint8_t options[64];
};
```

參數意義對照表：

```
FIELD      OCTETS       DESCRIPTION
   -----      ------       -----------

   op            1  Message op code / message type.
                    1 = BOOTREQUEST, 2 = BOOTREPLY
   htype         1  Hardware address type, see ARP section in "Assigned
                    Numbers" RFC; e.g., '1' = 10mb ethernet.
   hlen          1  Hardware address length (e.g.  '6' for 10mb
                    ethernet).
   hops          1  Client sets to zero, optionally used by relay agents
                    when booting via a relay agent.
   xid           4  Transaction ID, a random number chosen by the
                    client, used by the client and server to associate
                    messages and responses between a client and a
                    server.
   secs          2  Filled in by client, seconds elapsed since client
                    began address acquisition or renewal process.
   flags         2  Flags (see figure 2).
   ciaddr        4  Client IP address; only filled in if client is in
                    BOUND, RENEW or REBINDING state and can respond
                    to ARP requests.
   yiaddr        4  'your' (client) IP address.
   siaddr        4  IP address of next server to use in bootstrap;
                    returned in DHCPOFFER, DHCPACK by server.
   giaddr        4  Relay agent IP address, used in booting via a
                    relay agent.
   chaddr       16  Client hardware address.
   sname        64  Optional server host name, null terminated string.
   file        128  Boot file name, null terminated string; "generic"
                    name or null in DHCPDISCOVER, fully qualified
                    directory-path name in DHCPOFFER.
   options     var  Optional parameters field.  See the options
                    documents for a list of defined options.
```

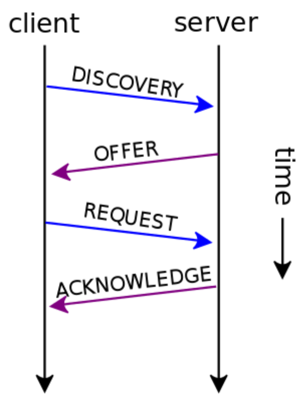
DHCP messages type:

```
 Message         Use
   -------         ---
   DHCPDISCOVER -  Client broadcast to locate available servers.

   DHCPOFFER    -  Server to client in response to DHCPDISCOVER with
                   offer of configuration parameters.

   DHCPREQUEST  -  Client message to servers either (a) requesting
                   offered parameters from one server and implicitly
                   declining offers from all others, (b) confirming
                   correctness of previously allocated address after,
                   e.g., system reboot, or (c) extending the lease on a
                   particular network address.

   DHCPACK      -  Server to client with configuration parameters,
                   including committed network address.

   DHCPNAK      -  Server to client indicating client's notion of network
                   address is incorrect (e.g., client has moved to new
                   subnet) or client's lease as expired

   DHCPDECLINE  -  Client to server indicating network address is already
                   in use.

   DHCPRELEASE  -  Client to server relinquishing network address and
                   cancelling remaining lease.

   DHCPINFORM   -  Client to server, asking only for local configuration
                   parameters; client already has externally configured
                   network address.
```

```
enum DHCPMessageType {
    DHCPDISCOVER = 1,
    DHCPOFFER = 2,
    DHCPREQUEST = 3,
    DHCPACK = 5
};

void handleDHCPRequest(int serverSocket) {
    struct sockaddr_in clientAddress;
    socklen_t clientAddressLength = sizeof(clientAddress);

    // Allocate memory for receiving the DHCP message
    DHCPMessage* request = (DHCPMessage*)malloc(sizeof(DHCPMessage));

    ssize_t bytesRead = recvfrom(serverSocket, request, sizeof(DHCPMessage), 0, (struct
sockaddr *)&clientAddress, &clientAddressLength);

    cout << "byteRead: " << bytesRead << '\n';
    if (bytesRead <= 0) {
        cerr << "Error receiving DHCP request" << endl;
        free(request);
        return;
    }

    if (request->op != 1 || request->options[0] != 53) {
        cout << "Received non-DHCPDISCOVER or non-DHCPREQUEST message. Ignoring." << endl;
        free(request);
        return;
    }

    DHCPMessageType messageType = (DHCPMessageType)request->options[2];

    if (bytesRead <= 0) {
        cerr << "Error receiving DHCP request" << endl;
        free(request);
        return;
    }

    cout << "DHCP Message Received:" << endl;
    cout << "----------------------" << endl;
    cout << "Operation: " << static_cast<int>(request->op) << endl;
    cout << "Hardware Type: " << static_cast<int>(request->htype) << endl;
    cout << "Hardware Address Length: " << static_cast<int>(request->hlen) << endl;
    cout << "Hops: " << static_cast<int>(request->hops) << endl;
    cout << "Transaction ID: " << std::hex << request->xid << endl;
    cout << "Seconds: " << request->secs << endl;
    cout << "Flags: " << request->flags << endl;
    cout << "Client IP Address: " << inet_ntoa(*(struct in_addr *)&(request->ciaddr)) <<
endl;
    cout << "Your IP Address: " << inet_ntoa(*(struct in_addr *)&(request->yiaddr)) <<
endl;
    cout << "Server IP Address: " << inet_ntoa(*(struct in_addr *)&(request->siaddr)) <<
endl;
    cout << "Gateway IP Address: " << inet_ntoa(*(struct in_addr *)&(request->giaddr)) <<
endl;

    string tMAC;
    tMAC.resize(12);
    cout << "Client MAC Address: ";
    for (int i = 0; i < 6; i++) {
        printf("%02x", request->chaddr[i]);
        sprintf(&tMAC[i * 2], "%02x", request->chaddr[i]);
        if (i < 5) {
            cout << ":";
        }
    }
    cout << endl;

    cout << "Client Hardware Padding: ";
    for (int i = 6; i < 16; i++) {
        printf("%02x", request->chaddr[i]);

        if (i < 15) {
            cout << ":";
        }
    }
    cout << endl;

    if (messageType == DHCPDISCOVER) {
        cout << "DHCPDISCOVER\n";
    } else if (messageType == DHCPREQUEST) {
        cout << "DHCPREQUEST\n";
    }
    cout << endl;
    ...
}
```

由此即可解析 Client 發送的 Request 的資訊了，其中注意到我們撰寫的格式中有一個 192 bytes 的 padding 這是由於 DHCP 的前身為 BOOTP 當初需要傳送開機檔案所留的位置，而此於 DHCP 就沒在用了，不過為了前後兼容於是保留了這樣一個區域。

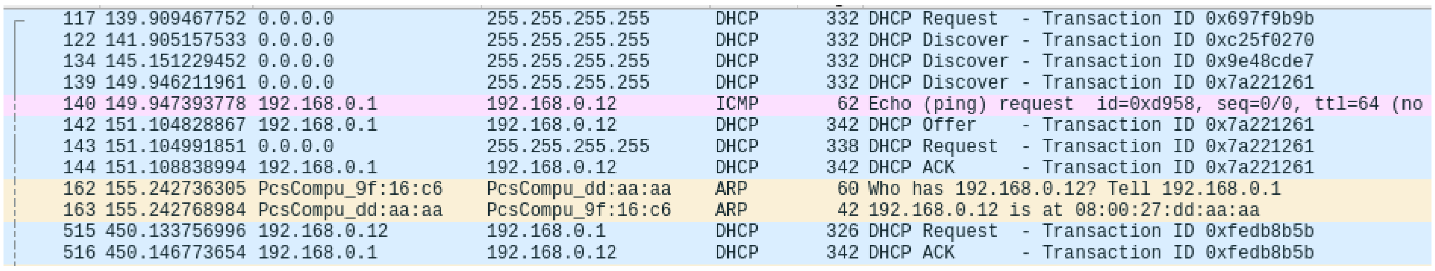
DHCP Client-Server 交互流程、方法：



接著根據上述 Server 、 Client 的基本交互流程，可以簡化成收到 DHCPDISCOVER 時回 DHCPOFFER、收到 DHCPREQUEST 時回 DHCPACK，即可實作出最基本的 DHCP Server。

於時將回應的封包按照對應的格式回傳回去，發現並不能正常傳送出去，Wireshark 也沒有收到，但 sendto system call 也沒有錯誤訊息。

於是透過 Wireshark 觀察了一下 DHCP Server 的運作：



```
  117 139.909467752 0.0.0.0            255.255.255.255    DHCP      332 DHCP Request  - Transaction ID 0x697f9b9b
  122 141.905157533 0.0.0.0            255.255.255.255    DHCP      332 DHCP Discover - Transaction ID 0xc25f0270
  134 145.151229452 0.0.0.0            255.255.255.255    DHCP      332 DHCP Discover - Transaction ID 0x9e48cde7
  139 149.946211961 0.0.0.0            255.255.255.255    DHCP      332 DHCP Discover - Transaction ID 0x7a221261
  140 149.947393778 192.168.0.1        192.168.0.12       ICMP       62 Echo (ping) request  id=0xd958, seq=0/0, ttl=64 (no
  142 151.104828867 192.168.0.1        192.168.0.12       DHCP      342 DHCP Offer    - Transaction ID 0x7a221261
  143 151.104991851 0.0.0.0            255.255.255.255    DHCP      338 DHCP Request  - Transaction ID 0x7a221261
  144 151.108838994 192.168.0.1        192.168.0.12       DHCP      342 DHCP ACK      - Transaction ID 0x7a221261
  162 155.242736305 PcsCompu_9f:16:c6  PcsCompu_dd:aa:aa  ARP        60 Who has 192.168.0.12? Tell 192.168.0.1
  163 155.242768984 PcsCompu_dd:aa:aa  PcsCompu_9f:16:c6  ARP        42 192.168.0.12 is at 08:00:27:dd:aa:aa
  515 450.133756996 192.168.0.12       192.168.0.1        DHCP      326 DHCP Request  - Transaction ID 0xfedb8b5b
  516 450.146773654 192.168.0.1        192.168.0.12       DHCP      342 DHCP ACK      - Transaction ID 0xfedb8b5b
```

可以發現在 Server 發完 DHCPACK 之後，才發送 ARP Request，Client 也才能回 ARP Reply，由於 Client 於收到 DHCPACK 之前都屬於尚未設定 IP 的狀態，因此無法響應 ARP，所以 Server 透過 UDP Socket 下去送封包時，可以將其放至 buffer，但無法填 Ethernet header ，因為等不到 ARP 的 Reply，所以需要自己填 Ethernet header。

此時可以回來看到 DHCP 的 header 當中其時包含了：

```
htype        1  Hardware address type, see ARP section in "Assigned
                Numbers" RFC; e.g., '1' = 10mb ethernet.
hlen         1  Hardware address length (e.g.  '6' for 10mb
                ethernet).
chaddr      16  Client hardware address.
```

用以提供 MAC 的資訊，方便我們填寫 Layer 2 header (Ethernet header)：

```
// Prepare the Ethernet frame
struct ethhdr ethernetHeader;
std::memcpy(ethernetHeader.h_source, srcMac, 6);     // Source MAC
std::memcpy(ethernetHeader.h_dest, dstMac, 6);       // Destination MAC
ethernetHeader.h_proto = htons(ETH_P_IP);            // Ethertype
```

再來填 Layer 3 header (IP header)：

```
// Prepare the IP packet
struct iphdr ipHeader;
std::memset(&ipHeader, 0, sizeof(struct iphdr));
ipHeader.version = 4;                                 // IPv4
ipHeader.ihl = 5;                                     // Header length in 32-bit words
ipHeader.tos = 0;                                     // Type of service (0 for default)
ipHeader.tot_len = htons(sizeof(struct iphdr));       // Total length of the IP packet
ipHeader.id = htons(12345); ;                         // Identification (0 for default)
ipHeader.frag_off = htons(IP_DF);                     // Fragment offset (0 for default)
ipHeader.ttl = 64;                                    // Time-to-live (64 for default)
ipHeader.protocol = IPPROTO_UDP;                      // Protocol (UDP)
ipHeader.check = 0;                                   // Checksum (0 for now)
ipHeader.saddr = inet_addr(server_ip);                // Source IP address
ipHeader.daddr = inet_addr(ip_s);                     // Destination IP address
```

其中 Source IP 為 Server IP，而 Destination IP 則為預計要分配給 Client 的 IP，另外當中比較重要的是 IP header 的 checksum 範圍僅 IP header、tot_len 則是包含 payload，因此要先填完後面的內容得知總長度，填回 tot_len 再單獨拿 IP_header 算 checksum 。

```
ipHeader.tot_len = htons(sizeof(struct iphdr) + sizeof(struct udphdr) + payloadLength);
ipHeader.check = in_cksum((unsigned short *)&ipHeader, sizeof(struct iphdr));
```

接著填 Layer 4 header (UDP header)：

```
struct udphdr udpHeader;
std::memset(&udpHeader, 0, sizeof(struct udphdr));
udpHeader.source = htons(67);                          // Source port
udpHeader.dest = htons(68);                            // Destination port
udpHeader.len = htons(sizeof(struct udphdr));          // Length of UDP header and payload
udpHeader.check = 0;                                   // Checksum (0 for now)
```

Server 固定為 67 Port，Client 固定為 68 Port，另外 len 也是包含 payload，不過 checksum 則不只是 header ，也包含 payload。checksum 要先將 checksum 位置填 0 ，整段拿去算完再填會去。

```
unsigned short in_cksum(unsigned short *addr, int len) {
    int nleft = len;
    int sum = 0;
    unsigned short *w = addr;
    unsigned short answer = 0;
    // Sum all 16-bit words
    while (nleft > 1) {
        sum += *w++;
        nleft -= 2;
    }
    // Add the padding byte if necessary
    if (nleft == 1) {
        *(unsigned char *)(&answer) = *(unsigned char *)w;
        sum += answer;
    }
    // Add the carry
    sum = (sum >> 16) + (sum & 0xFFFF);
    sum += (sum >> 16);
    answer = ~sum;
    return answer;
}
```

最後再將所有 header 、payload 組起來即可：

```
// Combine the headers and payload into a buffer
char buffer[sizeof(struct ethhdr) + sizeof(struct iphdr) + sizeof(struct udphdr) +
payloadLength];
std::memcpy(buffer, &ethernetHeader, sizeof(struct ethhdr));
std::memcpy(buffer + sizeof(struct ethhdr), &ipHeader, sizeof(struct iphdr));
std::memcpy(buffer + sizeof(struct ethhdr) + sizeof(struct iphdr), &udpHeader,
sizeof(struct udphdr));
std::memcpy(buffer + sizeof(struct ethhdr) + sizeof(struct iphdr) + sizeof(struct udphdr),
payload, payloadLength);
```

另外要傳出去，須設定 Interface，再 call sendto：

```
// Send the packet through the raw socket
struct sockaddr_ll socketAddress;
std::memset(&socketAddress, 0, sizeof(struct sockaddr_ll));
socketAddress.sll_family = AF_PACKET;
socketAddress.sll_protocol = htons(ETH_P_ALL);
socketAddress.sll_ifindex = if_nametoindex(interface);
socketAddress.sll_pkttype = PACKET_OTHERHOST;
socketAddress.sll_halen = ETH_ALEN;
std::memcpy(socketAddress.sll_addr, dstMac, 6);
if (sendto(rawSocket, buffer, sizeof(buffer), 0, (struct sockaddr*)&socketAddress,
sizeof(socketAddress)) == -1) {
    std::cerr << "Failed to send raw packet." << std::endl;
    return;
}
```

將其包成一個 function 方便使用：

```
void sendDHCP(unsigned char* dstMac, char ip_s[], DHCPMessage* payload){
    int rawSocket = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
    if (rawSocket == -1) {
        std::cerr << "Failed to create raw socket." << std::endl;
        return;
    }
    // Prepare the Ethernet frame
    ...
    // Prepare the IP packet
    ...
    // Prepare the UDP packet
    ...
    // Combine the headers and payload into a buffer
    ...
    // Send the packet through the raw socket
    ...
    // Close the raw socket
    close(rawSocket);
}
```

將收到的 Request ，依據存有的數據確認可分配 IP ，再將其填入 Response 中，其中 op 要設成 2 = BOOTREPLY、xid 要和 Request 一樣來用以識別、ciaddr (Client IP address) 只在 client 處於可以響應 ARP 時填寫 (用於續約時)、其它資訊則填至 options 中，要去確認各項資訊的 option 編號。

```
// Setting
// 192.168.0.0+X
int ip_begin = 100;
int ip_end = 200;
// netmask
uint32_t netmask = 0xFFFFFF00;
// Interface
string interface = "enp0s3";
// Server MAC
unsigned char srcMac[6] = {0x08, 0x00, 0x27, 0x9f, 0x16, 0xc6};
// Server IP
uint32_t serverIP = 0xC0A80001;
void handleDHCPRequest(int serverSocket) {
    struct sockaddr_in clientAddress;
    socklen_t clientAddressLength = sizeof(clientAddress);
    // Allocate memory for receiving the DHCP message
    DHCPMessage* request = (DHCPMessage*)malloc(sizeof(DHCPMessage));
    ssize_t bytesRead = recvfrom(serverSocket, request, sizeof(DHCPMessage), 0, (struct
sockaddr *)&clientAddress, &clientAddressLength);
    // Parse
    ...
    // Find Usage IP
    ...
    // Fill DHCP
    ...
    // Send
    if (messageType == DHCPDISCOVER) {
        response->options[2] = 2; // DHCP Offer
        sendDHCP((uint8_t*)request->chaddr, ip_s, response);
    } else if (messageType == DHCPREQUEST) {
        response->options[2] = 5; // DHCP Ack
        sendDHCP((uint8_t*)request->chaddr, ip_s, response);
    }
}
```

另外 option 中有一個是告知 client 多久需要續約，使得不會 client 沒在用，卻 IP 被占用的狀況發生。

以上即是簡易的 DHCP Server 的實作。

Reference

https://datatracker.ietf.org/doc/html/rfc2131