Entrega 2
INE-5424

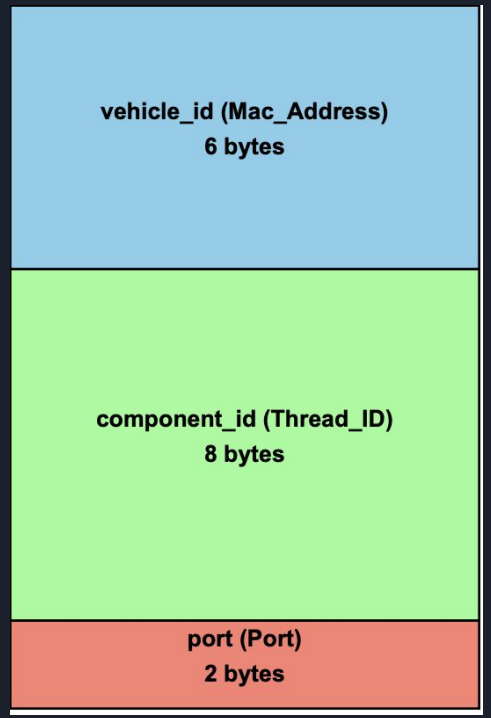
Identificação dos Componentes

Address:

- MAC address - Endereço da NIC
- Thread ID - Identificador da thread
- Port

Identificação dos Componentes

```
struct Address { // (16 bytes)
    Mac_Address vehicle_id; // (6 bytes)
    Thread_ID component_id; // (8 bytes)
    Port port; // (2 bytes)
    } __attribute__((packed));
```



Identificação dos Componentes

- Desse modo cada componente será identificado pela combinação desses três números, gerando um identificador único

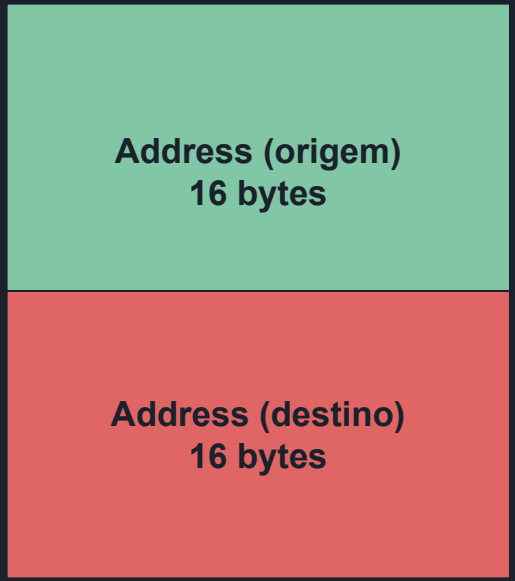
Identificação dos Componentes

```
struct Header { // (32 bytes)

        Address src_address; // Endereço de
origem (16 bytes)

        Address dst_address; // Endereço de
destino (16 bytes)

    } __attribute__((packed));
```



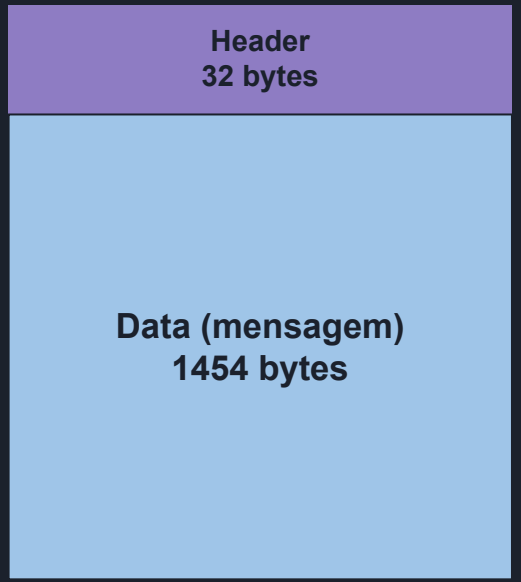
Identificação dos Componentes

```
struct Payload {

        Header header; // Cabeçalho do
protocolo 32 bytes

        uint8_t data[1454]; // Mensagem a ser
transmitida 1454 bytes

    } __attribute__((packed));
```

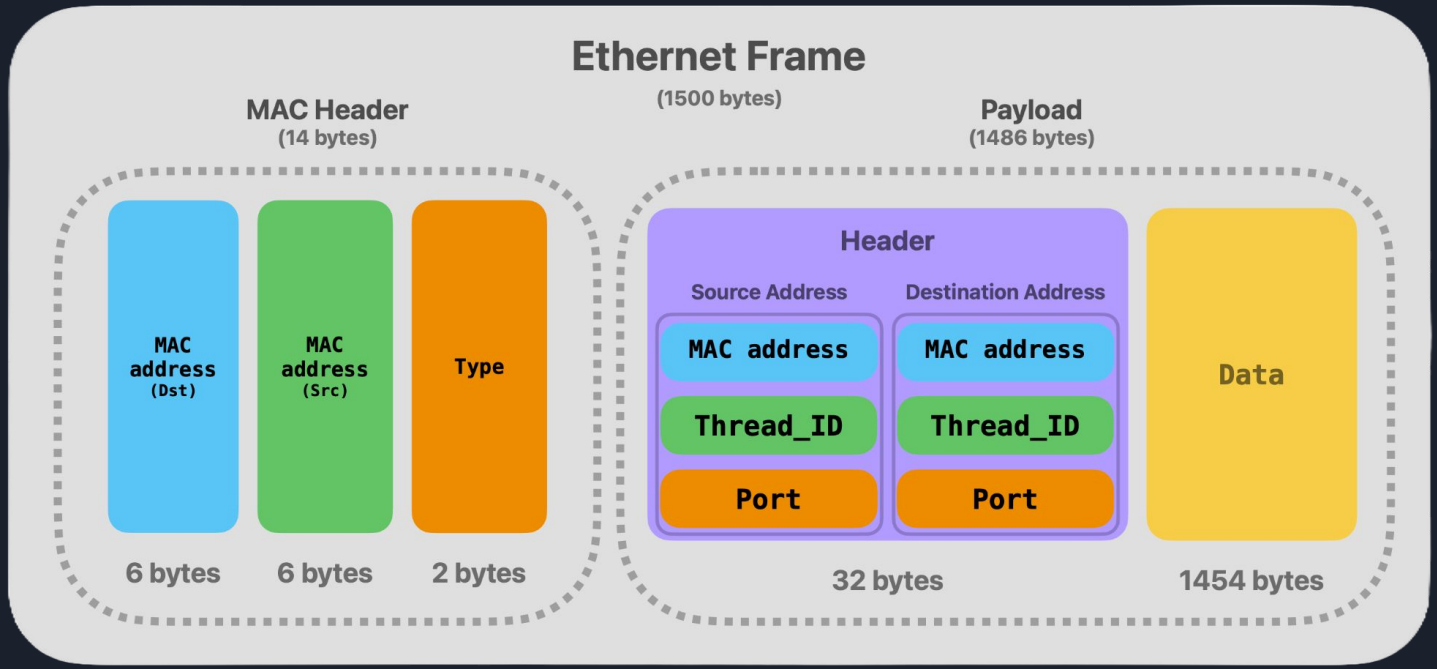


Identificação dos Componentes

- Estrutura do Frame Ethernet
  - MAC Address destino: Broadcast
  - MAC Address origem: NIC
  - Protocol Number: 0x88B5
  - Payload: Header + Mensagem

Identificação dos Componentes

Frame Ethernet:



Comunicação Interna

- Segunda engine especializada em comunicação interna
    - Cópia da engine original, utilizando uma fila única como memória compartilhada
    - Caminho de recebimento continua o mesmo, utilizando callback para NIC receive

Comunicação Interna

- Métodos da Engine Interna:

```cpp
// Method to add data to the queue
int InternalEngine::send(const void* data, size_t size) {
    std::vector<char> buffer(static_cast<const char*>(data), static_cast<const char*>(data) + size);

    {
        std::lock_guard<std::mutex> lock(queue_mutex);
        buffer_queue.emplace(std::move(buffer), size);
    }
    queue_cv.notify_one();
    return 1; // Return 1 to indicate success
}
```

```cpp
// Method to process the queue
void InternalEngine::process_queue() {
    while (true) {
        std::pair<std::vector<char>, size_t> item;

        // Wait until there is data in the queue or stop_processing is true
        {
            std::unique_lock<std::mutex> lock(queue_mutex);
            queue_cv.wait(lock, [this]() { return !buffer_queue.empty() || stop_processing; });

            if (stop_processing && buffer_queue.empty()) {
                break;
            }

            item = std::move(buffer_queue.front());
            buffer_queue.pop();
        }

        // Process the item using the callback
        if (_callback) {
            _callback(item.first.data(), item.second);
        }
    }
```

Comunicação Interna

- Envio de mensagens
  - Protocol irá decidir se a mensagem é de comunicação interna ou externa utilizando o header com as informações de origem e destino

```
// Verifica se o endereço MAC de origem e destino são iguais
if (from.vehicle_id == to.vehicle_id) {
    is_internal = true; // Define como interno se os endereços forem iguais
}
```

Comunicação Interna

- Envio de mensagens
  - Se o MAC destino for o mesmo de origem significa que o recipiente é um componente do mesmo carro

Testes : Comunicação interna

```
int teste_comunicacao_interna(std::string networkInterface, int totalMessages)
```

```
Veiculo veiculo_a(NETWORK_INTERFACE, "Veiculo A", MAC_VEICULO_A);
veiculo_a.cria_thread_receptor("Receptor 1", MAC_VEICULO_A, receptor_1_id, PORTA_RECEPTOR_1, NUM_MENSAGENS * 3);
veiculo_a.cria_thread_receptor("Receptor 2", MAC_VEICULO_A, receptor_2_id, PORTA_RECEPTOR_2, NUM_MENSAGENS * 3);

// Cria Enviadores internos.
veiculo_a.cria_thread_enviador("Enviador a1", MAC_VEICULO_A, a1_id, PORTA_ENVIADOR, endereco_receptor_1, NUM_MENSAGENS);
veiculo_a.cria_thread_enviador("Enviador a2", MAC_VEICULO_A, a2_id, PORTA_ENVIADOR, endereco_receptor_2, NUM_MENSAGENS);
veiculo_a.cria_thread_enviador("Enviador b1", MAC_VEICULO_A, b1_id, PORTA_ENVIADOR, endereco_receptor_1, NUM_MENSAGENS);
veiculo_a.cria_thread_enviador("Enviador b2", MAC_VEICULO_A, b2_id, PORTA_ENVIADOR, endereco_receptor_2, NUM_MENSAGENS);
veiculo_a.cria_thread_enviador("Enviador c1", MAC_VEICULO_A, c1_id, PORTA_ENVIADOR, endereco_receptor_1, NUM_MENSAGENS);
veiculo_a.cria_thread_enviador("Enviador c2", MAC_VEICULO_A, c2_id, PORTA_ENVIADOR, endereco_receptor_2, NUM_MENSAGENS);
```

```
Receptor 2: média de latência total = 0.333333 ms
```

Testes : Comunicação externa

```
int teste_comunicacao_externa(std::string networkInterface, int totalMessages)
```

```
pid_t pid_a = fork();

if (pid_a == 0) {
    // Cria Veículo A e Receptor
    Veiculo veiculo_a(NETWORK_INTERFACE, "Veiculo A", MAC_VEICULO_A);
    veiculo_a.cria_thread_receptor("Receptor A", MAC_VEICULO_A, receptor_a_id, PORTA_RECEPTOR_A, NUM_MENSAGENS * 3);

    std::this_thread::sleep_for(std::chrono::milliseconds(100));

    // Cria Enviadores internos (veículo A)
    veiculo_a.cria_thread_enviador("Enviador a1", MAC_VEICULO_A, a1_id, PORTA_ENVIADOR, endereco_receptor_b, NUM_MENSAGENS);
    veiculo_a.cria_thread_enviador("Enviador a2", MAC_VEICULO_A, a2_id, PORTA_ENVIADOR, endereco_receptor_b, NUM_MENSAGENS);
    veiculo_a.cria_thread_enviador("Enviador a3", MAC_VEICULO_A, a3_id, PORTA_ENVIADOR, endereco_receptor_b, NUM_MENSAGENS);
    pthread_join(receptor_a_id, nullptr);
    pthread_join(a1_id, nullptr);
    pthread_join(a2_id, nullptr);
    pthread_join(a3_id, nullptr);
    return 0;
}
```

```
Receptor B: média de latência total = 3 ms
```