Trabalho 1 – Lab. Redes

Integrandes: Luca Manfroi e Lucas Weiss

DeviceUDP.java

Descrição:
Classe principal que representa o dispositivo na rede. É responsável por gerenciar a comunicação UDP, escutando pacotes, identificando seus tipos (HEARTBEAT, TALK, FILE, CHUNK, END, ACK, etc.) e respondendo de acordo. Ela funciona como o "core" do sistema, lidando com a recepção e roteamento das mensagens para as classes auxiliares.

Objetivo principal:
Gerenciar toda a lógica de comunicação de rede, garantindo que os comandos e transferências sejam corretamente processados e encaminhados.

Principais métodos:

- processarPacote(DatagramPacket packet): Método que faz o parsing das mensagens recebidas e decide a ação (ex: TALK, FILE, ACK...).
- enviarAck(String id): Envia um ACK para confirmar o recebimento de pacotes importantes.
- **Outros:** Trechos que lidam com a atualização de lista de dispositivos, controle de pacotes desconhecidos e logs.

ComandoCLI.java

Descrição:
Classe que trata a interface de linha de comando (CLI). Monitora os comandos digitados pelo usuário e executa ações como listar dispositivos ativos, enviar mensagens (talk) ou iniciar envio de arquivos (sendfile).

Objetivo principal:
Permitir a interação do usuário com o sistema através de comandos simples, integrando com as demais classes.

Principais métodos:

- run(): Loop principal que lê e interpreta os comandos digitados.
- listarDispositivos(): Mostra a lista atualizada de dispositivos ativos.
- talk(String destino, String mensagem): Envia uma mensagem de texto para o dispositivo especificado.
- sendfile(String destino, String caminhoArquivo): Dispara a thread de envio de arquivo usando a classe FileSender.

HeartbeatSender.java

Descrição:
Thread responsável por enviar periodicamente pacotes do tipo HEARTBEAT em broadcast, permitindo que outros dispositivos saibam que este está ativo.

Objetivo principal:
Manter a presença do dispositivo na rede, garantindo que ele apareça como ativo para os outros participantes.

Principais métodos:

- run(): Envia a mensagem "HEARTBEAT [nome]" a cada 5 segundos continuamente enquanto a aplicação estiver rodando.

LimpezaDispositivos.java

Descrição:
Thread que verifica periodicamente a lista de dispositivos ativos e remove aqueles que não enviaram HEARTBEAT nos últimos 10 segundos.

Objetivo principal:
Garantir que a lista de dispositivos seja sempre atualizada e contenha apenas dispositivos realmente ativos.

Principais métodos:

- run(): Loop que verifica timestamps dos dispositivos e remove os que excederam o limite de inatividade.

FileSender.java

Descrição:
Classe que implementa a lógica de envio confiável de arquivos, dividindo o arquivo em blocos (CHUNKs), enviando cada um deles e aguardando ACK para confirmar a recepção. Implementa lógica de reenvio em caso de timeout/falha.

Objetivo principal:
Realizar a transmissão confiável de arquivos inteiros por UDP, garantindo que todos os blocos sejam entregues e confirmados.

Principais métodos:

- run(): Gerencia todo o fluxo de envio do arquivo (FILE, CHUNKs e END).
- enviarEConfirmar(String mensagem, String id): Envia um pacote e aguarda o ACK correspondente, com timeout e reenvio automático.
- bytesToHex(byte[] bytes): Converte o hash SHA-256 em string hexadecimal para envio na mensagem END.

FileReceiver.java

Descrição:
Classe que trata a recepção dos arquivos enviados. Monta o arquivo a partir dos CHUNKs recebidos, envia os ACKs e, no final, verifica a integridade comparando o hash SHA-256 recebido com o hash calculado.

Objetivo principal:
Reconstruir corretamente os arquivos recebidos, lidar com duplicação e fora de ordem (armazenando via TreeMap) e garantir a integridade final do arquivo.

Principais métodos:

- receberFile(String id, String nomeArquivo, long tamanho): Inicializa a recepção do arquivo.
- receberChunk(String id, int seq, String dadosBase64): Armazena cada CHUNK, garantindo que duplicatas sejam descartadas.
- receberEnd(String id, String hashEsperado): Finaliza o arquivo e verifica o hash de integridade, imprimindo sucesso ou falha.

Dispositivo.java

Descrição:
Classe modelo que representa um dispositivo ativo na rede, armazenando informações como IP, nome e o timestamp da última vez que enviou HEARTBEAT.

Objetivo principal:
Modelar um dispositivo para ser usado nas listas gerenciadas pelo sistema (ex: lista de dispositivos ativos).

Principais atributos/métodos:

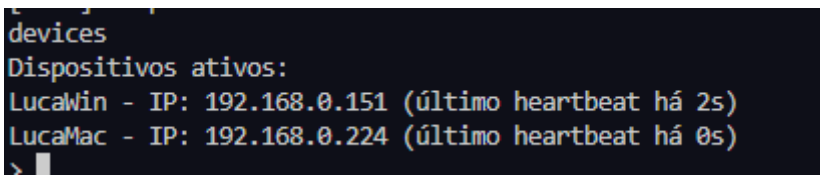
- String nome: Nome amigável do dispositivo.
- String ip: Endereço IP do dispositivo.
- long ultimoHeartbeat: Timestamp do último HEARTBEAT recebido.
- Getters e setters simples.

Funções:

- <u>Devices.</u>

Exibe a lista de dispositivos atualmente ativos na rede.

Para cada dispositivo, devem ser mostradas as seguintes informações:



- Talk .

Envia uma mensagem de texto para o dispositivo com o nome especi□cado.

Device1:

```
> talk LucaMac teste talk
Mensagem enviada para LucaMac
> Recebido: ACK TALK
```

Device1:

```
Recebido TALK: TALK teste talk
```

- Sendfile .

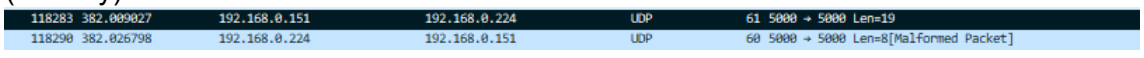
Inicia a transferência de um arquivo para o dispositivo com o nome indicado.

```
Uso: sendfile <nome> <nome-arquivo>
> sendfile LucaMac arquivoteste.txt
> Enviando: FILE e3fa622d-1fab-405c-b93b-098a111ff129 arquivoteste.txt 27
ACK recebido para ID: e3fa622d-1fab-405c-b93b-098a111ff129
Enviando: CHUNK e3fa622d-1fab-405c-b93b-098a111ff129 0 VGVzdGFuZG8gZW52aW8gZGUgYXJxdWl2b3Mu
ACK recebido para ID: e3fa622d-1fab-405c-b93b-098a111ff129
Enviando: END e3fa622d-1fab-405c-b93b-098a111ff129 23d47b3af467cfd78d8979d9d88948d602b37eae98d7c12907fab016182416f7
ACK recebido para ID: e3fa622d-1fab-405c-b93b-098a111ff129
Arquivo enviado com sucesso!
```

```
Recebido FILE: FILE e3fa622d-1fab-405c-b93b-098a111ff1
29 arquivoteste.txt 27
Recebido CHUNK seq=0
Chunk 0 recebido (27 bytes)
Hash esperado: 23d47b3af467cfd78d8979d9d88948d602b37ea
e98d7c12907fab016182416f7
Hash calculado: 23d47b3af467cfd78d8979d9d88948d602b37e
ae98d7c12907fab016182416f7
Arquivo arquivoteste.txt recebido com sucesso!
```

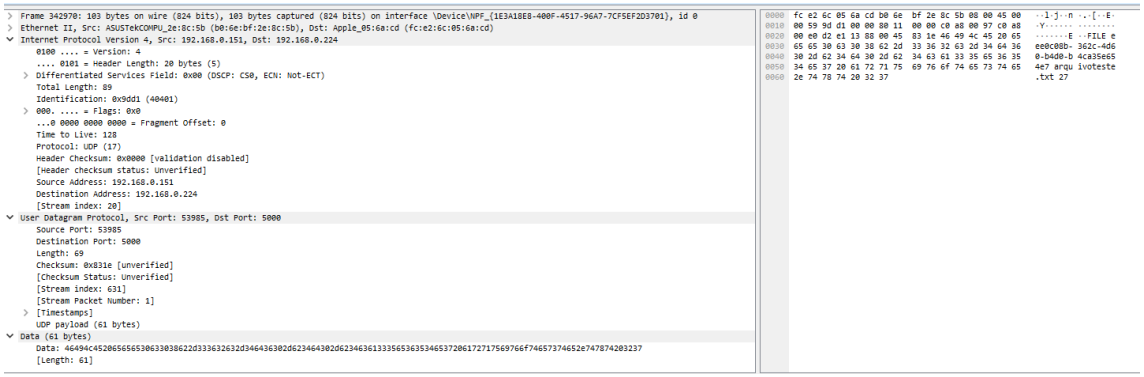
Testes de Integridade:

Pacote TALK sem interferência:
Captura do pacote UDP referente ao comando talk enviado do Windows (192.168.0.151) para o Mac (192.168.0.224) na porta 5000. O campo Data mostra claramente o conteúdo da mensagem enviada ("mensagem teste"). Esta captura serve como **base de referência** para comparações posteriores com rede instável simulada (Clumsy).



Pacotes CHUNK de arquivo sem interferência:
Captura de pacotes UDP durante o envio de arquivo entre Windows (192.168.0.151) para o Mac (192.168.0.224), porta 5000. Mostra os pacotes CHUNK contendo fragmentos do arquivo e o pacote de finalização (END). Essa referência permite comparar a transmissão normal com as condições de rede instável nos testes seguintes.

342970	1345.132390	192.168.0.151	192.168.0.224	UDP	103 53985 → 5000 Len=61
342976	1345.243274	192.168.0.224	192.168.0.151	UDP	82 5000 → 53985 Len=40
342977	1345.250454	192.168.0.151	192.168.0.224	UDP	123 53985 → 5000 Len=81
342983	1345.292905	192.168.0.224	192.168.0.151	UDP	82 5000 → 53985 Len=40
342984	1345.296642	192.168.0.151	192.168.0.224	UDP	147 53985 → 5000 Len=105
342987	1345.345470	192.168.0.224	192.168.0.151	UDP	82 5000 → 53985 Len=40



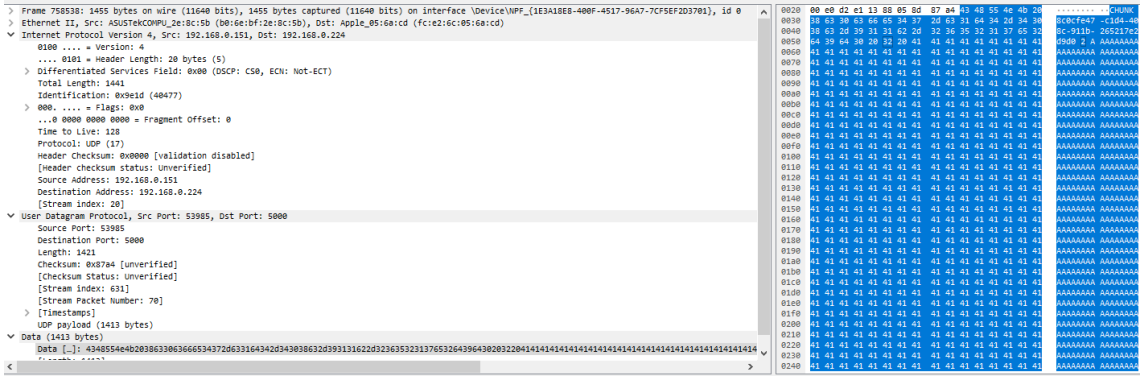
entrega fora de ordem,

atrasos e pacotes corrompidos 192.168.0.224

Teste Drop 20%:
Configuração do Clumsy aplicando 20% de perda de pacotes no tráfego UDP (porta 5000). No Wireshark, o pacote CHUNK Seq=2 foi perdido e reenviado após timeout, evidenciado por duas capturas do mesmo pacote. O console confirma o recebimento

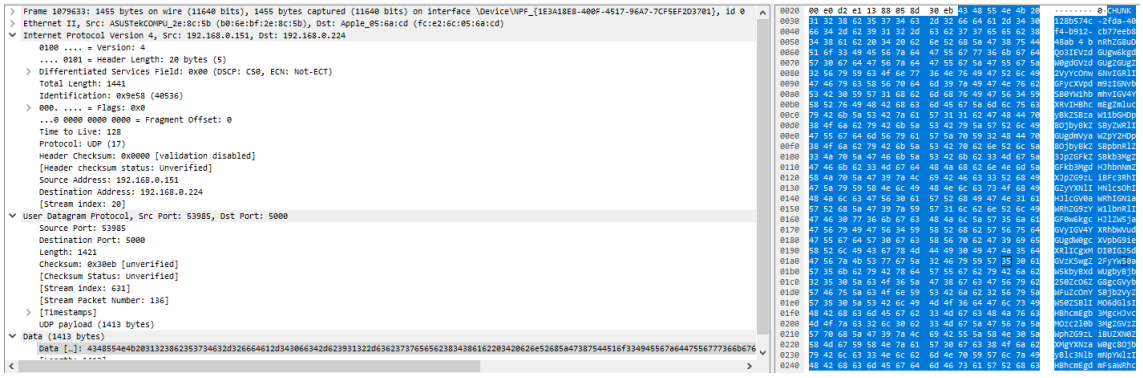
duplicado e a continuidade da transmissão sem perda de integridade.

757975	2844.836390	192.168.0.151	192.168.0.224	UDP	105 53985 → 5000 Len=63
757978	2844.888930	192.168.0.224	192.168.0.151	UDP	82 5000 → 53985 Len=40
757979	2844.893994	192.168.0.151	192.168.0.224	UDP	1455 53985 → 5000 Len=1413
757980	2844.901802	192.168.0.224	192.168.0.151	UDP	82 5000 → 53985 Len=40
757981	2844.903552	192.168.0.151	192.168.0.224	UDP	1455 53985 → 5000 Len=1413
757987	2844.909797	192.168.0.224	192.168.0.151	UDP	82 5000 → 53985 Len=40
758200	2846.928172	192.168.0.151	192.168.0.224	UDP	1455 53985 → 5000 Len=1413
758204	2846.939177	192.168.0.224	192.168.0.151	UDP	82 5000 → 53985 Len=40
758291	2848.268213	192.168.0.151	192.168.0.255	UDP	59 5000 → 5000 Len=17
758292	2848.268303	192.168.0.151	192.168.0.255	UDP	59 5000 → 5000 Len=17
758348	2848.786054	192.168.0.224	192.168.0.255	UDP	60 5000 → 5000 Len=17
758366	2848.939485	192.168.0.151	192.168.0.224	UDP	1455 53985 → 5000 Len=1413
758538	2850.953024	192.168.0.151	192.168.0.224	UDP	1455 53985 → 5000 Len=1413



Teste Duplicate 10%:
Configuração do Clumsy aplicando duplicação de pacotes (10% de chance) no tráfego UDP (porta 5000). No Wireshark, o pacote CHUNK Seq=4 foi duplicado, aparecendo duas vezes na captura. O receptor identificou a duplicação utilizando a estrutura TreeMap, que armazena cada CHUNK com base no número de sequência. Antes de gravar, o código verifica se o número já foi recebido (chunks.containsKey(seq)) e descarta automaticamente duplicatas. O arquivo foi reconstruído com sucesso e a integridade foi confirmada.

10796…	3837.549823	192.168.0.151	192.168.0.224	UDP	104 53985 → 5000 Len=62
10796…	3837.560945	192.168.0.224	192.168.0.151	UDP	82 5000 → 53985 Len=40
10796…	3837.563459	192.168.0.151	192.168.0.224	UDP	1455 53985 → 5000 Len=1413
10796…	3837.569202	192.168.0.224	192.168.0.151	UDP	82 5000 → 53985 Len=40
10796…	3837.571360	192.168.0.151	192.168.0.224	UDP	1455 53985 → 5000 Len=1413
10796…	3837.578416	192.168.0.224	192.168.0.151	UDP	82 5000 → 53985 Len=40
10796…	3837.580409	192.168.0.151	192.168.0.224	UDP	1455 53985 → 5000 Len=1413
10796…	3837.588103	192.168.0.224	192.168.0.151	UDP	82 5000 → 53985 Len=40
10796…	3837.589350	192.168.0.151	192.168.0.224	UDP	1455 53985 → 5000 Len=1413
10796…	3837.590625	192.168.0.151	192.168.0.224	UDP	1455 53985 → 5000 Len=1413
10796…	3837.590659	192.168.0.151	192.168.0.224	UDP	1455 53985 → 5000 Len=1413
10796…	3837.595202	192.168.0.224	192.168.0.151	UDP	82 5000 → 53985 Len=40
10796…	3837.596713	192.168.0.151	192.168.0.224	UDP	1455 53985 → 5000 Len=1413
10796…	3837.598901	192.168.0.224	192.168.0.151	UDP	82 5000 → 53985 Len=40
10796…	3837.600162	192.168.0.151	192.168.0.224	UDP	1019 53985 → 5000 Len=977
10796…	3837.603690	192.168.0.224	192.168.0.151	UDP	82 5000 → 53985 Len=40
10796…	3837.605515	192.168.0.151	192.168.0.224	UDP	147 53985 → 5000 Len=105
10796…	3837.605777	192.168.0.224	192.168.0.151	UDP	82 5000 → 53985 Len=40
10796…	3837.617111	192.168.0.224	192.168.0.151	UDP	82 5000 → 53985 Len=40



Teste Out of Order 20% - 80%:

Durante o teste de entrega fora de ordem, foi utilizada a função "Out of Order" do Clumsy com alta intensidade. Embora o receptor tenha sido projetado para aceitar pacotes fora de ordem e remontá-los corretamente (utilizando TreeMap), não foi possível visualizar pacotes fora de ordem na captura do Wireshark. Mesmo assim, o protocolo está preparado para esse cenário e manteve a integridade da transferência.

15190…	6074.517738	192.168.0.151	192.168.0.224	UDP	104 53985 → 5000 Len=62
15190…	6074.527234	192.168.0.224	192.168.0.151	UDP	82 5000 → 53985 Len=40
15192…	6074.988433	192.168.0.151	192.168.0.224	UDP	1455 53985 → 5000 Len=1413
15192…	6075.000886	192.168.0.224	192.168.0.151	UDP	82 5000 → 53985 Len=40
15195…	6075.925876	192.168.0.151	192.168.0.224	UDP	1455 53985 → 5000 Len=1413
15195…	6075.944280	192.168.0.224	192.168.0.151	UDP	82 5000 → 53985 Len=40
15195…	6075.945634	192.168.0.151	192.168.0.224	UDP	1455 53985 → 5000 Len=1413
15195…	6075.952809	192.168.0.224	192.168.0.151	UDP	82 5000 → 53985 Len=40
15197…	6076.865133	192.168.0.151	192.168.0.224	UDP	1455 53985 → 5000 Len=1413
15197…	6076.872733	192.168.0.224	192.168.0.151	UDP	82 5000 → 53985 Len=40
15199…	6077.336106	192.168.0.151	192.168.0.224	UDP	1455 53985 → 5000 Len=1413
15199…	6077.421649	192.168.0.224	192.168.0.151	UDP	82 5000 → 53985 Len=40
15201…	6078.182449	192.168.0.224	192.168.0.255	UDP	60 5000 → 5000 Len=17
15201…	6078.182588	192.168.0.151	192.168.0.224	UDP	1455 53985 → 5000 Len=1413
15201…	6078.190121	192.168.0.224	192.168.0.151	UDP	82 5000 → 53985 Len=40
15202…	6078.612966	192.168.0.151	192.168.0.255	UDP	59 5000 → 5000 Len=17
15202…	6078.614400	192.168.0.151	192.168.0.224	UDP	1019 53985 → 5000 Len=977
15202…	6078.614536	192.168.0.151	192.168.0.255	UDP	59 5000 → 5000 Len=17
15202…	6078.647678	192.168.0.224	192.168.0.151	UDP	82 5000 → 53985 Len=40
15204…	6079.578010	192.168.0.151	192.168.0.224	UDP	147 53985 → 5000 Len=105
15204…	6079.687164	192.168.0.224	192.168.0.151	UDP	82 5000 → 53985 Len=40

Teste Tamper (10%):

Configuração do Clumsy aplicando corrupção de pacotes (10% de chance) no tráfego UDP (porta 5000). No Wireshark, foi possível visualizar pacotes CHUNK com payload alterado (dados inválidos). No console, observamos que os pacotes corrompidos não receberam ACK e foram retransmitidos automaticamente até esgotar as tentativas. Os HEARTBEATs também foram corrompidos, resultando em logs de mensagens desconhecidas. Esse teste comprova que o protocolo implementa detecção de falhas e retransmissão em caso de corrupção. Além disso, mesmo que todos os CHUNKs

sejam recebidos, a integridade do arquivo é verificada no final por meio do hash SHA-256, garantindo que arquivos corrompidos sejam detectados e descartados.