# Laboratório de Redes de Computadores – Trabalho 1

**Alunos:**  
[Seu nome]  
[Nome do colega, se houver]

**Turma:** [Identificação da turma]  
**Data de entrega:** [Data]

# Sumário

1. [Introdução](#Xc2970a1a23df3ef3b34d1da3795ce3cec2678cb)
2. [Descrição da Implementação](#Xf79a7fd381e8d7f6146b8ea614a09e7e1637fb1)
   1. [Arquitetura Geral](#X5844ebbf9c2eec2a3ea5b724482cd134b0866c2)
   2. [Protocolo de Descoberta e Comunicação](#X0b73c385c98c206ce67fb2ca88c47115d077e43)
   3. [Mecanismos de Confiabilidade](#X95e46fc7edc355f7faa605dbe3089002c1a3132)
   4. [Estrutura do Código](#Xb1c90f8a49e1fa3c209d1e5024128c5d3193f29)
3. [Interface de Usuário](#Xec60a8487bb3393c7240f437f9bafac3e8203a0)
4. [Cenários de Teste e Validação](#X918afb11f2e79bbf818f08a36ae0de36c313820)
5. [Análise dos Resultados](#Xb62bce7fbdced784f1383023eb0cccae3eb33b0)
6. [Conclusão](#X5458287df17b821b98869ad3a9feb2ebf6c99b6)
7. [Anexos](#X82bde930feddbbc22ac137897c77ea2a9e9fe10)

# 1. Introdução

O presente trabalho tem como objetivo o desenvolvimento de um protocolo de comunicação customizado, encapsulado em pacotes UDP, para busca automática e comunicação entre dispositivos em uma rede local. O projeto visa proporcionar experiência prática com a pilha de protocolos de rede, uso de sockets, e implementação de mecanismos de confiabilidade em aplicações distribuídas peer-to-peer.

# 2. Descrição da Implementação

## 2.1 Arquitetura Geral

A aplicação é composta por múltiplos dispositivos, cada um executando um programa Python que utiliza sockets UDP para comunicação. Cada dispositivo é capaz de:

* Descobrir automaticamente outros dispositivos ativos na rede.
* Enviar e receber mensagens de texto.
* Transferir arquivos de forma confiável, mesmo em condições adversas de rede.

A arquitetura utiliza múltiplas threads para: - Envio periódico de mensagens HEARTBEAT. - Recebimento e processamento de mensagens UDP. - Limpeza automática de dispositivos inativos.

## 2.2 Protocolo de Descoberta e Comunicação

O protocolo implementa os seguintes tipos de mensagens:

* **HEARTBEAT** : Enviada via broadcast na inicialização e a cada 5 segundos, informando a presença do dispositivo na rede.
* **TALK** : Envia uma mensagem de texto para outro dispositivo, aguardando confirmação (ACK).
* **FILE** : Inicia a transferência de arquivo, aguardando ACK antes de enviar os blocos.
* **CHUNK** : Envia um bloco do arquivo, codificado em base64, com confirmação individual (ACK) e retransmissão em caso de perda.
* **END** : Finaliza a transferência, enviando o hash do arquivo para verificação de integridade.
* **ACK** : Confirma o recebimento de qualquer mensagem.
* **NACK** : Indica erro no recebimento ou processamento.

## 2.3 Mecanismos de Confiabilidade

* **Confirmação de recebimento (ACK) e retransmissão automática** em caso de perda de pacotes.
* **Detecção e eliminação de duplicatas** de mensagens e blocos.
* **Detecção e reordenação de pacotes fora de ordem** para blocos de arquivos.
* **Validação de integridade** ao final da transferência de arquivos, comparando o hash recebido com o calculado.
* **Envio de arquivos em blocos**, sem carregar o arquivo inteiro na memória.

## 2.4 Estrutura do Código

O código está dividido em dois arquivos principais:

* dispositivo.py: Implementa toda a lógica do protocolo, comunicação UDP, threads, envio/recebimento de mensagens e arquivos, e mecanismos de confiabilidade.
* main.py: Implementa a interface de linha de comando, permitindo ao usuário listar dispositivos, enviar mensagens e arquivos.
* grande\_teste.txt: Arquivo de texto grande para testes de transferência.

# 3. Interface de Usuário

A interface é baseada em linha de comando e permite:

* Listar dispositivos ativos (devices).
* Enviar mensagens de texto (talk <nome> <mensagem>).
* Enviar arquivos (sendfile <nome> <nome-arquivo>).
* Visualizar o progresso da transferência e mensagens de sucesso ou falha.

**Exemplo de uso:**

1. Listar dispositivos ativos  
2. Enviar mensagem (use: talk <nome> <mensagem>)  
3. Enviar arquivo (use: sendfile <nome> <arquivo>)  
4. Sair

**Prints da interface:**  
*Insira aqui prints de tela mostrando a interface em funcionamento.*

# 4. Cenários de Teste e Validação

A seguir, são apresentados os principais cenários de teste realizados para validar o funcionamento e a confiabilidade do protocolo.

## 4.1 Funcionamento Normal

* **Objetivo:** Demonstrar a descoberta automática de dispositivos, envio de mensagens e transferência de arquivos em condições normais.
* **Procedimento:**
  + Iniciar dois dispositivos.
  + Listar dispositivos ativos.
  + Enviar mensagem TALK.
  + Enviar arquivo (ex: grande\_teste.txt).
* **Evidências:**
  + *Inserir prints do Wireshark mostrando HEARTBEAT, TALK, FILE, CHUNK, END, ACK, NACK.*
  + *Inserir prints da interface e dos logs.*

## 4.2 Testes em Condições Adversas (Clumsy e Wireshark)

Para validar a robustez do protocolo, foram realizados testes com a ferramenta **Clumsy** (Windows) para simular falhas de rede e **Wireshark** para capturar os pacotes.

### Como usar o Clumsy

1. Baixe o Clumsy: https://jagt.github.io/clumsy/
2. Abra o Clumsy e selecione a interface de rede correta.
3. Marque as opções conforme o teste desejado:
   * **Drop:** Simula perda de pacotes (ex: 10%)
   * **Duplicate:** Simula duplicação de pacotes (ex: 10%)
   * **Lag:** Simula atraso (ex: 200ms)
   * **Out of order:** Simula entrega fora de ordem (ex: 10%)
   * **Tamper:** Simula corrupção de pacotes (ex: 10%)
4. Clique em Start para ativar as falhas.
5. Execute os testes normalmente no programa.
6. Desative o Clumsy após o teste.

### Exemplos de Cenários para Teste

| Caso de Teste | Configuração Clumsy | O que observar/capturar |
| --- | --- | --- |
| **Normal** | Nenhuma opção marcada | Transferência sem falhas, tudo rápido |
| **Perda de pacotes** | Drop (10%) | Retransmissão automática, sucesso no final |
| **Duplicação de pacotes** | Duplicate (10%) | Duplicatas descartadas, sem erro no destino |
| **Atraso** | Lag (200ms ou mais) | Protocolo espera, pode haver retransmissão |
| **Fora de ordem** | Out of order (10%) | Blocos fora de ordem aceitos corretamente |
| **Corrupção** | Tamper (10%) | Arquivo corrompido, NACK enviado, falha |
| **Combinado** | Drop + Lag + Duplicate | Protocolo lida com múltiplas adversidades |

### Usando o Wireshark

1. Abra o Wireshark e selecione a interface de rede.
2. Inicie a captura antes de rodar o programa.
3. Filtre por porta UDP usada (ex: udp.port == 5000).
4. Salve o arquivo .pcapng após o teste.

### Evidências

* Prints do Clumsy com as opções marcadas.
* Prints do Wireshark mostrando os pacotes UDP.
* Prints do terminal mostrando retransmissões, ACKs, NACKs, sucesso ou falha.

# 5. Análise dos Resultados

**[Aqui você irá descrever, para cada cenário, o que foi observado nas capturas e prints. Explique como o protocolo respondeu a cada situação adversa.]**

* O protocolo foi capaz de garantir a entrega confiável de mensagens e arquivos mesmo sob condições adversas, como perda, duplicação, atraso, reordenação e corrupção de pacotes.
* As retransmissões automáticas, confirmações (ACK), detecção de duplicatas e verificação de integridade (hash) funcionaram conforme esperado.
* O usuário foi informado sobre o progresso e o resultado das operações, tanto em caso de sucesso quanto de falha.

# 6. Conclusão

O trabalho permitiu compreender na prática o funcionamento de protocolos de rede, a utilização de sockets UDP e a implementação de mecanismos de confiabilidade em aplicações distribuídas. O protocolo desenvolvido demonstrou ser capaz de garantir a entrega confiável de mensagens e arquivos, mesmo em condições adversas de rede, atendendo a todos os requisitos propostos.

# 7. Anexos

* Códigos-fonte do programa.
* Arquivos de captura do Wireshark (.pcapng).
* Prints de tela e logs relevantes.
* Arquivo de teste: grande\_teste.txt.

# Roteiro para Apresentação (5 a 7 minutos)

1. **Introdução rápida:**
   * Objetivo do trabalho e contexto.
2. **Demonstração da interface:**
   * Listar dispositivos, enviar mensagem, enviar arquivo.
3. **Demonstração de um teste normal:**
   * Mostre a transferência e a captura no Wireshark.
4. **Demonstração de um teste adverso:**
   * Exemplo: perda de pacotes, retransmissão e sucesso final.
5. **Mostre prints/capturas:**
   * Prints do Clumsy, Wireshark e terminal.
6. **Conclusão:**
   * Destaque os aprendizados e resultados.

*Preencha os espaços marcados com prints, capturas e análises após realizar os testes!*