



Departamento de
Informática e Estatística
CTC • UFSC



Implementação da API de Comunicação: Comunicação Segura em Grupos

INE5424 - Sistemas Operacionais II

**Grupo A(Manhã): Vitor Calegari, Matheus Bigolin, Pedro Fountoura, Pedro
Taglialenha**

Objetivos da entrega 5

- **Segurança de Mensagens:**
 - Implementar Message Authentication Code (MAC) para mensagens.
 - Algoritmo: Poly1305 (OpenSSL).
 - Descartar mensagens com falha na verificação do MAC.
- **Gerenciamento de Chaves MAC por RSUs (Roadside Units):**
 - RSUs geram e distribuem chaves MAC para veículos em sua vizinhança.
- **Sincronização Temporal PTP (Evolução da E4):**
 - Interação por demanda: Veículo inicia sincronização com a RSU de seu quadrante.
 - RSUs atuam como referência temporal para os veículos.
- **Coordenadas:**
 - Introdução de coordenadas (_coord_x, _coord_y) nas mensagens.
 - Filtragem de mensagens por proximidade (alcance de comunicação simulado).

Introdução de Novas Classes e Entidades (1/2)

Componentes para Gerenciamento de Localização, Chaves e Papéis

- NavigatorCommon (e especializações NavigatorRandomWalk, NavigatorDirected):
 - Abstrai a lógica de movimentação e obtenção de coordenadas (`_x`, `_y`) para os veículos.
 - Utilizada por ProtocolCommom para preencher os campos `_coord_x`, `_coord_y` na Message.
 - Método `is_in_range(Coordinate)` usado em `Protocol::update` para filtrar mensagens de veículos fora do alcance de comunicação.
- Namespace MAC (`mac.hh`, `mac.cc`):
 - Encapsula a funcionalidade de criptografia MAC:
 - `MAC::Key` (`std::array<std::byte, 32>`)
 - `MAC::Tag` (`std::array<std::byte, 16>`)
 - `MAC::compute(key, message)`: Gera a tag Poly1305.
 - `MAC::verify(key, message, tag)`: Verifica a tag.
 - `MAC::generate_random_key()`: Gera uma chave aleatória.
- KeyKeeper (em `key_keeper.hh`, usado por Protocol - veículos):
 - Armazena um `std::map<int, MAC::Key>` (ID da RSU -> Chave MAC).
 - Método `setKeys(const std::vector<MacKeyEntry>&)`: Populado por mensagens `Control::Type::MAC` recebidas das RSUs.
 - Método `getKey(int rsu_id)`: Usado para obter a chave correta para calcular/verificar MACs.

Introdução de Novas Classes e Entidades (2/2)

- RSUs (Roadside Units):
 - Conceito introduzido, cada RSU responsável por um quadrante.
 - Implementadas como processos distintos com RSUProtocol.
 - RSUEngine<RSUProtocol> (em rsu_engine.hh):
 - Lógica específica das RSUs.
 - `_key_sender_thread`: Thread periódica para:
 - Gerar/Renovar sua própria `MAC::Key`.
 - A cada broadcast de mac, uma das RSUs incrementa um contador(em memória compartilhada entre processos) que determina quando todas devem gerar novas chaves MAC. (Sincronizações necessárias são implementadas utilizando `pthread_barrier_t` e `pthread_mutex_t`)
 - Obter chaves das RSUs adjacentes (vizinhança 3x3) da `SharedData->entries`.
 - Enviar estas chaves em broadcast (`Control::Type::MAC`).

Modificações na Estrutura Message e Protocol::Packet (1/2)

- **Novos Campos em Message<Addr> (e consequentemente em Protocol::Packet::Header):**
 - MAC::Tag_tag;
 - Armazena a tag MAC (16 bytes para Poly1305) calculada sobre a mensagem.
 - Acessado via Message<Addr>::tag().
 - double _coord_x, double _coord_y:
 - Armazenam as coordenadas geográficas do emissor da mensagem.
 - Acessadas via Message<Addr>::getCoordX() e Message<Addr>::getCoordY().
 - Preenchidas em ProtocolCommom::fillBuffer() utilizando a instância de NavigatorCommon.
- **Preenchimento e Verificação do MAC:**
 - Envio (em Protocol::fillBuffer para veículos):
 - Após preencher todos os outros campos da Protocol::Packet (incluindo _timestamp, _ctrl, coordenadas).
 - Obtém a MAC::Key do quadrante que o veículo que vai enviar a mensagem se localiza.
 - Calcula a MAC::Tag usando MAC::compute(key, message_bytes).
 - A tag calculada é inserida em pkt->header()->tag.

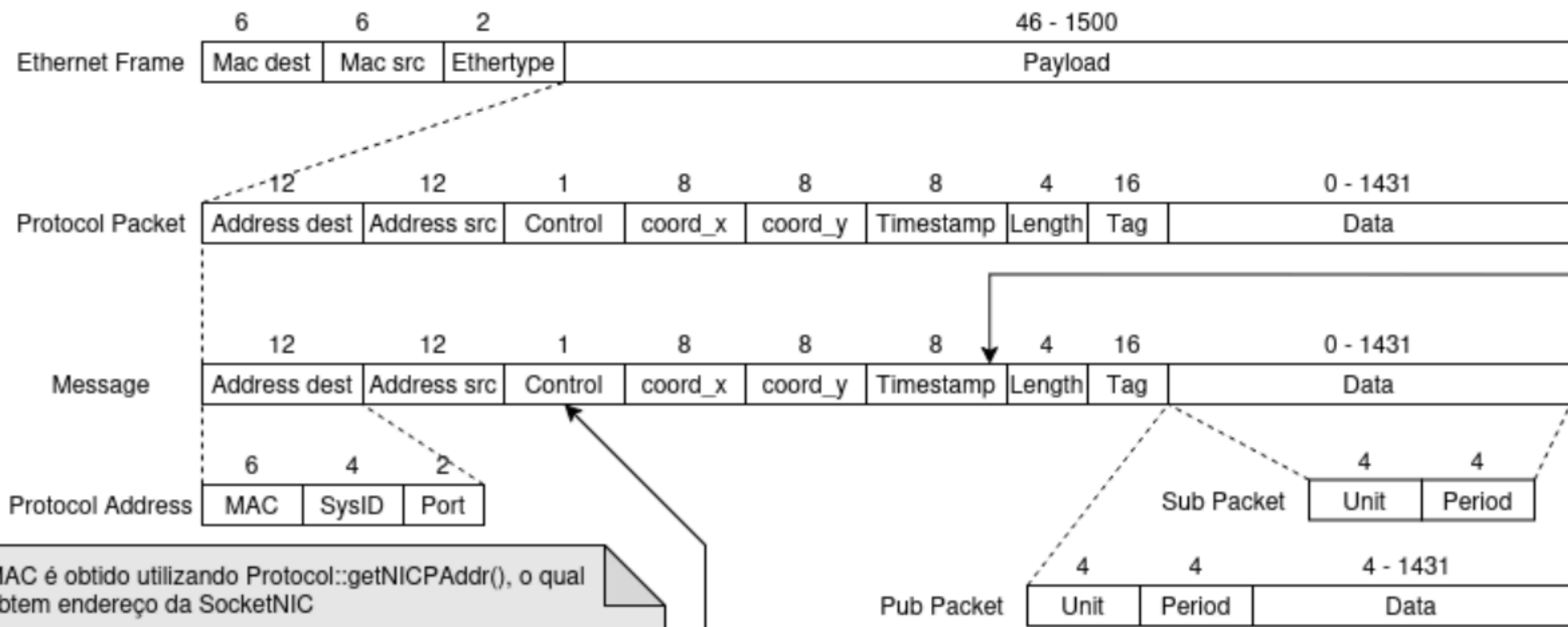
Modificações na Estrutura Message e Protocol::Packet (2/2)

- **Preenchimento e Verificação do MAC:**

- Recebimento (em Protocol::update para veículos):
 - Extrai a tag recebida de pkt->header()->tag.
 - Recalcula a tag localmente usando a chave apropriada e os bytes da mensagem recebida (excluindo a própria tag).
 - Verifica usando MAC::verify(key, message_bytes, received_tag).
 - Se a verificação falhar, a mensagem é descartada.

- **Campo Control em Message:**

- Tipo PTP foi substituído por tipo DELAY_RESP e LATE_SYNC.
- Adicionado bit **needSync**
- Adicionado Control::Type::MAC para mensagens específicas de distribuição de chaves MAC pelas RSUs.



Durante o envio, Protocol é encarregado de:

- 1) Alocar um Buffer por uma das NICs
- 2) Preencher o cabeçalho Ethernet
- 3) Copiar a mensagem para o Pacote do protocolo, o qual é o conteúdo do payload do quadro Ethernet
- 4) Preencher timestamp, bit de sincronização e **bit needSync**

Timestamp é interpretado como tempo de recebimento na mensagem Delay Response. Demais mensagens ele é interpretado como tempo em que a mensagem foi enviada.

Sub Packet e Pub Packet são da camada SmartData, mensagens do tipo COMMOM não estão limitadas a estrutura de Pub Packet ou Sub Packet.

Durante o recebimento o conteúdo do payload é copiado do Buffer pela NIC para um Pacote de Protocol.

Protocol preenche a mensagem com os dados do Pacote recebido.

MAC é obtido utilizando Protocol::getNICPAddr(), o qual obtem endereço da SocketNIC

SysId é obtido pelo método Protocol::getSysID(), o qual retorna o SysID fornecido pelo usuário durante a instanciação de Protocol

Porta do comunicador destino deve ser fornecida pelo usuário

Control:

- bit 0 é o de sincronização
- bits 1 a 3 são Msg Type
- **bit 4 é needSync**

Msg Type pode assumir 6 valores:

- COMMOM
- PUBLISH
- SUBSCRIBE
- **DELAY_RESP**
- **LATE_SYNC**
- **MAC**

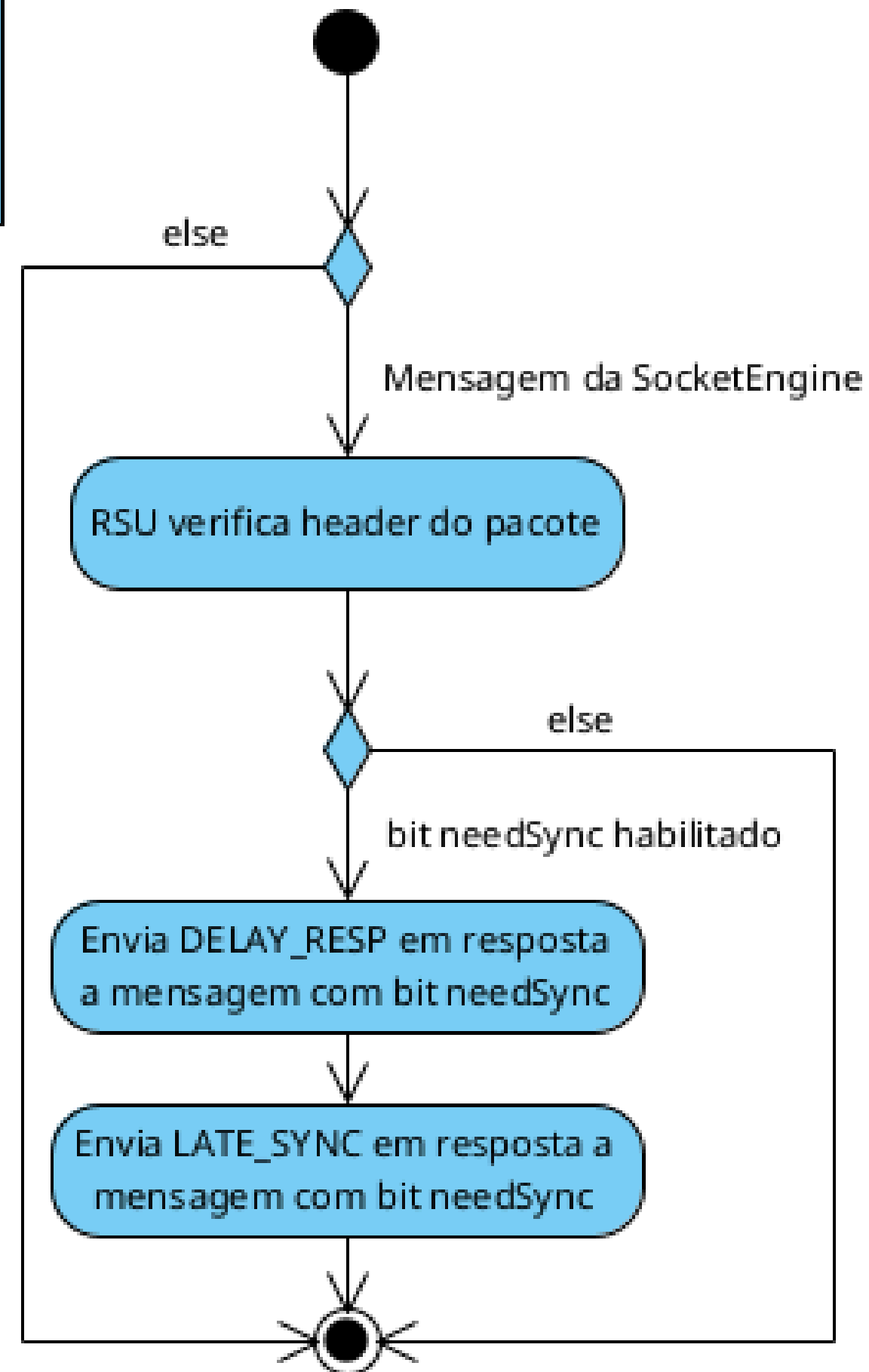
O tamanho de todos os campos são representados em Bytes

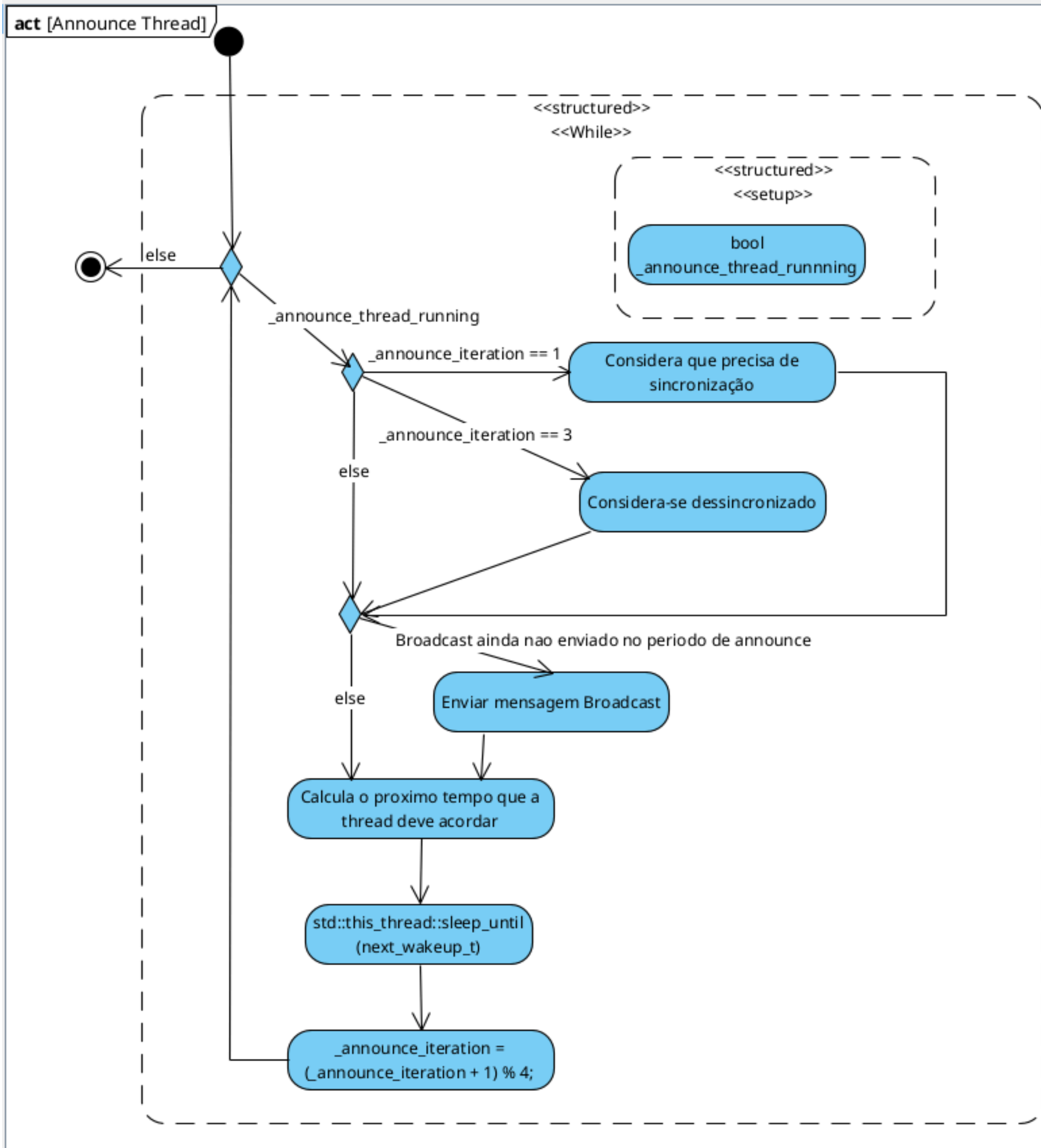
Sincronização Temporal (PTP) por Demanda com RSUs

Evolução do Modelo PTP da Entrega 4

- **RSUs atuam como mestres PTP para veículos.**
- **Veículos (Slaves PTP):**
 - `_announce_thread` (SyncEngine) envia ANNOUNCE periodicamente.
 - Se ANNOUNCE do veículo indica `!needSync()` (verificado pela RSU):
 - RSU (`RSUProtocol::update`) responde com:
 - `Control::Type::DELAY_RESP` (Header: T4 da RSU).
 - `Control::Type::LATE_SYNC` (Header: T1 da RSU).
 - Mensagens contém em seu payload a qual 'needSync' ela está relacionada
 - Veículo (`SyncEngine::handlePTP`) processa `DELAY_RESP` e `LATE_SYNC`:
 - `DELAY_RESP`: Armazena T4 mapeado para T3 em `unordered_map`;
 - `LATE_SYNC`: Usa T1 (da msg), T2 (local), T3 (payload) e T4 (do map) para calcular offset.
 - Ao fim do processo, atualiza offset e considera-se sincronizado;

O código relacionado a esse fluxo está presente em RSUProtocol::update(...) e acontece durante o recebimento de uma mensagem.





Hierarquia de Protocol

Especialização para Veículos e RSUs

- **ProtocolCommom<SocketNIC, SharedMemNIC> (Base):**
 - Funcionalidades Comuns: Estruturas Address, Header, Packet (com _coord_x, _coord_y, _tag). Interação com SyncEngine (com flag isRSU) e NavigatorCommon. Métodos base send, receive.
 - Método update(...) é virtual, forçando especialização.
- **Protocol<SocketNIC, SharedMemNIC> (Veículos):**
 - Herda de ProtocolCommom.
 - update(...): Filtra por _nav->is_in_range(); verifica MAC com _key_keeper; processa Control::Type::MAC para _key_keeper.setKeys(); delega PTP à _sync_engine.
 - fillBuffer(): Adiciona cálculo de MAC (via MAC::compute).
 - Atributo KeyKeeper para armazenamento de chaves MAC.
- **RSUProtocol<SocketNIC, SharedMemNIC> (RSUs):**
 - Herda de ProtocolCommom.
 - Atributo RSUEngine para lógica de chaves MAC.
 - update(): Atua como mestre PTP por demanda. Se recebe mensagem de veículo não sincronizado, envia DELAY_RESP e LATE_SYNC. Ignora outras mensagens PTP e MAC.

Distribuição de Chaves MAC

- **RSUs (RSUEngine): Geração e Distribuição de Chaves**

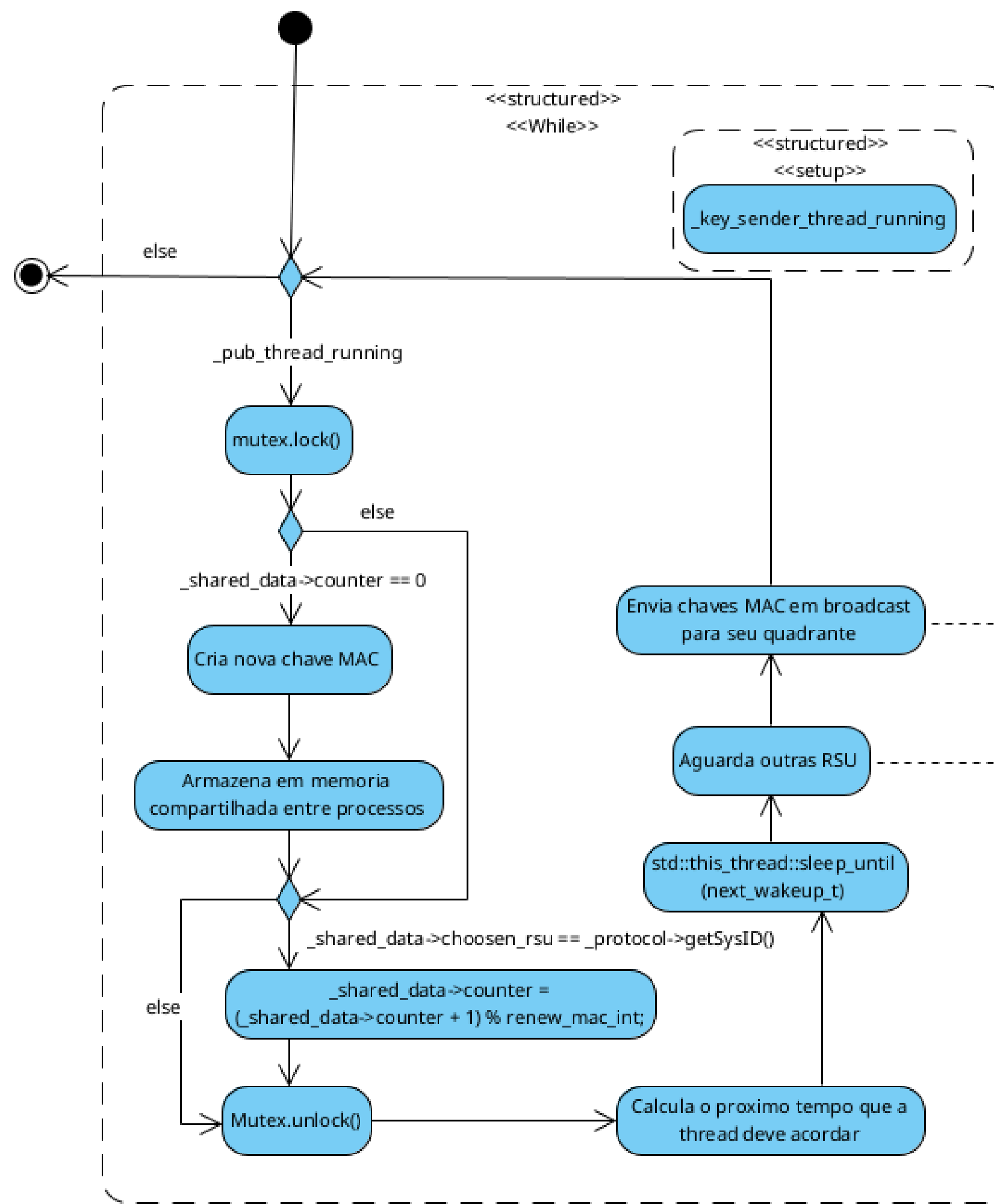
- `_key_sender_thread` (periódica):
 - Coordena com outras RSUs (via `SharedData`, `pthread_barrier_t`, `pthread_mutex_t`) para que uma RSU designada (`_shared_data->chosen_rsu`) atualize o contador `_shared_data->counter`.
 - Se `_shared_data->counter == 0`, a RSU atualiza sua `MacKeyEntry` em `_shared_data->entries` com `MAC::generate_random_key()`.
 - Lê chaves da vizinhança (3x3) de `_shared_data->entries`.
 - Envia `neighborhood_keys` em `Message` com `Control::Type::MAC` (broadcast).

- **Veículos (Protocol): Recebimento e Armazenamento**

- `Protocol::update()`: Ao receber `Control::Type::MAC`, usa `_key_keeper.setKeys(...)` com as chaves recebidas.

Uso de Chaves MAC

- **Veículos (Protocol::fillBuffer): Cálculo de MAC no Envio**
 - Obtém a MAC::Key do quadrante que o veículo que vai enviar a mensagem se localiza.
 - Calcula tag = MAC::compute(key, serialized_packet_bytes).
 - Preenche tag no cabeçalho da mensagem.
- **Veículos (Protocol::update): Verificação de MAC no Recebimento**
 - A partir da localização do veículo que enviou a mensagem, escolhe uma de suas chaves MAC para verificar a mensagem.
 - Verifica com MAC::verify(key, serialized_packet_bytes, pkt->header()->tag).
 - Descarta mensagem se MAC não for válido.

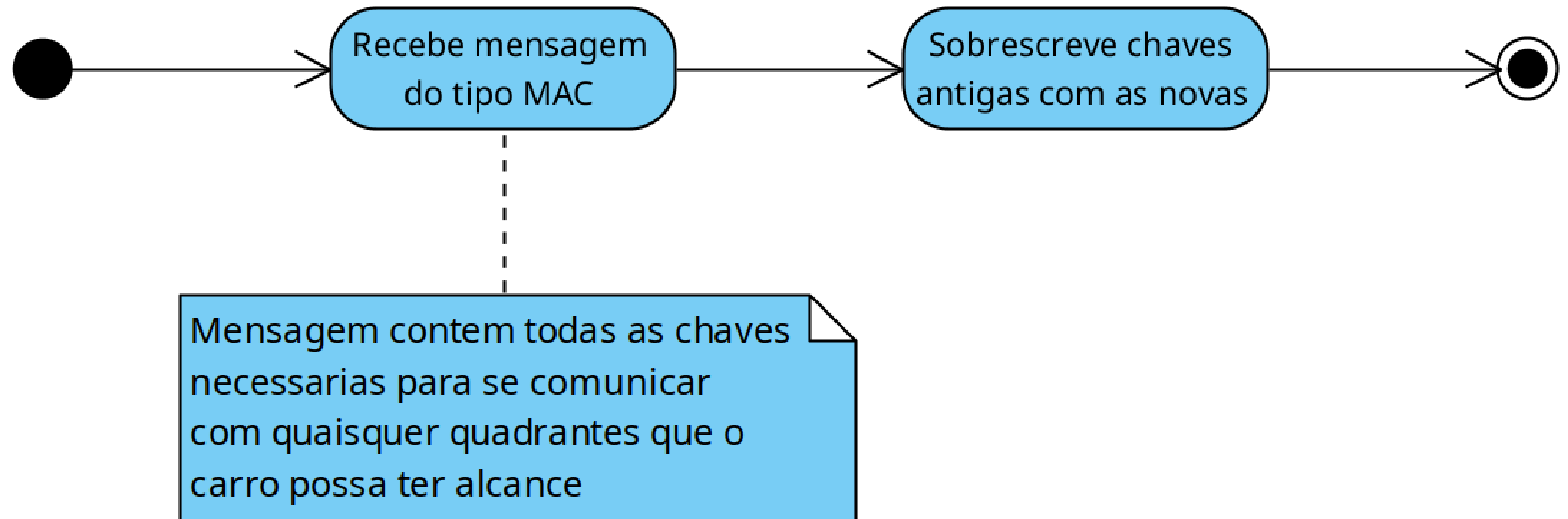


Dado uma RSU representada por ■, e RSUs adjacentes representadas por □, uma RSU (■) envia sua chave e as chaves das RSU adjacentes(□):

□ □ □
□ ■ □
□ □ □

`pthread_barrier_wait(&_shared_data->barrier)`

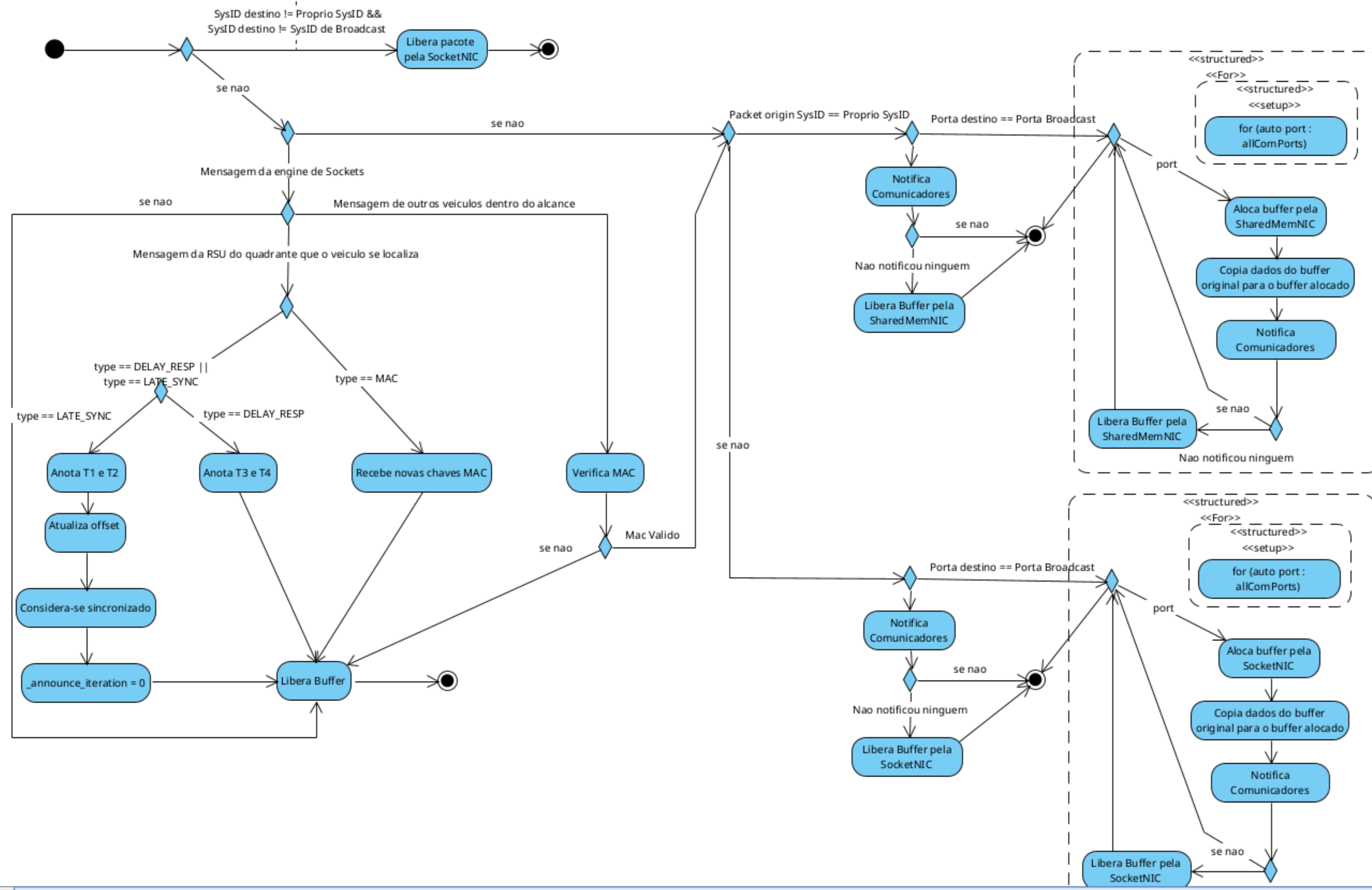
act [Recebimento de chaves MAC]

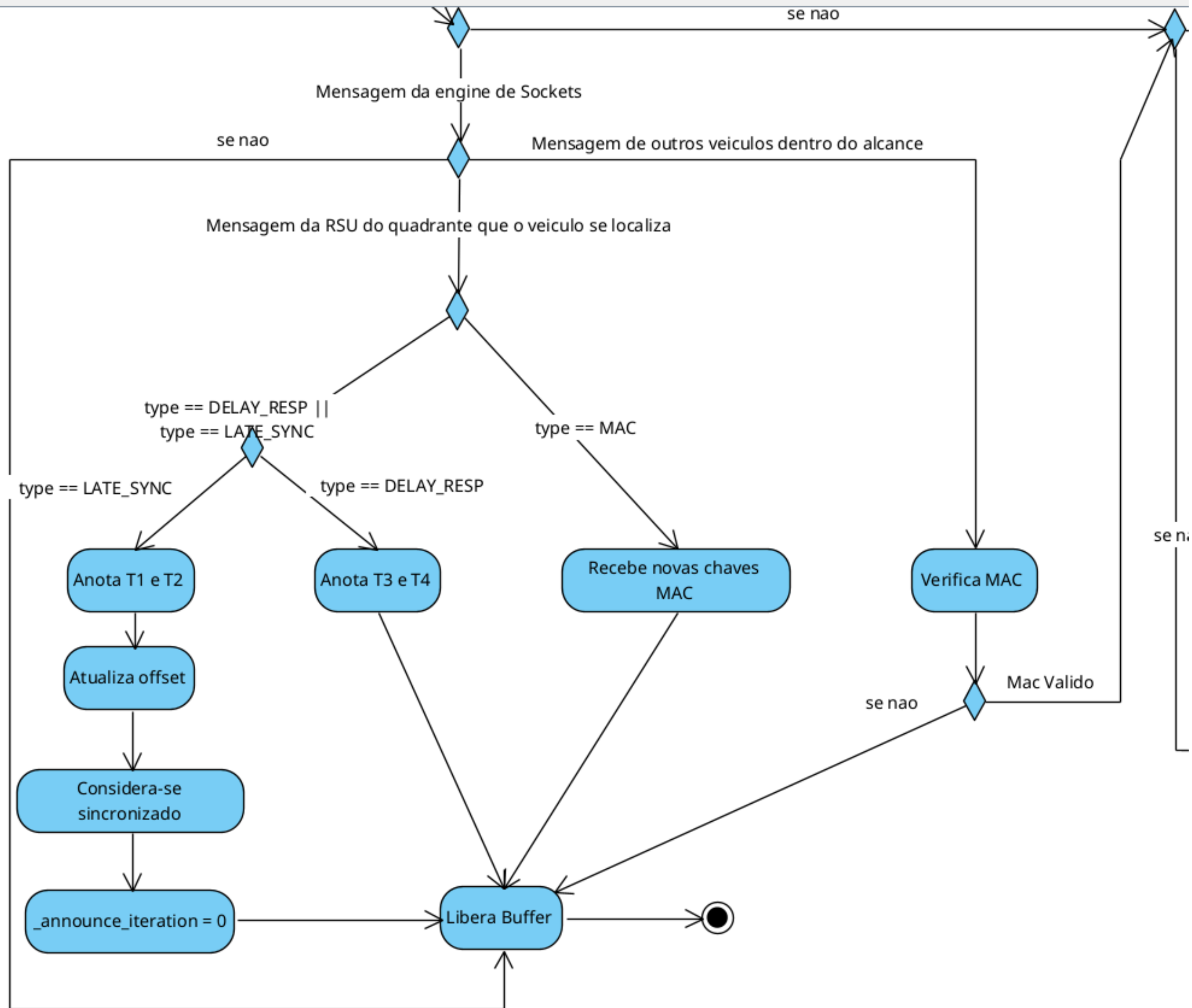


act [protocol.update(...)]

Esse metodo é chamado quando uma NIC notifica o protocolo de uma nova mensagem

O protocolo armazena um SysID em seu interior que representa o Identificador unico do veiculo



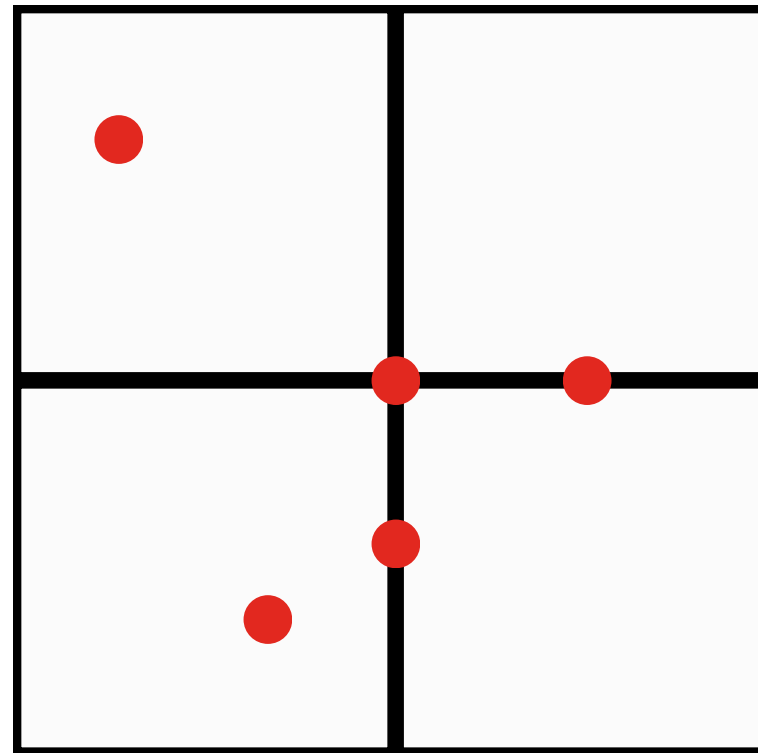


Testes Desenvolvidos e Validação

- Teste de quadrante

● Carros

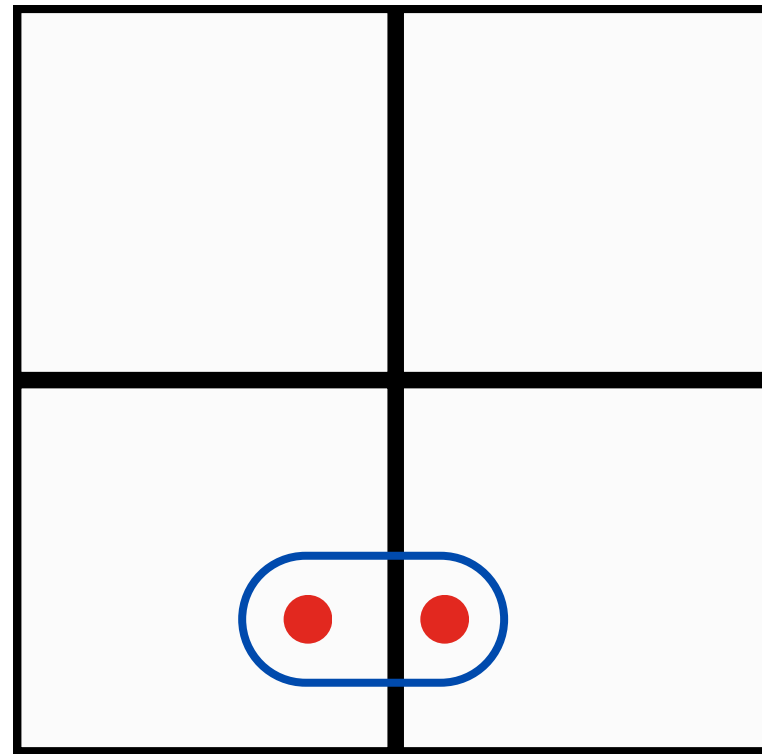
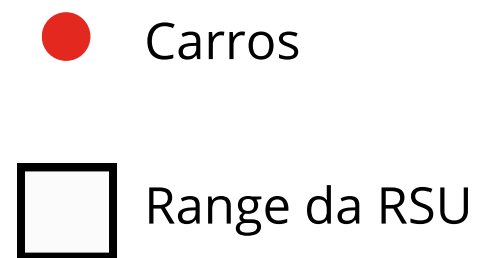
□ Range da RSU



Verifica em que quadrante está o ponto e verifica o que acontece se estiver em intersecções

Testes Desenvolvidos e Validação

- Teste de comunicação entre carros em diferentes quadrantes



O que está sendo testado:

- Comunicação inter-quadrante
- Funcionamento de múltiplas RSUs
- Correto recebimento das mensagens na topologia

Testes Desenvolvidos e Validação

- Teste de validação do MAC

Teste que passa por todos os métodos de MAC para verificação de seu funcionamento

- Testa todos os métodos públicos da classe MAC
- Cobre cenários positivos e negativos
- Valida integridade e autenticidade
- Verifica detecção de ataques/corrupção

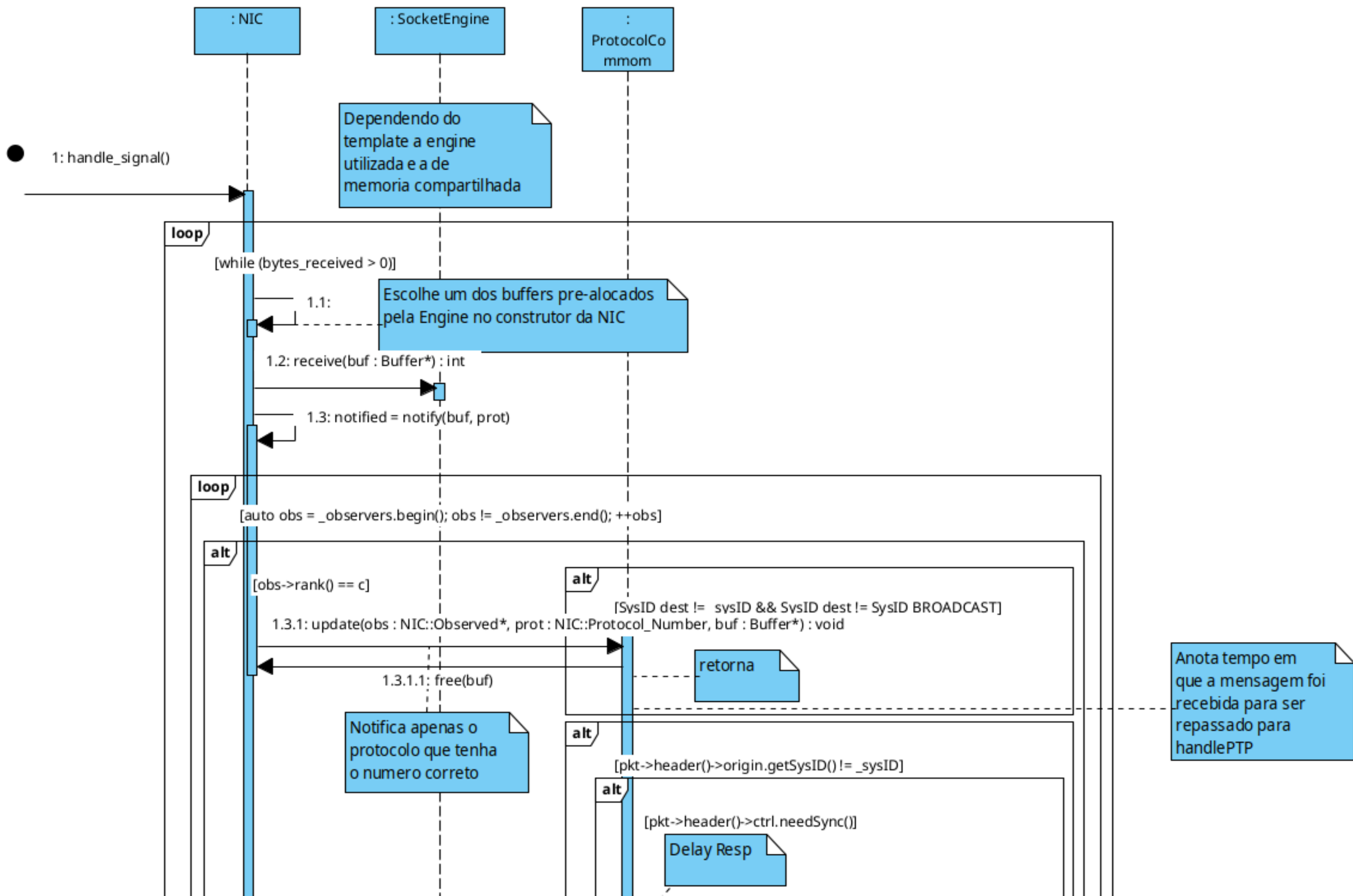
- Teste de Descarte de Mensagens

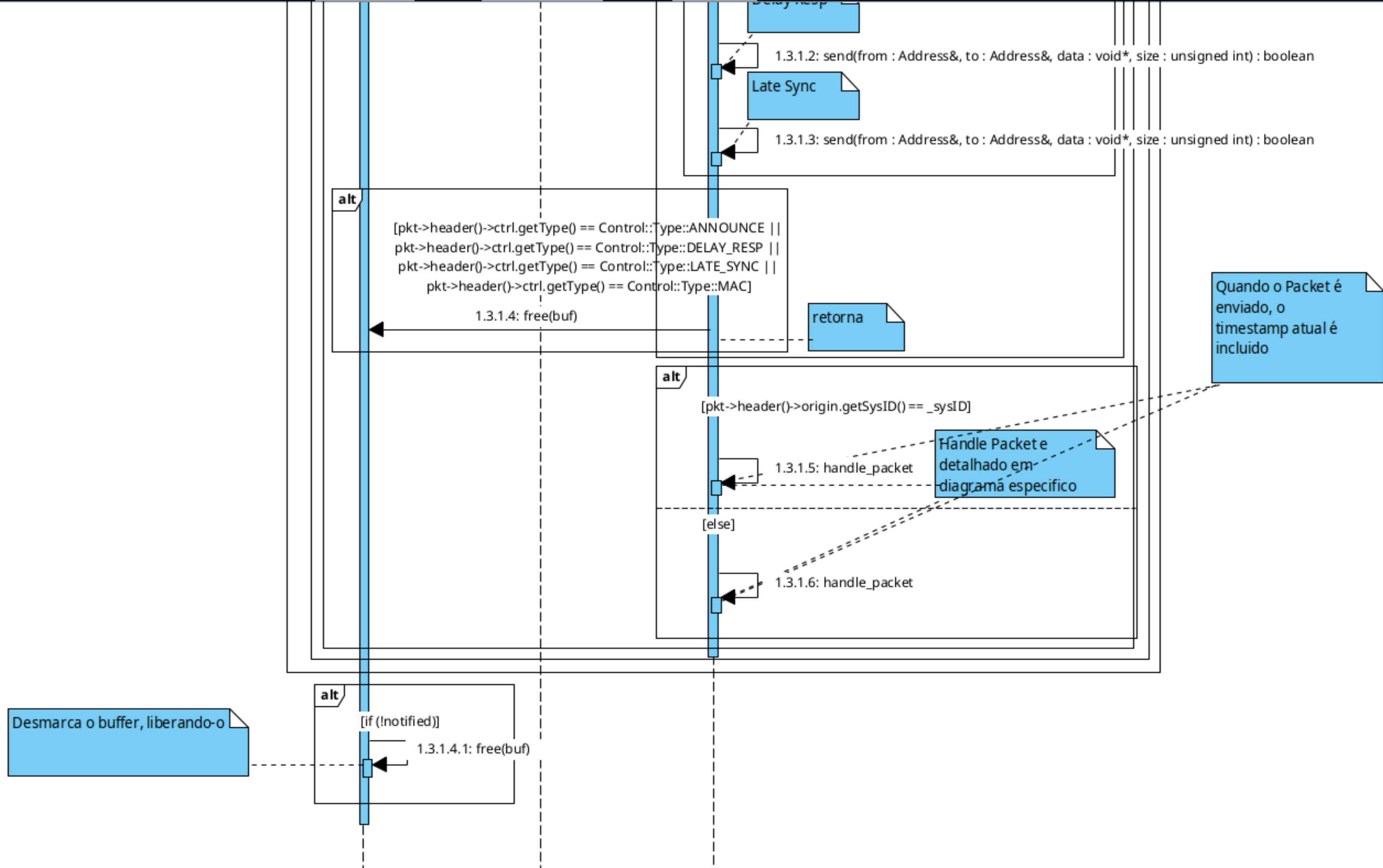
Teste que valida o descarte automático de mensagens com MAC inválido em comunicação V2V unicast, garantindo que apenas mensagens autenticadas sejam processadas pelo receptor.

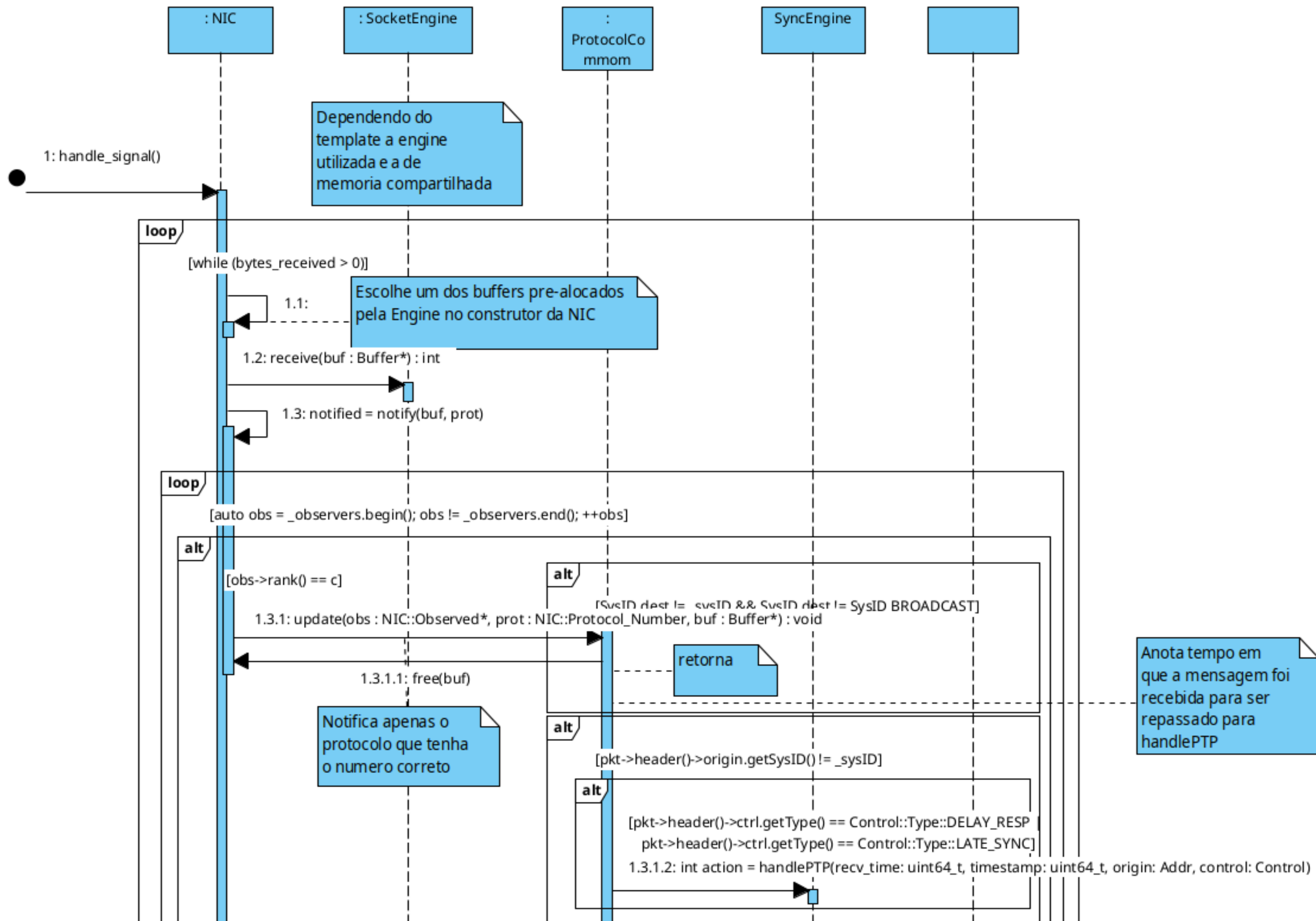
Objetivo: testar se o sistema automaticamente filtra/descarta mensagens que:

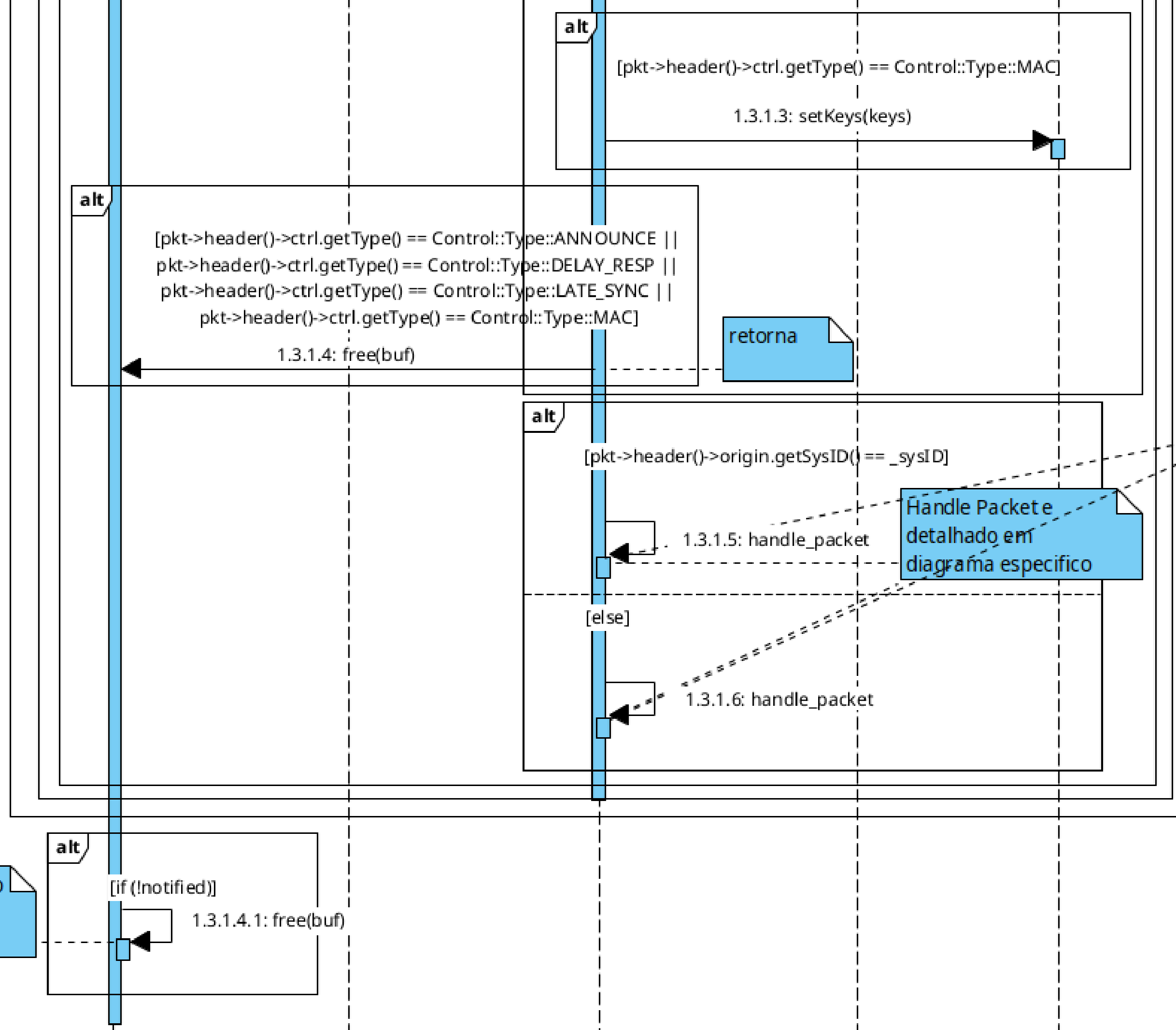
- Não passaram na validação MAC
- Foram corrompidas durante transmissão
- Vieram de fontes não autenticadas

**Diagramas Novos ou
Atualizados:**









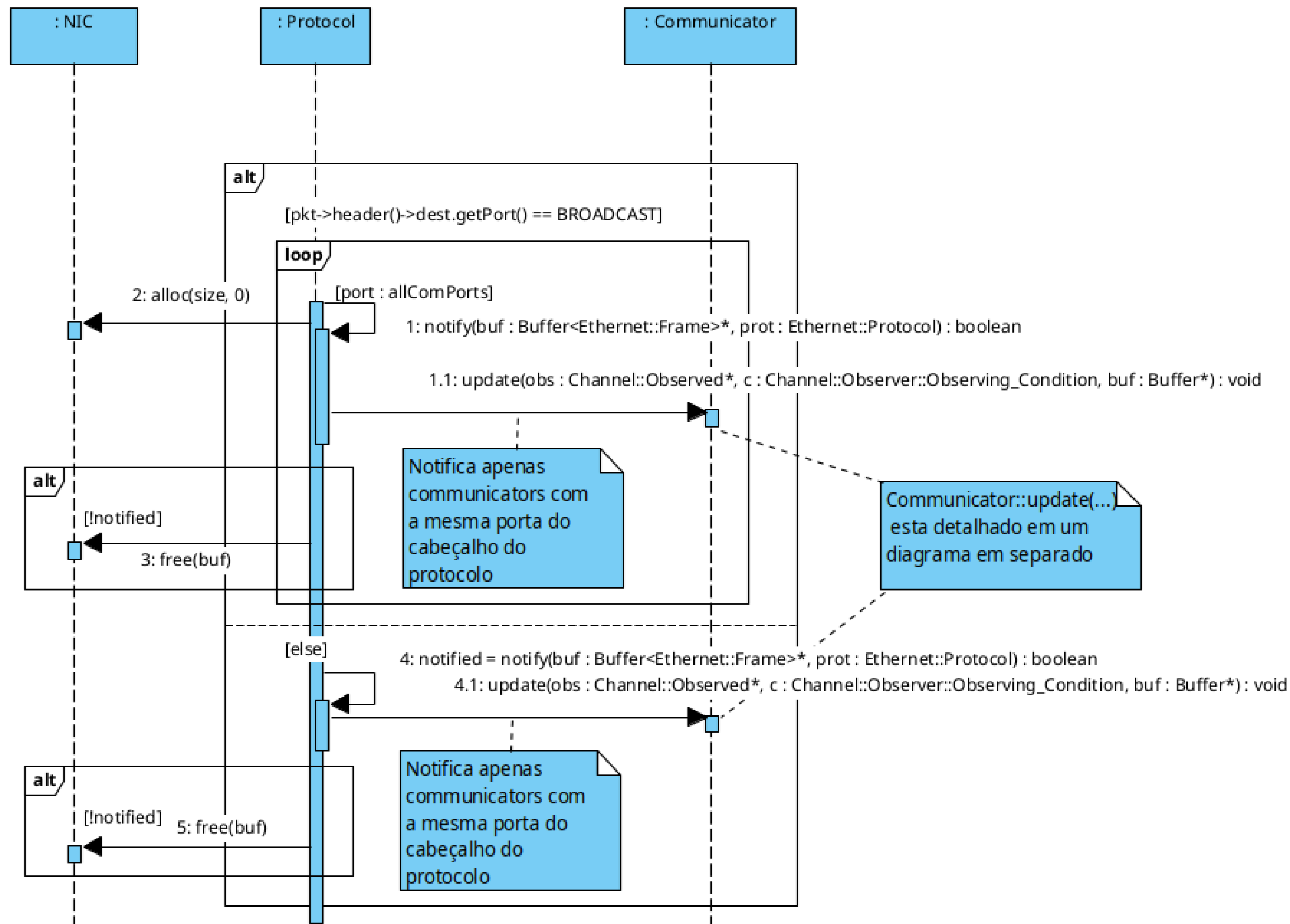
Quando o Packet é enviado, o timestamp atual é incluído

Handle Packet e detalhado em diagrama especifico

Desmarca o buffer, liberando-o

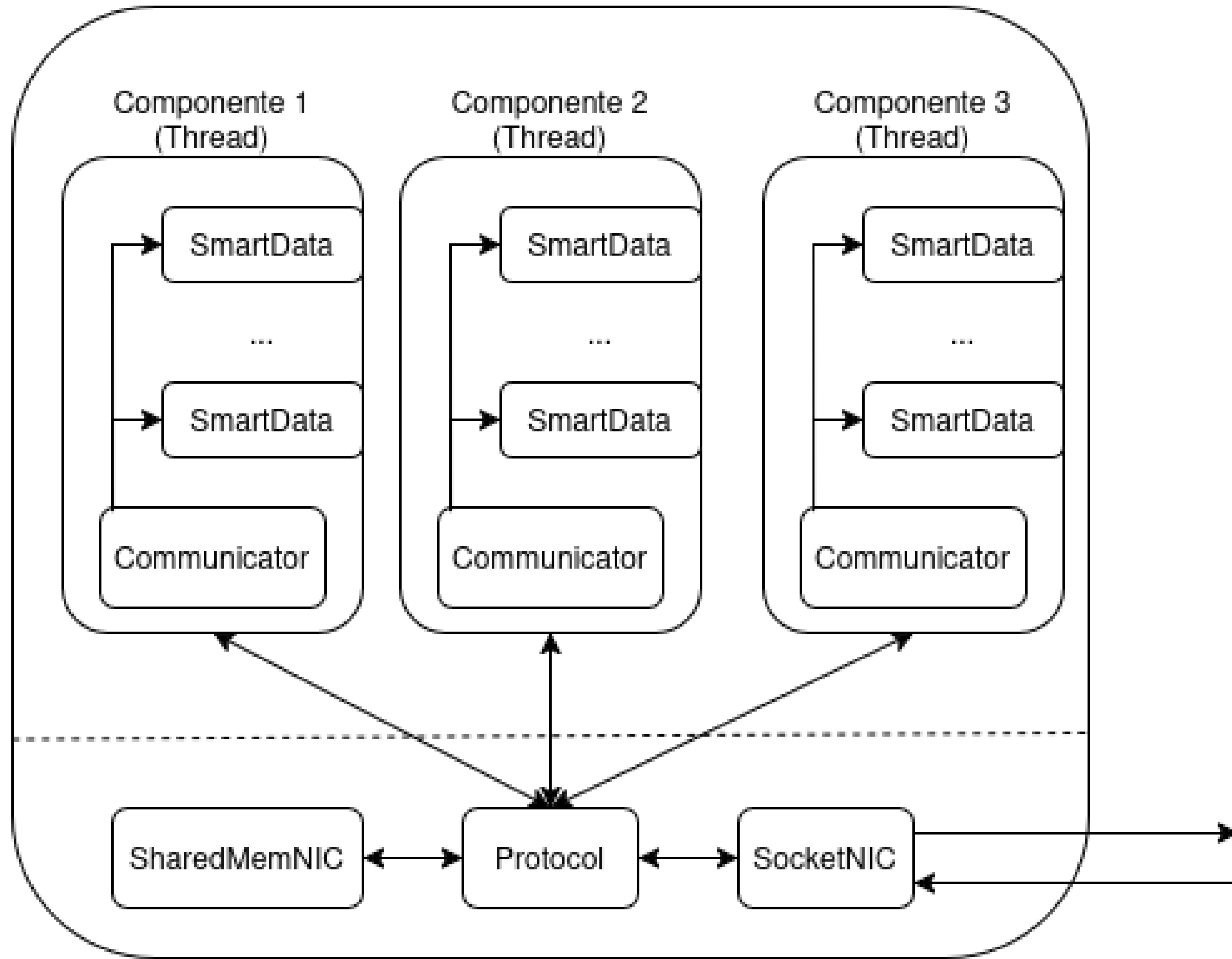
Outros Diagramas:

sd [handlePacket]

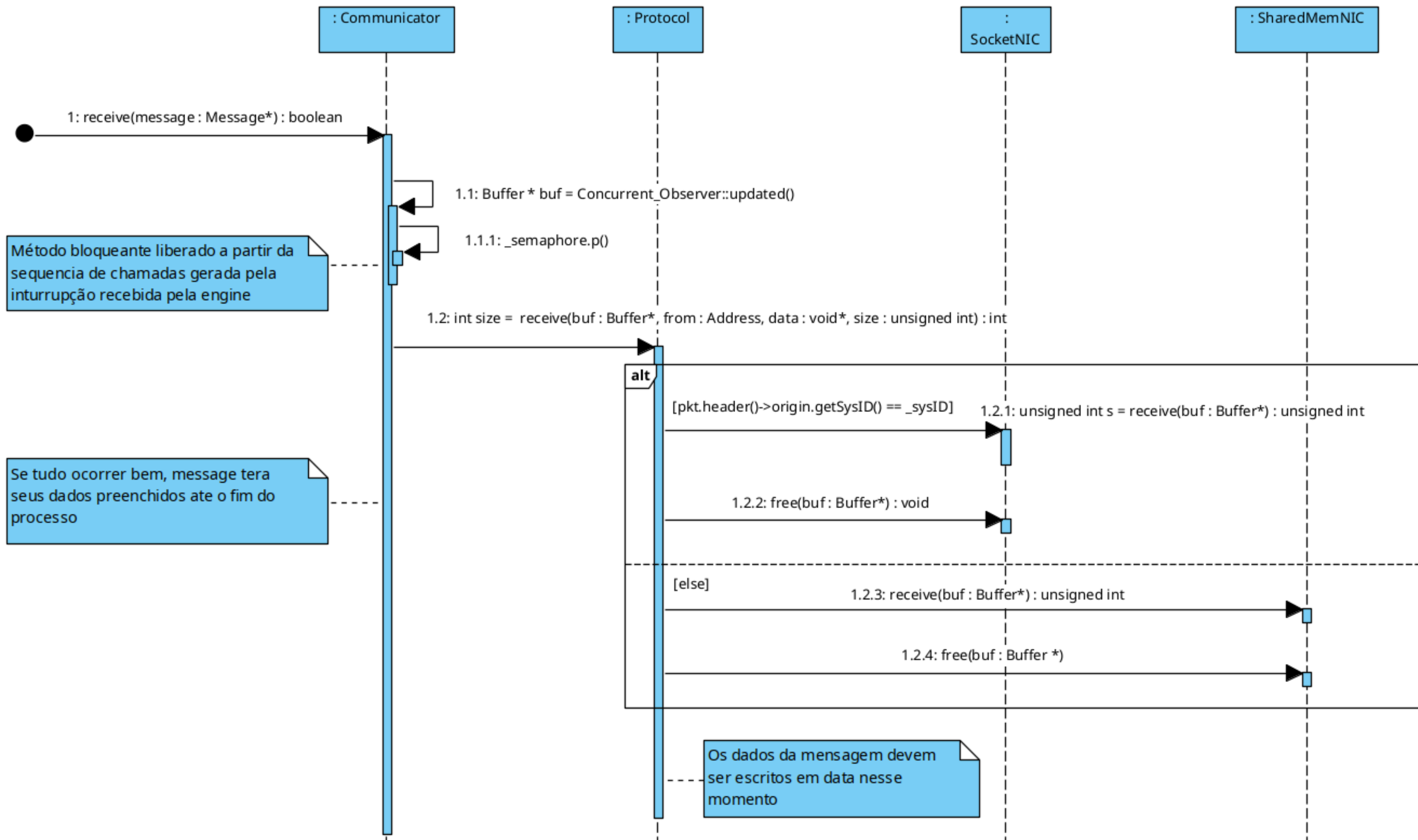


Estrutura do Veículo

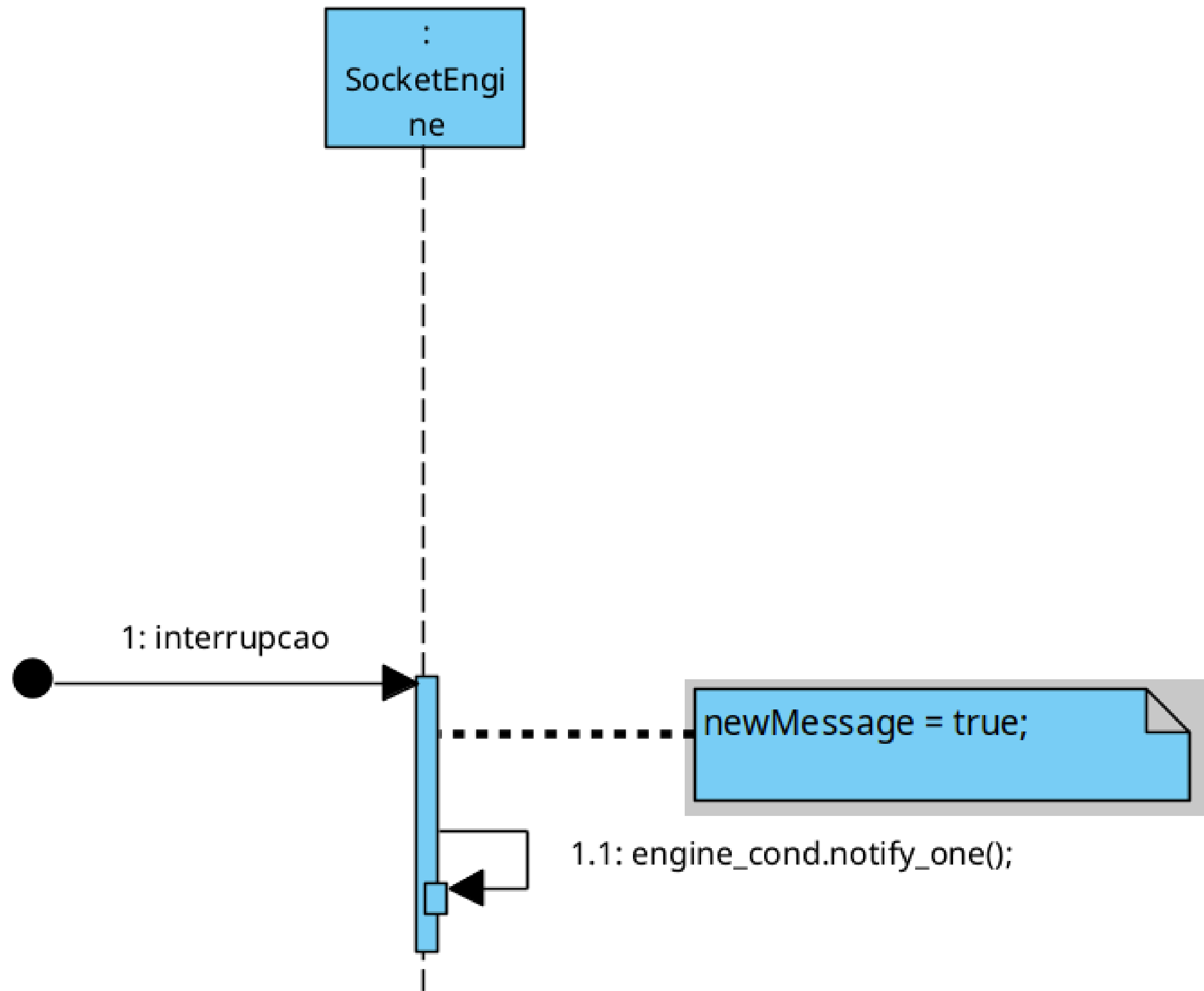
Veículo (Processo)

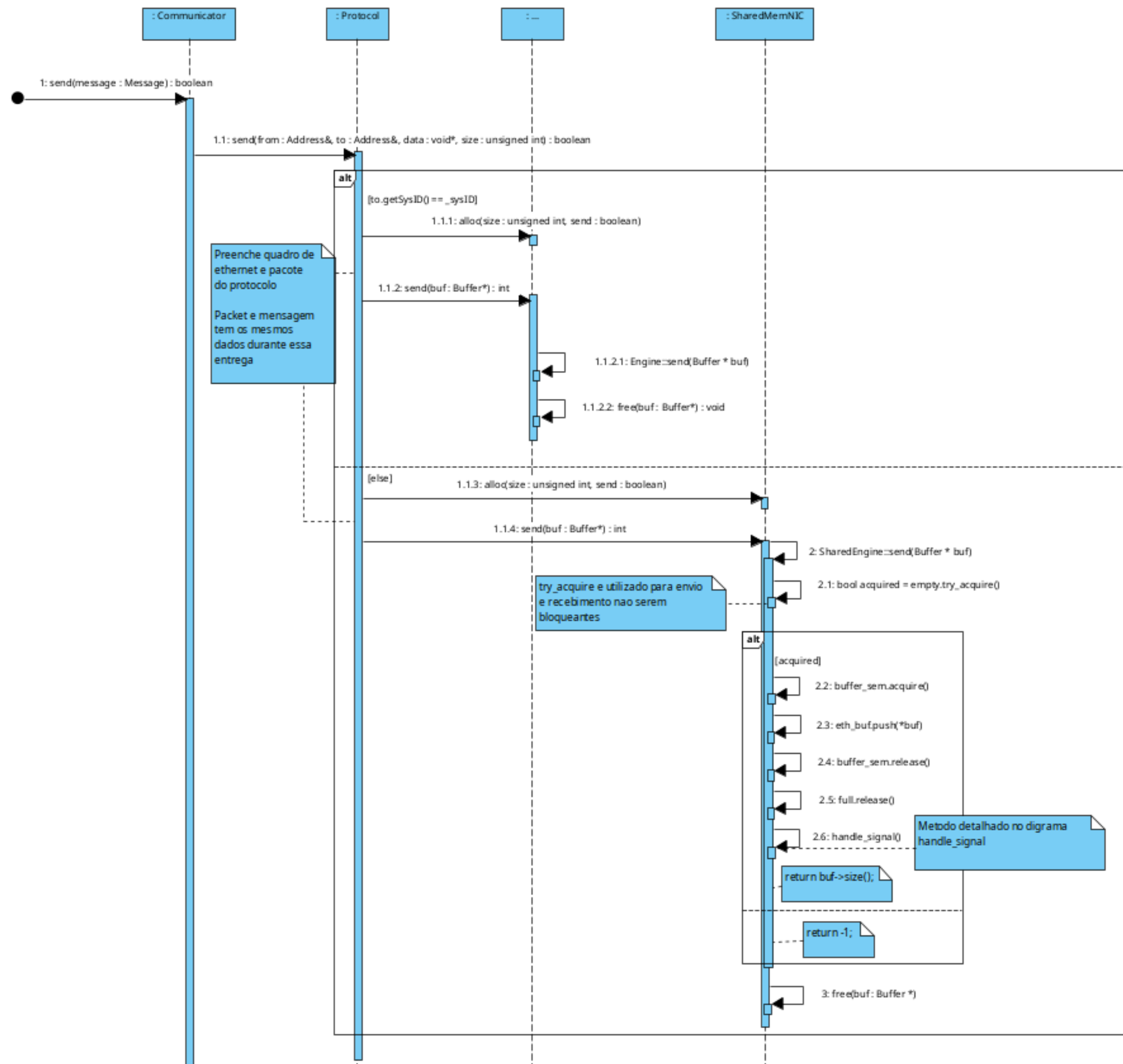


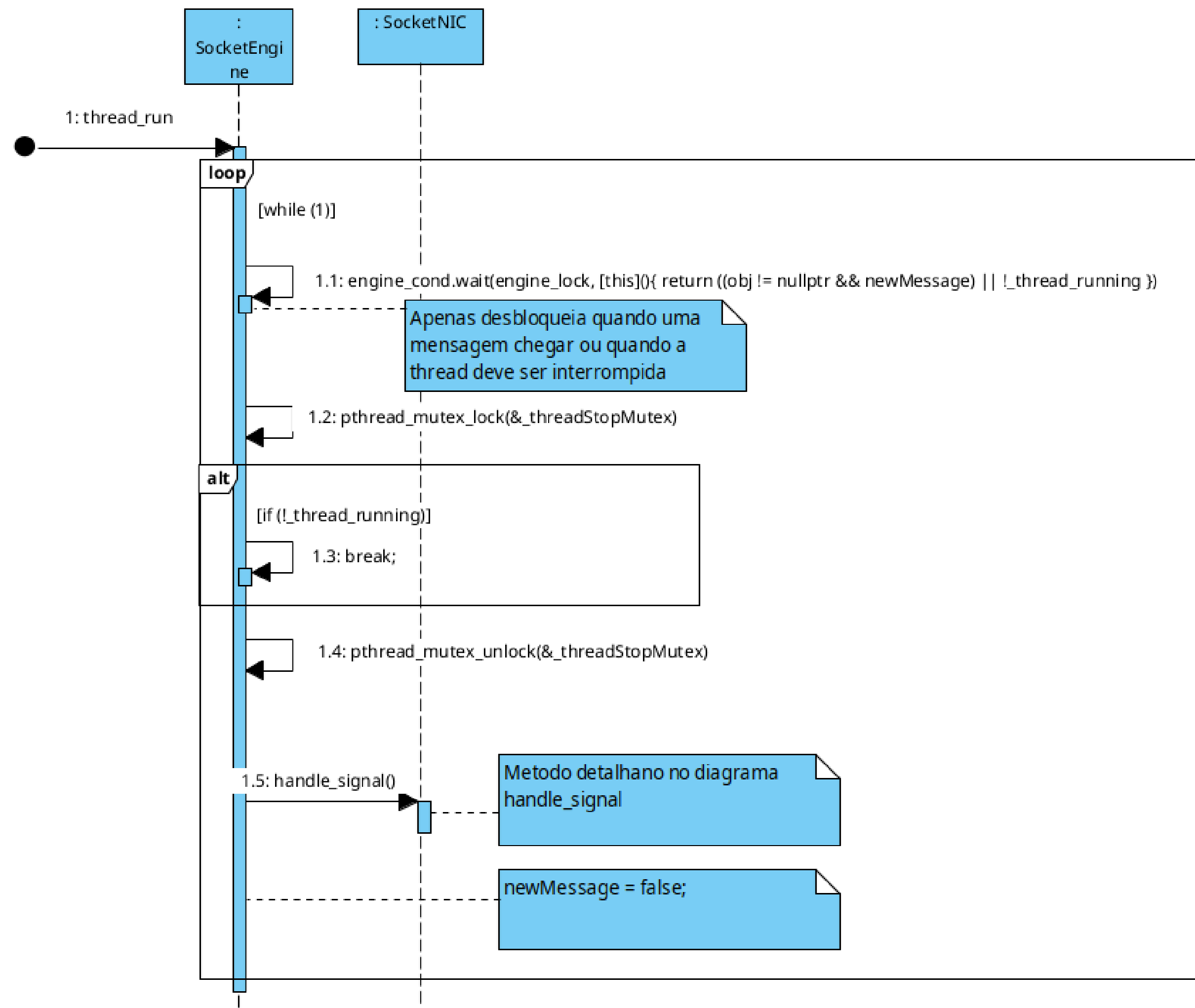
Diagramas de Sequência

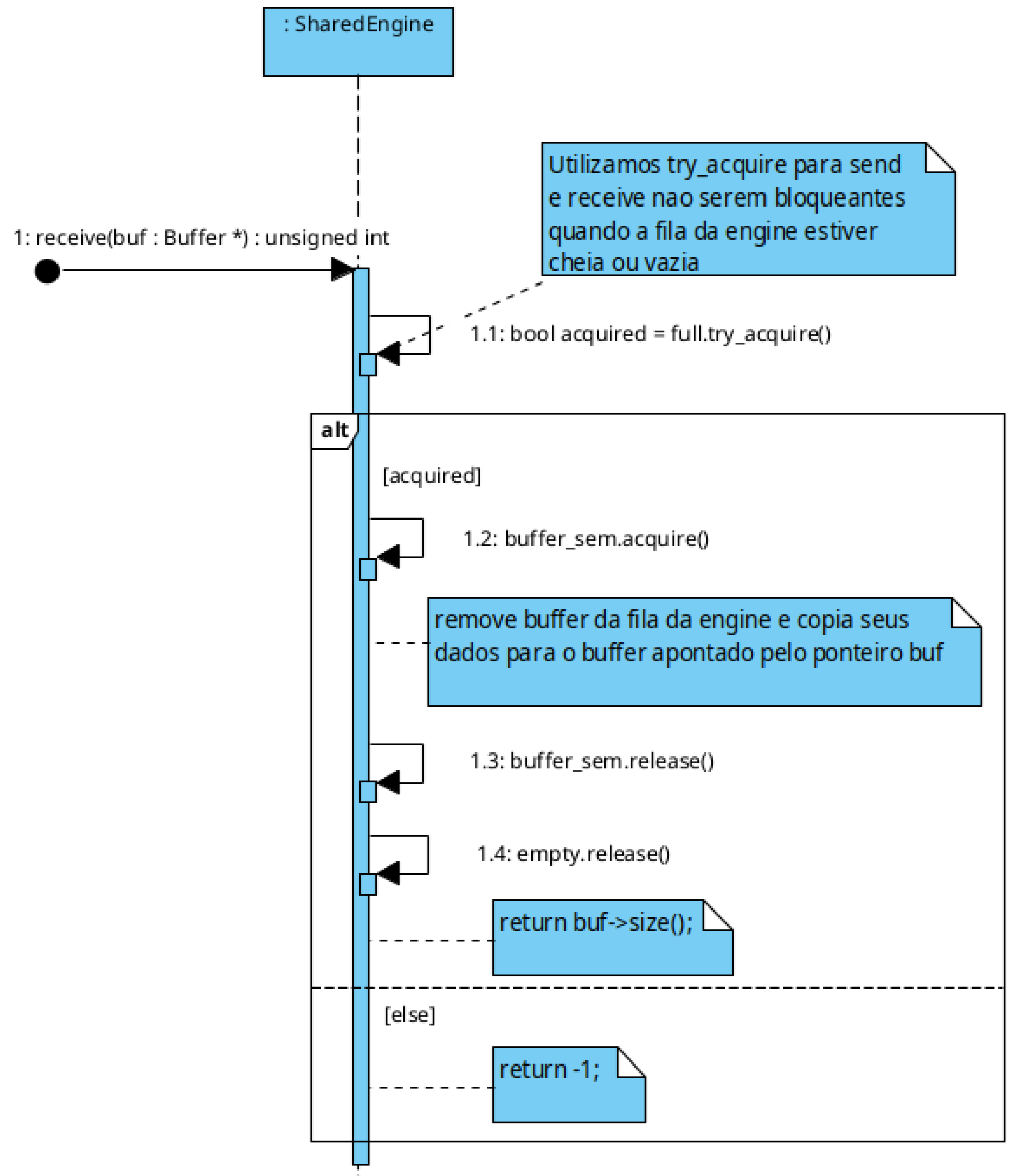


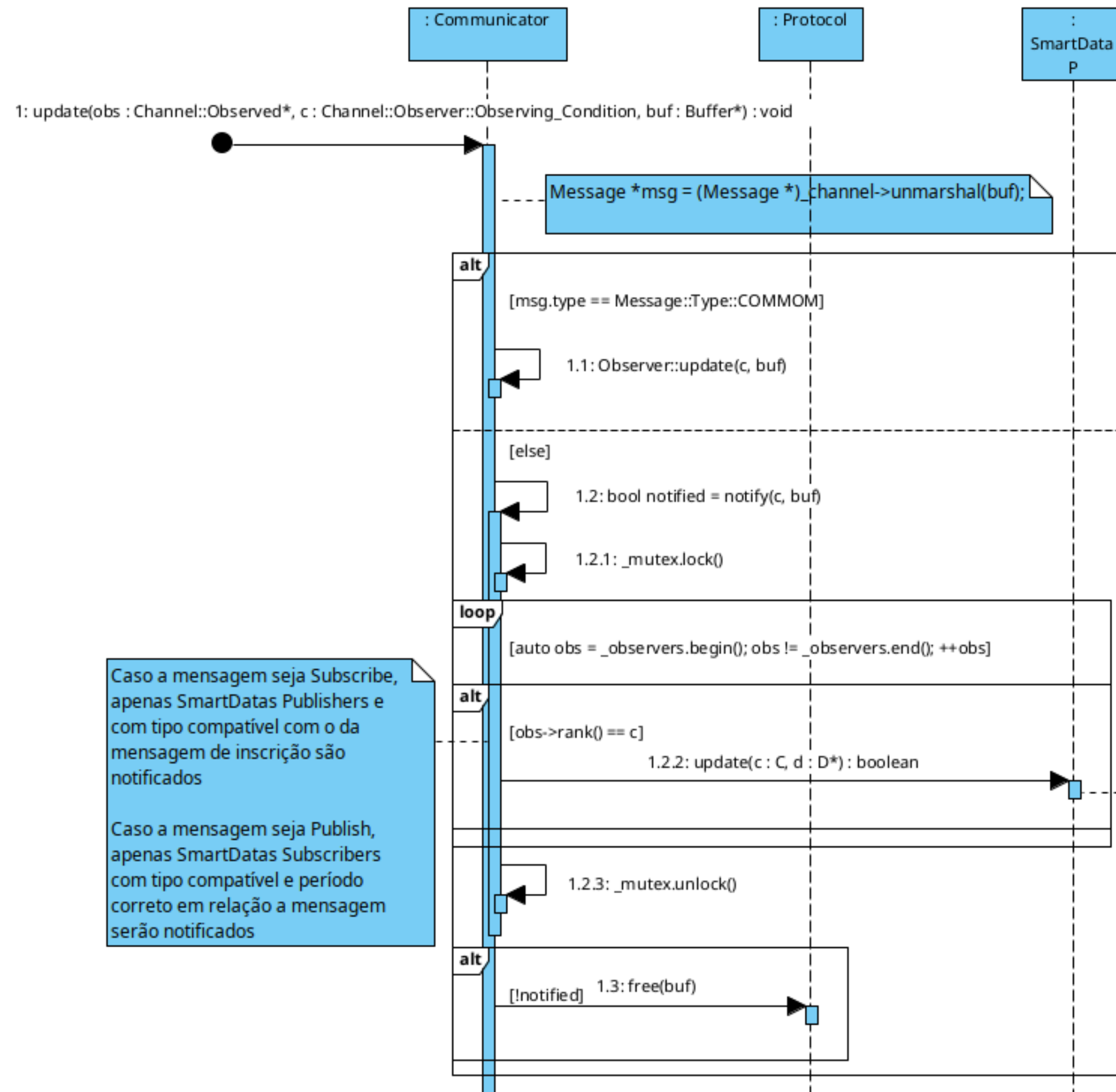
sd [SIGIO handler]











Caso a mensagem seja Subscribe, apenas SmartDatas Publishers e com tipo compatível com o da mensagem de inscrição são notificados

Caso a mensagem seja Publish, apenas SmartDatas Subscribers com tipo compatível e período correto em relação a mensagem serão notificados

Caso seja um SmartData Publisher, trata a mensagem de subscribe.

Caso seja um SmartData Subscriber, trata a mensagem de publish.

Diagramas de Atividade

act [User View Send]

Cada Communicator é
identificado unicamente
pelo seu Address

Mensagem tem o seguinte formato:

```
Address ::= SEQUENCE {  
    mac OCTET STRING (SIZE(6)),  
    sysID OCTET STRING (SIZE(4)),  
    port OCTET STRING (SIZE(2))  
}  
  
Message ::= SEQUENCE {  
    destAddress Address,  
    srcAddress Address,  
    type OCTET STRING(SIZE(1)),  
    data_length OCTET STRING (SIZE(4)),  
    data OCTET STRING (SIZE(0..1471))  
}
```

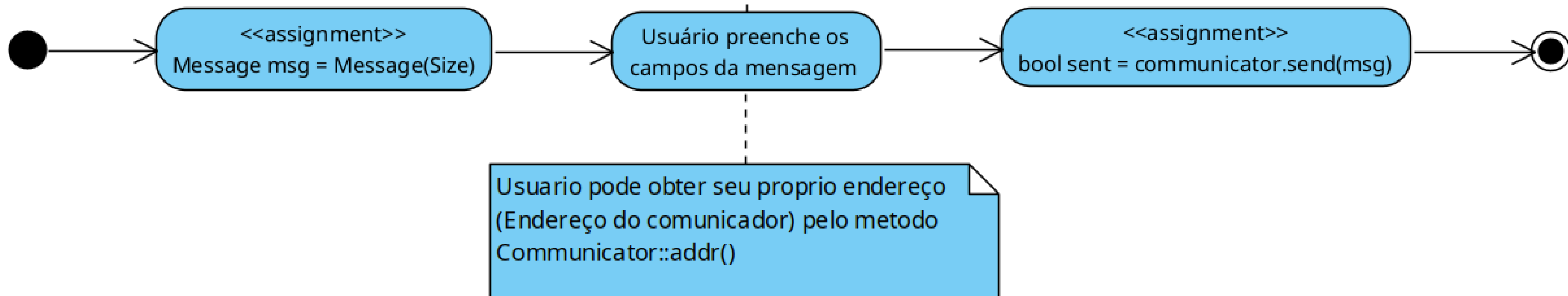
Endereços especiais:

0 = Broadcast_SysID

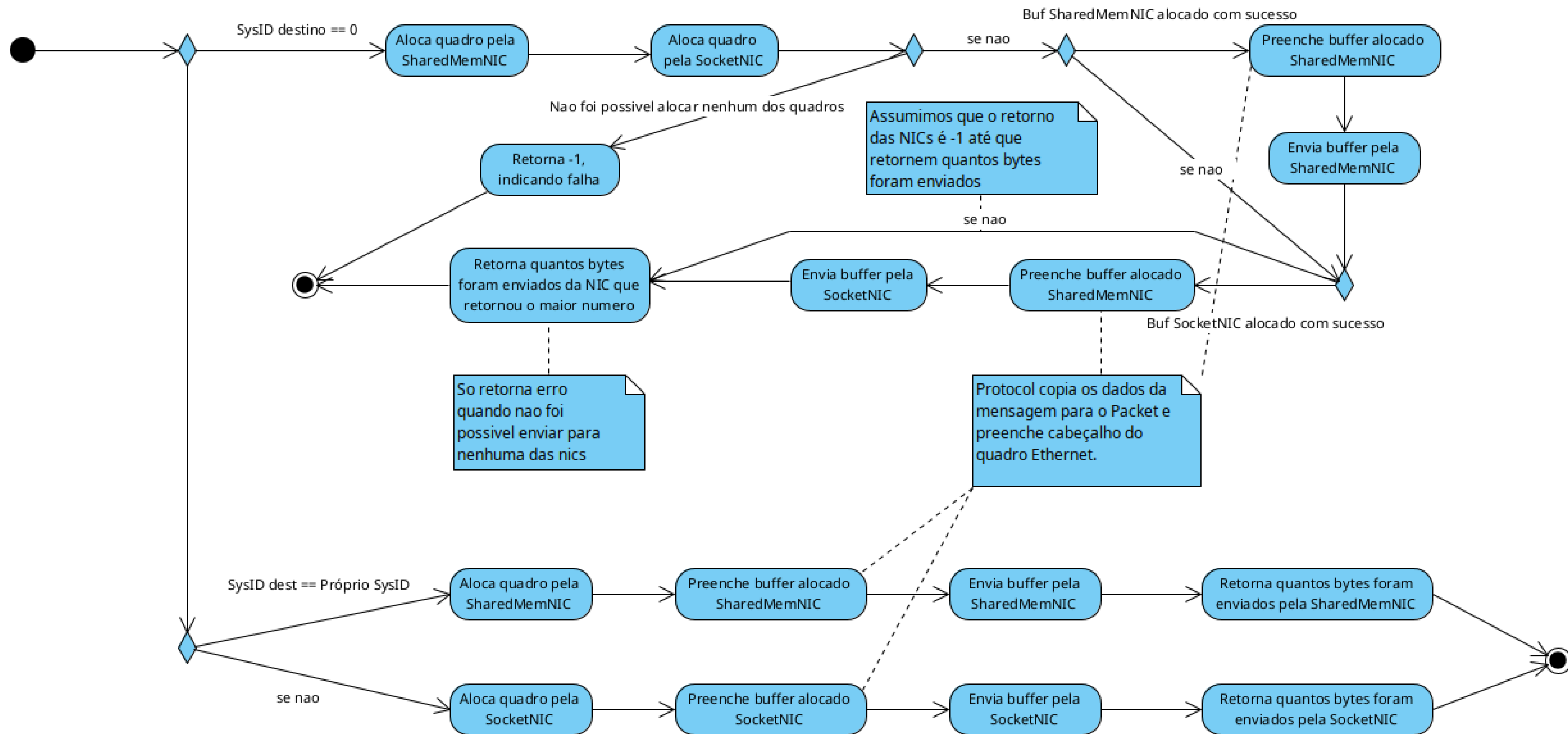
Caso uma mensagem seja enviada
com o SysID 0, todos os veículos
(Processos) alcançáveis dentro da
rede recebem a mensagem.

0xFFFF = Broadcast_Port

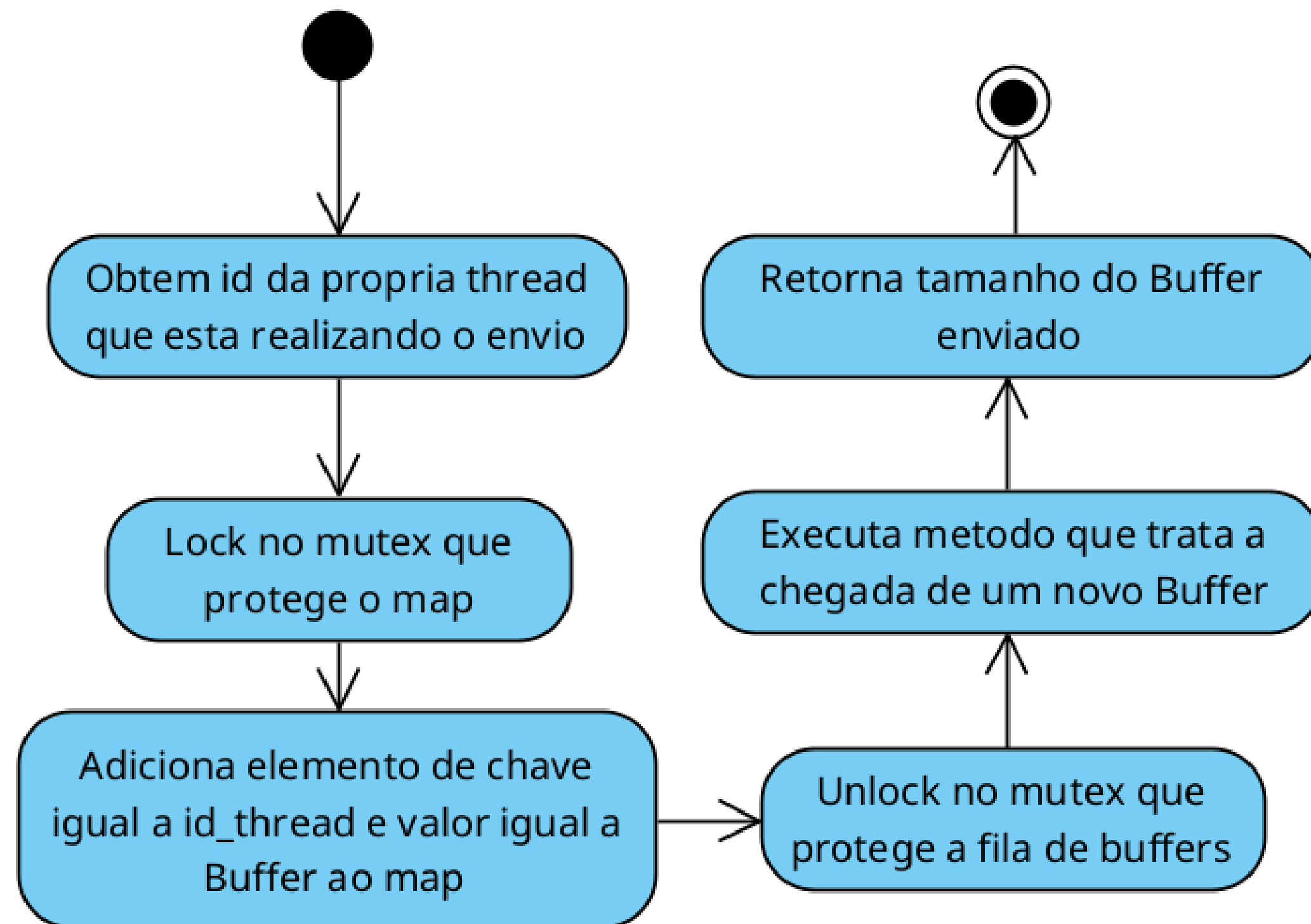
Caso uma mensagem seja enviada
com Port 0xFFFF, todos os
componentes(Threads) de um
veículo (Processo) recebem a
mensagem.

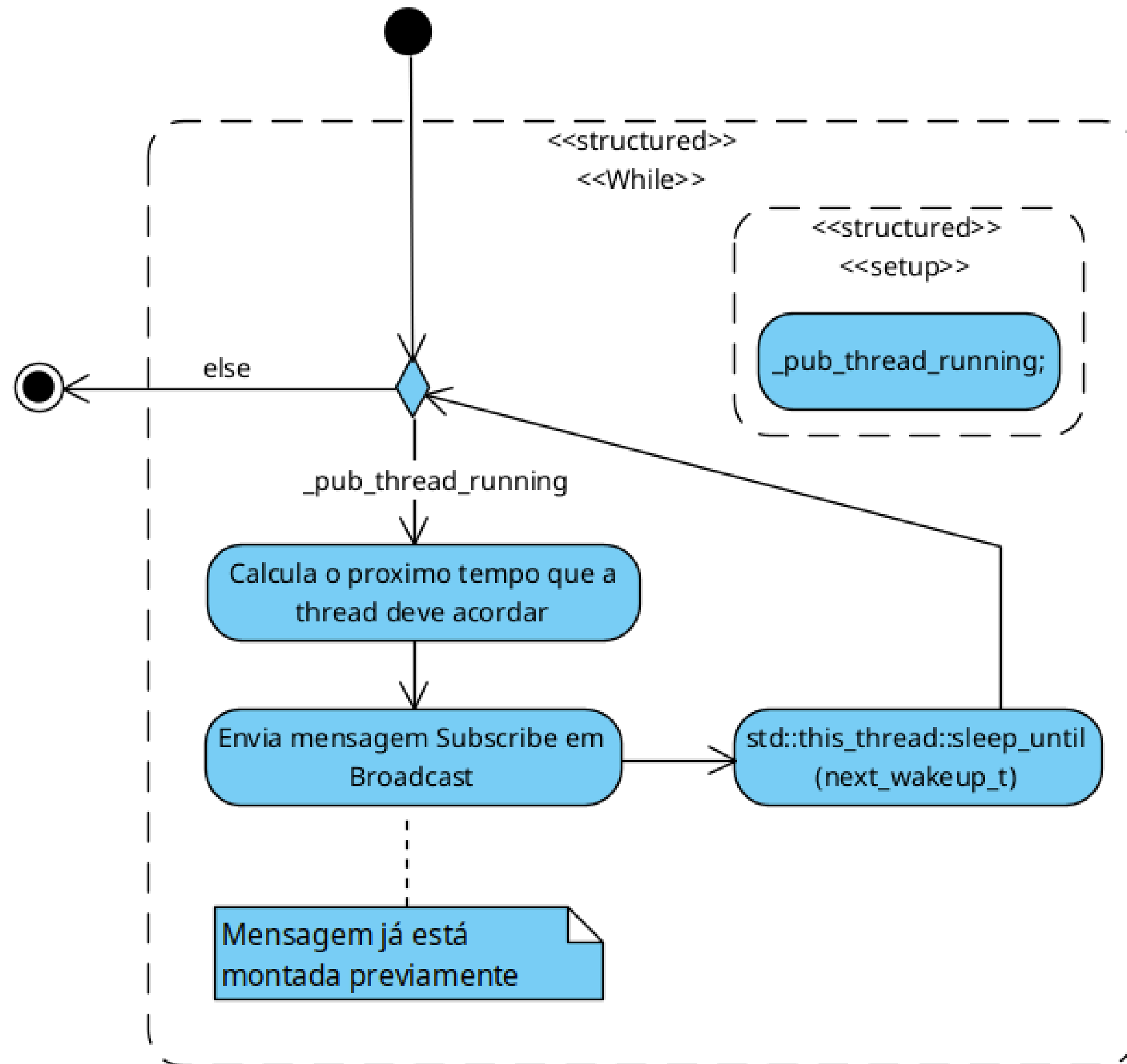


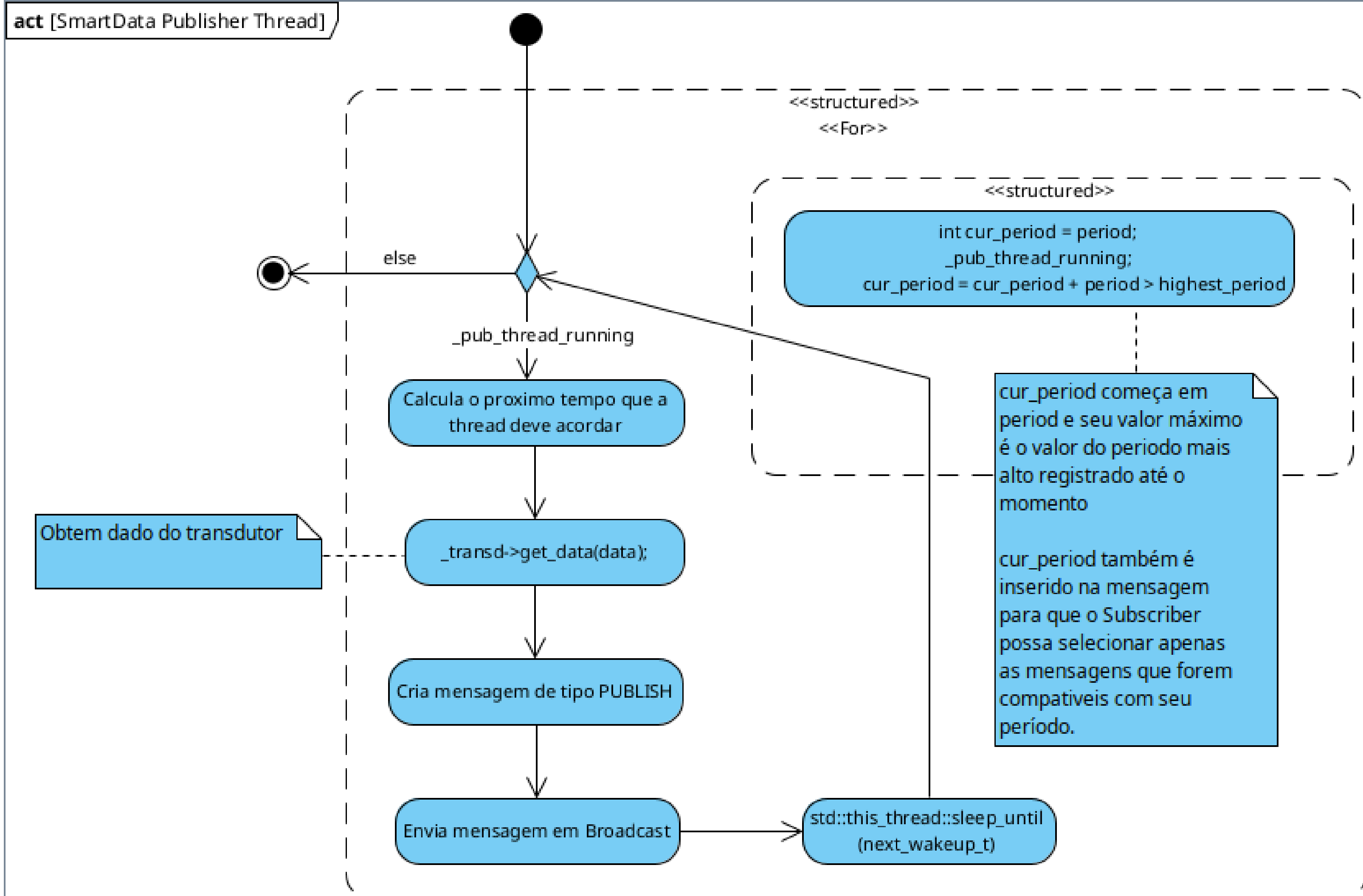
act [protocol.send(...)]



```
std::unordered_map<std::thread::id, Buffer>
```



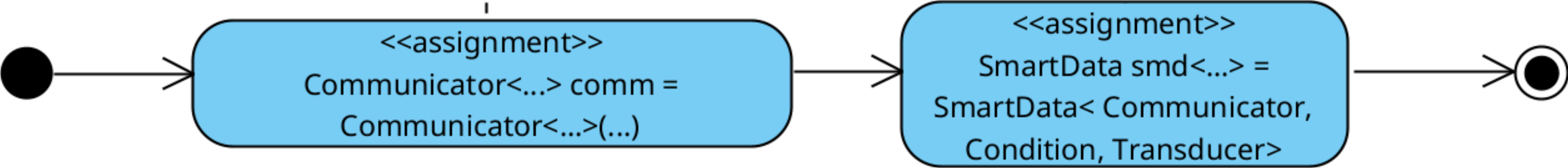




act [User View Publish]

Considere que outras entidades da pilha a partir de Protocol ja estao instanciadas no carro.

Ao instanciar um SmartData com template Communicator, Condition e Transducer, ele estará pronto para receber Subscribers da rede.



act [SmartData.update(...)]

Subscriber SmartData

Publisher SmartData

