Caça ao Tesouro com Raw Sockets: Implementação de Protocolo de Camada 2 em Rede Ponto-a-Ponto

João Meyer, Vitor Faria

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)
Curitiba – PR – Brasil

jmm23@inf.ufpr.br, vfms23@inf.ufpr.br

Abstract. *This report describes the development of a networked treasure hunt game using raw sockets and a custom data link layer protocol. The project involves client-server communication over a point-to-point connection without routing through switches. A stop-and-wait protocol ensures reliable transmission, with Kermit-inspired framing used for message formatting and file transfer.*

Resumo. *Este relatório descreve o desenvolvimento de um jogo de caça ao tesouro em rede, utilizando raw sockets e protocolo customizado na camada de enlace. O projeto envolve comunicação cliente-servidor sobre conexão ponto-a-ponto, sem roteamento por switches. Um protocolo stop-and-wait garante a confiabilidade, com quadros inspirados no Kermit para mensagens e transferência de arquivos.*

1. Introdução

O trabalho T1 da disciplina de Redes I consiste na implementação de um jogo cliente-servidor com comunicação direta entre duas máquinas via cabo de rede. O foco está na manipulação de protocolos da camada de enlace, com implementação de sockets brutos (raw sockets) e controle de fluxo confiável com tratamento de erros.

2. Objetivo

Implementar um jogo interativo com movimentação em um grid 8x8, onde o cliente comanda um agente em busca de tesouros escondidos, os quais são controlados e revelados pelo servidor. A comunicação entre as partes deve obedecer a um protocolo definido em sala, baseado no Kermit, e operando diretamente sobre frames Ethernet.

3. Protocolo

O protocolo implementado possui os seguintes campos:

- **Marcador de início** (1 byte): 0x7E
- **Tamanho** (7 bits), **Sequência** (5 bits), **Tipo** (4 bits)
- **Checksum** (1 byte): XOR dos campos anteriores
- **Dados** (0–127 bytes)

Foram implementados 16 tipos de mensagens, como `ACK`, `NACK`, movimentações (cima, baixo, etc.), envio de arquivos e erros.

4. Cliente e Servidor

4.1. Servidor

O servidor é responsável por:

- Sortear 8 tesouros no grid;
- Exibir suas posições no console;
- Processar os comandos do cliente;
- Enviar arquivos de tesouros quando encontrados;
- Garantir confiabilidade com reenvio em caso de falha.

Cada tesouro corresponde a um arquivo (.txt, .jpg, .mp4) localizado na pasta `objetos/`. O servidor usa a função `stat()` para obter informações e validar permissões de leitura.

4.2. Cliente

O cliente exibe o mapa, permite movimentação com teclas `WASD`, detecta tesouros e trata a exibição de arquivos. Possui um loop com timeout para receber e validar mensagens, e envia `ACKs` ou `NACKs` conforme o protocolo.

Arquivos são salvos localmente e, se for o caso, abertos com `xdg-open`. O cliente também verifica espaço livre usando `statvfs()`.

5. Transmissão de Arquivos

O envio de arquivos do servidor ao cliente é realizado em etapas:

1. Envio do nome do arquivo;
2. Envio do tamanho;
3. Envio dos dados, em blocos de até 127 bytes;
4. Envio de mensagem de finalização.

A cada etapa, o servidor aguarda um `ACK` e reenvia em caso de falha, respeitando a política stop-and-wait.

6. Controle de Fluxo e Erros

Foram implementados mecanismos de:

- Timeout e retransmissão;
- `NACKs` em pacotes corrompidos;
- Reenvio automático;
- Verificação de sequência e duplicação.

O cliente é responsável por manter a posição do jogador e marcações no mapa: pontos visitados, pontos com tesouros encontrados e posição atual.

7. Escolhas Feitas

Durante o desenvolvimento do projeto, algumas decisões de implementação se mostraram estratégicas:

- **Separação clara de papéis**: cliente e servidor foram implementados como programas distintos, facilitando testes e modularização.
- **Protocolos tipados**: todos os tipos de mensagens foram enumerados em `kermit.h`, facilitando manutenção e legibilidade.
- **Reenvio com timeout curto**: optou-se por janelas de retransmissão de 50ms para responsividade.
- **Interface textual simples**: o mapa 8x8 foi representado no terminal com símbolos, garantindo compatibilidade em qualquer ambiente Unix.
- **Sistema de ACK explícito com validação de sequência**: assegurou confiabilidade mesmo com perdas ou duplicações.
- **Detecção automática de tipo de arquivo**: o servidor envia tipo e nome com base na extensão do arquivo, permitindo ao cliente lidar com imagens, vídeos ou textos sem hardcode.

8. Desafios

Um dos principais desafios técnicos foi relacionado à transmissão de vídeos em redes reais. Durante os testes, notamos que arquivos de vídeo com maior volume frequentemente falhavam na recepção ou eram corrompidos. A causa identificada foi a inserção automática de uma **tag VLAN de 4 bytes** por parte da interface de rede, deslocando os dados e invalidando a estrutura do protocolo.

Para contornar esse problema, foi adotada uma estratégia de **duplicação do buffer enviado**: a cada byte real transmitido, adicionava-se um byte `0xFF` entre os dados. Essa abordagem, ainda que aumente o tráfego, impediu que alterações acidentais nos primeiros bytes invalidassem a estrutura do pacote, contornando interferências causadas por encapsulamento ou alteração em nível físico, como VLAN tagging.

9. Conclusão

O projeto permitiu o aprofundamento em protocolos de enlace, manipulação de sockets brutos e tratamento de arquivos binários e texto. A confiabilidade da comunicação foi garantida por um protocolo simples, porém robusto, e a separação clara das responsabilidades entre cliente e servidor facilitou o desenvolvimento.