



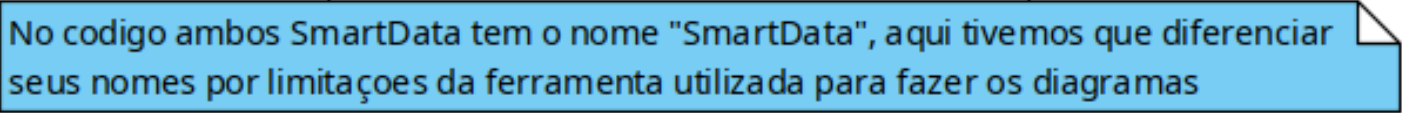
Departamento de  
Informática e Estatística  
**CTC • UFSC**



# Visão geral do projeto

**INE5424 - Sistemas Operacionais II**

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



# Principais Estruturas de Dados

Buffer	4	4	4	1	8	1514
	Type	Size	MaxSize	in_use	timestamp	Data

Ethernet Frame	6	6	2	46 - 1500
	Mac dest	Mac src	Ethertype	Payload

Protocol FullPacket	12	12	1	8	8	8	4	16	0 - 1431
	Address dest	Address src	Control	coord_x	coord_y	Timestamp	Length	Tag	Data

SharedMem Frame	2	0 - 1500
	Ethertype	Payload

Protocol LitePacket	2	2	1	4	0 - 1431
	Port dest	Port src	Control	Length	Data

Message	12	12	1	8	8	8	4	0 - 1431
	Address dest	Address src	Control	coord_x	coord_y	Timestamp	Length	Data

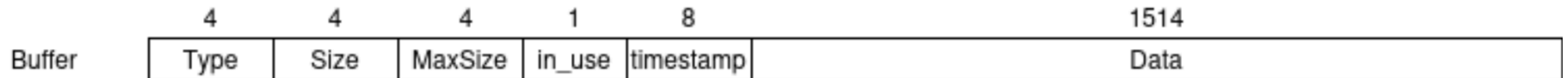
Protocol Address	6	4	2
	MAC	SysID	Port

O tamanho de todos os campos  
são representados em Bytes

# Estrutura de dados: Buffer

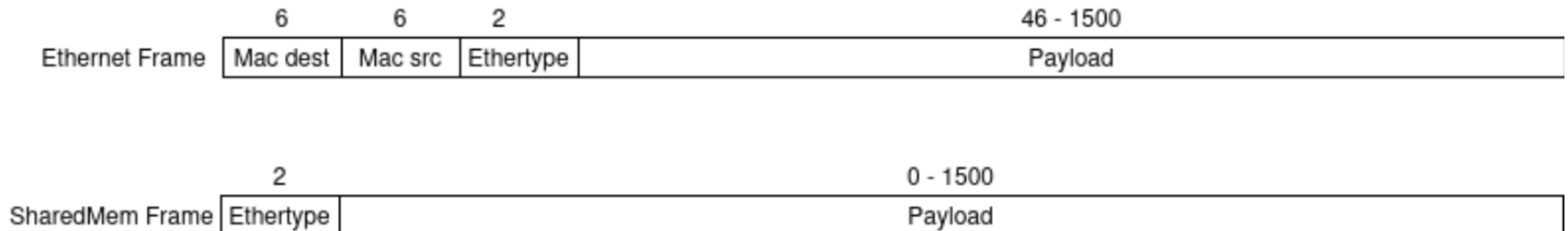
Nossa arquitetura é baseada em uma série de estruturas alinhadas, onde cada uma encapsula a próxima, permitindo que cada camada da pilha de comunicação opere com o nível de abstração apropriado.

- **Definição:** Contêiner de transporte de dados. É uma classe que encapsula um array de bytes de tamanho fixo e dados de controle.
- **Dados de controle:**
  - `_type`: Enumerador que define o conteúdo como `EthernetFrame` ou `SharedMemFrame`.
  - `_in_use`: Booleano para o gerenciamento do pool de memória. Controla se o buffer está alocado ou livre.
  - `_receive_time`: Timestamp (`int64_t`) capturado pela NIC no momento do recebimento de um quadro.



# Estrutura de dados: Frames

- **Definição:** Representa o quadro da camada de enlace. Existem duas variantes:
  - **Ethernet::Frame:** Estrutura de 14 bytes de cabeçalho (MAC Destino, MAC Origem, EtherType) e payload, correspondendo a um quadro Ethernet padrão.
  - **SharedMem::Frame:** Estrutura com um cabeçalho de 2 bytes contendo apenas o EtherType e payload.
- **Operação:** O EtherType é presente em ambas as variantes para permitir que a NIC utilize um mecanismo de notificação uniforme para o Protocol, independentemente do Frame ter vindo da SocketEngine ou da SharedEngine.



# Estrutura de Endereçamento Hierárquico

A identificação dos agentes de comunicação é feita por uma estrutura de endereço de 12 bytes, dividida em três níveis:

- **Nível Físico - MAC Address (6 bytes):** Identifica a interface de rede física. É utilizado pela SocketEngine para construir o cabeçalho Ethernet. É ignorado na comunicação interna.
- **Nível de Sistema - SysID (4 bytes, pid\_t):** Identifica o sistema autônomo, modelado como um processo POSIX. Permite Protocol distinguir entre tráfego interno e externo.
- **Nível de Componente - Port (2 bytes, unsigned short):** Identifica um componente, que corresponde a uma instância de Communicator em uma thread específica.

Endereços especiais:

SysID:

0 = Broadcast entre veículos e componentes

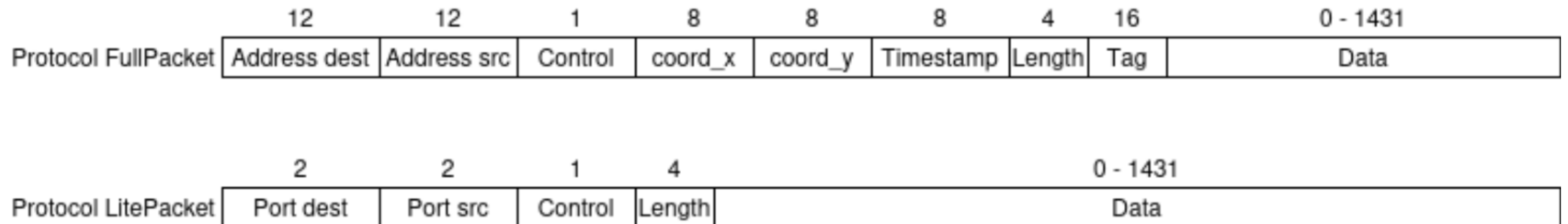
-2 = Broadcast entre veículos

Port:

0xFFFF = Broadcast entre componentes de um veículo

# Estrutura de dados: Packet

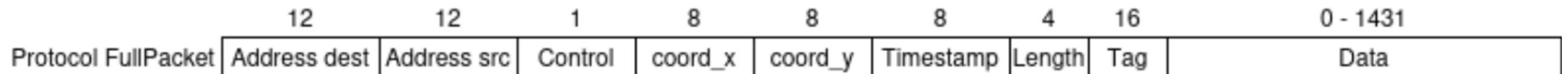
- Os pacotes estão contidos no payload dos Frames e é onde os dados necessários para a lógica do nosso protocolo residem. Novamente, existem duas versões.
  - **FullPacket:** Utilizado para comunicação externa, seu cabeçalho contém endereços completos, coordenadas, timestamp e tag de segurança.
  - **LitePacket:** Utilizado para comunicação interna, contém um cabeçalho otimizado apenas com as informações essenciais.



# Formato do Pacote Externo: FullPacket

- **Campos do FullHeader:**

- Address origin e Address dest: Endereços completos de 12 bytes.
- Control ctrl: Byte de controle com flags de tipo e estado.
- double coord\_x, coord\_y: Coordenadas do remetente, obtidas do Navigator.
- uint64\_t timestamp: Timestamp de envio, obtido do SyncEngine.
- MAC::Tag tag: Tag de 16 bytes do Poly1305.





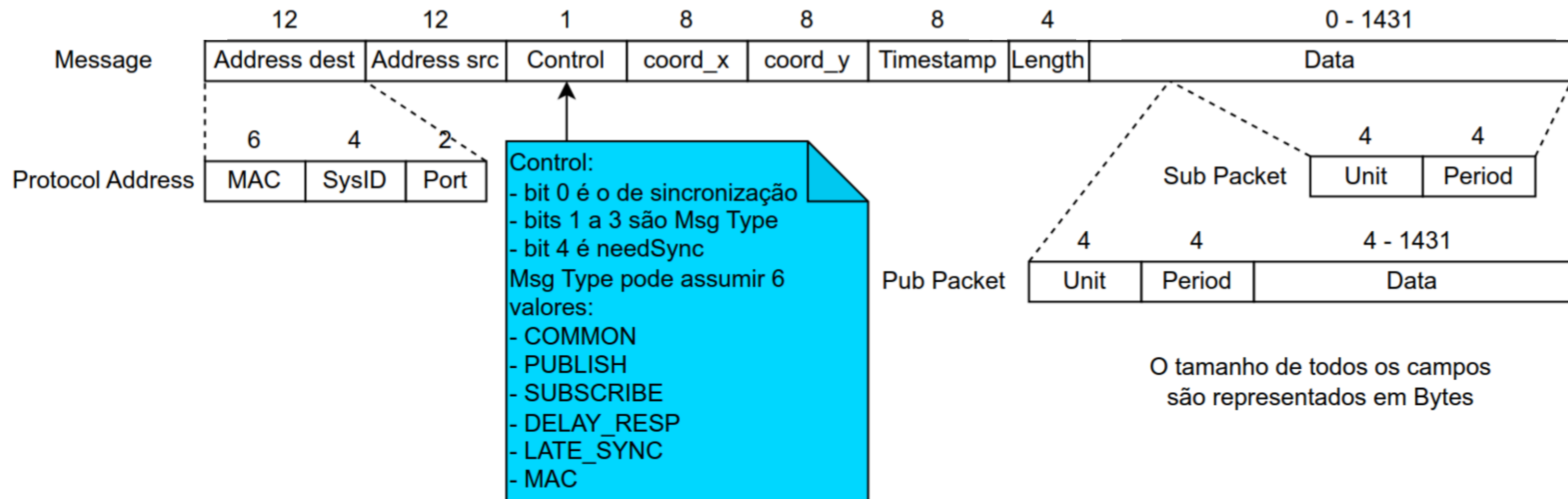
# Formato do Pacote Interno: LitePacket e Otimização

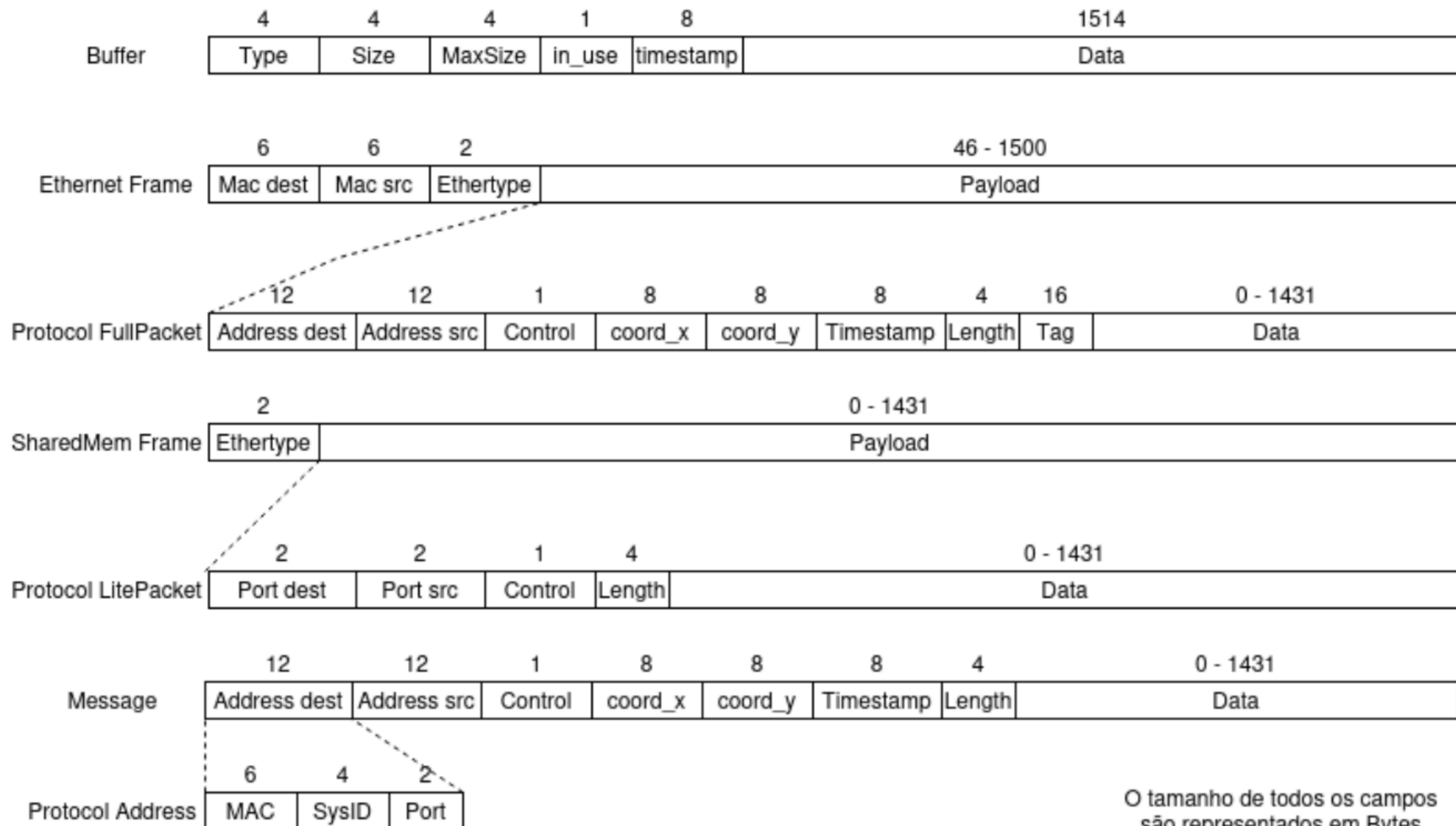
- **Campos Mantidos:** Apenas a Port de origem e destino, o campo de Control e o tamanho do payload.
- **Campos Omitidos:**
  - **Endereço Físico e SysID:** São implícitos, pois a comunicação ocorre dentro do mesmo processo.
  - **Coordenadas:** A classe Message foi projetada para obter esses valores sob demanda, consultando diretamente o Navigator do Protocol.
  - **Timestamp:** Timestamp é o obtido a partir do timestamp de recebimento do Buffer.
  - **Tag de Autenticação:** Removida, pois a verificação de segurança é desnecessária dentro do mesmo domínio.



# Estrutura de dados: Message

- Message - A Interface para a Aplicação
  - **Função:** É a estrutura de dados que a camada de aplicação manipula diretamente. Ela abstrai os formatos de pacote subjacentes.
  - **Mecanismo de Marshalling/Unmarshalling:** No envio, o Protocol serializa (marshal) os dados de um objeto Message para dentro de um FullPacket ou LitePacket. Na recepção, ele desserializa (unmarshal) os dados de um Packet para preencher um objeto Message.

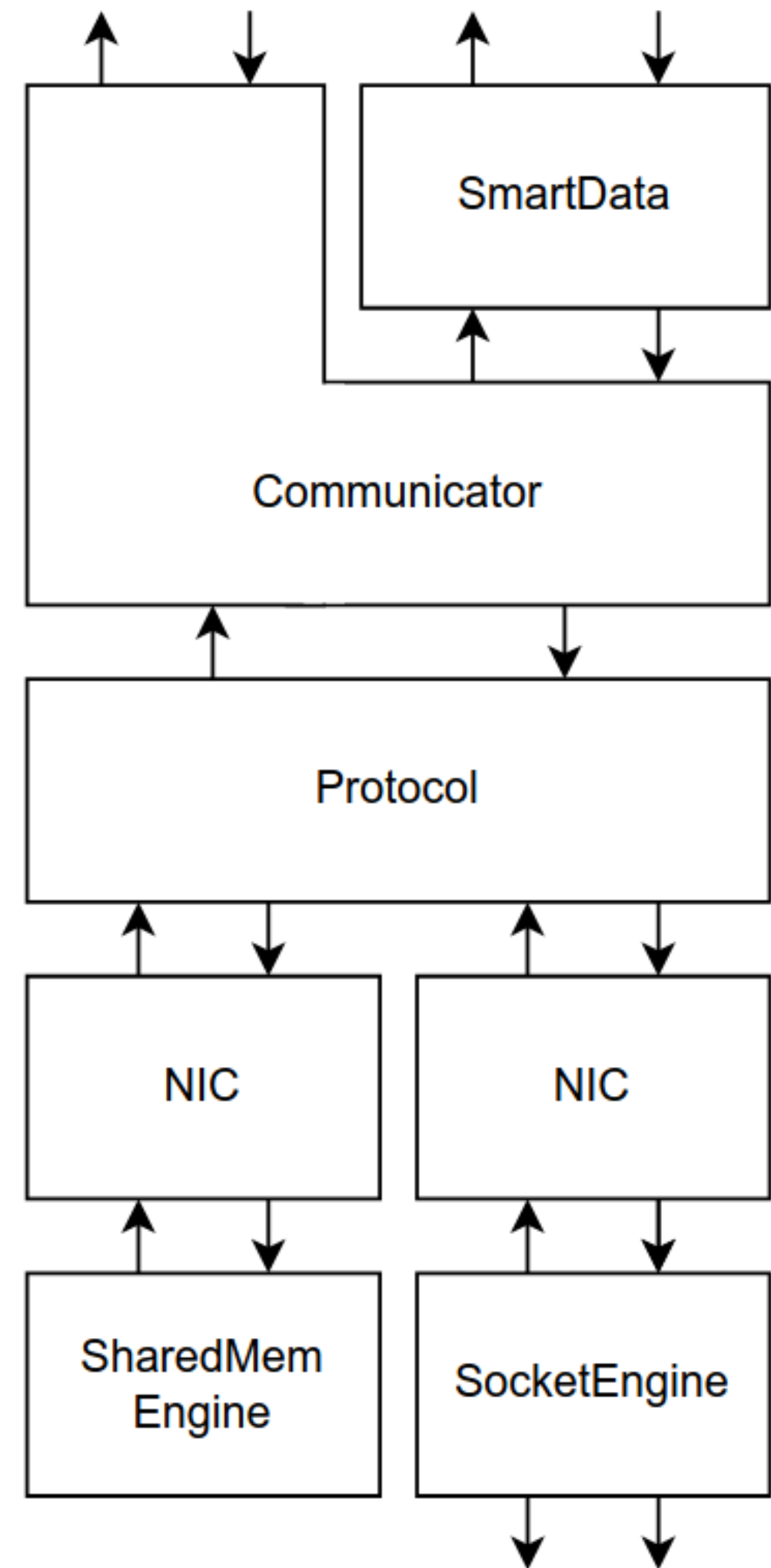




O tamanho de todos os campos  
são representados em Bytes

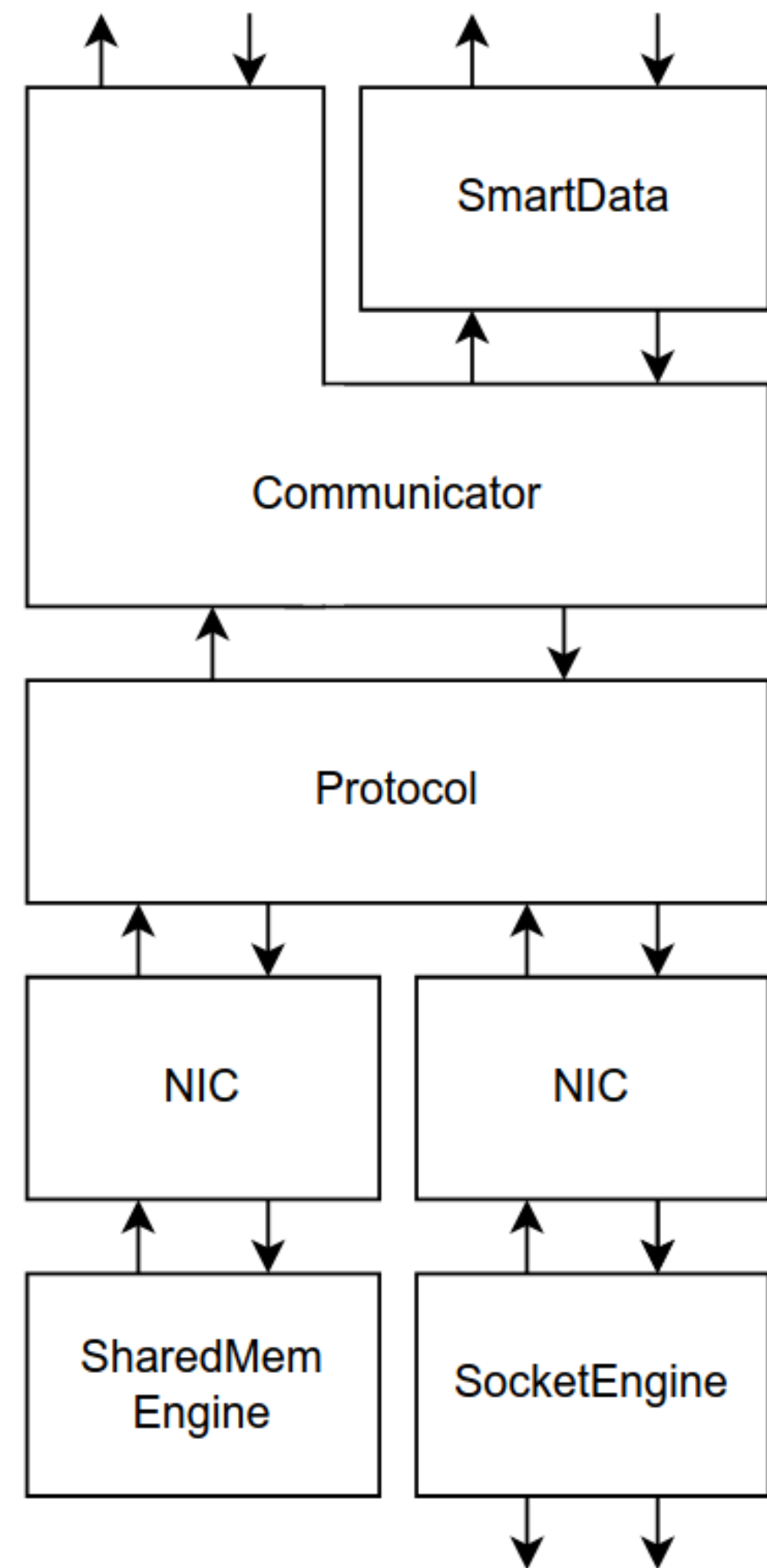
# Visão Geral da Arquitetura

- **Communicator e SmartData:** Camada de interface com a aplicação. SmartData constrói sobre o Communicator para implementar a lógica de Publish-Subscribe.
- **Protocol:** Roteamento de mensagens (interno vs. externo) com base no SysID, demultiplexação por porta para os Communicators, implementação da lógica de segurança (verificação de MAC) e de sincronização temporal (PTP).



# Visão Geral da Arquitetura

- **NIC:** Atua como uma fachada sobre a Engine. É responsável pelo gerenciamento do ciclo de vida dos buffers e notifica a camada Protocol com base no EtherType dos quadros recebidos.
- **Engines:** Encapsulam comunicação por rawsockets ou memória compartilhada.



# SocketEngine - comunicação entre veículos

- **Mecanismo de Comunicação:** Utiliza um Raw Socket para envio e recebimento de quadros Ethernet.
- **Recepção Assíncrona:** A chegada de pacotes é notificada pelo sinal SIGIO. A Engine se registra para receber este sinal usando fcntl com a flag F\_SETOWN.
- **Tratamento de Sinal:** O *signal handler* registrado para SIGIO executa uma única função: notificar uma `std::condition_variable`. A thread de recebimento, que está bloqueada nesta variável, é então acordada e invoca a lógica de processamento (`NIC::handle_signal`).
- **Justificativa:** Esta abordagem foi escolhida para contornar as restrições dos *signal handlers* em POSIX. Funções não Async-Signal-Safe, como funções não reentrantes, alocação de memória ou aquisição de mutexes, não podem ser chamadas de forma segura dentro de um handler. Ao delegar o trabalho a uma thread separada, garantimos que o processamento dos pacotes ocorra em um contexto controlado.

# SharedMemEngine - comunicação interna

A SharedMemEngine é a implementação da Engine para comunicação intra-processo.

- **Mecanismo de Comunicação:** A transferência de dados ocorre inteiramente em memória compartilhada. A estrutura de dados central é um mapa que associa o identificador de uma thread a um **ponteiro para um Buffer**. O acesso a este mapa é protegido através de um semáforo binário.
  - Para manter a API consistente com a SocketEngine, o método `send(...)` da SharedMemEngine, após inserir o buffer no mapa, invoca diretamente o método `NIC::handle_signal()` da SharedMemNIC. Isso permite que a camada NIC trate ambos os tipos de Engine de forma idêntica.

# NIC e o Gerenciamento de Buffers

A NIC abstrai a Engine e implementa uma política de gerenciamento de memória para os buffers de comunicação.

- **Pools de buffers:** A NIC pré-aloca, em sua inicialização, dois arrays de objetos Buffer: um para envio e outro para recepção. O tamanho desses pools é fixo e definido em tempo de compilação.
- **Ciclo de Vida do Buffer:**
  - `alloc(...)`: A camada Protocol solicita um buffer. O método `alloc(...)` percorre o pool de forma circular, protegido por um `std::mutex`, para encontrar o primeiro buffer que não está em uso (`!is_in_use()`).
  - O buffer é marcado como "em uso" e retornado.
  - `free(...)`: Após o uso, o buffer é devolvido ao pool, e sua flag `in_use` é resetada.
- **Justificativa:** Evitar a alocação dinâmica de memória (`new/delete`) durante o fluxo crítico de comunicação. Alocações dinâmicas introduzem latência, com o pooling, a obtenção de um buffer se torna uma operação mais performática.



# Protocol

- **Envio de pacotes:**

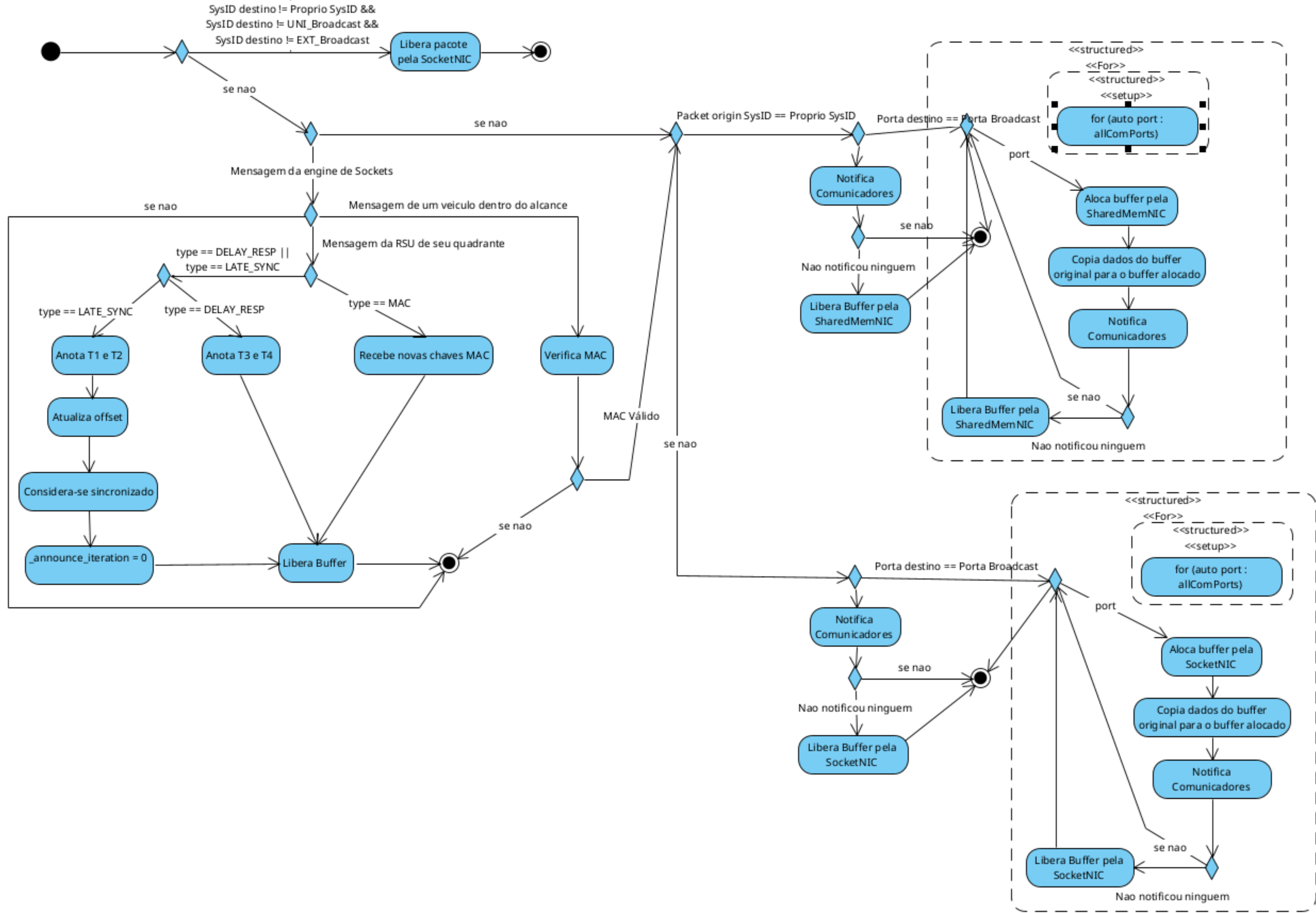
- Se o SysID de destino é igual ao SysID local, a mensagem é interna e encaminhada para a instância da SharedMemNIC.
- Caso contrário, a mensagem é externa e encaminhada para a instância da SocketNIC.
- Implementa broadcast interno ao veículo, externo e ambos simultaneamente.

- **Recebimento de pacotes:**

- Protocol atua como observador das NICs. Quando seu método update(...) é invocado por uma das NICs, ele extrai a Port de destino do cabeçalho do pacote recebido. Em seguida, ele notifica apenas o Communicator que se registrou para aquela Port específica.

- **Sincronização Temporal**

- **Comunicação segura**



# RSUs

- RSUs fazem o papel de líder do PTP para os veículos dentro de seu quadrante.
- RSUs distribuem periodicamente as chaves MAC necessárias para os veículos se comunicarem, ou seja, periodicamente os veículos recebem a chave de sua RSU e das RSUs vizinhas.

## Mapa Exemplo

RSU	RSU	RSU
RSU	RSU	RSU
RSU	RSU	RSU

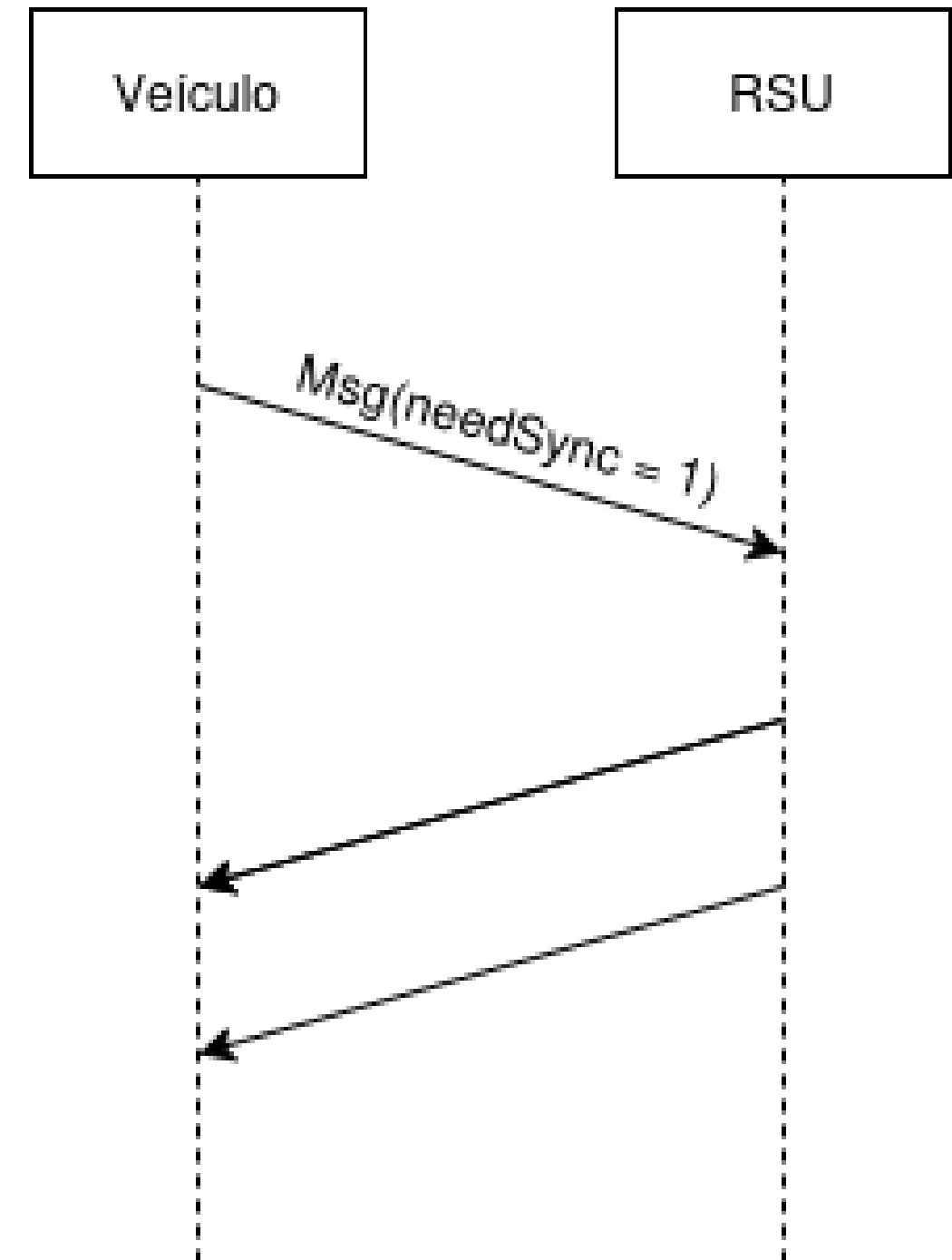
# Uso das Chaves no Veículo

- **KeyKeeper:** Classe auxiliar dentro de Protocol que armazena um mapa que associa o ID de um quadrante à sua chave MAC.
- **Navigator:** Fornece a localização geográfica do veículo e a topologia da rede.
- **Seleção da Chave Correta:**
  - **No envio:** O Protocol utiliza o Navigator para determinar seu quadrante atual. O ID deste quadrante é usado para consultar a chave correta no KeyKeeper e calcular a tag da mensagem.
  - **No recebimento:** O Protocol extrai as coordenadas do remetente do cabeçalho do pacote, determina o quadrante de origem da mensagem e utiliza a chave daquele quadrante para verificar a tag. Mensagens com tags inconsistentes são descartadas.

# Sincronização Temporal:

A sincronização temporal é implementada através de um modelo sob demanda.

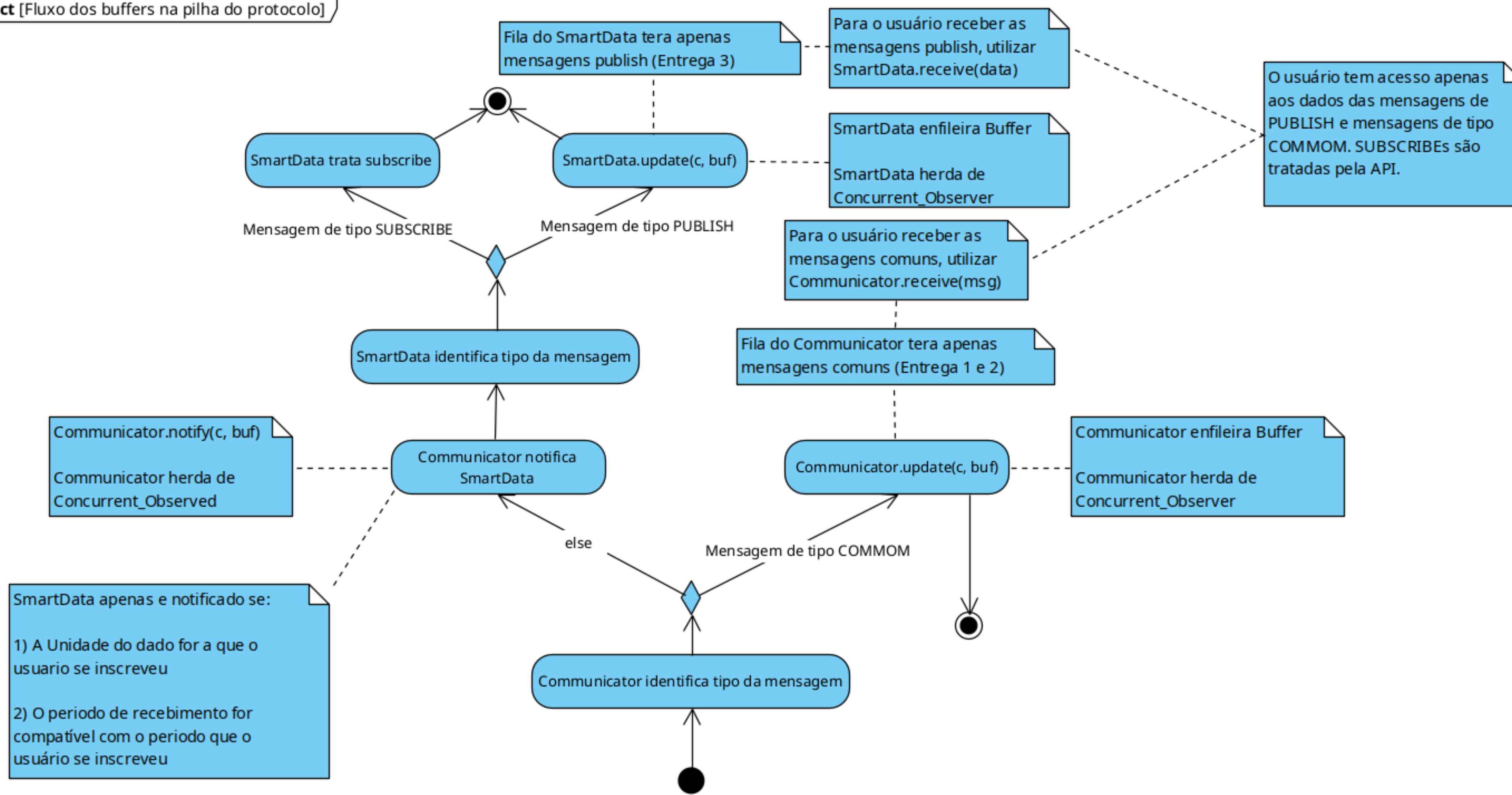
- **Papéis:** RSUs atuam como Mestres; Veículos atuam como Escravos.
- **Modelo de Interação:**
  - Um veículo que precisa de sincronização passa a enviar mensagens com a flag needSync ativada.
  - Se não está transmitindo Broadcasts, envia uma mensagem de Announce periodicamente.
  - A RSU, ao receber este pedido, responde com um par de mensagens DELAY\_RESP e LATE\_SYNC, que contêm os timestamps necessários para o cálculo do offset.



# Communicator

- Interface que o usuário pode utilizar para se comunicar por meio de mensagens do tipo **COMMON**
- Notifica camada superior (SmartData) quando necessário.

**act** [Fluxo dos buffers na pilha do protocolo]



# SmartData: Lógica do Publisher

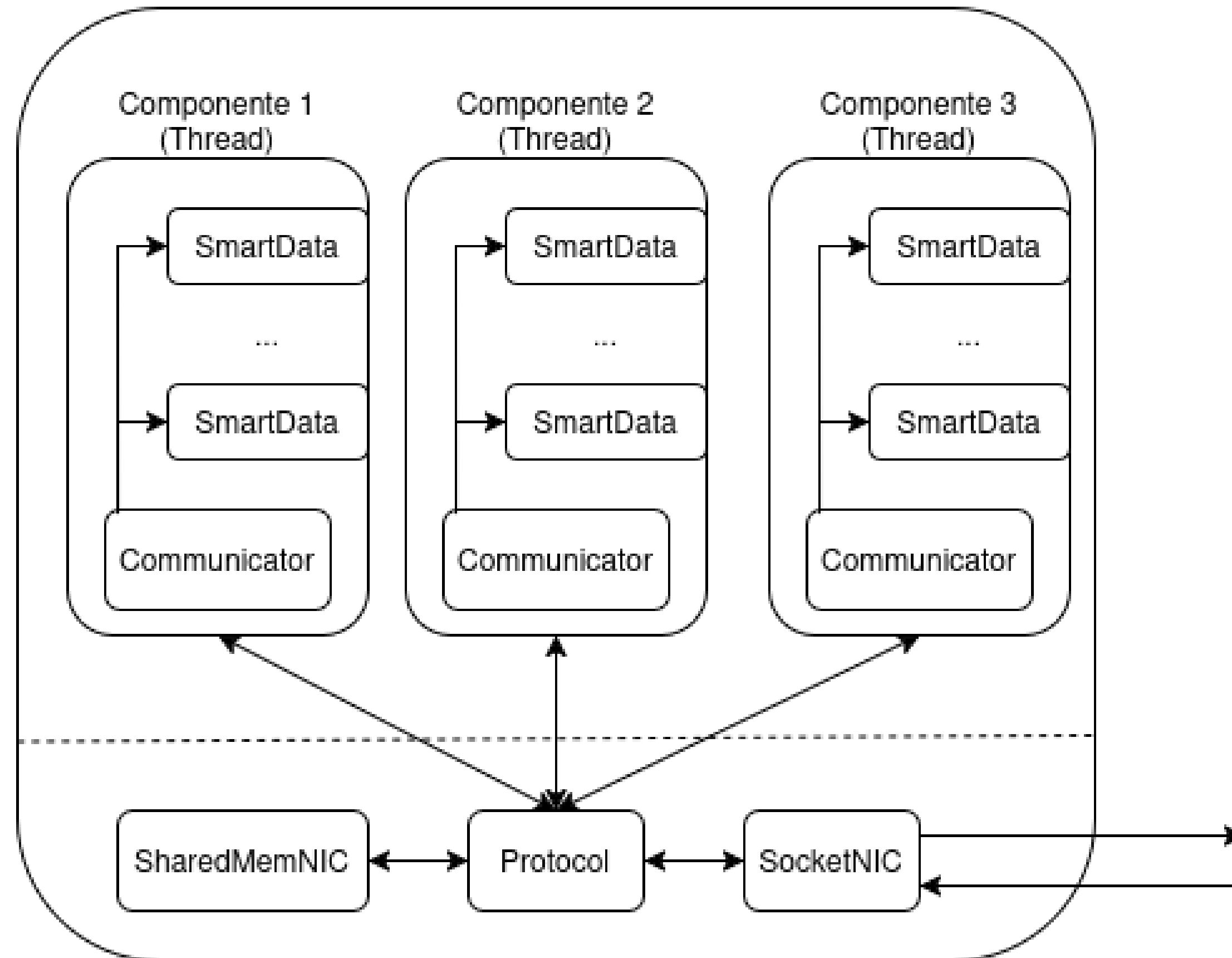
- **Entidade:** SmartData<..., ..., Transducer>.
  - Um SmartData Publisher tem um ponteiro para o transdutor que produz o seu dado.
- **Gerenciamento de Período:** O período é recalculado dinamicamente como o Máximo Divisor Comum (MDC) dos períodos de todos os assinantes ativos.
- **Contador de Ciclo:** Cada mensagem PUBLISH carrega um contador de ciclo. Este contador é incrementado pelo valor do MDC a cada publicação.
- **Justificativa:** O uso do MDC é uma otimização que garante que a frequência de publicação seja a mais baixa possível que ainda satisfaz o assinante mais exigente. O contador de ciclo permite que cada assinante saiba quais publicações lhe dizem respeito.



# SmartData: Lógica do Subscriber

- **Entidade:** SmartData<..., ..., void>.
- **Filtragem na Recepção:** Um Subscriber só processa uma mensagem PUBLISH se duas condições forem atendidas:
  - Correspondência de Tipo: A "unidade" de dados na mensagem é a mesma que a de sua subscrição.
  - Correspondência de Momento: O contador de ciclo da publicação é um múltiplo do período que ele solicitou.
- **Justificativa:** A verificação do contador de ciclo permite que o subscriber descarte publicações que não são para ele, mesmo que sejam do tipo de dado correto. Isso evita processamento de dados redundantes e permitindo que cada componente opere em sua própria taxa.

# Veiculo (Processo)



# Testes

## Teste de Latência

	Mínimo	Média	Máximo
Socket	13	390	71273
SharedMem	1	207	68452

tempos em microssegundos

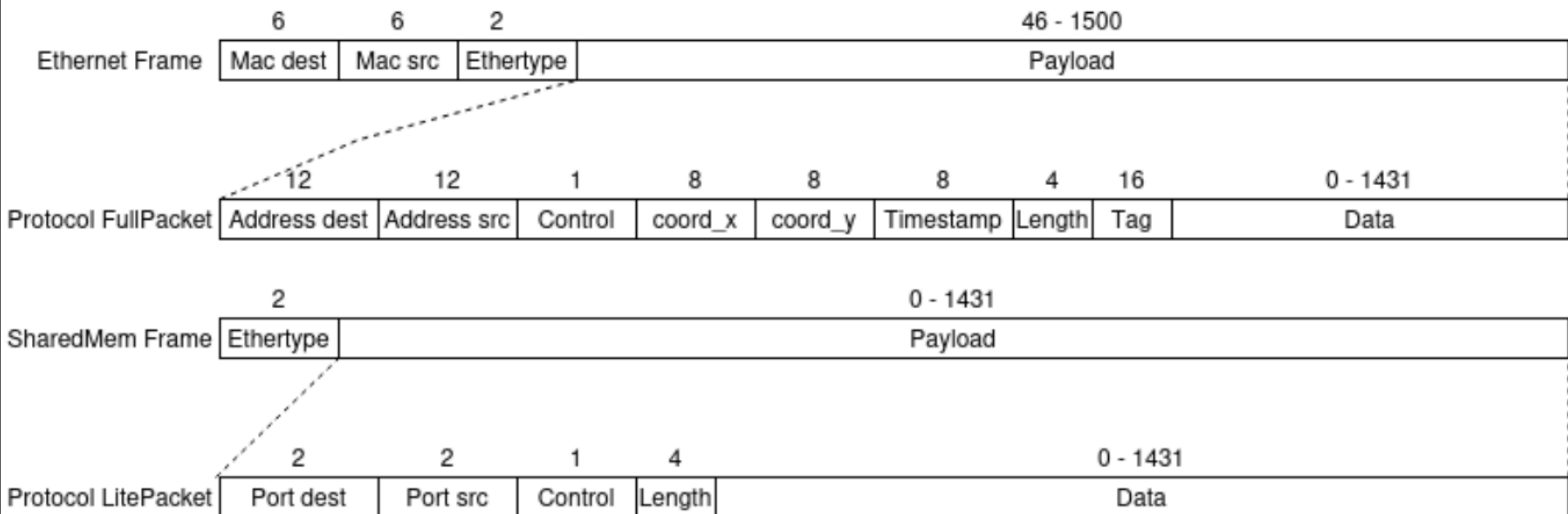
## Teste Simulação

Teste com somente um receptor

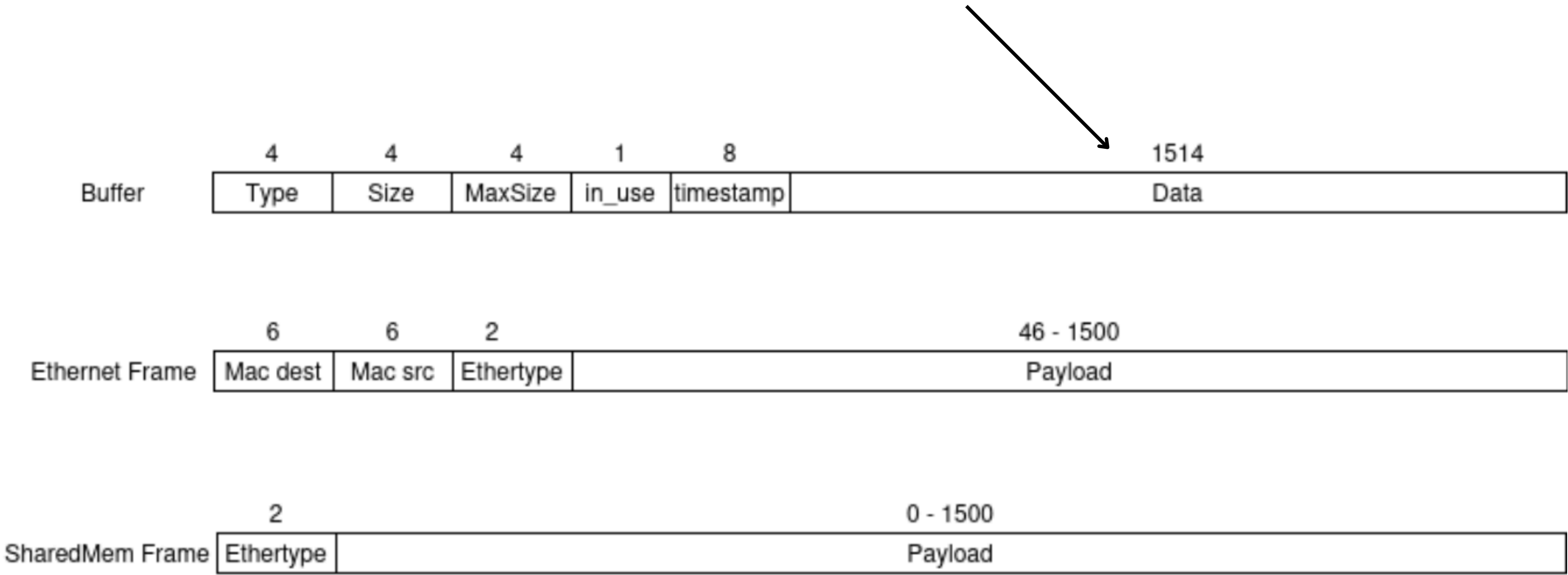
Entrega 6:

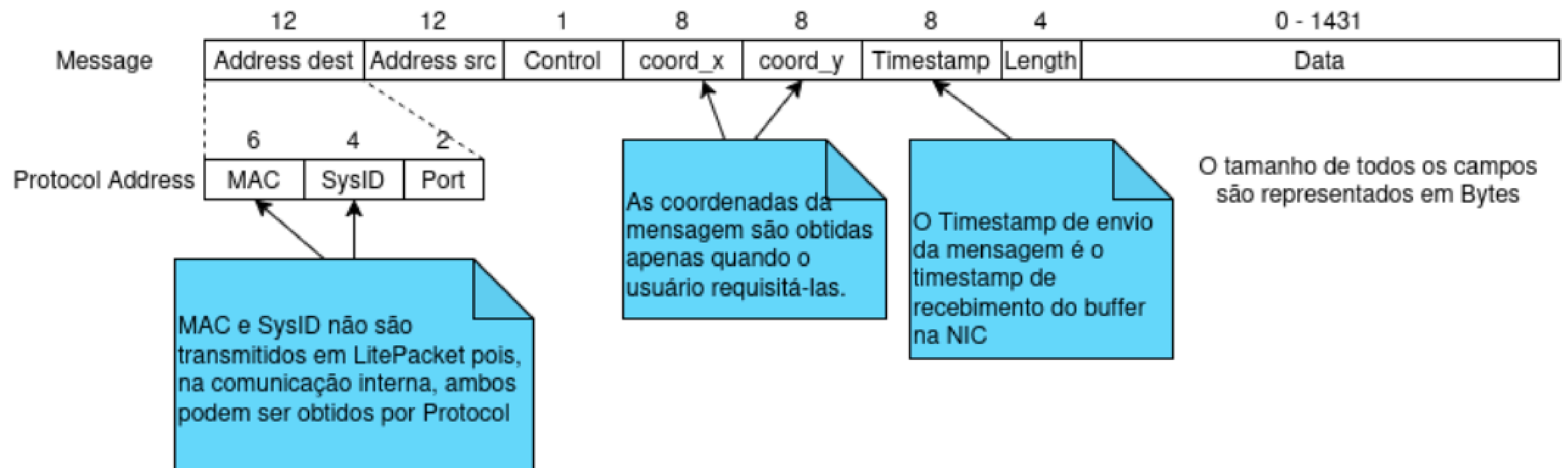
Teste com comunicação interna  
e externa

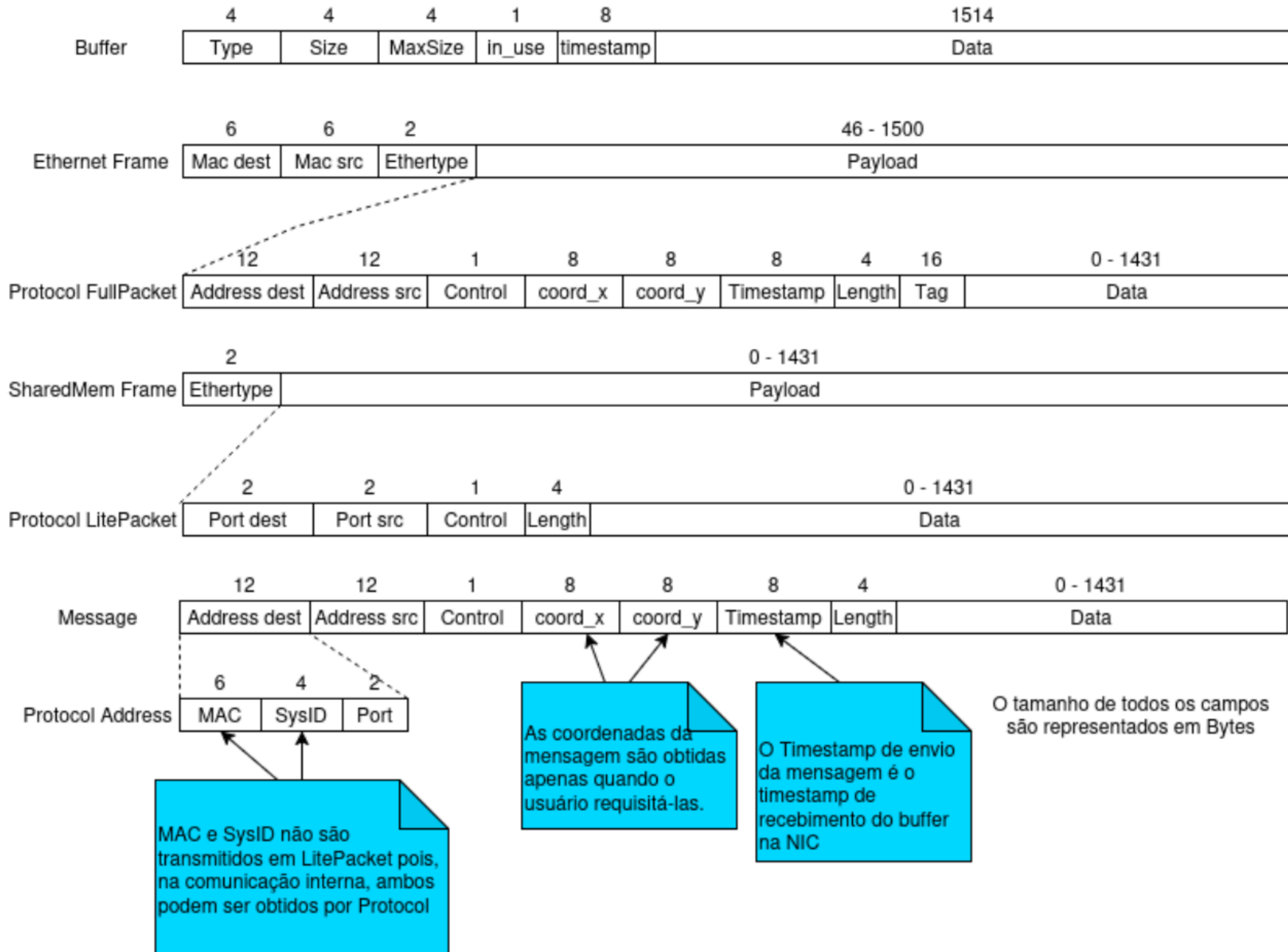
**Outros Diagramas:**



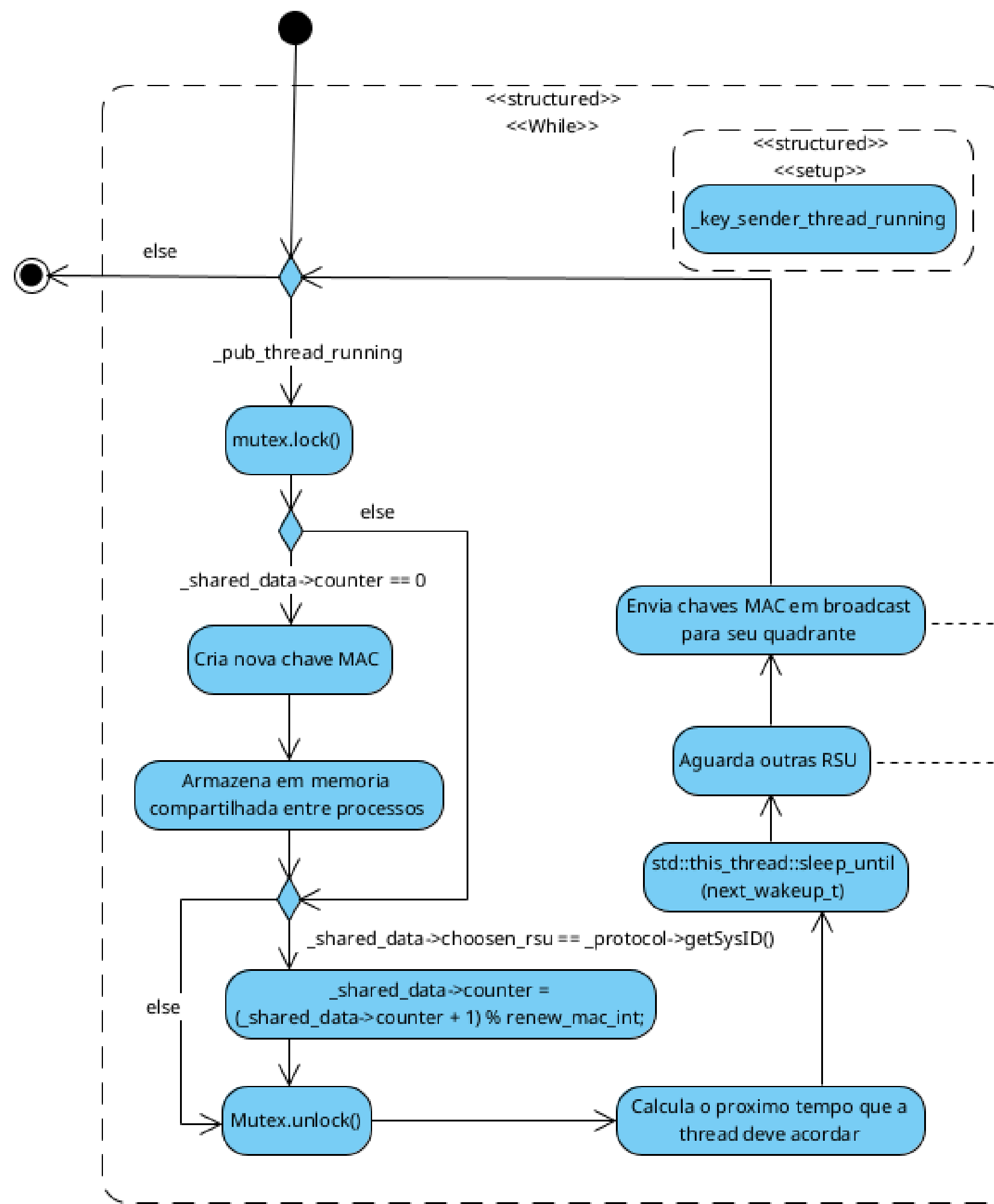
# Buffer.data armazena Ethernet::Frame ou SharedMem::Frame







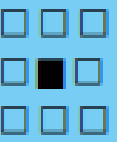






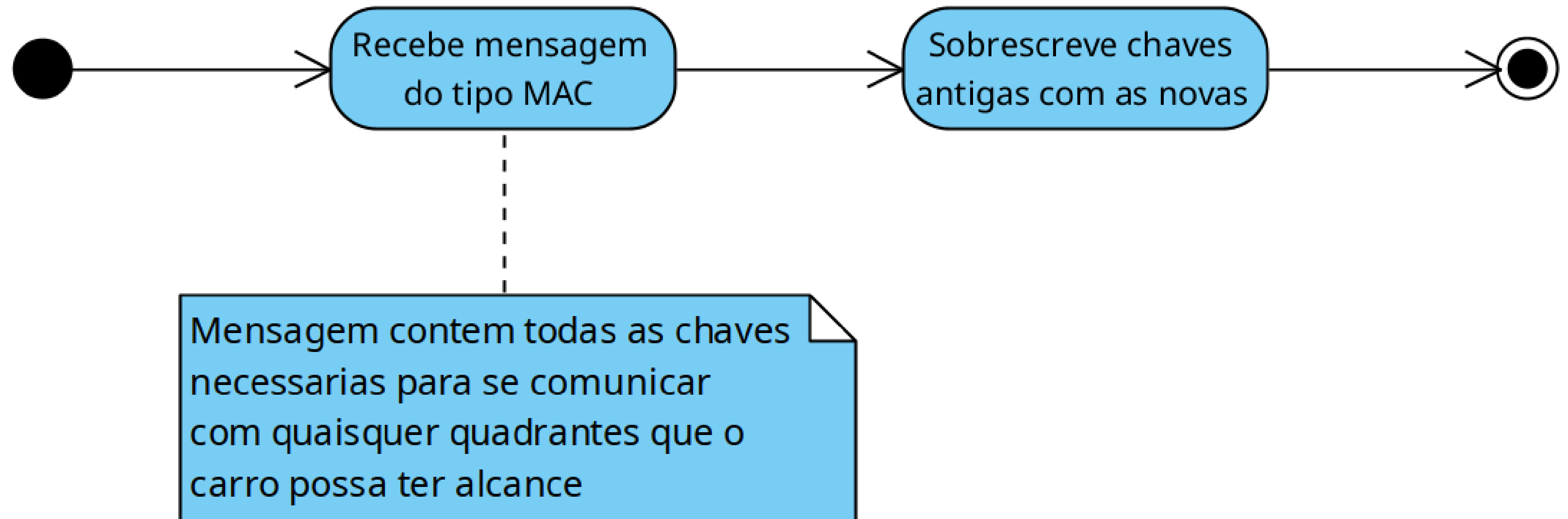


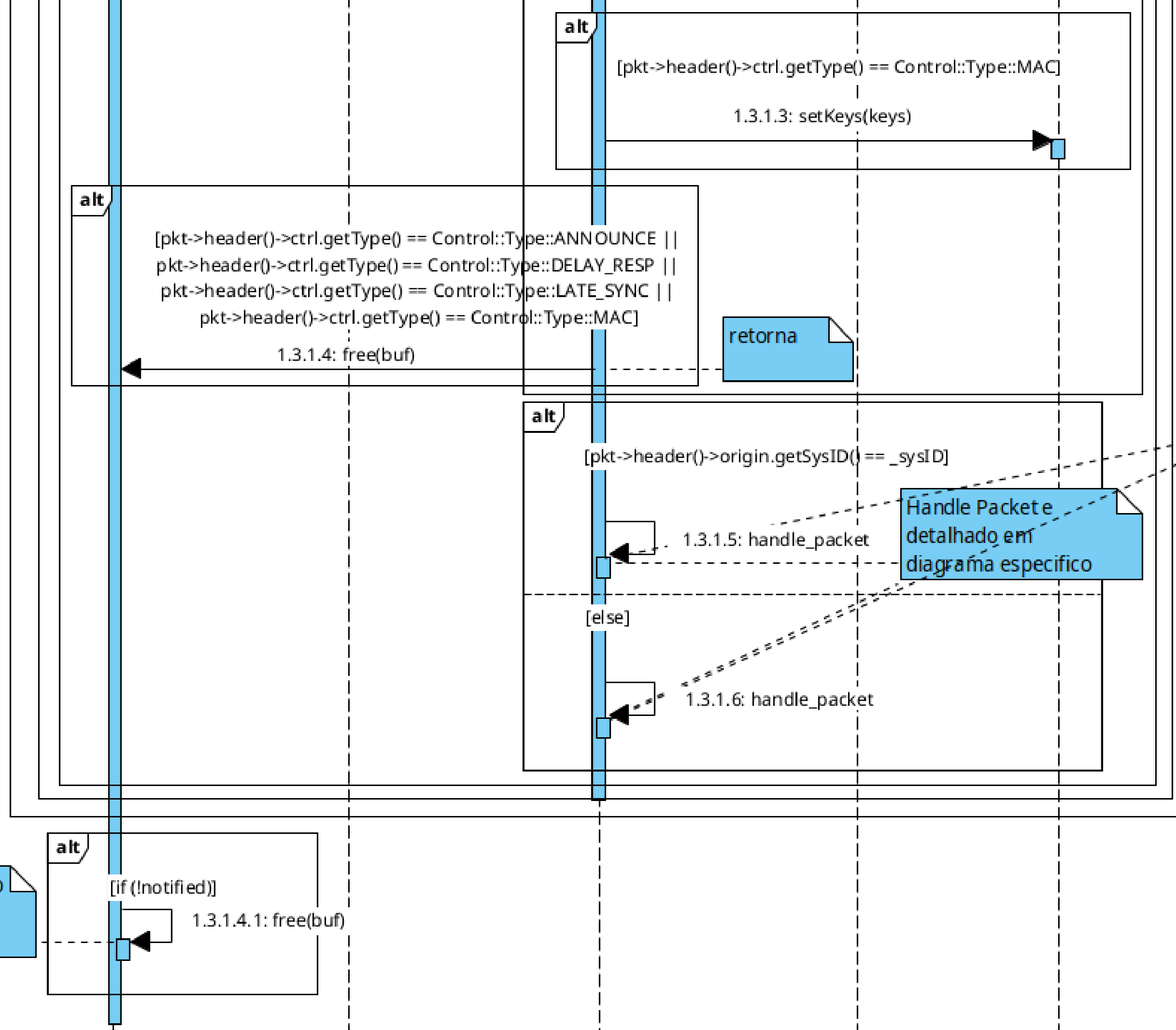
Dado uma RSU representada por , e RSUs adjacentes representadas por , uma RSU () envia sua chave e as chaves das RSU adjacentes():



`pthread_barrier_wait(&_shared_data->barrier)`

**act** [Recebimento de chaves MAC]

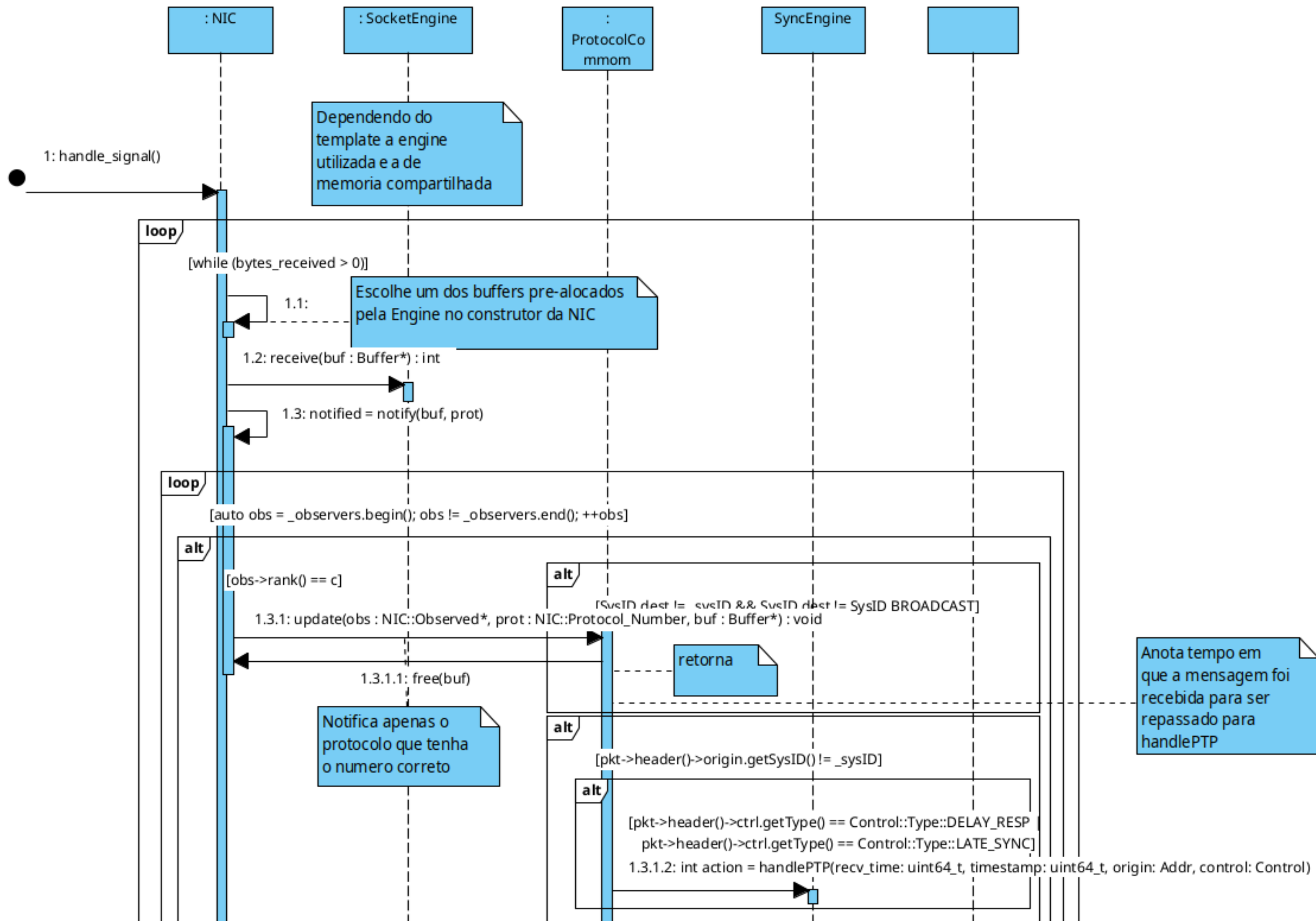


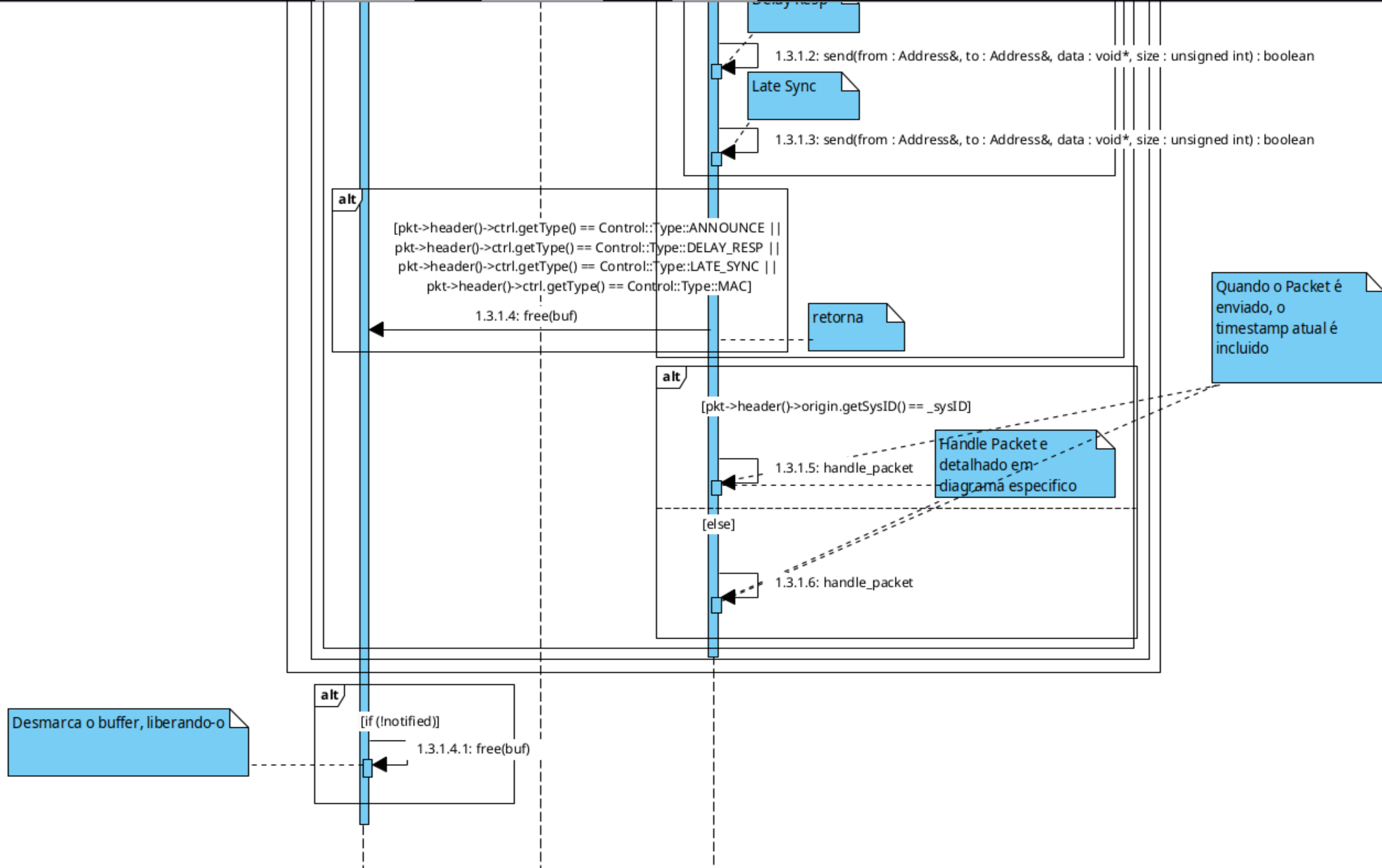


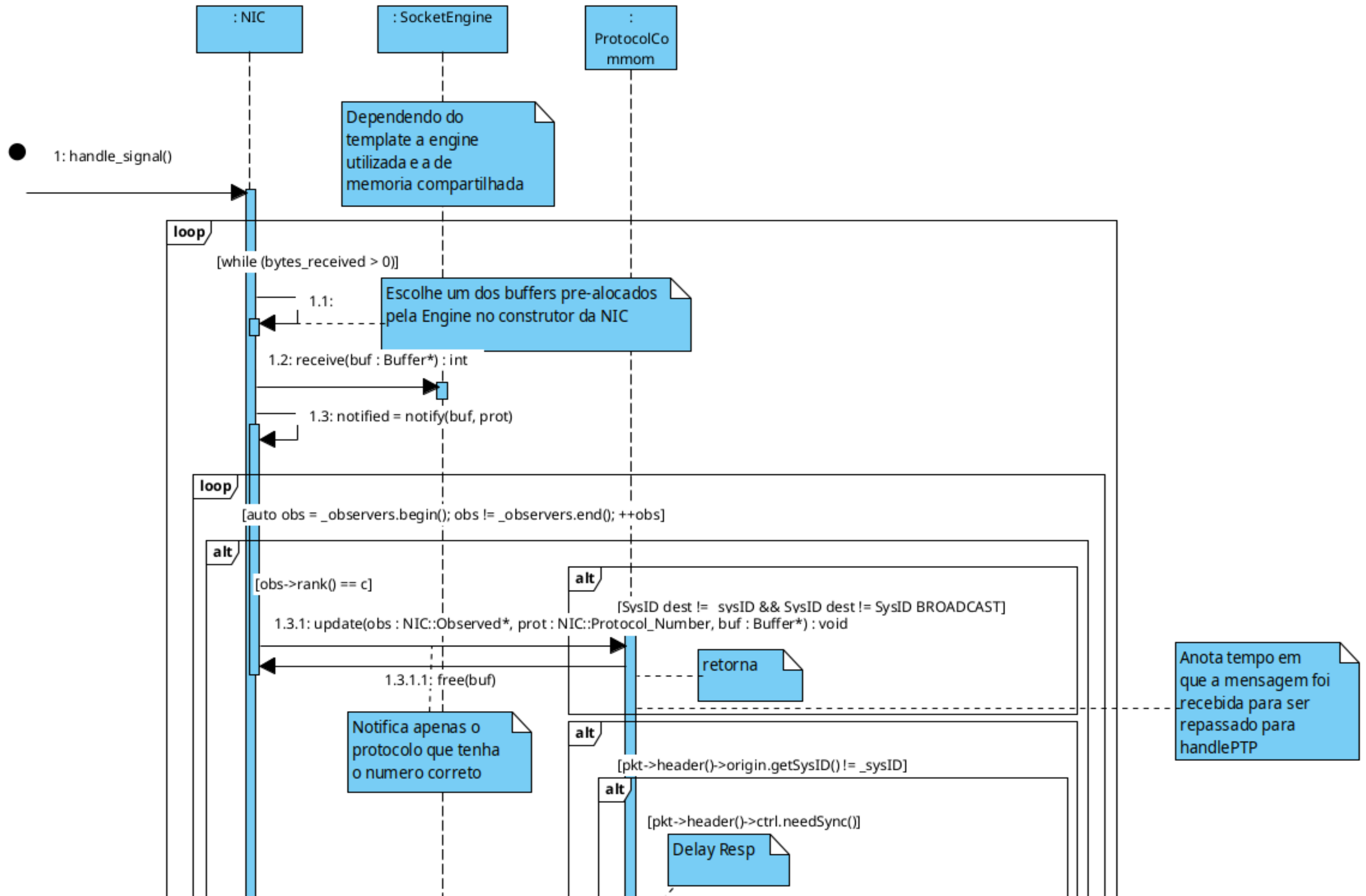
Quando o Packet é enviado, o timestamp atual é incluído

Handle Packet e detalhado em diagrama especifico

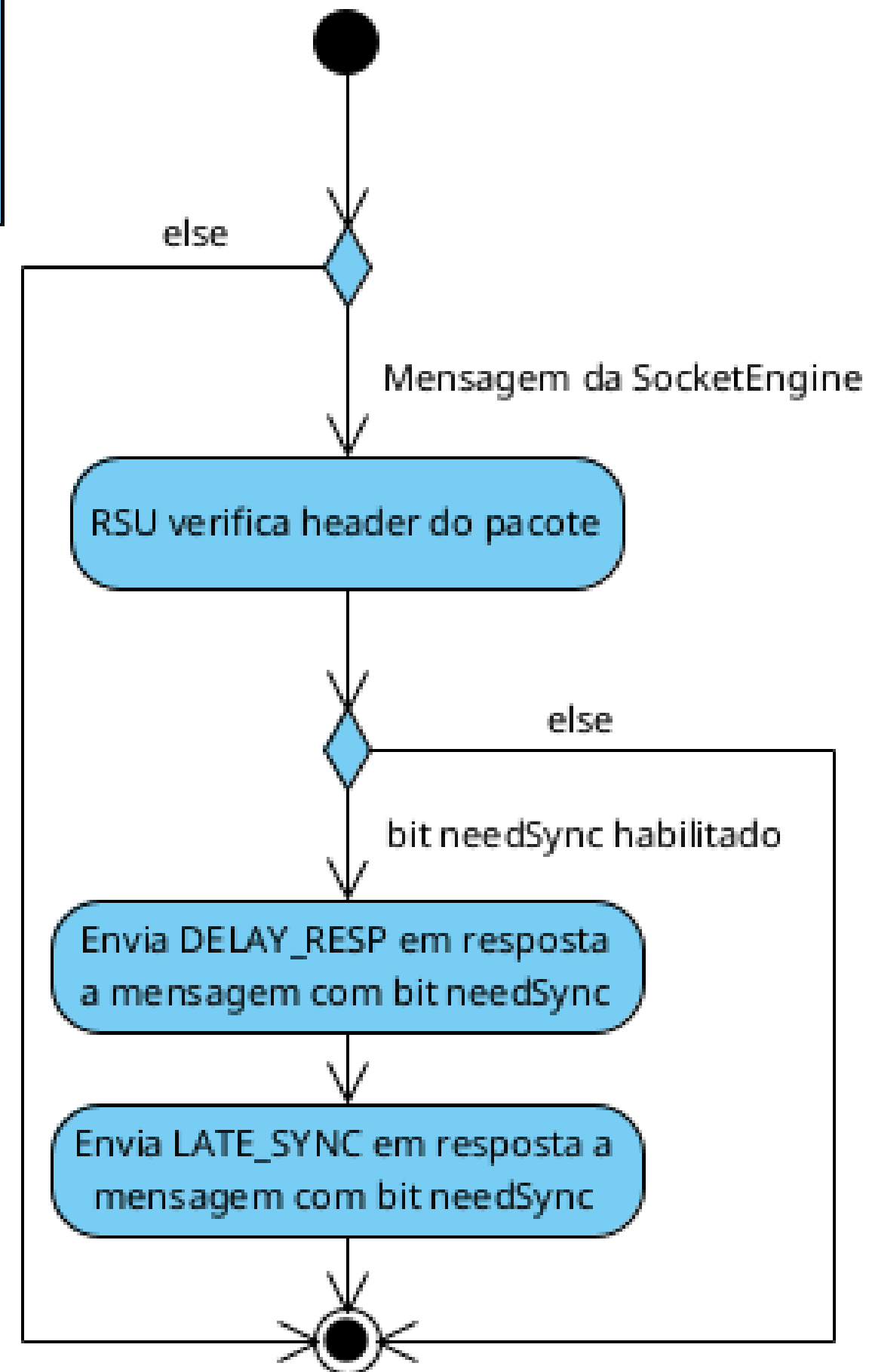
Desmarca o buffer, liberando-o

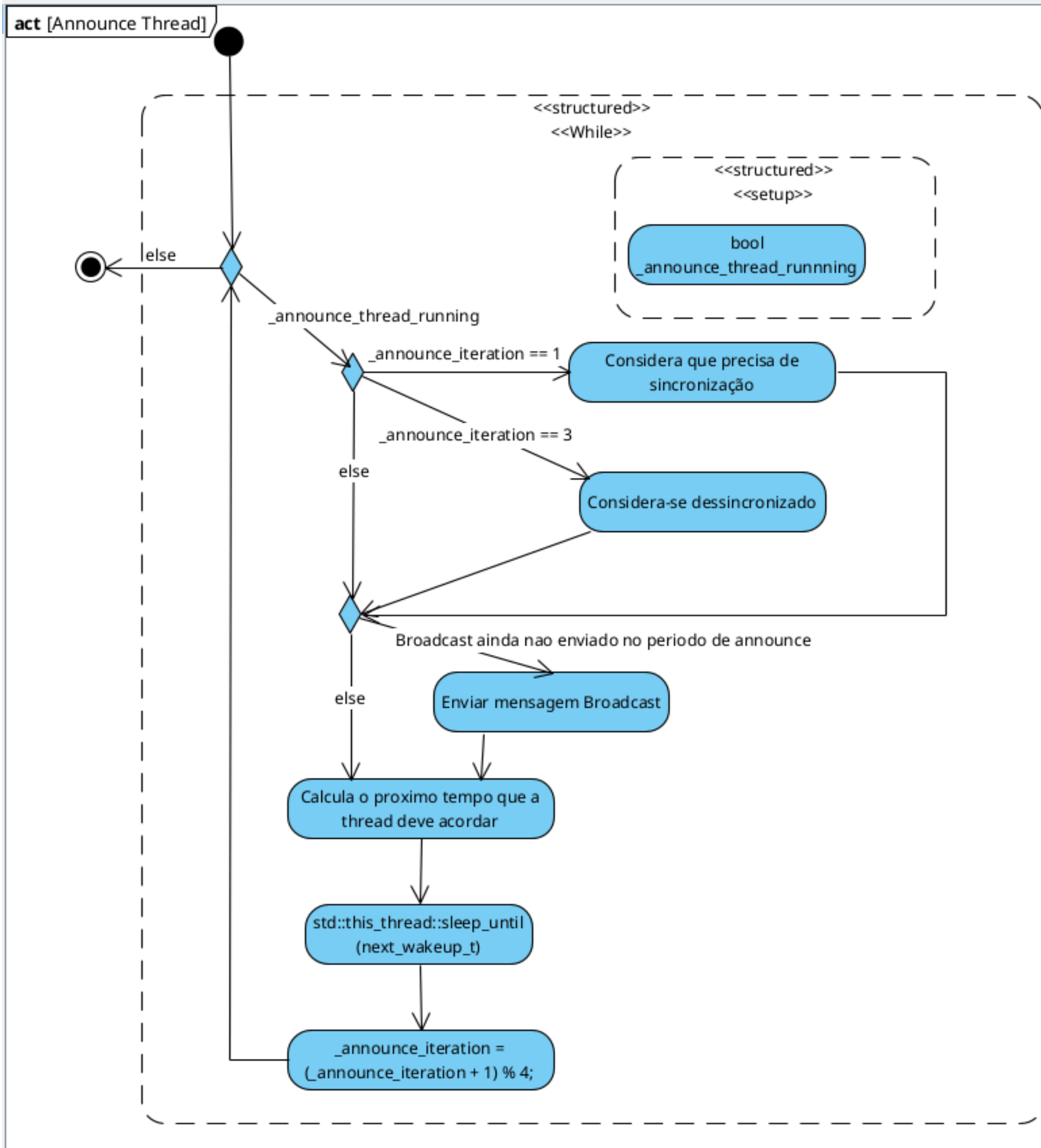






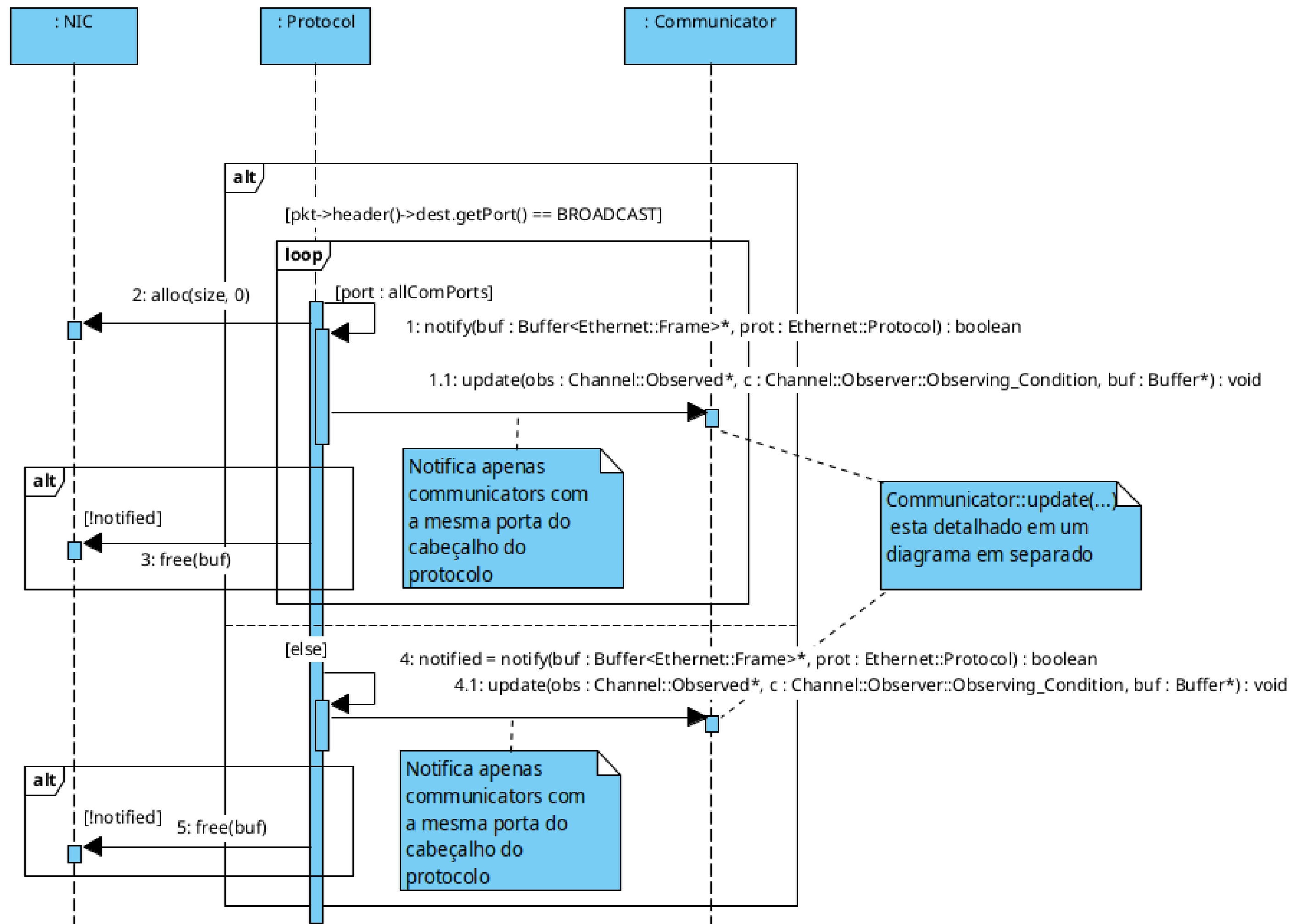
O código relacionado a esse fluxo está presente em RSUProtocol::update(...) e acontece durante o recebimento de uma mensagem.





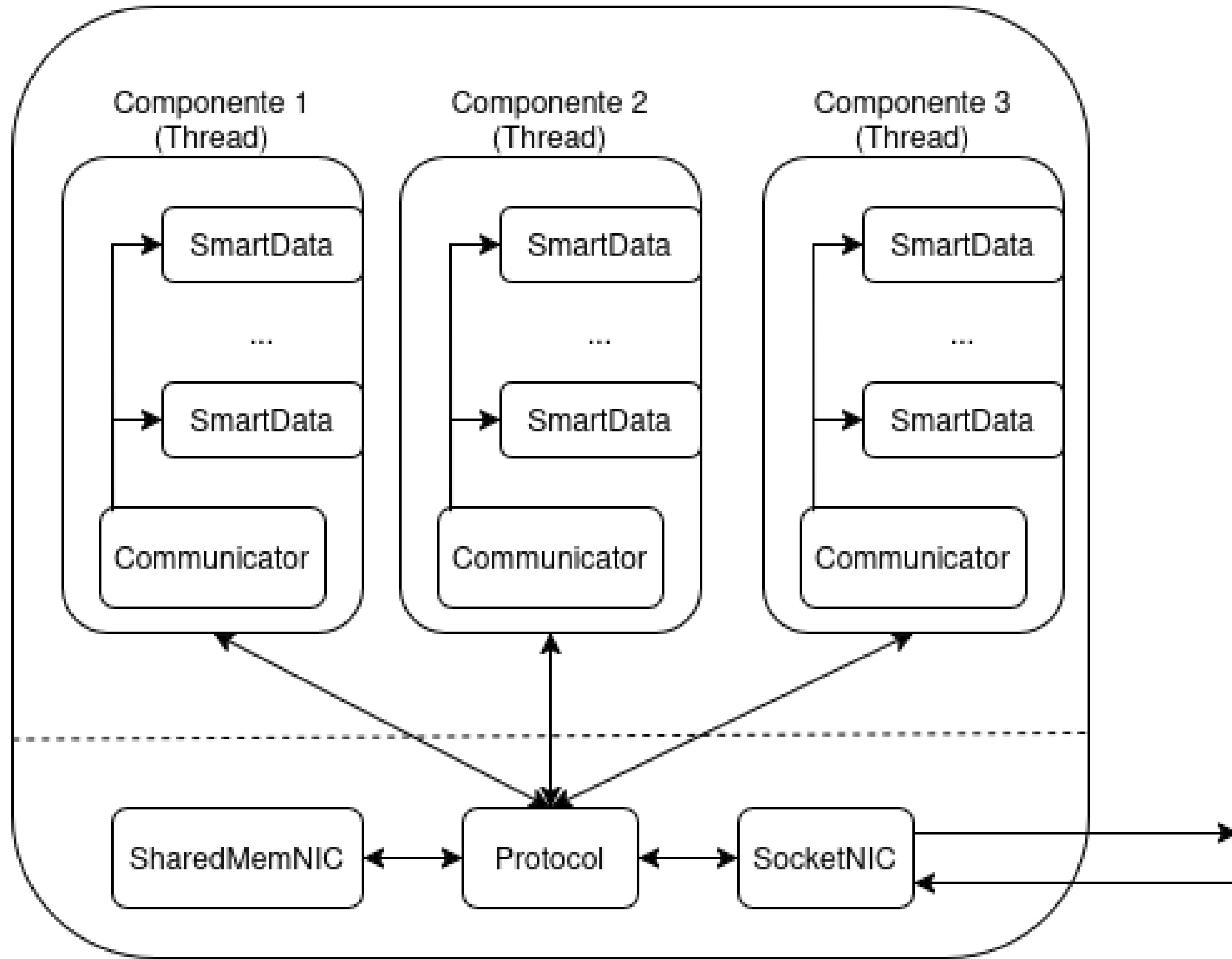


sd [handlePacket]

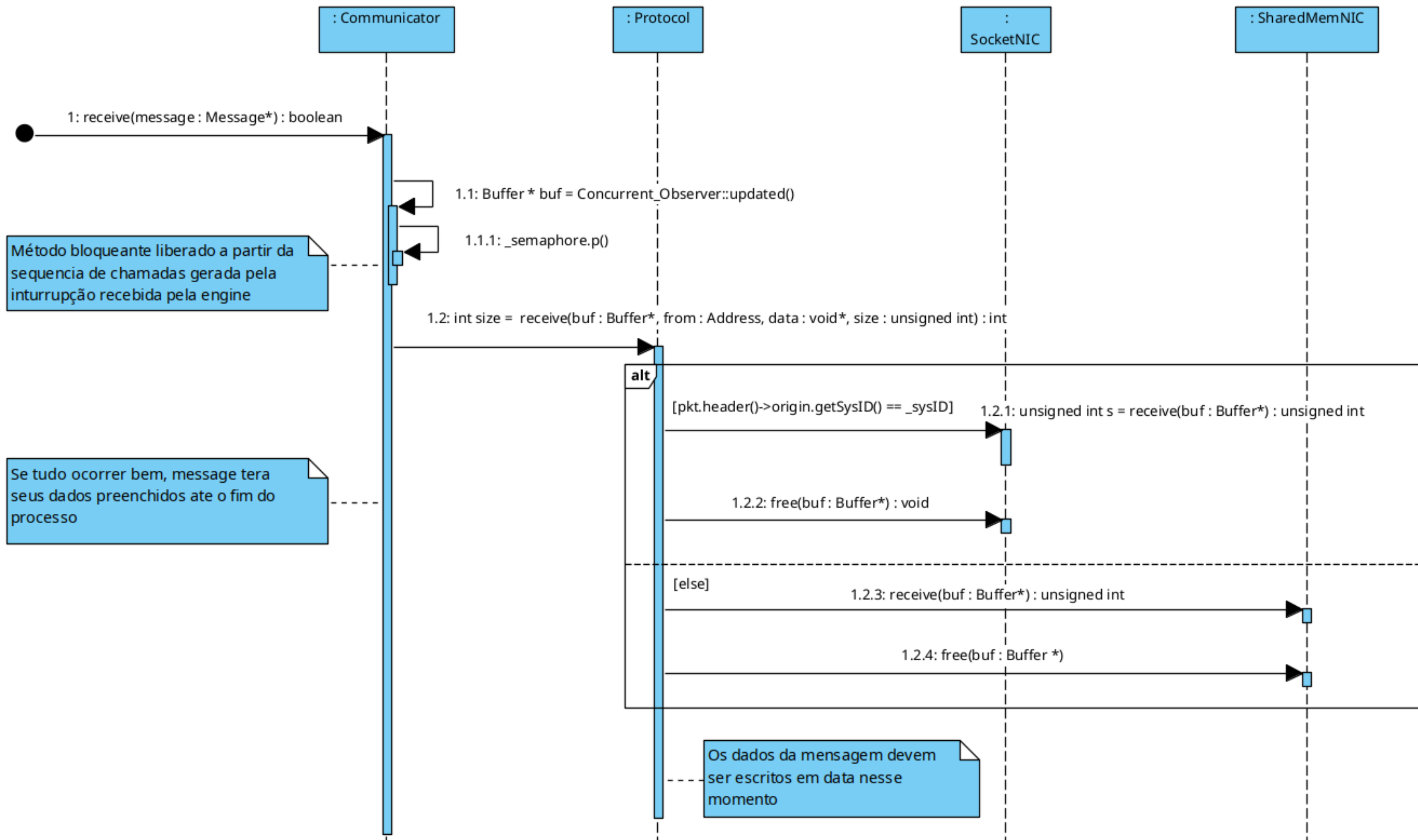


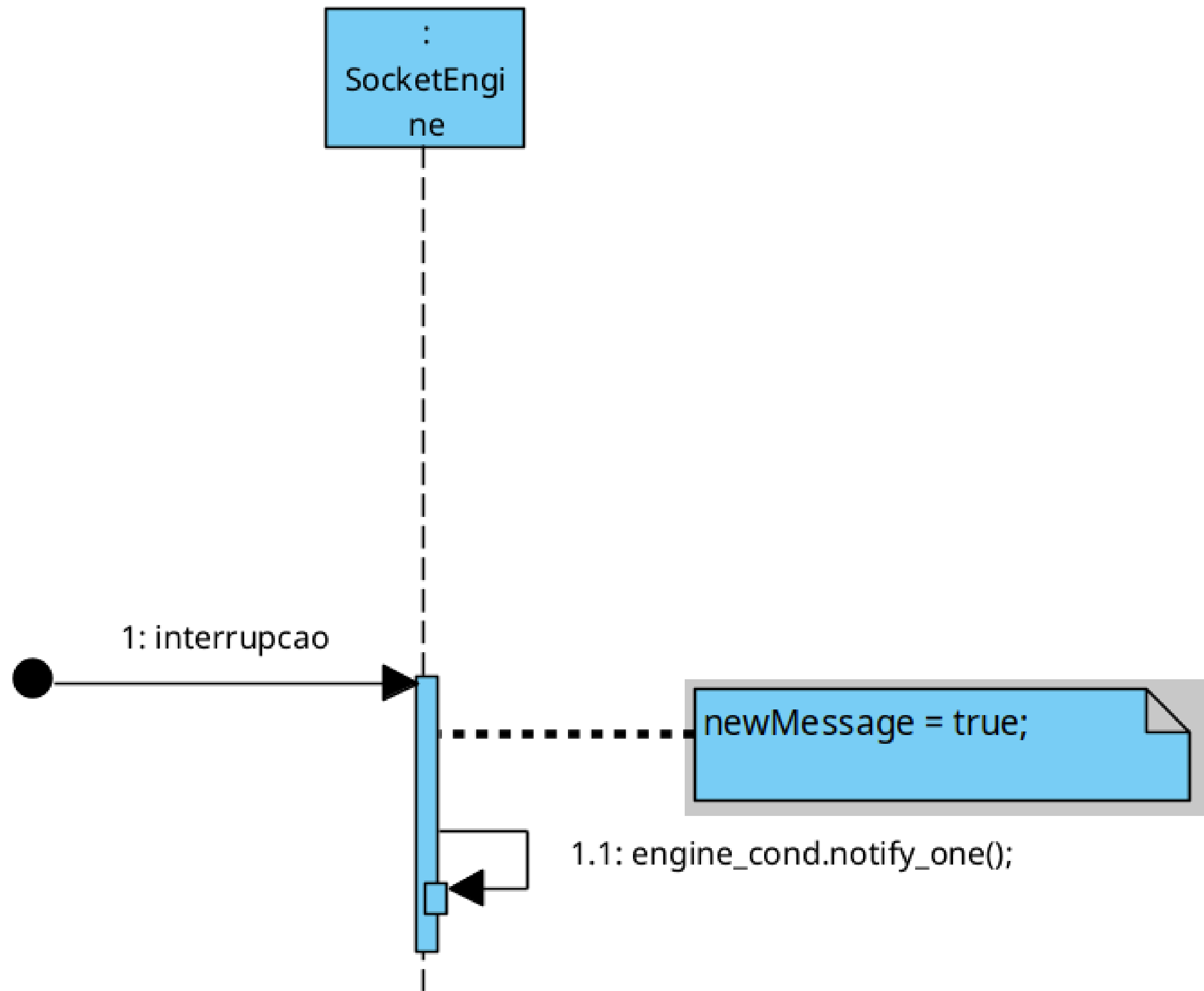
# Estrutura do Veículo

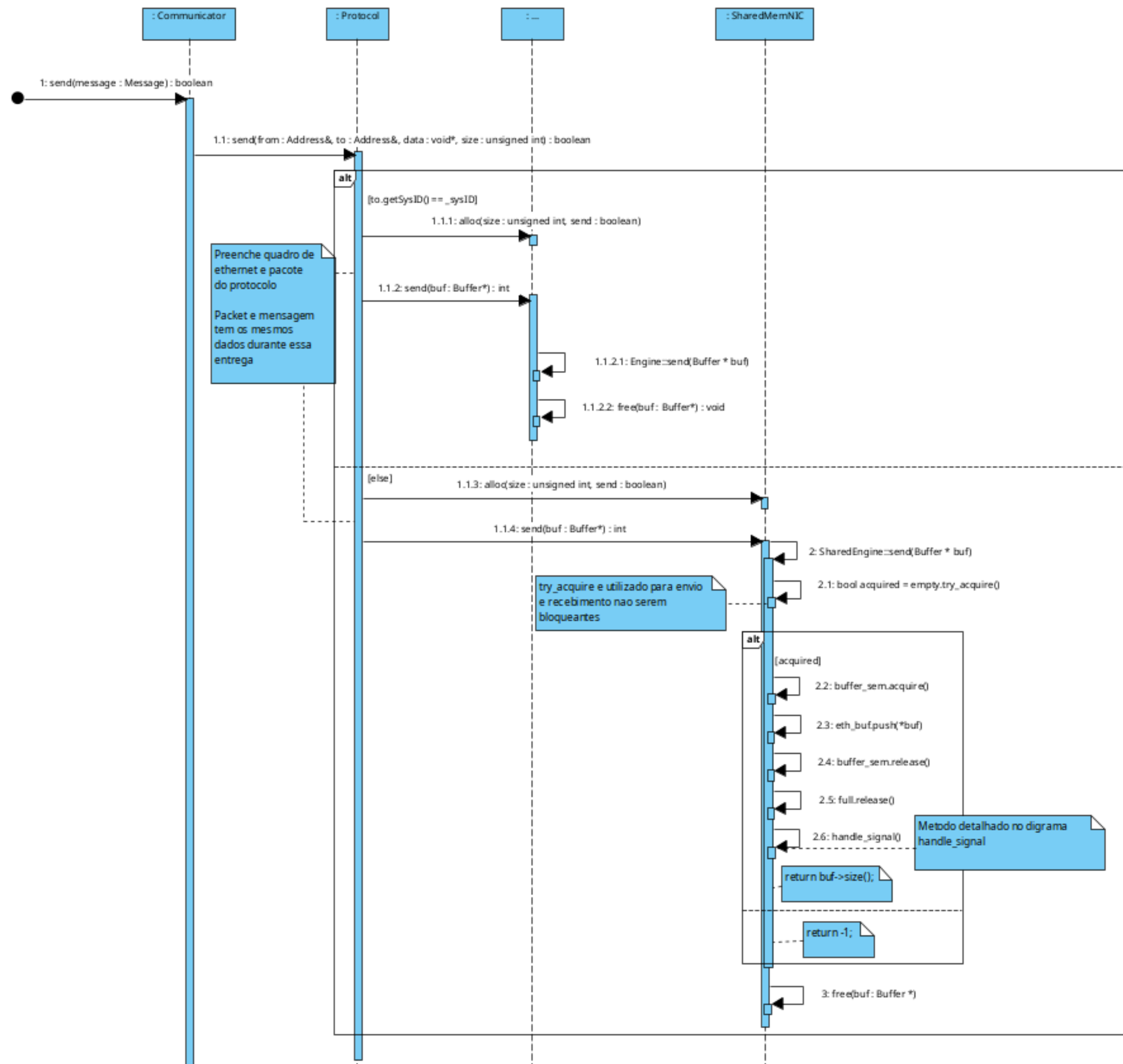
Veículo (Processo)

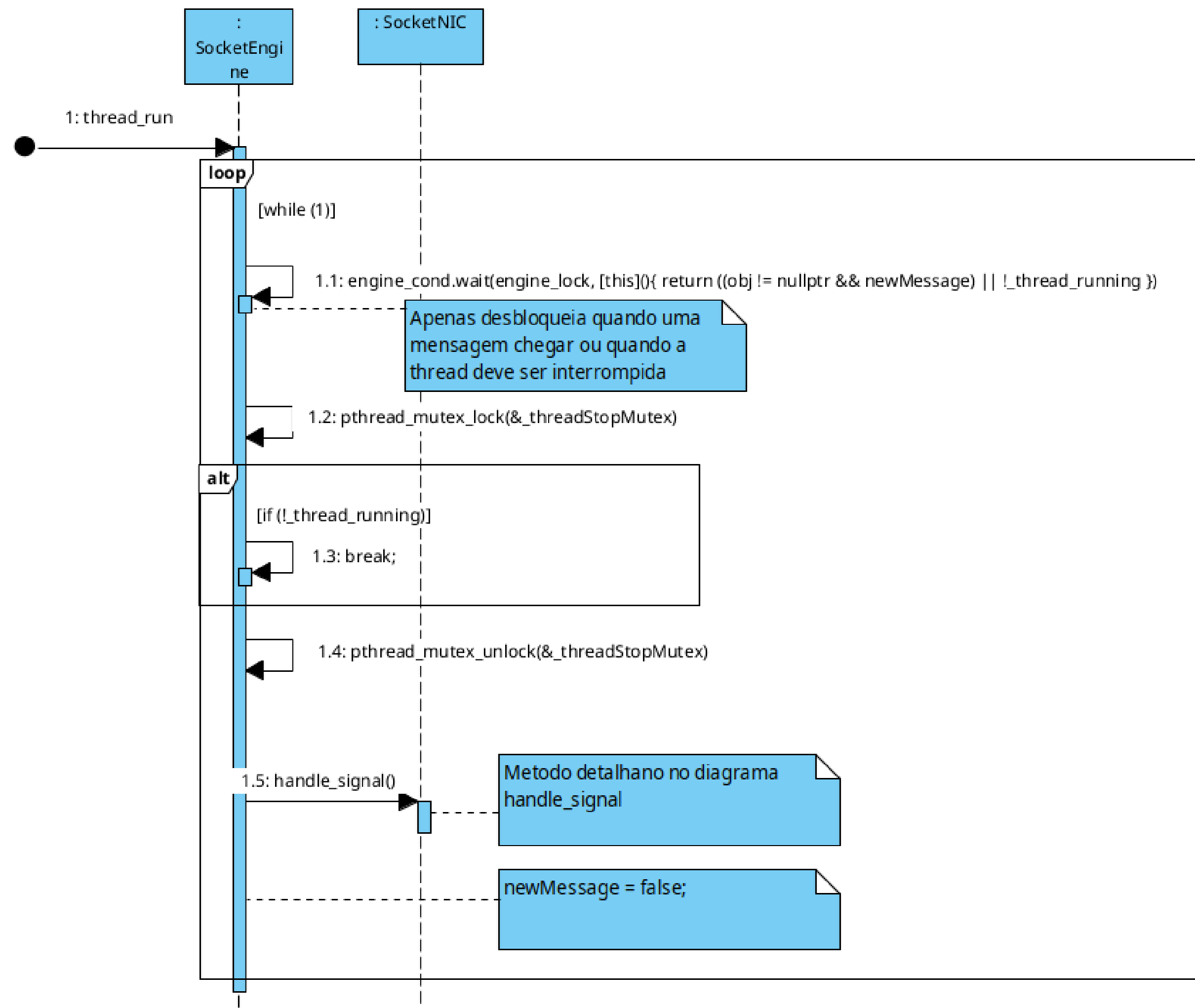


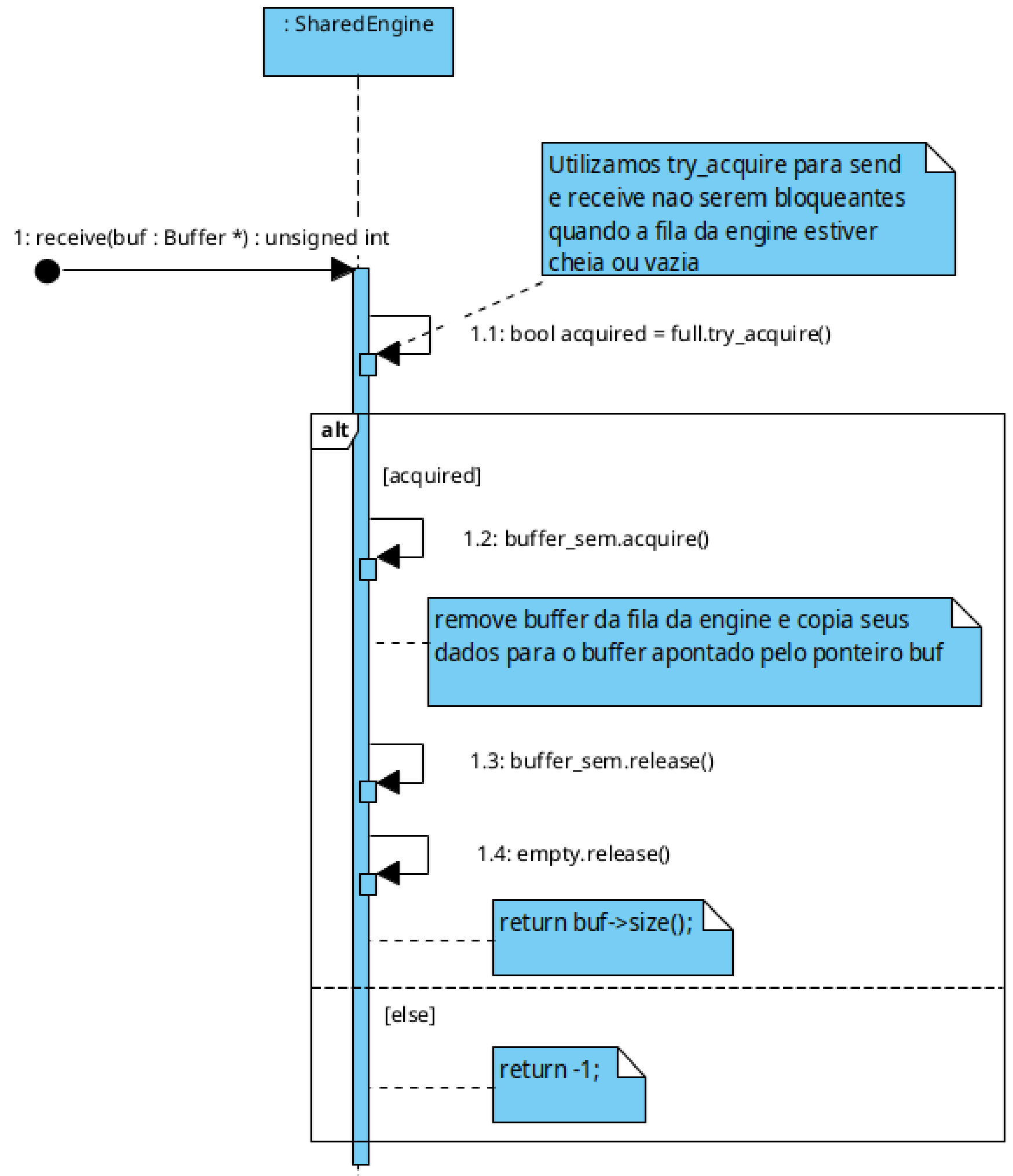
# Diagramas de Sequência



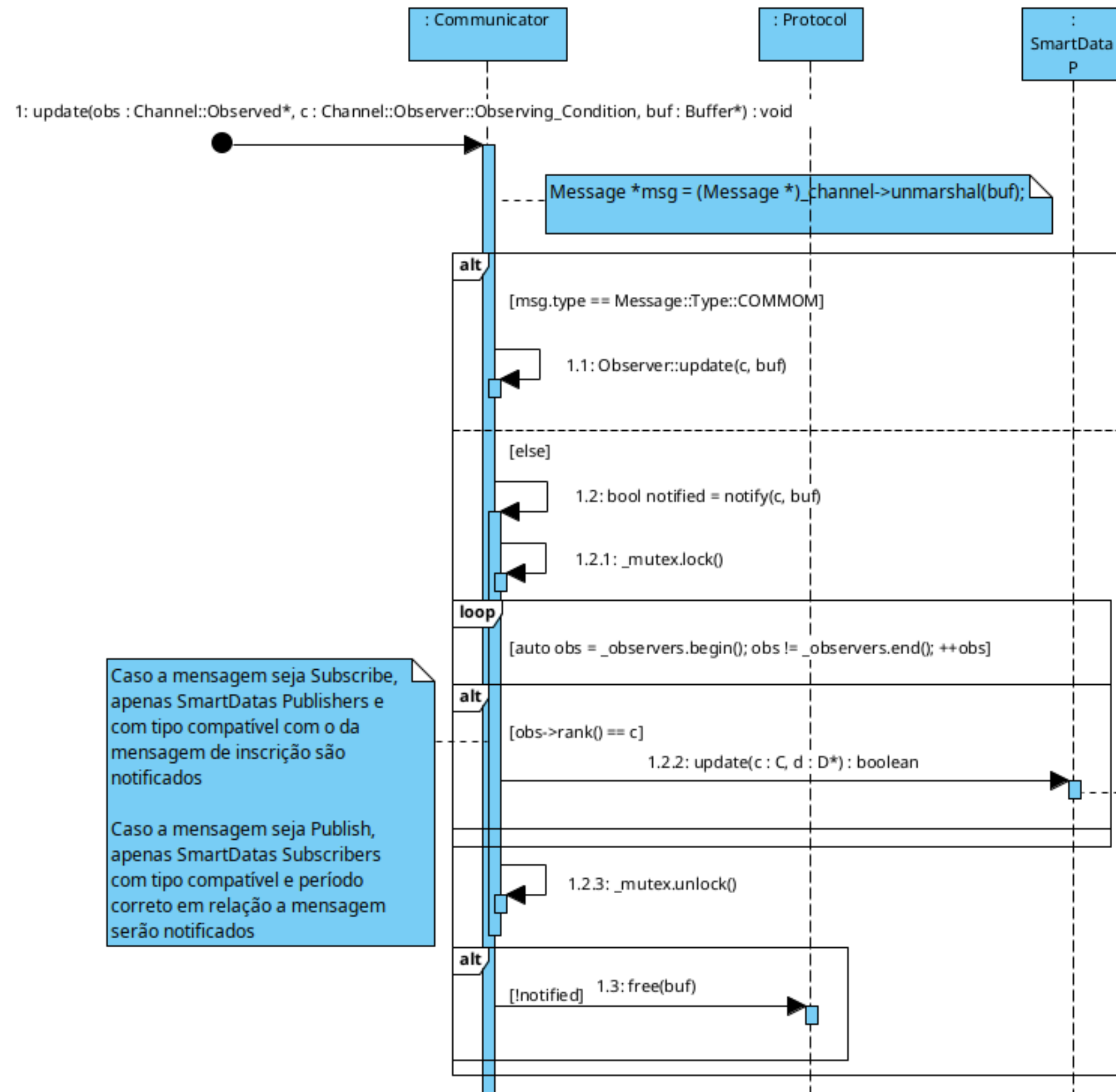












Caso a mensagem seja Subscribe, apenas SmartDatas Publishers e com tipo compatível com o da mensagem de inscrição são notificados

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))  
}
```

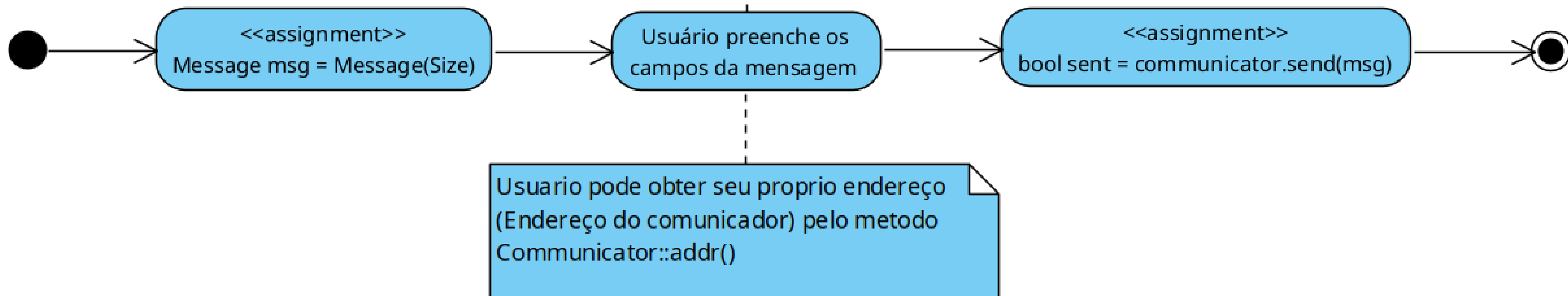
Endereços especiais:

0 = Broadcast\_SysID

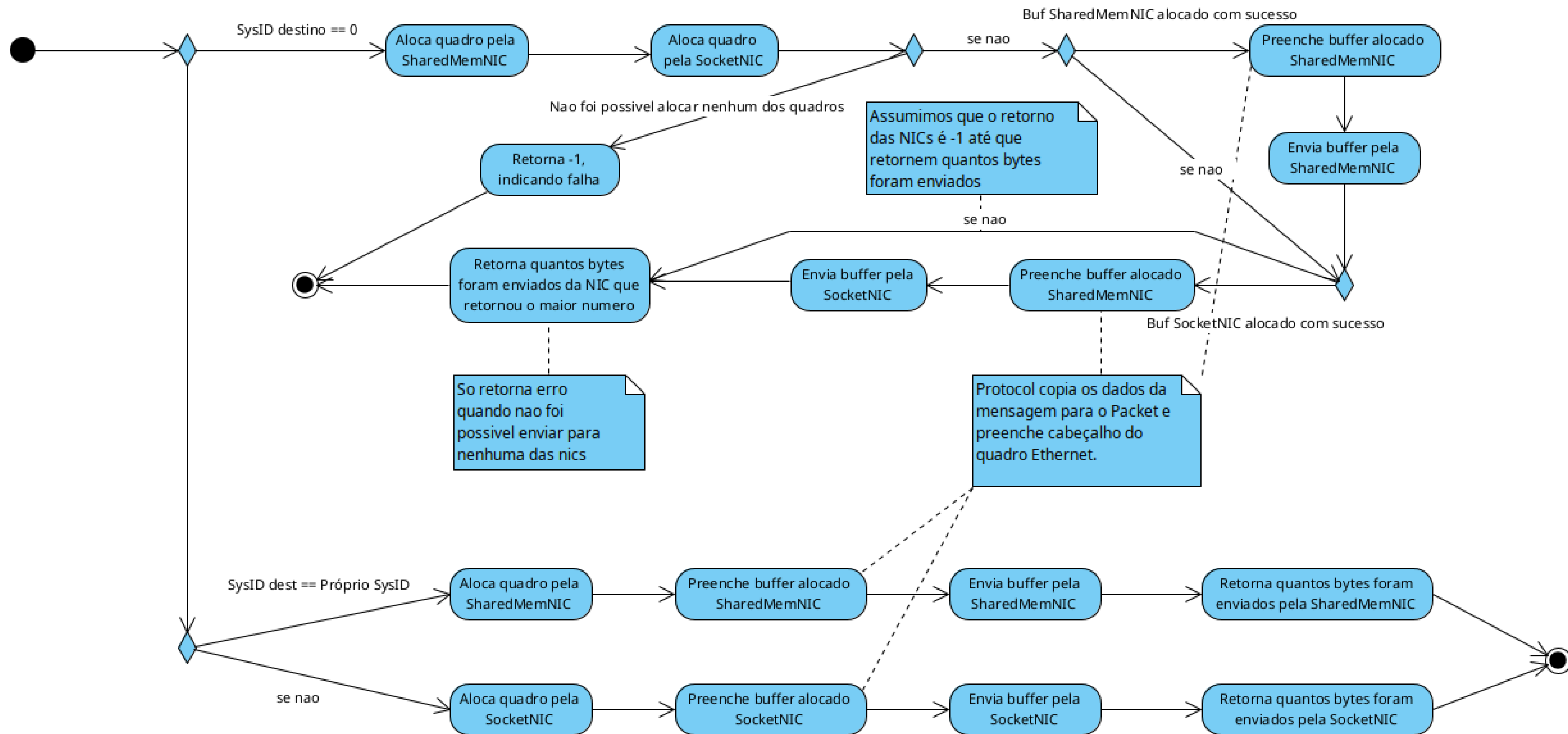
Caso uma mensagem seja enviada  
com o SysID 0, todos os veículos  
(Processos) alcançáveis dentro da  
rede recebem a mensagem.

0xFFFF = Broadcast\_Port

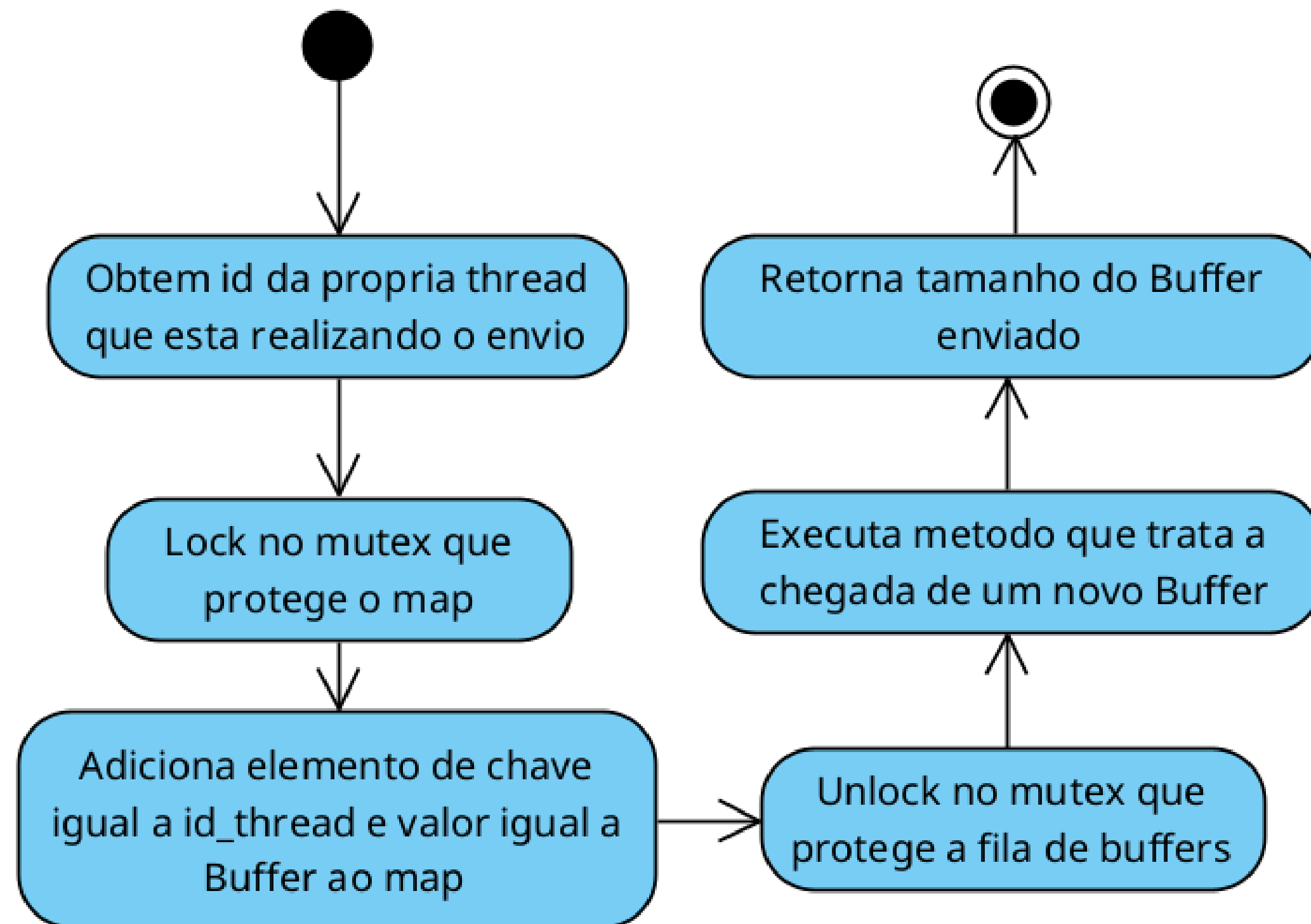
Caso uma mensagem seja enviada  
com Port 0xFFFF, todos os  
componentes(Threads) de um  
veículo (Processo) recebem a  
mensagem.

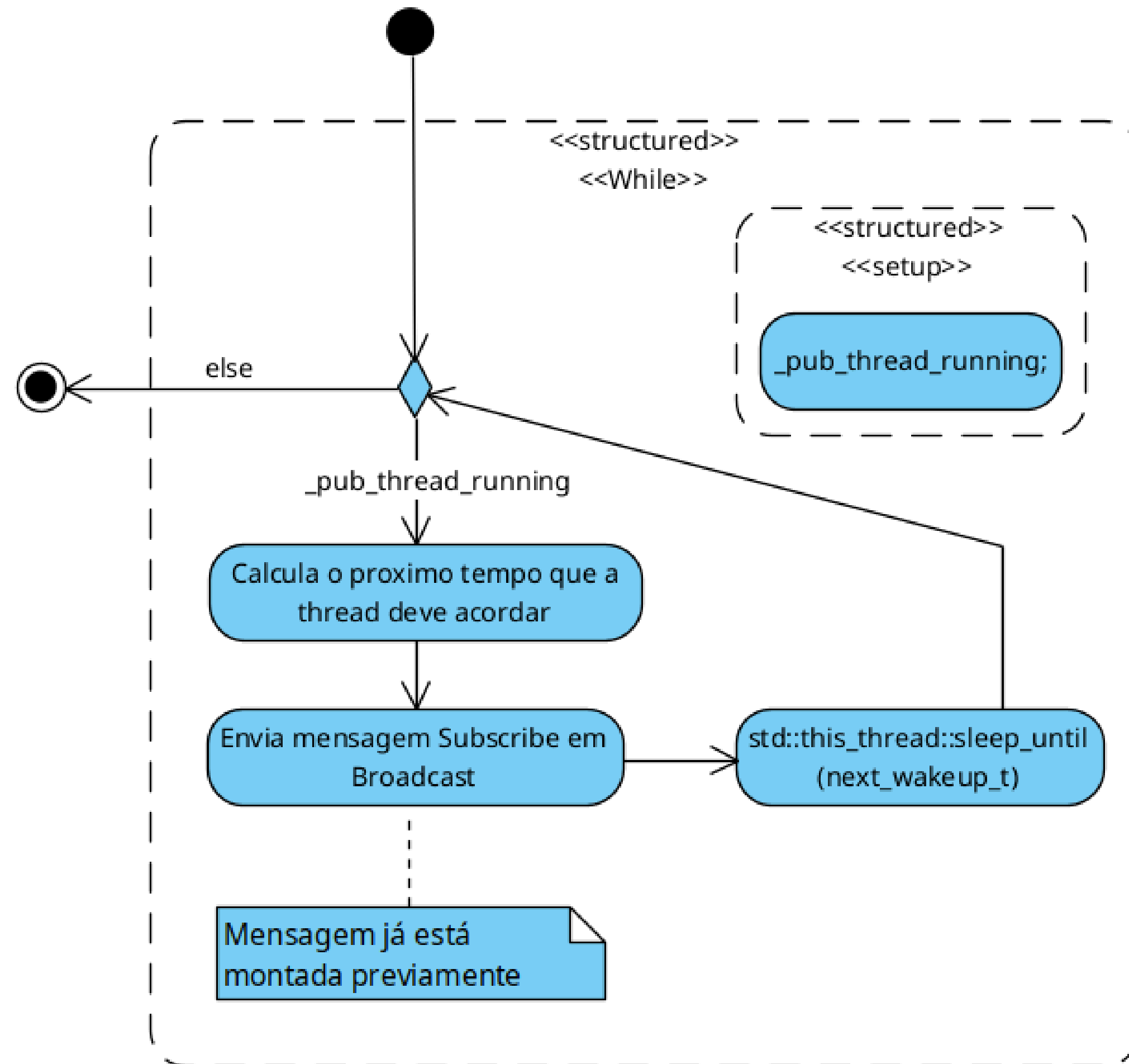


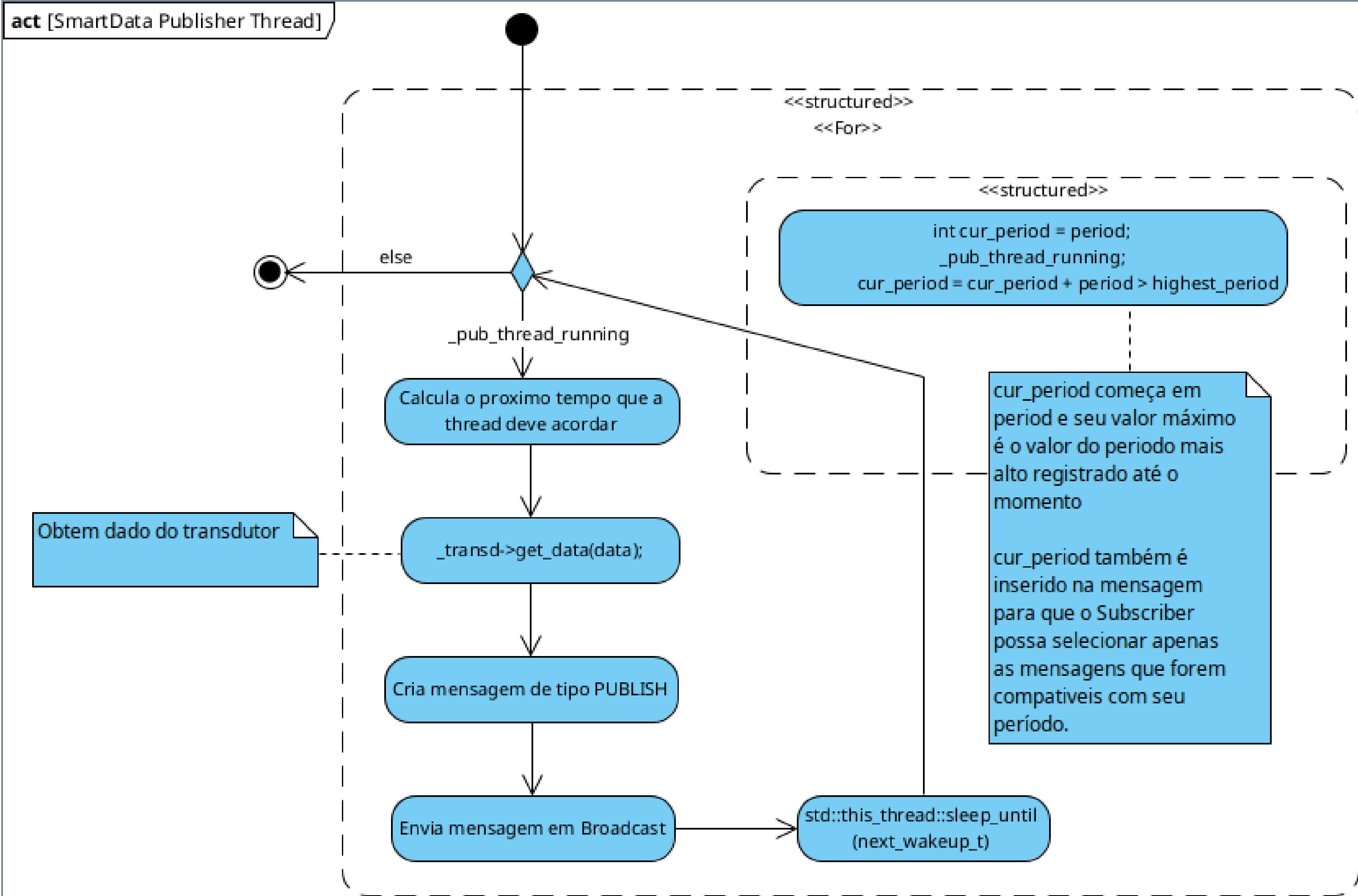
act [protocol.send(...)]



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



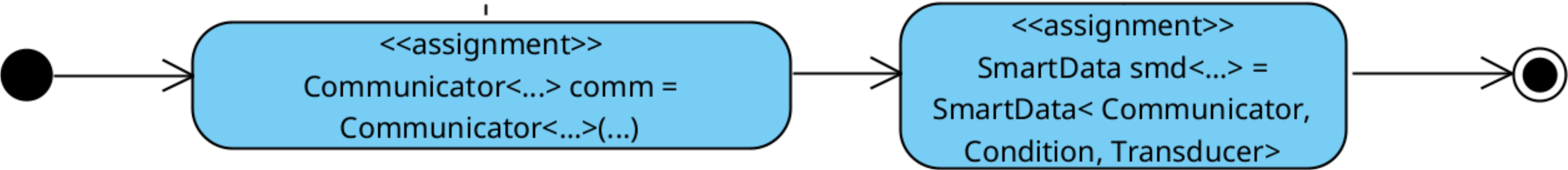




**act** [User View Publish]

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.

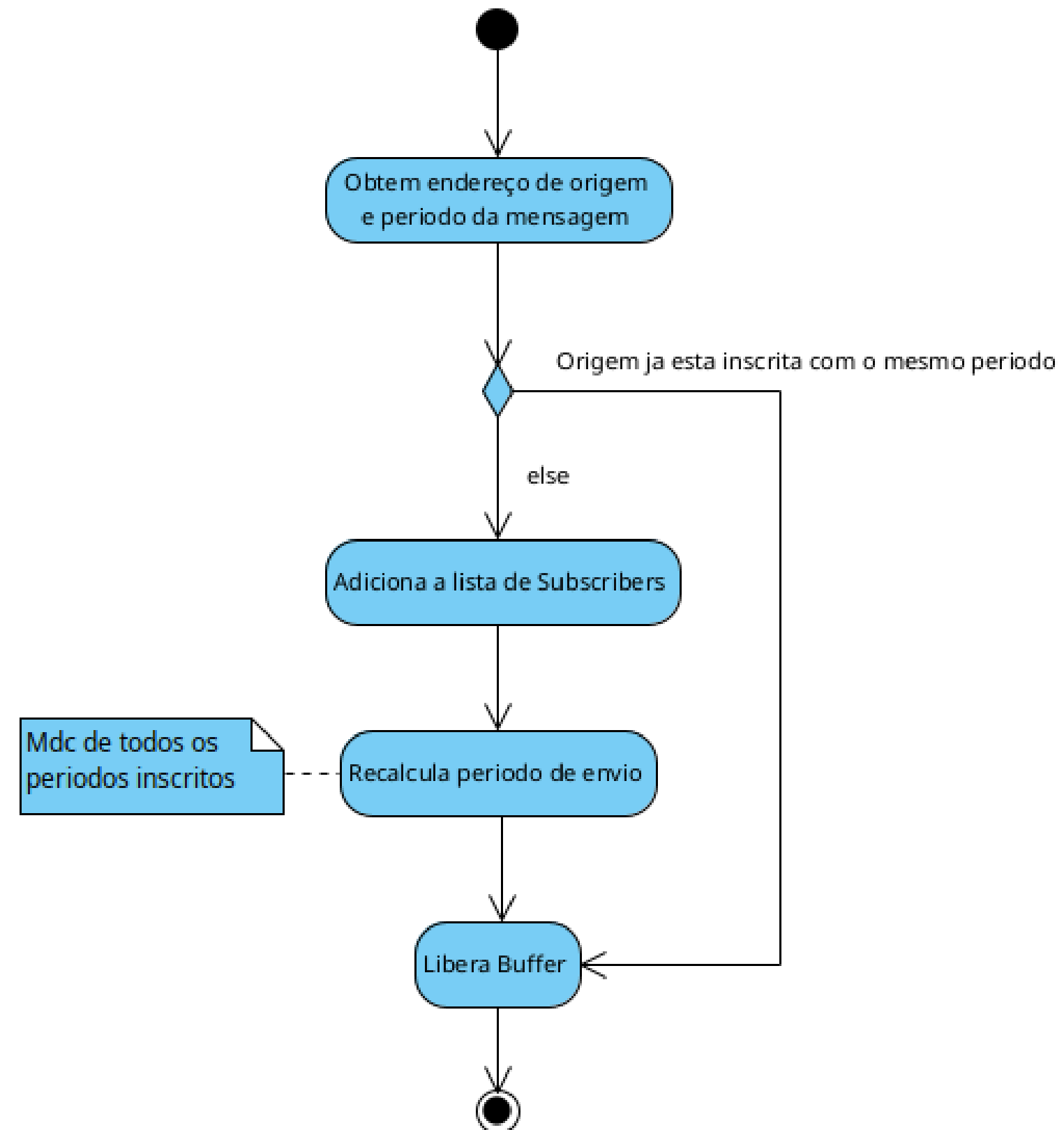
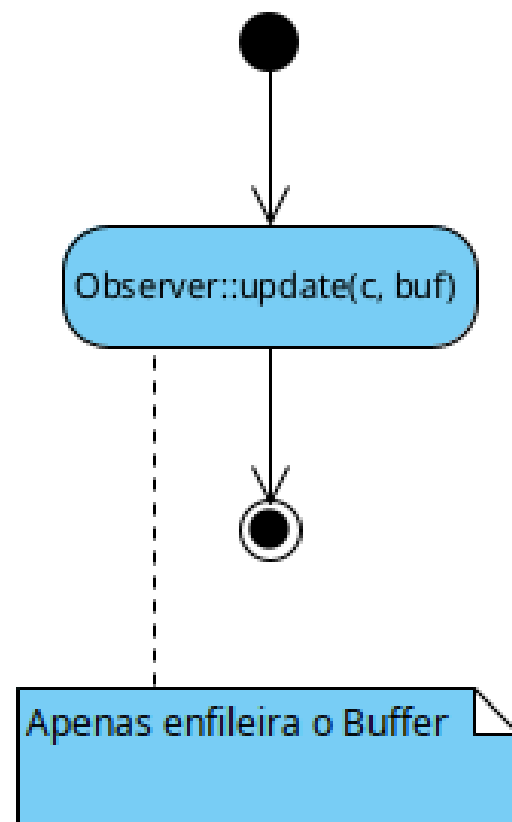


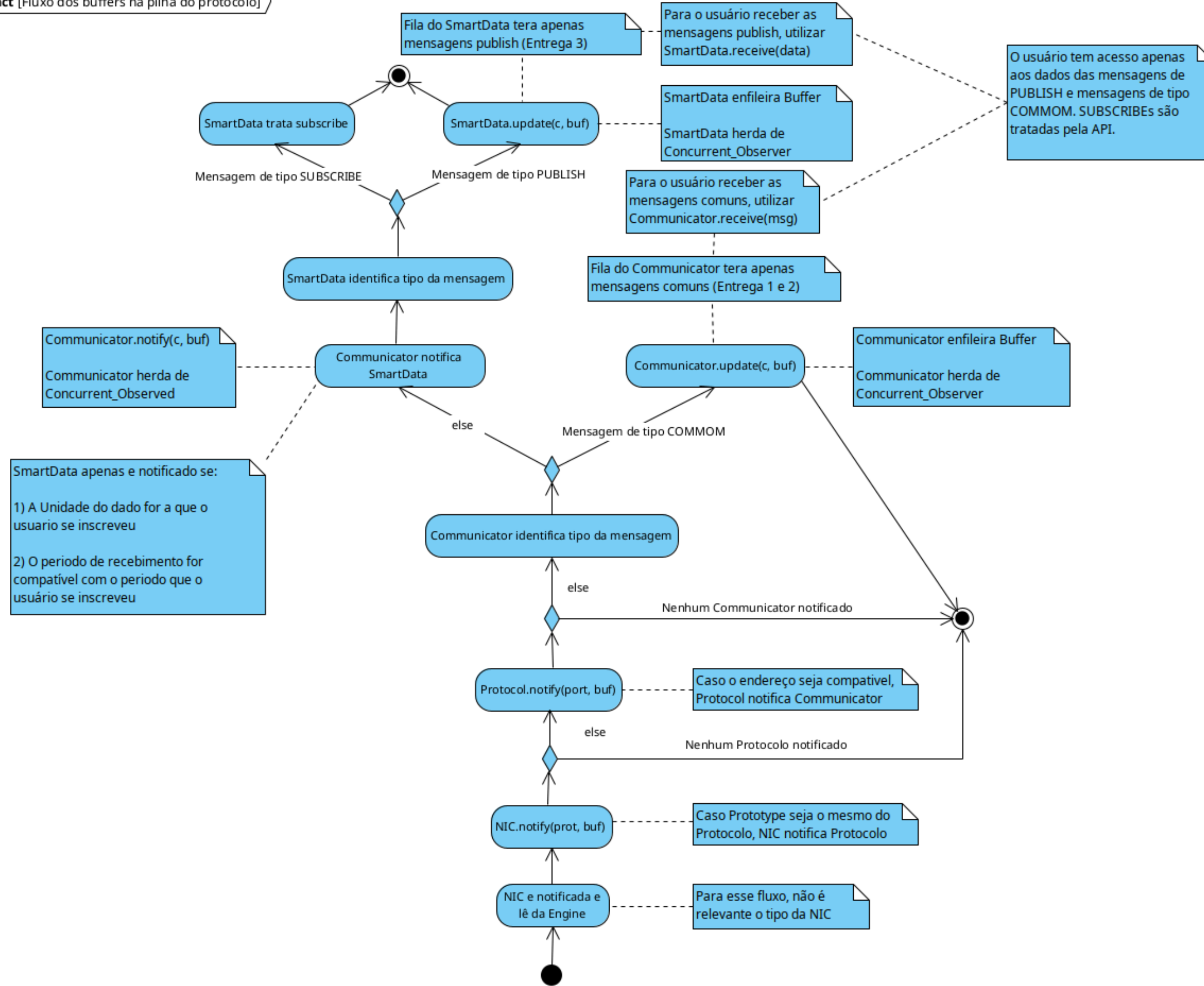


act [SmartData.update(...)]

Subscriber SmartData

Publisher SmartData





# Sincronização Temporal: Detalhes do Cálculo

O cálculo do offset no veículo depende da correlação de quatro timestamps.

- **Timestamps:**
  - **T1:** Tempo de envio da LATE\_SYNC pela RSU.
  - **T2:** Tempo de recebimento da LATE\_SYNC pelo veículo.
  - **T3:** Tempo de envio da ANNOUNCE original pelo veículo.
  - **T4:** Tempo de recebimento da ANNOUNCE pela RSU.
- **Mecanismo de Correlação:** O veículo, ao enviar ANNOUNCE, armazena T3. A RSU inclui T3 no payload de suas respostas. Ao receber LATE\_SYNC e DELAY\_RESP, o veículo usa o T3 do payload para consultar T4 em um mapa interno.
- **Cálculo:** Com os quatro valores, as fórmulas padrão do PTP são aplicadas para calcular o atraso de propagação e o offset. O offset é então aplicado ao relógio local.

# Comunicação Segura: Coordenação de RSUs

A geração e distribuição sincronizada de chaves MAC entre as RSUs (processos distintos) é realizada com memória compartilhada.

- **Estrutura SharedData:** Um segmento de memória compartilhada contém:
  - **pthread\_mutex\_t:** Para serializar o acesso ao contador e à matriz de chaves.
  - **pthread\_barrier\_t:** Para sincronizar a execução das threads de todas as RSUs.
  - Um contador de ciclo para a renovação das chaves.
  - Uma matriz que armazena a chave MAC de cada RSU.
- **Fluxo de Sincronização:**
  - Cada RSU gera sua chave e a escreve em sua posição na matriz compartilhada.
  - Todas as RSUs bloqueiam na barreira.
  - Somente após todas as RSUs passarem pela barreira, elas prosseguem para ler as chaves de sua vizinhança da memória compartilhada e distribuí-las em broadcast.
- **Justificativa:** O uso de uma barreira POSIX garante que nenhuma RSU distribua chaves antigas enquanto outras já geraram as novas, mantendo a consistência do sistema de chaves em toda a rede.