Sending your first message

This page: *how to initialize your Ethernet driver and send your first message. A lot of constants can be seen in the explanatory code. Those can be found in* `components/common/networking_constants`*. They are also added at the end of this file.*

> The functions that are used for Ethernet can be found in `components/common/networking/inc/ethernet.h`.

> The headers in the folder `components/common/networking/stm/` are only to be used inside the Ethernet driver!

> **NOTE:** The implementation only works with FreeRTOS, but it works fine with only 1 thread. **Before you implement anything, do an introductory tutorial on FreeRTOS!**

Code Usage

1) Linker file updating

In the linker file, you need to say what memory addresses LWIP uses. You set these values in Cubemx and the sections are generated for you, but you have to add it to the linker file yourself.

> You can find your linker (`.ld`) file in `components/{env_name}/firmware/`

The codeblock added for the default values is:

```
.lwip_sec (NOLOAD) :
  {
      . = ABSOLUTE(0x30000000);
      *(.RxDecripSection)

      . = ABSOLUTE(0x30000080);
```

```
      *(.TxDecripSection)

      . = ABSOLUTE(0x30000100);
      *(.Rx_PoolSection)

  } >RAM_D2
```

You can just use the generated linker file and append to it, because it never gets regenerated, but you can also create a new one. If you create a new one, you have to put in the platformio configuration file that you use the new linker file. An example of how to do it is shown below. The path is relative to where the platformio configuration file is located.

```
board_build.ldscript = components/network_board/firmware/STM32H753XX_FLASH.ld
```

---

2) Code initialization

When you first use the Ethernet driver, you have to initialize a few things.

First make sure that HAL is initialized, the D and I cache are enabled, the system clock and the memory protection unit as well. These auto-generated functions start processes that Ethernet uses.

```
//Initialize HAL
HAL_Init();

//Enable D&I cache (for ETH)
SCB_EnableICache();
SCB_EnableDCache();

//Start the system clock
SystemClock_config();

//Memory protection unit
MPU_Config_wrapper();
```

Note for debugging with SCB_EnableDCache()

When debugging and trying to step over `SCB_EnableDCache()`, the program will keep infinitely running **unless** you place a breakpoint on the next line **while in active debugging mode**.

ETH_Init(...)

```
result_t ETH_init(linkstatus_callback_t link_state_change_callback,
                  uint8_t ip[4], uint8_t netmask[4],
                  uint8_t gateway[4],
                  uint8_t mac_address[6])
```

For `ETH_init`, the `link_status_change_callback` can be set to `NULL`, to use the default function. The others have to be set. For initial testing you can set these to the same values as set in CubeMX.

Example usage of ETH_init(...)

```
//Example values
uint8_t ip[4] = {192, 168, 0, 223};
uint8_t mac[6] = {255, 255, 255, 255, 255, 255};
uint8_t gateway[4] = {192, 168, 0, 1};
uint8_t netmask[4] = {255, 255, 255, 0};

ETH_init(NULL, ip, netmask, gateway, mac);
```

> **NOTE on multithreading:** make sure you set the stack size big enough (I use `.stack_size = 1024 * 8`) when setting the `task_attributes` for the thread Ethernet will be running in!

ETH_udp_init()

```
void ETH_udp_init(uint8_t sender_prio_buf,
                  QueueHandle_t *send_queues,
                  receive_callback_t receiver_callback)
```

- `sender_prio_buf`
  Defines the **number** of priority queues you use for queuing the send messages. The queues we use are freeRTOS queues. More information can be found here.
- `send_queues`
  You need to pass freeRTOS queues as a pointer to an array. The queues need to be implemented before passing them to `ETH_udp_init(...)`, see below.

Example queue implementation

```
int SendQueueSize = 80;

//Create queue 1
static StaticQueue_t xStaticQueue1;
uint8_t ucQueueStorageArea1[SendQueueSize * ETHERNET_SQ_ITEM_SIZE]; //NOTE: constants from
<ethernet_constants.h>
QueueHandle_t udp_receiver_queue1 = xQueueCreateStatic(SendQueueSize,
ETHERNET_SQ_ITEM_SIZE, ucQueueStorageArea1, &xStaticQueue1);

//Create queue 2
static StaticQueue_t xStaticQueue2;
uint8_t ucQueueStorageArea2[SendQueueSize * ETHERNET_SQ_ITEM_SIZE];
QueueHandle_t udp_receiver_queue2 = xQueueCreateStatic(SendQueueSize,
ETHERNET_SQ_ITEM_SIZE, ucQueueStorageArea2, &xStaticQueue2);

//queues object we pass as an argument later
QueueHandle_t queues[2] = {udp_receiver_queue1, udp_receiver_queue2};
```

- `receiver_callback`
  This can be any function that takes a `receive_frame_t`, found in `ethernet_udp.h`. However, we also implemented a "packet dispatcher", to handle incoming packets.

> More information about the packet dispatcher is in the Packet Dispatcher documentation.

Example implementation WITHOUT packet dispatcher

```
void HandlePacket(receive_frame_t *receive_frame) {
 printf("Wayoo, message received");
}

int main(void) {
  ...
  /*The other code examples above would go HERE */
  ETH_udp_init(2, queues, HandlePacket);
  ...
```

```
}
```

Now Ethernet is started and you can **receive packages**!

---

3) Sending Packets

ETH_add_arp(...)

```
result_t ETH_add_arp(uint8_t ip[4], uint8_t mac[6], int retry_count)
```

To send to a certain IP, you have to set the ARP table. Otherwise, it will only send ARP resolution messages. You do that by using the `ETH_add_arp(...)` function with the IP you want to send to, and any random mac address.

> **WARNING:** Make sure you have initialized udp **before** using ETH_add_arp(...)!

ETH_udp_send(...)

```
void ETH_udp_send(uint8_t ip[4],
                  uint8_t port,
                  uint8_t *payload,
                  uint16_t payload_len,
                  uint8_t prio_num)
```

- `payload`
  The payload can be any byte array, where the size of that byte array is the 4th argument: `payload_len`.
- `prio_num`
  Priority level for transmission. Must be less than `sender_prio_buf` specified in `ETH_udp_init(...)`.

Now you can send messages!

Example: sending a message

```
//NOTE: These are the receiving ip/mac,
//not the sending ones we specified earlier
uint8_t ip[4] = SAMPLE_BOARD_IP;
uint8_t mac[6] = SAMPLE_BOARD_MAC; //constants from <ip_mac_constants.h>

ETH_add_arp(ip, mac, 5);

uint8_t packet1_payload[4] = {14,06,20,04};
ETH_udp_send(ip, 8, packet1_payload, 4, 1); //Send a single packet
```

Appendix

1) Networking_constants

```
#ifndef IP_MAC_CONSTANTS
#define IP_MAC_CONSTANTS

#define SAMPLE_BOARD_IP {192, 168, 0, 111}
#define SAMPEL_BOARD_MAC {0x00, 0x80, 0xe1, 0x00, 0x00, 0x00}

#define NETWORK_IP {192, 168, 0, 223}
#define NETWORK_MAC {0x00, 0x43, 0x23, 0xee, 0x21, 0x64}

#define GATEWAY {192, 168, 0, 1}
#define NETMASK {255, 255, 255, 0}
#endif //! IP_MAC_CONSTANTS
```

2) Full Example Code

> Import `"networking_constants.h"` and `"ip_mac_constants.h"` to use the predefined test constants

```
#include "ethernet.h"
#include "networking_constants.h"
#include "ip_mac_constants.h"


/* Callback function that handles a specific packet*/
void HandlePacket(receive_frame_t *receive_frame) {
    printf("Wayoo, message received");
}

int outgoing_counter = 0;
int main(void) {
    /*Inits*/
    HAL_Init();
    SystemClock_Config();

    MPU_Config_wrapper();

    SCB_EnableICache();
    SCB_EnableDCache();

    MX_GPIO_Init();

    /*Config + init sending side*/
    uint8_t mac[6] = NETWORK_MAC;
    uint8_t ip[4] = NETWORK_IP;
    uint8_t netmask[4] = NETMASK;
    uint8_t gateway[4] = GATEWAY;

    ETH_init(NULL, ip, netmask, gateway, mac);

    /*Making queues*/
    int SendQueueSize = 80;

    static StaticQueue_t xStaticQueue1;
    uint8_t ucQueueStorageArea1[SendQueueSize * ETHERNET_SQ_ITEM_SIZE];
    QueueHandle_t udp_receiver_queue1 = xQueueCreateStatic(SendQueueSize,
ETHERNET_SQ_ITEM_SIZE, ucQueueStorageArea1, &xStaticQueue1);

    static StaticQueue_t xStaticQueue2;
```

```
    uint8_t ucQueueStorageArea2[SendQueueSize * ETHERNET_SQ_ITEM_SIZE];
    QueueHandle_t udp_receiver_queue2 = xQueueCreateStatic(SendQueueSize,
ETHERNET_SQ_ITEM_SIZE, ucQueueStorageArea2, &xStaticQueue2);

    QueueHandle_t queues[2] = {udp_receiver_queue1, udp_receiver_queue2};

    ETH_udp_init(2, queues, HandlePacket);

    /*Config + add ARP receiving side*/
    uint8_t ip[4] = SAMPLE_BOARD_IP;
    uint8_t mac[6] = SAMPLE_BOARD_MAC;

    ETH_add_arp(ip, mac, 5);

    /*Sending a message*/
    uint8_t packet1_payload[4] = {14,06,20,04};

    /*Test sending*/
    while (outgoing_counter < 100) { //NOTE: after 80 packages the queue will be full!
          ETH_udp_send(ip, 8, packet1_payload, 4, 1);
          osDelay(10);
          outgoing_counter += 1;
          LOGI(TAG, "%d", outgoing_counter);
      }

}
```

Revision #41
Created 2026-04-16 13:25:02 UTC by Jaron Lendering
Updated 2026-05-12 20:40:01 UTC by Lisa te Braak