Entrega 3
INE-5424

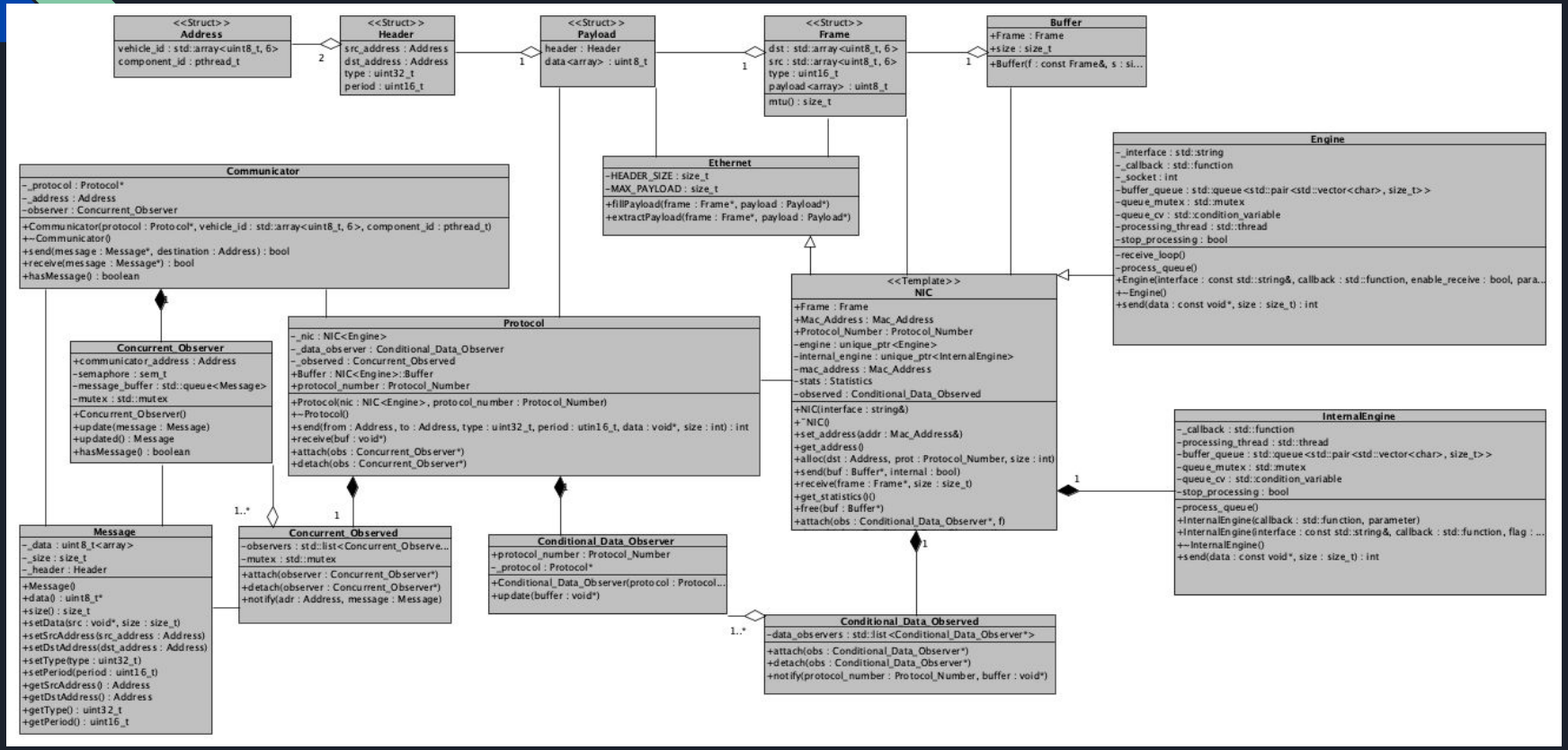
Header e Endereço dos Componentes

Mudanças Realizadas:

- Adição de dois novos campos ao cabeçalho da mensagem para dar suporte às mensagens de Interesse e Resposta:
  - Type: identifica o tipo de dado da mensagem.
  - Period: armazena o intervalo desejado (em milissegundos).

- Remoção do campo Porta dos endereços dos componentes, por não ser mais necessário na identificação do tipo das mensagens.

Diagrama de Classes



Mensagem de Interesse

Diferença da Mensagem de Interesse:

- Sem identificador de destino
  → A mensagem de interesse não especifica o componente de destino.

- Broadcast interno no veículo
  → Ao receber uma mensagem de interesse, o veículo repassa internamente para todos os seus componentes.

- Filtragem por tipo
  → Cada componente verifica o tipo da mensagem de interesse e decide se deve responder, conforme seu tipo.

Mensagem de Interesse

Exemplo envio:

```
// Prepara mensagem de interesse para o sensor temperatura.
mensagem.setDstAddress({dados->id_veiculo, (pthread_t)0});  // Preenche endereço de destino
mensagem.setType(TIPO_SENSOR_TEMPERATURA);                  // Preenche tipo do dado

// Preenche periodo de interesse.
int periodo = PERIODO_MIN + (std::rand() % (PERIODO_MAX - PERIODO_MIN + 1));
mensagem.setPeriod(periodo);
// Envia mensagem de interesse.
comunicador.send(&mensagem);
std::cout << "📫 " << dados->nome << ": enviou interesse para o sensor temperatura com periodo: " << periodo << std::endl;
```

Exemplo recebimento:

```
comunicador.receive(&mensagem);

// Verifica se a mensagem eh de interesse (nao preencheu id componente no endereco de destino).
if (pthread_equal(mensagem.getDstAddress().component_id, (pthread_t)0)) {
    // Verifica se a mensagem de interesse eh para ele.
    if (mensagem.getType() == TIPO_SENSOR_GPS) {
        // Responde a mensagem.
        mensagem.setDstAddress(mensagem.getSrcAddress());
        mensagem.setData(reinterpret_cast<DadosSensorGPS*>(&posicao), sizeof(DadosSensorGPS));
        comunicador.send(&mensagem);
```

Mensagem de Resposta

Diferença da Mensagem de Resposta:

- Possuem identificador de destino
  → A mensagem de resposta especifica o componente de destino.

- Logo a mensagem é encaminhada para um destino específico.
  → Não realiza broadcast interno. Apenas o componente com endereço de destino recebe.

- Não utilizam o campo Período, porém não é possível utilizar isso para diferenciar uma mensagem de resposta, já que pode existir interesses com periodo = 0 (interesse em respostas unicas).

Mensagem de Resposta

Exemplo envio:

```
// Responde a mensagem.
mensagem.setDstAddress(mensagem.getSrcAddress());
mensagem.setType(TIPO_SENSOR_GPS);
mensagem.setData(reinterpret_cast<DadosSensorGPS*>(&posicao), sizeof(DadosSensorGPS));
comunicador.send(&mensagem);
```

Exemplo recebimento:

```
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() == TIPO_SENSOR_GPS) {
        num_respostas++; // incrementa numero de respostas.
        // Extrai dado recebido.
        DadosSensorGPS posicao = *reinterpret_cast<DadosSensorGPS*>(mensagem.data());
```

Interesse vs Resposta

A diferença entre as mensagens de Interesse e Resposta é:

- Na mensagem de Interesse, o identificador do componente do endereço de destino é 0, significando que não está direcionado a um componente específico.

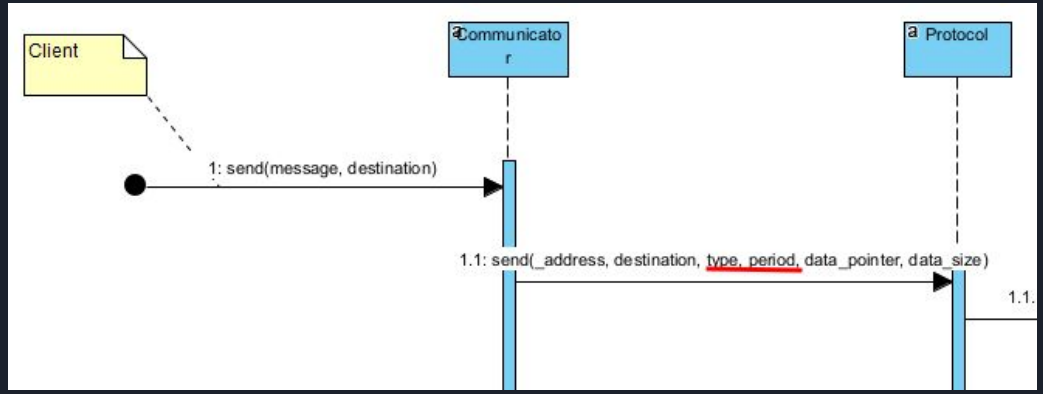
- Na mensagem de Resposta, o identificador do componente do endereço de destino é != 0, significando que a mensagem é direcionada a um componente específico.

Envio de Mensagens

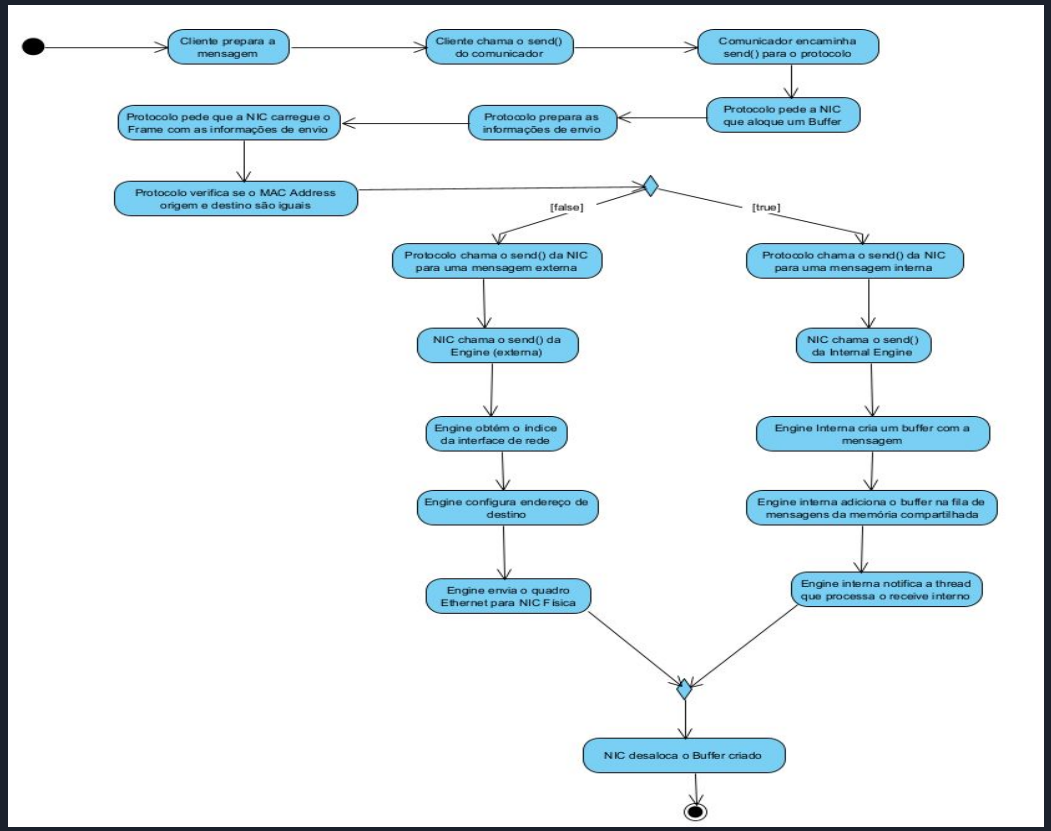
```
Message mensagem;

// Prepara mensagem de interesse para o sensor temperatura.
mensagem.setDstAddress({dados->id_veiculo, (pthread_t)0});  // Preenche endereço de destino
mensagem.setType(TIPO_SENSOR_TEMPERATURA);                  // Preenche tipo do dado

// Preenche periodo de interesse.
int periodo = PERIODO_MIN + (std::rand() % (PERIODO_MAX - PERIODO_MIN + 1));
mensagem.setPeriod(periodo);
// Envia mensagem de interesse.
comunicador.send(&mensagem);
```

Componente prepara a mensagem a ser enviada, definindo tipo e período

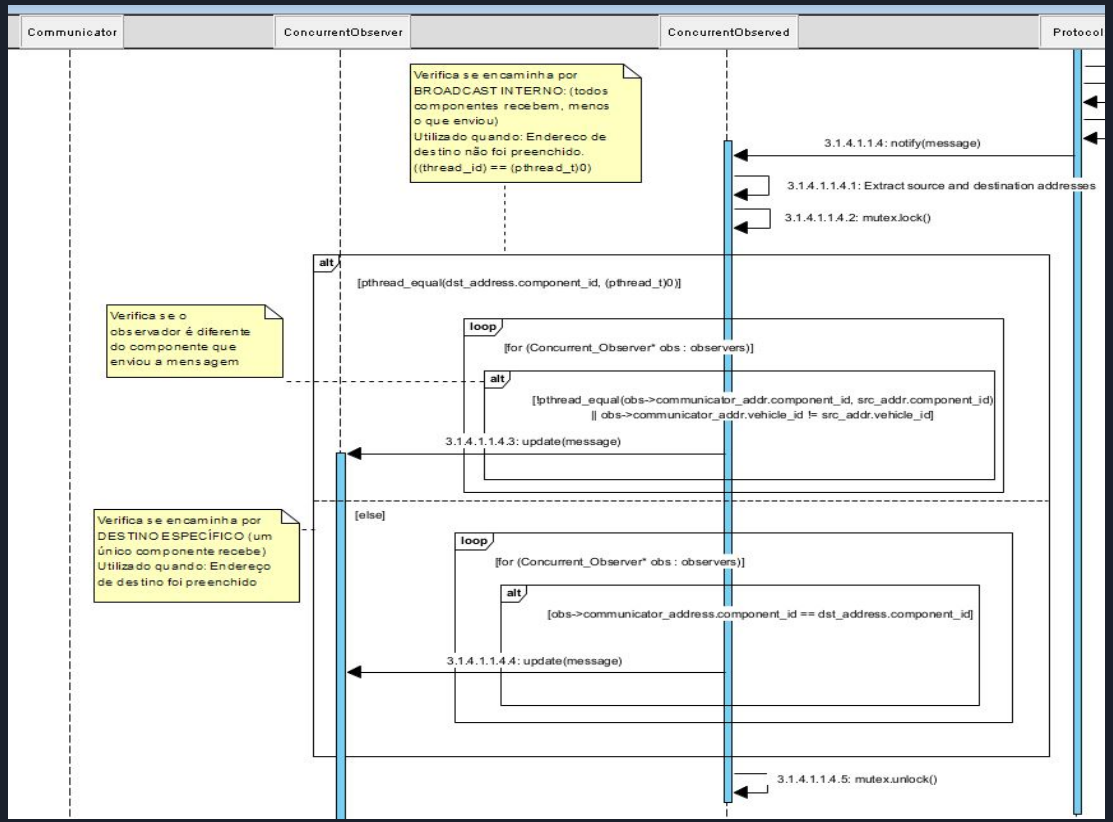
Componente ativa send do communicator, propagando a mensagem até seu envio externo ou interno



Envio de Mensagens



Recebimento de Mensagens

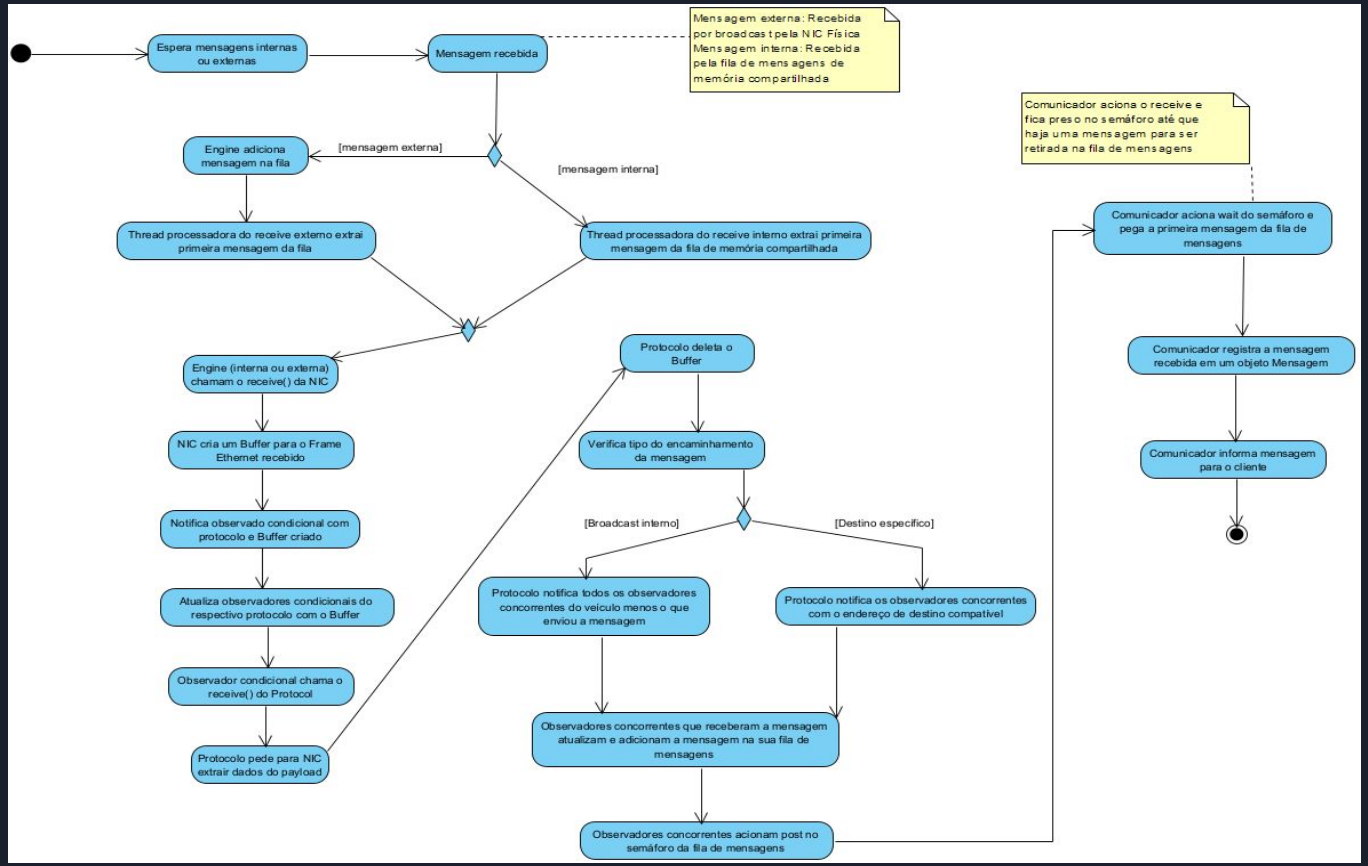


Concurrent observed verifica qual tipo de encaminhamento deve ser utilizado para notificar observadores

Quando o endereço de destino não foi preenchido, propaga o broadcast interno

Caso contrário, notifica apenas os observadores com destino específico

Recebimento de Mensagens



Classe Agendador

Objetivo:

Centralizar o controle dos períodos de envio de mensagens entre os componentes, evitando que cada um implemente sua própria lógica de temporização.

Como funciona:

- Cada componente registra seu interesse em ser notificado após um determinado intervalo de tempo.
- O Agendador executa uma thread dedicada que permanece em loop, monitorando os registros e notificando os componentes assim que seus períodos forem atingidos.
- Essa thread utiliza um conjunto ordenado pelo tempo do próximo envio, garantindo que ela acorde apenas no momento exato em que uma notificação precisa ser feita.

```cpp
// Loop principal da thread para controlar os envios periódicos
void Agendador::loop() {
    std::unique_lock<std::mutex> lock(mutex_);

    while (!stop_) {
        if (periodos_.empty()) {
            cv_.wait(lock);  // Espera até que haja algum período registrado
            continue;
        }

        // Pega o período que deve ser executado primeiro
        auto it = periodos_.begin();
        Periodo p = *it;
        auto agora = Clock::now();

        if (agora >= p.proximo_envio) {
            // Remove período da lista temporariamente
            periodos_.erase(it);

            // Marca o destino como pronto para o componente
            auto& atingidos = componentes_registrados_[p.componente_id];
            {
                std::lock_guard<std::mutex> lock_atingidos(atingidos.mutex);
                atingidos.destinos_prontos.push_back(p.destino);
                atingidos.pronto.store(true); // sinaliza que tem destino pronto
            }

            // Armazena estatística de tempo de envio
            estatisticas_[{p.componente_id, p}].tempos.push_back(agora);

            // Atualiza o próximo envio e reinserir período para reuso
            p.proximo_envio += p.periodo;
            periodos_.insert(p);

            continue;
        }

        // Espera até o próximo envio do período
        cv_.wait_until(lock, p.proximo_envio);
    }
}
```

Exemplo de uso do Agendador

Registro de periodos:

```
// Verifica se recebeu alguma mensagem.
if (comunicador.hasMessage()) {
    // Carrega mensagem recebida.
    comunicador.receive(&mensagem);
    // Verifica se a mensagem recebida eh de interesse (id componente nao foi preenchido).
    if (pthread_equal(mensagem.getDstAddress().component_id, (pthread_t)0)) {
        // Verifica se mensagem de interesse eh para ele.
        if (mensagem.getType() == TIPO_SENSOR_TEMPERATURA) {
            std::cout << "🦸 " << dados->nome << ": recebeu interesse." << std::endl;
            // Registra periodo no Agendador.
            dados->agendador->registrar_interesse(pthread_self(), mensagem.getSrcAddress(), mensagem.getPeriod());
```

Checagem de periodos atingidos:

```
// Verifica se algum periodo de resposta ja foi atingido.
if (dados->agendador->possui_periodos_atingidos(pthread_self())) {
    // Extrai endereco de destino dos periodos ja atingidos.
    std::vector<Ethernet::Address> destinos = dados->agendador->obter_destinos_prontos(pthread_self());

    for (const auto& destino : destinos) {
        // Prepara mensagem de resposta.
        mensagem.setType(TIPO_SENSOR_TEMPERATURA);
        mensagem.setDstAddress(destino);
        mensagem.setData(reinterpret_cast<char*>(&temperatura), sizeof(int));
        // Envia mensagem de resposta.
        comunicador.send(&mensagem);
```

Resultados teste utilizando Agendador

```
============================================================
🧪  TESTE: Reconhecimento e Comunicação entre componentes do mesmo veículo (interna)
------------------------------------------------------------
 Controladores requisitam dados aos Sensores de Temperatura:
============================================================

Parâmetros do teste:
 Interface de rede: enp0s1
 Número de controladores: 100
 Número de sensores: 10
 Número de respostas: 100
 Período mínimo (ms): 1
 Período máximo (ms): 100
```

```
=== ESTATÍSTICAS DO AGENDADOR ===
ID Registro: 1 | Periodo desejado: 33 ms | Média intervalos: 32.5051 ms | Diferença média: 0.494949 ms | Nº de notificações: 100
ID Registro: 2 | Periodo desejado: 73 ms | Média intervalos: 72.5051 ms | Diferença média: 0.494949 ms | Nº de notificações: 100
ID Registro: 3 | Periodo desejado: 44 ms | Média intervalos: 43.4646 ms | Diferença média: 0.535354 ms | Nº de notificações: 100
ID Registro: 4 | Periodo desejado: 35 ms | Média intervalos: 34.6061 ms | Diferença média: 0.393939 ms | Nº de notificações: 100
ID Registro: 5 | Periodo desejado: 42 ms | Média intervalos: 41.5354 ms | Diferença média: 0.464646 ms | Nº de notificações: 100
```

•
•
•

```
ID Registro: 995 | Periodo desejado: 9 ms | Média intervalos: 8.49495 ms | Diferença média: 0.505051 ms | Nº de notificações: 100
ID Registro: 996 | Periodo desejado: 35 ms | Média intervalos: 34.5152 ms | Diferença média: 0.484848 ms | Nº de notificações: 100
ID Registro: 997 | Periodo desejado: 41 ms | Média intervalos: 40.4848 ms | Diferença média: 0.515152 ms | Nº de notificações: 100
ID Registro: 998 | Periodo desejado: 31 ms | Média intervalos: 30.4444 ms | Diferença média: 0.555556 ms | Nº de notificações: 100
ID Registro: 999 | Periodo desejado: 53 ms | Média intervalos: 52.4747 ms | Diferença média: 0.525253 ms | Nº de notificações: 100
ID Registro: 1000 | Periodo desejado: 99 ms | Média intervalos: 98.4949 ms | Diferença média: 0.505051 ms | Nº de notificações: 100

Média global das diferenças entre períodos desejados e intervalos de notificação: 0.516187 ms
```

Resultados teste descobrimento

- Componente Detector de Veículos quer descobrir veículos próximos a ele.
- Para isso ele envia Mensagens de Interesse com Período = 0 e Tipo = posição (dado fornecido pelo Sensor GPS).
- Os Componentes Sensor GPS dos demais veículos recebem esses interesses e respondem uma única vez.
- Dessa forma, o Detector de Veículos consegue detectar apenas os veículos ao seu redor.
- *Possivel Aprimoramento: Detector de Veículos processar as posições recebidas de modo que ele detectasse os veículos vizinhos a ele.

```
============================================================
🧪  TESTE: Reconhecimento e Comunicação entre componentes de diferentes veículos (externa)
------------------------------------------------------------
 Veículo 1 detecta veiculos proximos a ele:
 Detector de Veiculos requisita dados aos Sensores GPS.
============================================================

Parâmetros do teste:
 Interface de rede: enp0s1
 Número de veículos: 2
 Número de respostas: 3
 Número de aparições: 3
 Intervalo entre aparições (ms): 1000
 Intervalo entre interesses (ms): 500

🧠 Detector Veiculos: enviou interesse.
🧠 Detector Veiculos: enviou interesse.
🧠 Detector Veiculos: enviou interesse.
✳Veiculo 1 adicionado
✳Veiculo 2 adicionado
🧠 Detector Veiculos: enviou interesse.
🧠 Detector Veiculos: detectou veiculo 1 na posicao: (0, 0)
🧠 Detector Veiculos: detectou veiculo 2 na posicao: (0, 0)
🧠 Detector Veiculos: enviou interesse.
✳Veiculo 4 adicionado
✳Veiculo 3 adicionado
🧠 Detector Veiculos: detectou veiculo 2 na posicao: (1, 1)
🧠 Detector Veiculos: detectou veiculo 1 na posicao: (1, 1)
🧠 Detector Veiculos: enviou interesse.
🧠 Detector Veiculos: detectou veiculo 2 na posicao: (2, 2)
🧠 Detector Veiculos: detectou veiculo 1 na posicao: (2, 2)
🧠 Detector Veiculos: detectou veiculo 4 na posicao: (0, 0)
🧠 Detector Veiculos: detectou veiculo 3 na posicao: (0, 0)
🧠 Detector Veiculos: enviou interesse.
✳Veiculo 5 adicionado
✳Veiculo 6 adicionado
🧠 Detector Veiculos: detectou veiculo 3 na posicao: (1, 1)
🧠 Detector Veiculos: detectou veiculo 4 na posicao: (1, 1)
🧠 Detector Veiculos: enviou interesse.
🧠 Detector Veiculos: detectou veiculo 4 na posicao: (2, 2)
🧠 Detector Veiculos: detectou veiculo 5 na posicao: (0, 0)
🧠 Detector Veiculos: detectou veiculo 6 na posicao: (0, 0)
🧠 Detector Veiculos: detectou veiculo 3 na posicao: (2, 2)
🧠 Detector Veiculos: enviou interesse.
🧠 Detector Veiculos: detectou veiculo 6 na posicao: (1, 1)
🧠 Detector Veiculos: detectou veiculo 5 na posicao: (1, 1)
🧠 Detector Veiculos: enviou interesse.
🧠 Detector Veiculos: detectou veiculo 6 na posicao: (2, 2)
🧠 Detector Veiculos: detectou veiculo 5 na posicao: (2, 2)

============================
✅ Teste finalizado.
============================
```