

# Entrega 6

## INE-5424



# Interesses: Internos ou Externos

- Utiliza a mesma lógica de diferenciação de comunicação interna e externa.
- Essa decisão é tomada de acordo com o preenchimento do endereço de origem e destino da mensagem a ser enviada.
- Mensagens internas: O ID do veículo é o mesmo, tanto no endereços de origem quanto no de destino. Sinalizando que o veículo de origem é o destino.
- Logo, essas mensagens são transmitidas por comunicação interna.
- Lembrando: O Protocol avalia essa condição durante o processo de envio e informa para a NIC qual é o tipo de comunicação (interna ou externa).



# Interesses: Internos ou Externos

- Determinando tipo da comunicação:

```
int Protocol::send(Address from, Address to, Type type, Period |
    // Verifica se o destino da mensagem é interno ou externo.
    bool is_internal = false;
    if (from.vehicle_id == to.vehicle_id) {
        is_internal = true;
    }

    // Envia o frame Ethernet para a NIC
    return _nic->send(buf, is_internal);
```

- Send Protocol:

```
// Verifica se o endereço de origem e destino são iguais
if (internal) {
    // Se o endereço de origem e destino forem iguais, envia pelo internal_engine
    result = internal_engine->send(frame, sizeof(*frame));
} else {
    // Se o endereço de origem e destino forem diferentes, envia pelo engine normal
    result = engine->send(frame, sizeof(*frame));
}
```

- Send NIC:

# Interesses: Internos ou Externos

- Exemplo de uso na Aplicação:

```
void* rotina_detector_veiculos(void* arg) {  
    Veiculo::DadosComponente* dados = (Veiculo::DadosComponente*)arg;  
    Communicator comunicador(dados->protocolo, dados->id_veiculo, pthread_self());  
  
    while (true) {  
        // Prepara mensagem de interesse para o sensor gps.  
        Message mensagem;  
  
        mensagem.setDstAddress({0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, (pthread_t)0); // Interesse externo.  
        mensagem.setType(Ethernet::TYPE_POSITION_DATA); // Preenche tipo do dado  
  
        // Preenche periodo de interesse.  
        mensagem.setPeriod(0); // Periodo = 0 => Ping (uma unica resposta).  
  
        std::cout << "🚗 " << dados->nome << ": enviou interesse." << std::endl;  
        // Envia mensagem de interesse.  
        comunicador.send(&mensagem);  
    }  
}
```

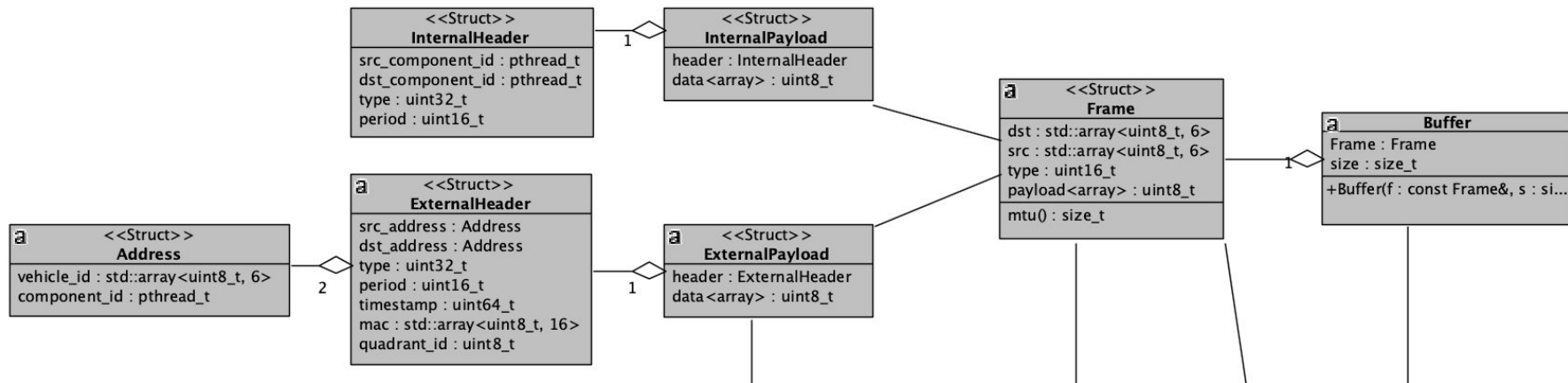
**\*\*Componente DetectorVeiculos envia interesse externo em dados de posição do veículo.\*\***

```
// Envia mensagem de interesse nos dados do GPS interno.  
Message message;  
message.setDstAddress({self->address.vehicle_id, (pthread_t)0});  
message.setPeriod(0);  
message.setType(Ethernet::TYPE_POSITION_DATA);  
communicator.send(&message);
```

**\*\*RSU Handler envia interesse interno em dados de posição do veículo.\*\***

# Estruturas atualizadas

- Estruturas utilizadas no envio e recebimento das mensagens:



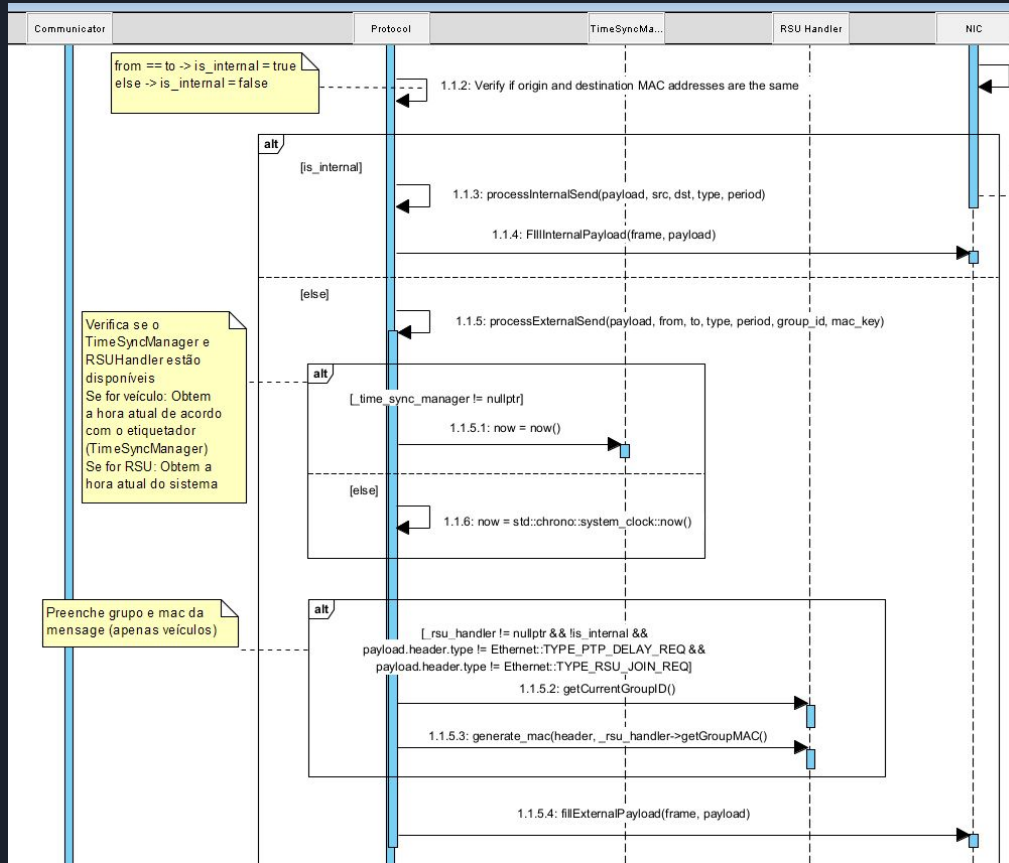
# Divisão do processo de envio e recebimento (interno/externo)

- Classe Protocol atualizada com novos métodos:

```
Protocol
- _nic : NIC<Engine>
- _data_observer : Conditional_Data_Observer
- _observed : Concurrent_Observed
- _time_sync_manager : TimeSyncManager
+ Buffer : NIC<Engine>::Buffer
+ protocol_number : Protocol_Number
- _data_publisher : DataPublisher*

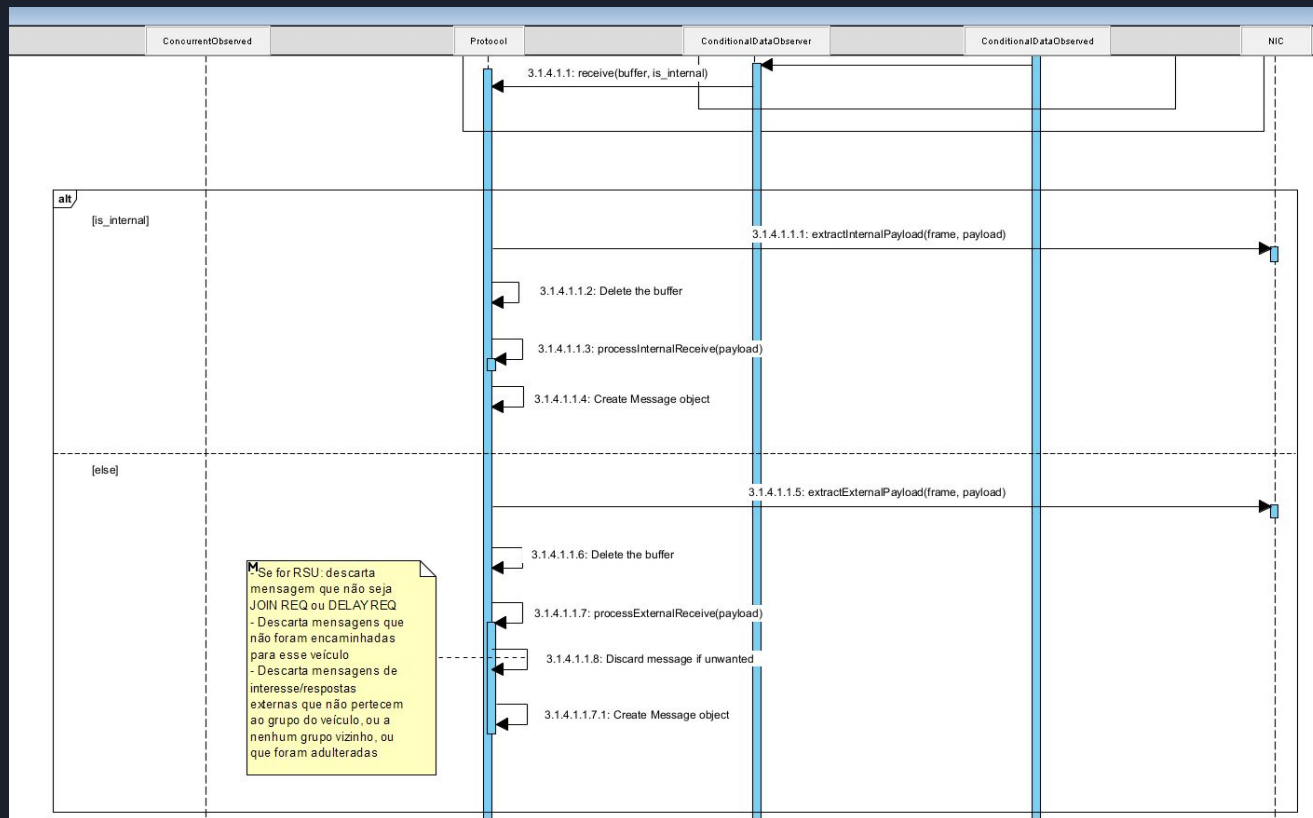
+ Protocol(nic : NIC<Engine>, protocol_number : Protocol_Number)
+ ~Protocol()
+ send(from : Address, to : Address, type : uint32_t, period : uint16_t, group_id : uint8_t, mac : std::array<uint8_t, 16>, data : void*, size : int) : int
+ receive(buf : void*, is_internal : boolean)
+ attach(obs : Concurrent_Observer*)
+ detach(obs : Concurrent_Observer*)
- processInternalSend(payload : InternalPayload, src_component : pthread_t, dst_component : pthread_t, type : uint32_t, period : uint16_t)
- processExternalSend(payload : ExternalPayload, from : Address, to : Address, type : uint32_t, period : uint16_t, group_id : uint8_t, mac : std::array<uint8_t, 16>)
- processInternalReceive(payload : InternalPayload)
- processExternalReceive(payload : ExternalPayload)
```

# Mudanças no Send



- Distinção feita no Protocol
- Remoção de complexidade do send interno
- Verificações relacionadas ao grupo, RSU Handler e preenchimento do mac limitadas ao send externo

# Mudanças no Receive



- Distinção feita no Protocol
- Descarte de mensagens indesejadas exclusivo ao receive externo






# Envio de mensagens internas

- As seguintes informações foram retiradas do Header das mensagens internas:
  - ID Veículo de origem e destino
  - Timestamp
  - MAC
  - Quadrant\_ID

```
// Método de preenchimento das mensagens de envio interno.  
void Protocol::processInternalSend(Ethernet::InternalPayload* payload,  
    Ethernet::Thread_ID src_component, Ethernet::Thread_ID dst_component, Type type, Period period) {  
  
    // Preenche Payload com o cabeçalho e a mensagem  
    payload->header.src_component_id = src_component;  
    payload->header.dst_component_id = dst_component;  
    payload->header.type = type;  
    payload->header.period = period;  
}
```



# Recebimento das mensagens internas

- ID do Veículo de origem e destino é preenchido utilizando o ID do veículo local.
- Timestamp é calculado e preenchido apenas no processo de recebimento.
- A Chave de Grupo é preenchida com a chave do grupo atual ao qual o veículo pertence.
  - Usada como base para gerar o MAC da mensagem quando requisitado.

```
// Método de processamento para as mensagens recebidas internamente.
void Protocol::processInternalReceive(Ethernet::InternalPayload payload) {
    // Preenche mensagem com o cabeçalho e os dados recebidos.
    Message message;
    message.setSrcAddress({_nic->get_address(), payload.header.src_component_id}); // Endereço de origem
    message.setDstAddress({_nic->get_address(), payload.header.dst_component_id}); // Endereço de destino
    message.setType(payload.header.type); // Tipo da mensagem
    message.setPeriod(payload.header.period); // Período de transmissão
    message.setTimestamp(_time_sync_manager->now()); // Horário de envio (mesmo do recebimento)
    message.setGroupKey(_rsu_handler->getGroupMAC(_rsu_handler->getCurrentGroupID())); // Chave MAC do grupo atual (usada para gerar MAC)
    message.setData(payload.data, sizeof(payload.data)); // Copia os dados para a mensagem
}
```

# Requisição do MAC das mensagens recebidas internamente

- Nesse caso, o MAC é gerado apenas quando requisitado pela Aplicação.
  - Utilizando a Chave de Grupo preenchida durante o recebimento como base.
  - O MAC é gerado utilizando a mesma função do RSU Handler (XOR do Header)

```
Message
- _data : uint8_t<array>
- _size : size_t
- _header : ExternalHeader
- _group_key : std::array<uint8_t, 16>

+ Message()
+ data() : uint8_t*
+ size() : size_t
+ setData(src : void*, size : size_t)
+ setSrcAddress(src_address : Address)
+ setDstAddress(dst_address : Address)
+ setType(type : uint32_t)
+ setPeriod(period : uint16_t)
+ getSrcAddress() : Address
+ getDstAddress() : Address
+ getType() : uint32_t
+ getPeriod() : uint16_t
+ setTimestamp(std::chrono::system_clock::time_point tp)
+ setMAC(mac_key : std::array<uint8_t, 16>)
+ setGroupID(group_id : uint8_t)
+ setGroupKey(key : std::array<uint8_t, 16>)
+ getTimestamp() : std::chrono::system_clock::time_point
+ getMAC() : std::array<uint8_t, 16>
+ getGroupID() : uint8_t
+ generate_mac(header : ExternalHeader, group_key : uint8_t : std::array<uint8_t, 16>)
```

```
// Retorna o MAC (Message Authentication Code) da mensagem
Ethernet::MAC_key getMAC() {
    // Gera MAC se for mensagem de comunicação interna.
    if (_header.src_address.vehicle_id == _header.dst_address.vehicle_id) {
        Ethernet::MAC_key key = {0};
        // Verifica se MAC nao foi gerado anteriormente.
        if (_header.mac == key) {
            _header.mac = generate_mac(_header, _group_key);
        }
    }
    return _header.mac;
}
```

# Formato das Mensagens recebidas

- Campo de Endereços é utilizado para diferenciar mensagens de interesse das respostas.
  - ID Componente ã preenchido => Mensagem Interesse (para todos os componentes produzem o tipo de dado)
  - ID Componente preenchido => Mensagem Resposta (para componente específico)
- Tipo sempre é utilizado para identificar qual tipo de dado produzido responder ou carregar.

## Recebimento interesse:

```
void* rotina_gps_estatico(void* arg) {
    while (true) {
        if (comunicador.hasMessage()) {
            Message mensagem;
            comunicador.receive(&mensagem);

            // Verifica se a mensagem é de interesse (não preencheu id componente no endereço de destino).
            if (pthread_equal(mensagem.getDstAddress().component_id, (pthread_t)0)) {
                // Responde a mensagem.
                mensagem.setDstAddress(mensagem.getSrcAddress());
                mensagem.setData(reinterpret_cast<Ethernet::Position*>(&posicao), sizeof(Ethernet::Position));
                comunicador.send(&mensagem);
                //std::cout << " " << dados->nome << ": Enviou posição." << std::endl;
                num_respostas_enviadas++;
            }
        }
    }
}
```

## Recebimento Resposta:

```
void* rotina_detector_veiculos(void* arg) {
    while (true) {
        // Processa mensagens recebidas.
        while (comunicador.hasMessage()) {
            Message mensagem;
            comunicador.receive(&mensagem);

            // Verifica se a mensagem recebida é resposta (id componente eh o do componente)
            if (pthread_equal(mensagem.getDstAddress().component_id, pthread_self())) {
                // Verifica tipo de resposta recebida.
                if (mensagem.getType() == Ethernet::TYPE_POSITION_DATA) {
                    // Extrai dado recebido.
                    Ethernet::Position posicao = *reinterpret_cast<Ethernet::Position*>(mensagem.data());
                    std::cout << " " << dados->nome << ": detectou veiculo na posicao: (" << posicao.x <<
                }
            }
        }
    }
}
```

**\*\*Exemplos do recebimento de mensagens dos componentes do teste de grupos. \*\***

# Formato das Mensagens recebidas

- Tipo e Período são usados pelo DataPublisher para criar as Threads Periódicas.
  - Componentes inscritos em determinados tipos.
  - Recebem interesse em determinados períodos.

Protocol encaminha mensagens de interesse para o DataPublisher:

```
// Encaminha mensagens de interesse direto para o DataPublisher.  
if (pthread_equal(payload.header.dst_address.component_id, (pthread_t)0)) {  
    _data_publisher->notify(message);  
} else {  
    // Notifica os observadores com o endereço de destino e a mensagem  
    _observed.notify(message);  
}
```

DataPublisher cria Threads Periódicas com a mensagem:

```
// Verifica quais observadores estão interessados na mensagem e os notifica.  
void DataPublisher::notify(Message message) {  
    Ethernet::Type msg_type = message.getType();  
    Ethernet::Period period = message.getPeriod();  
  
    std::lock_guard<std::mutex> lock(subscribers_mutex);  
    for (auto& sub : subscribers) {  
        for (auto& type : *(sub.second)) {  
            if (type == msg_type) {  
                // Envia diretamente se não for periódico  
                if (period <= 0) {  
                    sub.first->update(message);  
                } else {  
                    // Cria thread periódica para mensagens com período > 0  
                    create_periodic_thread(sub.first, message);  
                }  
                break; // Já encontrou o tipo desejado, pula para o próximo inscrito  
            }  
        }  
    }  
}
```