Using the DHCP client to reconfigure network settings.

What is DHCP?

The dynamic host configuration protocol is one of two main ways a host can assign an IP address to itself when it is present in a local area network (the other being statically). DHCP follows a state machine model to obtain, maintain, and update the IP address over time. The server leases the address to the host for a certain amount of time determined by the server and the state machine handles the lease expiry and other eventualities such as a drop in the link or if the server becomes inaccessible.

When the DHCP process begins (typically after the host boots), the client sends out a series of DHCPDISCOVER messages to the broadcast address (255.255.255.255) prompting a server to provide it with configuration parameters. The server is expected to supply the client with as much of the requested information as it can provide and does this by populating a DHCPOFFER message with the information and sending it to the client. If the client chooses to accept these parameters, it sends the server a DHCPREQUEST and after the server acknowledges it the IP address is considered to be bound to the host.

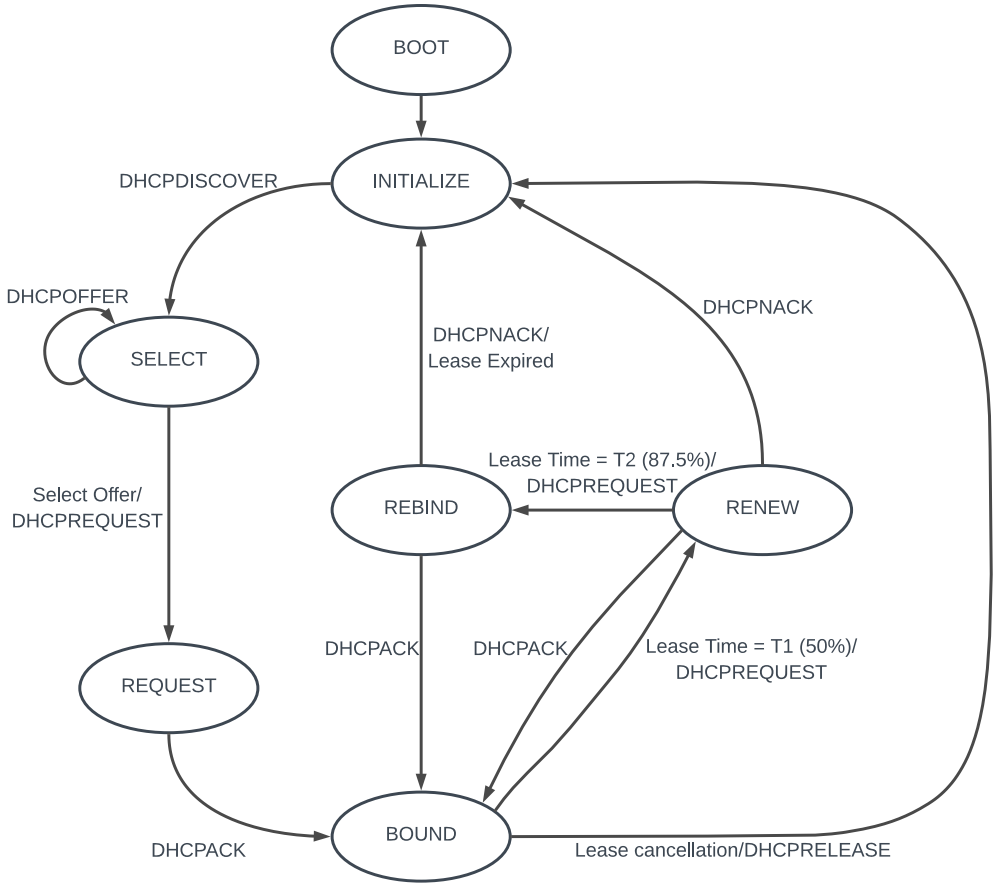

Figure 1: DHCP state machine. Only the INITIALIZE, SELECT, REQUEST, and BOUND states are relevant to this blog.

How may uC modules be reconfigured?

Internally, uC/DHCPc prompts the server for a predetermined set of request parameters (DHCP_OPT_PARAMETER_REQUEST_LIST) that include the subnet mask, the server identifier (router), the domain name server, and the time zone offset. These, and other request parameters can be solicited and later used to re-configure other uC modules such as uC/DNSc, uC/TFTPc, uC/SNTPc, uC/SNMPc, and uC/Clk at run time in case there are changes in the local area network. This can be achieved by way of invoking the `DHCPc_Stop()`, `DHCPc_Start()`, and the `DHCPc_GetOptVal()` API functions as shown in Figure 3. Please note that the number of parameters to be requested cannot exceed the value configured by the `DHCPc_CFG_PARAM_REQ_TBL_SIZE` macro. Also, in order for this to work at runtime, the instantiated configuration structures for each module should NOT be declared with the `const` keyword.

From the application's perspective

A uC/TCPIP-based application can synchronously or asynchronously update these parameters. For instance, there can be a task that is dedicated to stop and start the DHCP process directly or alternatively after the TCPIP stack is initialized the application may use `NetIF_LinkStateSubscribe()` to monitor link state changes via a callback function that posts to a task pending on a global semaphore object. The former scenario is demonstrated in Figure 3 where a task (`AppTask_DHCP_Update()`) running every 24 hours requests some parameters and reconfigures the DNS and NTP server information. Note that `req_param[0]` and `req_param[2]` are not fed to `DHCPc_GetOptVal()` and are only there for illustrative purposes. It is also important to mention that as previously stated, the server might not know a subset of the requested parameters and thus when the client is bound and `DHCPc_GetOptVal()` is called requesting said parameter, the API returns a `DHCPc_ERR_IF_OPT_NONE` error and the old value is kept in the module's configuration structure.

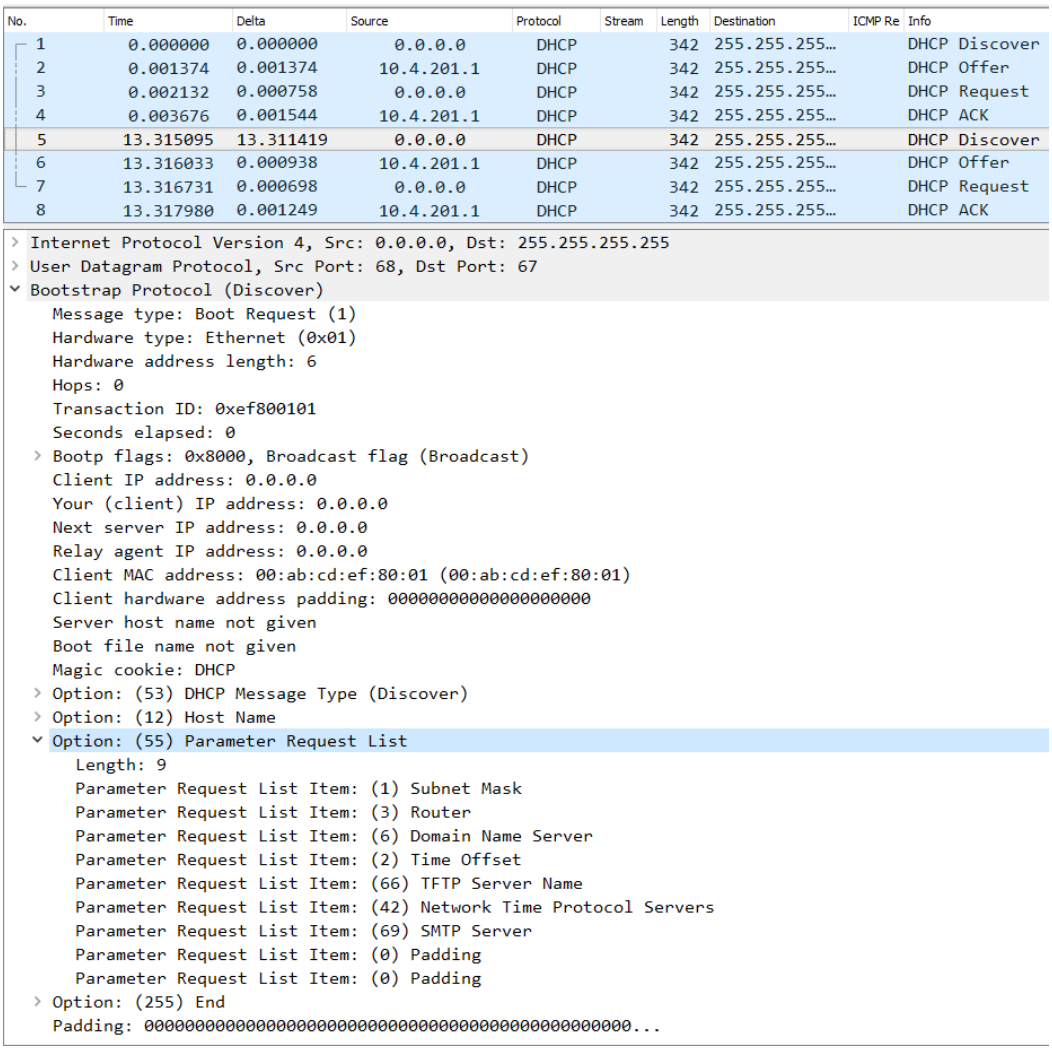

Figure 2: Wireshark capture showing new parameters requested by second DHCPDISCOVER message when DHCPc is restarted.

```
/*
*********************************************************************************************************
*                                        AppTask_DHCP_Update()
*
* Description : Initialize DHCP client for specified interface.
*
* Argument(s) : p_arg   is the argument passed to 'AppTask_DHCP_Update()' by 'OSTaskCreate()'.
*
* Return(s)   : none.
*
* Note(s)     : none.
*********************************************************************************************************
*/

void  AppTask_DHCP_Update(void *p_arg)
{
    DHCPc_OPT_CODE     req_param[DHCPc_CFG_PARAM_REQ_TBL_SIZE];
    DHCPc_ERR          err_dhcp;
    DHCPc_STATUS       dhcp_status;
    CPU_BOOLEAN        cfg_done;
    CPU_BOOLEAN        dly;
    NET_IPv4_ADDR      local_ntp_server_ip;
    CPU_INT16U         ntp_server_ip_len;
    CPU_CHAR           ntp_server_ip_str[NET_ASCII_LEN_MAX_ADDR_IPv4];
    NET_IPv4_ADDR      local_dns_server_ip;
    CPU_INT16U         dns_server_ip_len;
    CPU_CHAR           dns_server_ip_str[NET_ASCII_LEN_MAX_ADDR_IPv4];
    NET_IF_NBR         if_nbr;


    while (DEF_TRUE) {                                          /* Task body, always written as an infinite loop.       */
        Mem_Clr(req_param, DHCPc_CFG_PARAM_REQ_TBL_SIZE);

        req_param[0] = DHCP_OPT_TFTP_SERVER_NAME;
        req_param[1] = DHCP_OPT_NETWORK_TIME_PROTOCOL_SERVER;
        req_param[2] = DHCP_OPT_SIMPLE_MAIL_TRANSPORT_PROTOCOL_SERVER;

        if_nbr = Eth1_IF_Nbr;                                   /* Global var containing IF nbr assigned by NetIF_Add().*/
        DHCPc_Stop(if_nbr, &err_dhcp);
        OSTimeDly(1u, OS_OPT_NONE, &err);                       /* Allow DHCP Task to run.                              */

        DHCPc_Start((NET_IF_NBR      ) if_nbr,
                    (DHCPc_OPT_CODE *)&req_param[0],
                    (CPU_INT08U      ) sizeof(req_param),
                    (DHCPc_ERR      *)&err_dhcp);
        if (err_dhcp != DHCPc_ERR_NONE) {
            APP_TRACE_INFO(("DHCPc_Start() failed w/err = %d \r\n", err_dhcp));
        }

        APP_TRACE_INFO(("DHCP address configuration started\n\r"));

                                                                /* ------ CHECK INTERFACE'S DHCP CONFIGURE STATUS ----- */
        dhcp_status = DHCP_STATUS_NONE;
        cfg_done    = DEF_NO;
        dly         = DEF_NO;

        while (cfg_done != DEF_YES) {
            if (dly == DEF_YES) {
                OSTimeDlyHMSM( 0u, 0u, 0u, 100u,
                               OS_OPT_TIME_HMSM_STRICT,
                              &err);
            }

            dhcp_status = DHCPc_ChkStatus(if_nbr, &err_dhcp);
            switch (dhcp_status) {
                case DHCP_STATUS_CFGD:
                     APP_TRACE_INFO(("DHCP address configured\n\r"));
                     cfg_done          = DEF_YES;
                     ntp_server_ip_len = NET_IPv4_ADDR_LEN;
                     DHCPc_GetOptVal(              if_nbr,
                                                   DHCP_OPT_NETWORK_TIME_PROTOCOL_SERVER,
                                     (CPU_INT08U *)&local_ntp_server_ip,
                                                   &ntp_server_ip_len,
                                                   &err_dhcp);
                     if (err_dhcp == DHCPc_ERR_NONE) {
                         local_ntp_server_ip = NET_UTIL_NET_TO_HOST_32(local_ntp_server_ip);
                         Mem_Clr(ntp_server_ip_str, NET_ASCII_LEN_MAX_ADDR_IPv4);
                         NetASCII_IPv4_to_Str( local_ntp_server_ip,
                                               ntp_server_ip_str,
                                               DEF_NO,
                                              &net_err);
                         if (net_err == NET_ASCII_ERR_NONE) {
                             SNTPc_Cfg.ServerHostnamePtr = (CPU_CHAR *)&ntp_server_ip_str[0u];
                             APP_TRACE_INFO(("Successfully re-configured default NTP server to: %s.\r\n",
                                             ntp_server_ip_str));
                         }
                     }

                     dns_server_ip_len = NET_IPv4_ADDR_LEN;
                     DHCPc_GetOptVal(              if_nbr,
                                                   DHCP_OPT_DOMAIN_NAME_SERVER,
                                     (CPU_INT08U *)&local_dns_server_ip,
                                                   &dns_server_ip_len,
                                                   &err_dhcp);
                     if (err_dhcp == DHCPc_ERR_NONE) {
                         local_dns_server_ip = NET_UTIL_NET_TO_HOST_32(local_dns_server_ip);
                         Mem_Clr(dns_server_ip_str, NET_ASCII_LEN_MAX_ADDR_IPv4);
                         NetASCII_IPv4_to_Str( local_dns_server_ip,
                                               dns_server_ip_str,
                                               DEF_NO,
                                              &net_err);
                         if (net_err == NET_ASCII_ERR_NONE) {
                             DNSc_Cfg.ServerDfltPtr = (CPU_CHAR *)&dns_server_ip_str[0u];
                             APP_TRACE_INFO(("Successfully re-configured default DNS server to: %s.\r\n",
                                             dns_server_ip_str));
                         }
                     }
                     break;
```

```
109
110
111                     case DHCP_STATUS_CFGD_NO_TMR:
112                          APP_TRACE_INFO(("DHCP address configured (no timer)\n\r"));
113                          cfg_done = DEF_YES;
114                          break;
115
116
117                     case DHCP_STATUS_CFGD_LOCAL_LINK:
118                          APP_TRACE_INFO(("DHCP address configured (link-local)\n\r"));
119                          cfg_done = DEF_YES;
120                          break;
121
122
123                     case DHCP_STATUS_FAIL:
124                          APP_TRACE_INFO(("DHCP address configuration failed\n\r"));
125                          cfg_done = DEF_YES;
126                          break;
127
128
129                     case DHCP_STATUS_CFG_IN_PROGRESS:
130                     default:
131                          dly = DEF_YES;
132                          break;
133             }
134         }
135
136         OSTimeDlyHMSM(24u, 0u, 0u, 0u,
137                       OS_OPT_TIME_HMSM_STRICT,
138                       &err);
139     }
140 }
```

Figure 3: Sample application task that restarts DHCP process with new request parameters.