

# P1 - Comunicação 1:1 (funções send e receive)

Ismael Coral Hoepers Heinzelmann, Marcos Tomaszewski  
Matheus Paulon Novais, Sergio Bonini

September 2024

## 1 Introdução

A solução implementada tem como foco criar um sistema de comunicação confiável que opera sobre protocolos de rede potencialmente não confiáveis, como o UDP (User Datagram Protocol). O UDP é conhecido por não ter mecanismos embutidos para garantir a confiabilidade na entrega de mensagens, o que significa que pacotes podem ser perdidos, duplicados ou entregues fora de ordem. A solução implementada aborda esses desafios ao adicionar uma lógica personalizada para garantir a entrega confiável, não duplicação e não criação de mensagens. Este relatório aborda como cada parte do sistema funciona e enfatiza os mecanismos que asseguram as propriedades desejadas na transmissão de mensagens.

## 2 Visão Geral da Arquitetura

A solução consiste em diversos componentes-chave que trabalham juntos para proporcionar uma comunicação confiável:

- **Classes Datagram e Message:** Essas classes gerenciam a transmissão e montagem de datagramas em mensagens completas.
- **Classe ReliableCommunication:** Interface para acesso a biblioteca de comunicação confiável.
- **Classe MessageReceiver:** Gerencia a recepção de datagramas, garantindo que as mensagens sejam montadas corretamente, verificadas e entregues na ordem correta.

- **Classe `MessageSender`:** Lida com o envio e recebimento de datagramas de controle, com lógica embutida para reconhecimento (ACK), retransmissão e detecção de erros.
- **CRC32 Checksum:** Garantirá a integridade dos dados ao calcular e verificará checksums para cada datagrama.
- **BlockingQueue:** Proporciona um mecanismo thread-safe para enfileiramento e processamento de mensagens e requisições entre diferentes componentes.

## 3 Características-Chave que Garantem a Comunicação Confiável

### 3.1 Entrega Confiável

A entrega confiável é garantida pela incorporação de mecanismos de reconhecimento (ACK) e retransmissões. Os principais passos incluem:

- **Reconhecimentos (ACK):** Quando um nó envia um datagrama, ele aguarda um reconhecimento (ACK) do destinatário. Se o remetente não receber um ACK dentro de um tempo limite predefinido, o datagrama é retransmitido.
- **Sequenciamento de Datagramas:** A classe `Datagram` atribui a cada datagrama um número de versão, garantindo que todos os datagramas sejam transmitidos e recebidos na sequência correta.
- **Handshake de Duas Partes:** A comunicação começa com uma mensagem de sincronização (SYN), seguida por um reconhecimento (SYN-ACK) do destinatário. Esse handshake garante que ambas as partes estejam cientes da troca de mensagens que ocorrerá.

A classe `ReliableCommunication` lida com esse processo em seus métodos `send` e `receive`. Um datagrama é enviado, e, se o reconhecimento não for recebido dentro do tempo limite especificado, o datagrama é retransmitido.

## 3.2 Não Duplicação de Mensagens

A não duplicação de mensagens é garantida pelo rastreamento dos números de sequência (versões) de cada datagrama dentro de uma mensagem. Os principais mecanismos incluem:

- **Montagem de Mensagens e Rastreamento de Versão:** A classe `Message` mantém o número de versão do último datagrama recebido (`lastVersionReceived`). Cada novo datagrama é verificado para garantir que sua versão seja superior à última recebida, impedindo que datagramas duplicados sejam adicionados à mensagem.
- **Verificação pelo `MessageReceiver`:** A classe `MessageReceiver` verifica a sequência dos datagramas recebidos através do método `verifyMessage`. Este método verifica se a versão do datagrama está dentro do intervalo esperado e descarta duplicatas.

No caso de um datagrama ser duplicado durante a transmissão, o sistema o identifica por meio de seu número de versão e o descarta, garantindo que dados duplicados não sejam adicionados à mensagem.

## 3.3 Não Criação de Mensagens

O sistema impede a criação de mensagens fantasmas ou errôneas ao verificar a integridade e validade de cada datagrama antes que ele seja aceito na mensagem. Isso é alcançado por meio de:

- **Verificação de Checksum:** A classe `CRC32` será usada para calcular e verificar um checksum para cada datagrama. O checksum garantirá que o datagrama não foi adulterado ou corrompido durante a transmissão. O método `verifyChecksum` na classe `Protocol` irá comparar o checksum calculado com o checksum recebido para garantir a integridade.
- **Tratamento de SYN e FIN:** O sistema usa as flags SYN (Sincronização) e FIN (Finalização) para marcar o início e o fim de uma transmissão de mensagem. A classe `MessageReceiver` lida com o primeiro datagrama SYN ao criar uma nova mensagem, e o último datagrama

(com a flag FIN) marca o fim da mensagem. Somente datagramas válidos com as flags SYN/FIN apropriadas são permitidos para criar ou fechar mensagens.

- **Timeouts e Limpeza:** A classe `MessageReceiver` também lida com timeouts através do método `cleanse`, que remove mensagens parcialmente recebidas se nenhum datagrama for recebido por um período predefinido. Esse mecanismo garante que transmissões incompletas ou inválidas sejam limpas e não resultem na criação de mensagens incompletas.

## 4 Fluxo Detalhado de Mensagens

A seguir, descreve-se o fluxo de mensagens neste sistema de comunicação confiável:

### 1. Envio de Mensagens:

- Uma mensagem é dividida em vários datagramas pela classe `ReliableCommunication`.
- Cada datagrama é enviado com um número de sequência (versão) e um checksum para garantir a integridade.
- Após o envio de cada datagrama, o remetente aguarda um ACK do destinatário. Se nenhum ACK for recebido, o datagrama é retransmitido.

### 2. Recebimento de Mensagens:

- A classe `MessageReceiver` recebe cada datagrama e verifica seu número de sequência para garantir que ele não seja duplicado ou fora de ordem.
- O checksum é verificado para garantir que o datagrama esteja intacto e não corrompido.
- Os datagramas são adicionados à mensagem correspondente, e a mensagem é montada na ordem correta. Se a mensagem estiver completa (todos os datagramas esperados foram recebidos), o destinatário envia um FIN-ACK para indicar o recebimento bem-sucedido.

### 3. Tratamento de Erros e Retransmissões:

- Se um datagrama estiver faltando ou corrompido, o destinatário pode enviar um NACK (Negative Acknowledgment) para solicitar a retransmissão.
- Timeouts são usados para detectar datagramas perdidos, e o remetente retransmite se um ACK não for recebido dentro do período especificado.

### 4. Entrega Final da Mensagem:

- Uma vez que o destinatário tenha recebido todos os datagramas, a mensagem é montada, e a classe MessageReceiver empurra a mensagem completa para a BlockingQueue para processamento posterior pela camada de aplicação.

## 5 Detecção e Tratamento de Erros

O sistema garante a detecção e o tratamento de erros por meio de:

- **Verificação de Checksum:** Cada datagrama incluirá um checksum CRC32 que será verificado no lado do receptor para garantir que os dados não foram corrompidos durante a transmissão.
- **NACK (Negative Acknowledgment):** Se um datagrama estiver corrompido ou faltando, o destinatário envia um NACK ao remetente, solicitando a retransmissão do datagrama. Esse tratamento provê uma melhora no desempenho, uma vez que o pacote será retransmitido antes do tempo padrão de timeout.
- O não recebimento de um ACK ou NACK após dado tempo, implica na retransmissão automática do datagrama, garantindo que pacotes perdidos na rede sejam retransmitidos.

## **6 Detalhes de implementação**

### **6.1 Envio em lote**

Observamos que aguardar o recebimento de um datagrama para o envio do próximo acaba por demonstrar uma demora elevada para a transmissão das mensagens. Visando um melhor desempenho foi implementada uma estratégia de envio em lote. Para isto o remetente envia sequencialmente um conjunto de datagramas e aguarda, o destinatário responde cada uma conforme recebe e o remetente mantém uma lista dos pacotes que foram recebidos com sucesso. Após determinado tempo, o remetente enviará novamente em lote pacotes perdidos ou corrompidos, até que a mensagem chegue inteiramente.

### **6.2 Portas efêmeras**

A biblioteca define dinamicamente, junto ao sistema operacional, uma porta para o recebimento das mensagens de controle NACK e ACK pelo remetente. Isso garante o isolamento do tráfego entre os datagramas e mensagens de controle.

### **6.3 Fragmentação**

Ao enviar mensagens maiores que um pacote, a biblioteca fragmenta de forma transparente em vários pacotes, para que seja possível o envio via UDP.

## **7 Conclusão**

A solução implementada atinge a entrega confiável, não duplicação e não criação de mensagens ao incorporar mecanismos de reconhecimento (ACK), rastreamento de sequência, verificação de checksum e verificação de integridade das mensagens. O uso de CRC32 para detecção de erros, flags SYN/ACK/FIN para controle de fluxo de mensagens e gerenciamento de timeouts garantirá que o sistema lide com problemas típicos de rede, como pacotes perdidos, duplicados ou fora de ordem. Combinando essas estratégias, o sistema oferece uma solução robusta para comunicação confiável em ambientes onde a integridade e a ordem das mensagens são críticas.

## 8 Diagramas de mensagens

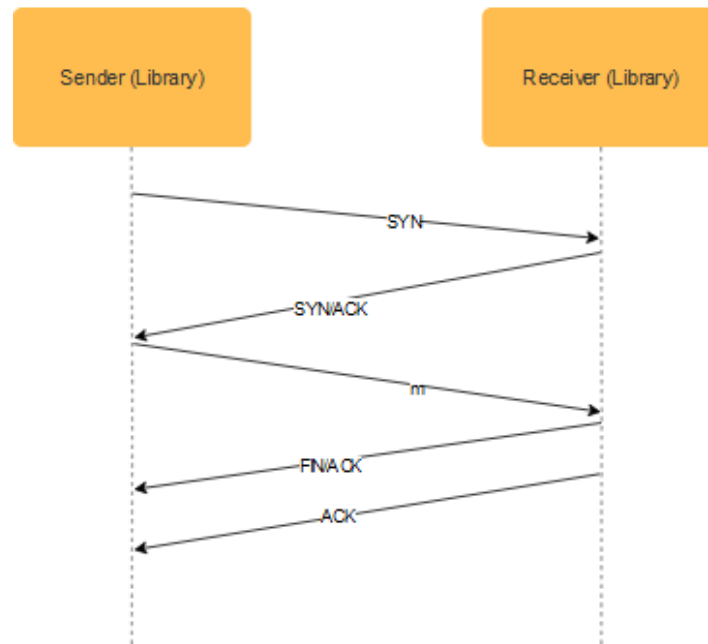


Figura 1: Mensagem em um datagrama único

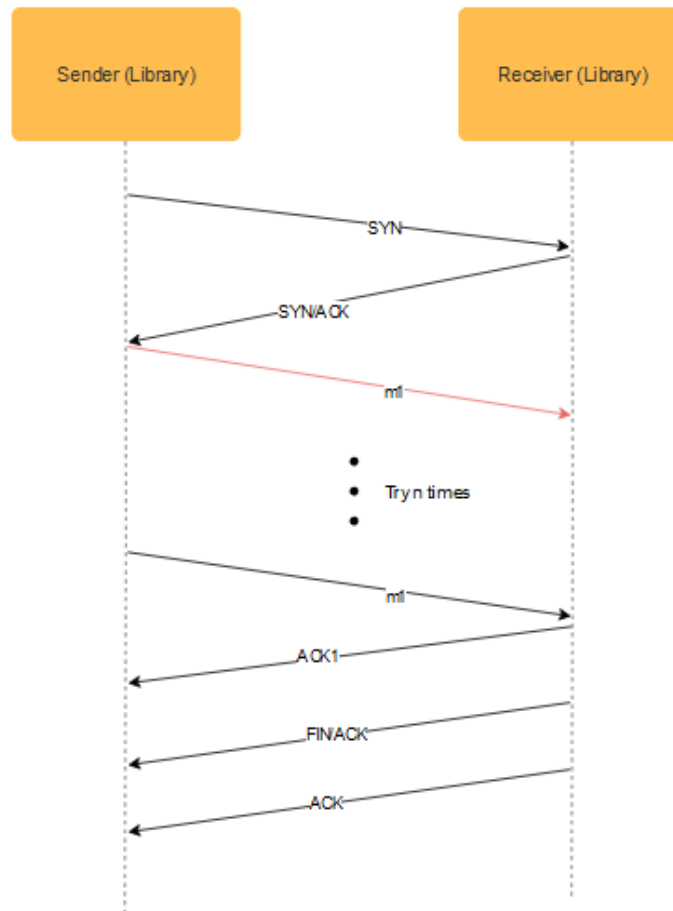


Figura 2: Mensagem com retransmissão



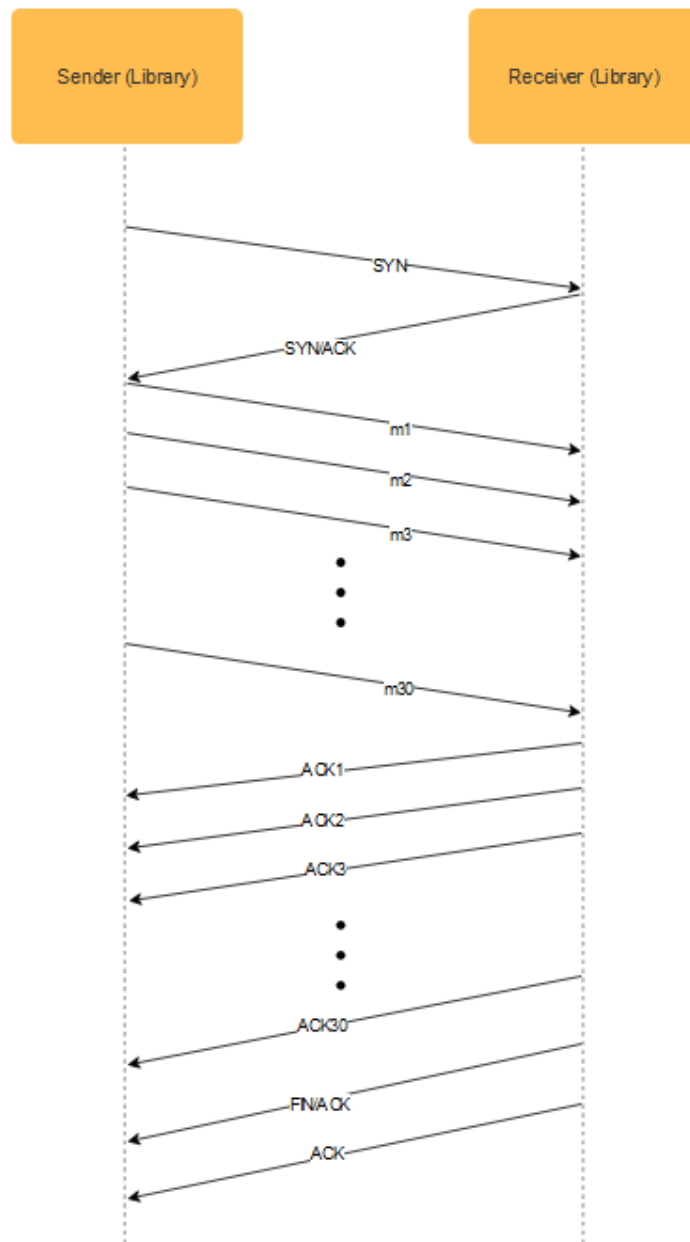


Figura 3: Mensagem com fragmentação

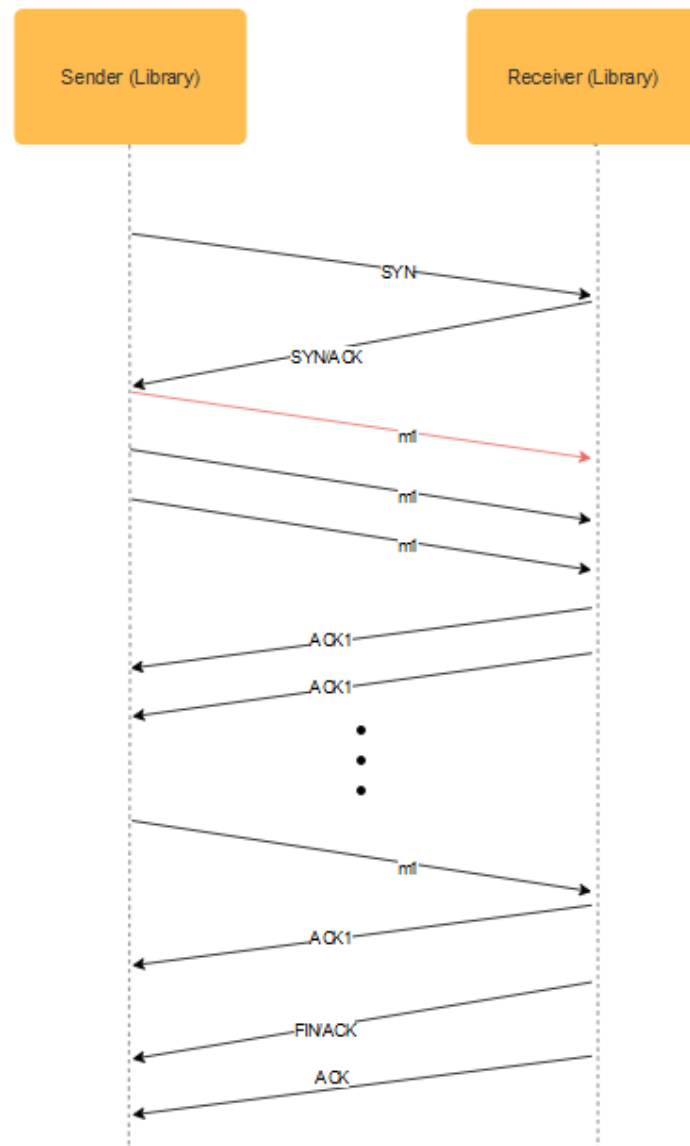


Figura 4: Mensagem com fragmentação e retransmissão