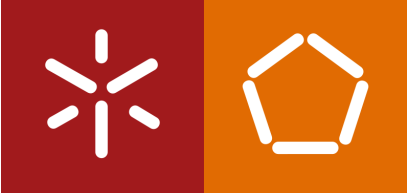




Catarina Pereira
Inês Neves
Leonardo Martins

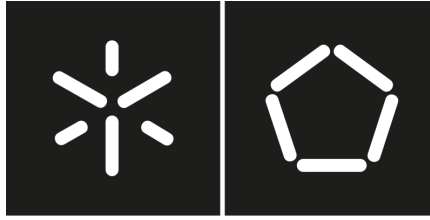
Protocolo Transporte Multimédia



Universidade do Minho
Escola de Engenharia

Catarina da Cunha Malheiro da Silva Pereira
Inês Cabral Neves
Leonardo Dias Martins

Protocolo Transporte Multimédia



Universidade do Minho

Escola de Engenharia

Catarina da Cunha Malheiro da Silva Pereira
Inês Cabral Neves
Leonardo Dias Martins

Protocolo Transporte Multimédia


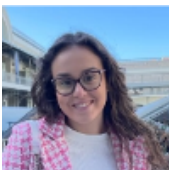

Serviços de Rede e Aplicações Multimédia
Trabalho prático de Mestrado
Engenharia em Telecomunicações e Informática

Trabalho efetuado sob a orientação de:

Professor Doutor Bruno Alexandre Fernandes Dias

Identificação do Grupo

O Grupo 08 é constituído por três deles que são do 1º ano do Mestrado em Engenharia de Telecomunicações e Informática (METI), sendo identificados por Pós-Graduação (PG) seguido dos seus números mecanográficos.

Imagem	Nome / Número Mecanográfico / E-mail institucional
	Catarina da Cunha Malheiro da Silva Pereira PG53733 pg537336@alunos.uminho.pt
	Inês Cabral Neves PG53864 pg53864@alunos.uminho.pt
	Leonardo Dias Martins PG53996 pg53996@alunos.uminho.pt

Índice

Identificação do Grupo	ii
Lista de Figuras	iv
Lista de Tabelas	v
Siglas	vi
1 Introdução	1
2 Revisão da Literatura	2
3 Abordagem Metodológica	3
4 Trabalho Realizado	5
4.1 Reencaminhador	5
4.2 Servidor	6
4.3 Cliente	9
5 Testes	11
6 Conclusão	12

Lista de Figuras

1	Resultados Servidor.	11
2	Resultados Reencaminhador.	11
3	Resultados Cliente.	11

Lista de Tabelas

Tabela 1: Estrutura do PDU 4

Siglas

METI Mestrado em Engenharia de Telecomunicações e Informática.

PDU Protocol Data Unit.

PG Pós-Graduação.

RTP Real-time Transport Protocol.

TCP Transmission Control Protocol.

UC Unidade Curricular.

UDP User Datagram Protocol.

1 Introdução

Este relatório faz parte do segundo trabalho prático da Unidade Curricular (UC) Serviços de Rede e Aplicações Multimédia, do 2º semestre do 1º ano do Mestrado em Engenharia de Telecomunicações e Informática. Este projeto foi desenvolvido como resposta a um problema apresentado pelo docente.

O sistema é composto por três componentes principais: o servidor, o reencaminhador e o cliente. O servidor gera a informação multimédia a um determinado ritmo e transmite-a continuamente para o reencaminhador, que a reenvia para o cliente. O cliente reproduz a informação ao ritmo em que é gerada pelo servidor e com um atraso máximo definido pelo servidor.

O sistema é projetado para ser atómico, assíncrono, inseguro e não fiável, com um Protocol Data Unit (PDU) que contenha toda a informação de controlo e de dados necessária para implementar os requisitos da interação dos componentes. O protocolo de transporte deve ser encapsulado num datagrama User Datagram Protocol (UDP) e deve ser definido e explicado em detalhe no relatório do trabalho.

2 Revisão da Literatura

Este capítulo apresenta os conceitos teóricos necessários ao desenvolvimento deste projeto, assim como a respetiva revisão da literatura.

1. Protocolos de Transporte de Informação Multimédia: A transmissão de vídeo e áudio digital pela Internet apresenta desafios únicos devido à necessidade de manter a sincronização temporal e minimizar atrasos. Os protocolos aplicacionais, como o Real-time Transport Protocol (RTP), frequentemente utilizam o UDP em vez do Transmission Control Protocol (TCP) devido à menor latência e à maior eficiência na transmissão contínua de dados, sem a necessidade de retransmissão de pacotes perdidos.
2. Modelos de Sistemas Multimédia: Os sistemas multimédia geralmente consistem num ou mais servidores e diversos clientes. A utilização de buffers pelos clientes é essencial para gerir a diferença entre o ritmo de transmissão do servidor e o ritmo de consumo no cliente, possibilitando uma experiência de reprodução contínua. A quantidade de dados que pode ser armazenada no buffer depende do atraso máximo permitido pelo tipo de serviço e pela diferença entre os ritmos de transmissão e consumo.
3. Objetivos do Sistema de Streaming: O objetivo principal do trabalho é desenvolver um sistema simples de streaming entre um servidor e um cliente, com um componente intermediário chamado reencaminhador. Este sistema deve simular as condições de atraso e perda de pacotes típicas da rede. O servidor gera frames multimédia em intervalos regulares e o cliente deve reproduzi-las sincronizadamente, considerando o atraso máximo especificado pelo servidor.

3 Abordagem Metodológica

A abordagem metodológica deste trabalho realizado em Java consiste nos seguintes pontos principais:

1. Construção de um Sistema de Streaming

- Desenvolver um sistema de streaming simples entre um servidor e um cliente.
- O servidor gera informações multimédia a um ritmo específico e transmite continuamente para o cliente.
- O cliente deve reproduzir a informação no ritmo em que é gerada pelo servidor, respeitando um atraso máximo definido pelo servidor.

2. Integração de um Reencaminhador

- Implementar um componente intermediário, o reencaminhador, que recebe a informação do servidor e a reenvia para o cliente.
- O reencaminhador simula o atraso da rede ao transmitir a informação a um ritmo igual ou inferior ao ritmo de receção.
- Este componente pode também simular a perda de PDU.

3. Criação de Informação Multimédia

- O servidor cria frames de informação a cada passo do relógio digital (por exemplo, a cada segundo).
- Cada frame contém uma sequência de imagens que representam os dígitos da hora atual.
- As frames são enviadas encapsuladas em PDUs.

4. Implementação do Cliente

- O cliente recebe as frames encapsuladas em PDUs e gere um buffer de entrada.
- A reprodução da informação é feita no tempo exato baseado no ritmo de reprodução e atraso máximo permitido.

5. Codificação e Protocolo de Transmissão

- Definição de um PDU que contenha toda a informação de controlo e de dados necessária.
- O protocolo deve ser atómico (informação de uma frame num único PDU), assíncrono, inseguro e não fiável.
- Utilização de UDP para a transmissão dos PDUs.

Esta abordagem assegura que todos os componentes do sistema multimédia são desenvolvidos e testados de forma estruturada, garantindo a sincronização e a transmissão eficiente das informações do servidor para o cliente através do reencaminhador.

A estrutura do PDU está representado na Tabela 1. Onde:

Tabela 1: Estrutura do PDU.

PDU
F (int)
A (int)
data (ArrayList)
string (String)
timeSentMillis (long)

- F é um inteiro que representa a precisão do relógio.
- A é um inteiro que representa o atraso máximo que é aconselhado ao cliente na reprodução.
- data é uma lista de arrays de bytes que armazena os dados da mensagem, neste caso as imagens dos dígitos.
- string é uma string que armazena uma mensagem textual (opcional - foi utilizado para casos de teste).
- timeSentMillis é um tempo em milissegundos que representa o tempo em que a mensagem foi enviada, usado para o cliente calcular o atraso.

4 Trabalho Realizado

Neste capítulo, são descritos detalhadamente a lógica de funcionamento do sistema desenvolvido, com ênfase na implementação de cada classe (um servidor, um reencaminhador, e um cliente).

4.1 Reencaminhador

O reencaminhador é projetado para receber dados de um servidor e repassá-los a um cliente. A função principal é atuar como um intermediário entre o servidor e o cliente.

- 1. Inicialização:** O reencaminhador inicia um “DatagramSocket” na porta “54321” para comunicar com o cliente e define o endereço do servidor (“127.0.0.1” na porta “12345”).

Código 1: Código do “DatagramSocket”.

```
1 forwarderSocket = new DatagramSocket(FORWARDER_PORT);
2 serverAddress = InetAddress.getByName("127.0.0.1");
3 System.out.println("Forwarder started on port: " + FORWARDER_PORT);
```

2. Inscrição do Cliente:

- Espera que um cliente envie uma mensagem de inscrição.
- Quando recebe a mensagem, armazena o endereço e a porta do cliente.

Código 2: Código da Inscrição do Cliente.

```
1 byte[] subscriptionBuffer = new byte[256];
2 DatagramPacket subscriptionPacket = new
  ↳ DatagramPacket(subscriptionBuffer, subscriptionBuffer.length);
3 forwarderSocket.receive(subscriptionPacket);
4 clientAddress = subscriptionPacket.getAddress();
5 int clientPort = subscriptionPacket.getPort();
6 System.out.println("Client subscribed from: " + clientAddress + ":" +
  ↳ clientPort);
```

- 3. Inscrição do Servidor:** Envia uma mensagem de inscrição ao servidor para começar a receber dados.

Código 3: Código da Inscrição do Servidor.

```
1 subscribeToServer(); // Subscribe to the server
```

4. Receção e Reencaminhamento de Dados:

- Entra num loop onde recebe pacotes de dados do servidor.
- Implementa uma lógica para descartar pacotes a cada 30 segundos ("M").
- Armazena os pacotes num buffer.
- Se o buffer atinge 10 pacotes ("N"), pausa por 2 segundos ("P"), e depois envia todos os pacotes do buffer para o cliente.
- Se o buffer não está cheio, envia imediatamente o pacote recebido para o cliente.

Código 4: Código de Receção e Reencaminhamento de Dados.

```

1 while (true) {
2     byte[] buffer = new byte[2048 * 15];
3     DatagramPacket packet = new DatagramPacket(buffer,
4         ↪ buffer.length);
5     serverSubscriptionSocket.receive(packet);
6     long currentTime = System.currentTimeMillis();
7     if (currentTime - lastIgnoreTime >= M * 1000) {
8         lastIgnoreTime = currentTime;
9         System.out.println("Ignoring PDU");
10        continue;
11    }
12    this.buffer.add(packet.getData());
13    if (this.buffer.size() >= N) {
14        System.out.println("Pausing for " + P + " seconds");
15        Thread.sleep(P * 1000);
16        while (!this.buffer.isEmpty()) {
17            byte[] data = this.buffer.poll();
18            DatagramPacket forwardPacket = new DatagramPacket(data,
19                ↪ data.length, clientAddress, clientPort);
20            forwarderSocket.send(forwardPacket);
21        }
22    } else {
23        DatagramPacket forwardPacket = new
24            ↪ DatagramPacket(packet.getData(), packet.getLength(),
25            ↪ clientAddress, clientPort);
26        forwarderSocket.send(forwardPacket);
27    }
28 }

```

4.2 Servidor

O servidor envia periodicamente pacotes contendo imagens e informações de tempo para clientes conectados. O funcionamento é descrito abaixo:

1. **Inicialização:** O servidor inicia um "DatagramSocket" na porta "12345" e lê uma série de imagens de dígitos (0-9 e separador) para um "ArrayList".

Código 5: Código da Inicialização.

```

1 private static final int PORT = 12345;
2 private DatagramSocket serverSocket;
3 private static final int F = 0;
4 private static final int A = 2;
5 private ArrayList<byte[]> digits = new ArrayList<>();
6 private String[] imagePaths = new String[] {"zero.png", "um.png",
↪ "dois.png", "tres.png", "quatro.png", "cinco.png", "seis.png",
↪ "sete.png", "oito.png", "nove.png", "separador.png"};

```

2. Espera das Conexões dos Clientes:

- Entra num loop aguardando que clientes enviem mensagens de inscrição.
- Quando um cliente inscreve-se, cria um novo “ServerHandler” para lidar com a comunicação com este cliente num novo thread.

Código 6: Código da Espera das Conexões dos Clientes.

```

1 while (true) {
2     try {
3         byte[] buffer = new byte[256];
4         //Wait until a client requests to "subscribe" the server
5         DatagramPacket request = new DatagramPacket(buffer,
↪ buffer.length);
6         serverSocket.receive(request);
7         //Get client "socket" informations
8         InetAddress clientAddress = request.getAddress();
9         int clientPort = request.getPort();
10        System.out.println("Client Connected from: " + clientAddress
↪ + ":" + clientPort);
11        // Handle the incoming clients on a new thread
12        ServerHandler HandlerThread = new ServerHandler(serverSocket,
↪ clientAddress, clientPort, getF(), getA(), getDigits());
13        new Thread(HandlerThread).start();
14    } catch (Exception e) {e.printStackTrace();}
15 }

```

3. Manipulação de Clientes (Em cada “ServerHandler”):

- Armazena informações do cliente (endereço e porta).
- Define o formato do tempo com base em uma precisão (“F”).
- Num loop, envia pacotes de dados ao cliente em intervalos definidos por “F”.

Código 7: Código da Manipulação de Clientes.

```

1 public ServerHandler(DatagramSocket serverSocket, InetAddress
  ↪ clientAddress, int clientPort, int F, int A, ArrayList<byte[]>
  ↪ digits) {
2     this.serverSocket = serverSocket;
3     this.clientAddress = clientAddress;
4     this.clientPort = clientPort;
5     this.F = F;
6     this.A = A;
7     this.digits = digits;
8     this.formatter = FormatDecission();
9 }

```

4. Envio de dados (A cada intervalo de tempo (“F”), o “ServerHandler”):

- Obtém o tempo atual e formata-o.
- Cria uma PDU contendo o tempo formatado e as imagens dos dígitos.
- Serializa a PDU e envia ao cliente.

Código 8: Código do Envio dos Dados.

```

1 while (true) {
2     ByteArrayOutputStream baos = new ByteArrayOutputStream();
3     ObjectOutputStream oos = new ObjectOutputStream(baos);
4     long end = System.currentTimeMillis();
5     if (end - start >= sleepIntervalls[this.F]) {
6         LocalTime now = LocalTime.now();
7         PDU toSend = new PDU(this.F, this.A);
8         String formattedTime = this.formatter.format(now);
9         char[] charArray = formattedTime.toCharArray();
10        toSend.addString(formattedTime);
11        System.out.println(toSend.getString());
12        for (char c : charArray) {
13            toSend.addDigit(digits.get(c - '0'));
14        }
15
16        toSend.setTimeSent(); oos.writeObject(toSend); oos.flush();
17
18        byte[] serializedPDU = baos.toByteArray();
19        DatagramPacket packet = new DatagramPacket(serializedPDU,
20            ↪ serializedPDU.length, clientAddress, clientPort);
21        serverSocket.send(packet);
22
23        baos.close(); oos.close(); start = end;
24    }
}

```


4.3 Cliente

O cliente recebe dados (incluindo imagens e informações de tempo) de um servidor (ou de um reencaminhador) e exibe esses dados numa interface gráfica. As funções do cliente:

- **Configuração da Interface Gráfica**

- Cria uma janela JFrame com título “CLOCK” e define o layout como FlowLayout.
- Configura o fundo da janela para preto.
- Adiciona vários JLabel ao frame para exibir as imagens dos dígitos do relógio.
- Define o tamanho da janela e o comportamento de fechamento.

Código 9: Código da Configuração da Interface Gráfica.

```

1 public Cliente() throws IOException, ClassNotFoundException,
   ↳ InterruptedException, InvocationTargetException {
2     setTitle("CLOCK");
3     setLayout(new FlowLayout());
4     getContentPane().setBackground(Color.BLACK);
5     add(Hour1Label);
6     add(Hour2Label);
7     add(HourMinuteSeparatorLabel);
8     add(Minute1Label);
9     add(Minute2Label);
10    add(MinuteSecondSeparatorLabel);
11    add(Second1Label);
12    add(Second2Label);
13    add(SecondDeciSeparatorLabel);
14    add(DeciLabel);
15    add(CentiLabel);
16    add(MilliLabel);
17    labels.add(Hour1Label);
18    labels.add(Hour2Label);
19    labels.add(HourMinuteSeparatorLabel);
20    labels.add(Minute1Label);
21    labels.add(Minute2Label);
22    labels.add(MinuteSecondSeparatorLabel);
23    labels.add(Second1Label);
24    labels.add(Second2Label);
25    labels.add(SecondDeciSeparatorLabel);
26    labels.add(DeciLabel);
27    labels.add(CentiLabel);
28    labels.add(MilliLabel);
29    setSize(600, 300);
30    setVisible(true);
31    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
32    receiveData();
33 }

```

- **Receção de Dados:**

- Inicia um novo thread para receber dados do reencaminhador.
- Conecta-se ao reencaminhador usando DatagramSocket.
- Envia um pacote de inscrição ao reencaminhador.
- Entra num loop para receber pacotes de dados.

- **Processamento de Dados:**

- Ao receber um pacote, desserializar o objeto PDU.
- Verifica a validade dos dados baseando-se no tempo ("A").
- Atualiza as imagens dos JLabel na interface gráfica usando ImageIO e Swing.

Código 10: Código da Receção e Processamento de Dados.

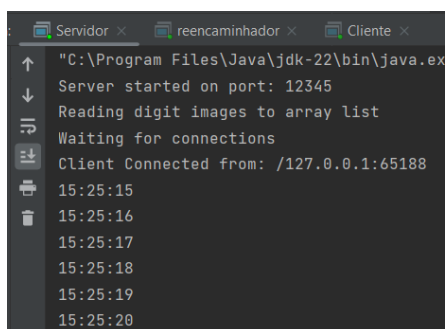
```

1 String hostname = "127.0.0.1"; int port = 54321; //forwarder port
2 InetAddress address = InetAddress.getByName(hostname);
3 DatagramSocket socket = new DatagramSocket();
4 byte[] buffer = new byte[2];
5 DatagramPacket request = new DatagramPacket(buffer, buffer.length,
   ↪ address, port);
6 socket.send(request);
7 while(true){
8     byte[] bufferResponse = new byte[2048 * 15];
9     DatagramPacket response = new DatagramPacket(bufferResponse,
   ↪ bufferResponse.length);
10    socket.receive(response);
11    System.out.println("Resposta recebida");
12    ByteArrayInputStream bais = new
   ↪ ByteArrayInputStream(response.getData());
13    ObjectInputStream ois = new ObjectInputStream(bais);
14    PDU deserializedObject = (PDU) ois.readObject();
15    bais.reset(); ois.close();
16    if(deserializedObject.getA() >= ((System.currentTimeMillis()
   ↪ -deserializedObject.getTimeSentMillis())/1000) % 60) {
17        System.out.println(deserializedObject.getString());
18        for (int i = 0; i < deserializedObject.getData().size(); i++)
   ↪ {
19            img = ImageIO.read(new
   ↪ ByteArrayInputStream(deserializedObject.getData()
   ↪ .get(i)));
20            final ImageIcon nextIcon = new ImageIcon(img);
21            img.flush(); int finalI = i;
22            java.awt.EventQueue.invokeLater(() -> {
23                labels.get(finalI).setIcon(nextIcon);
24                labels.get(finalI).repaint();
25            });
26    } } }

```

5 Testes

Este capítulo apresenta os resultados obtidos com a implementação do código desenvolvido. Nos testes, é possível verificar que o servidor, Figura 1, envia com sucesso as imagens de cada um dos dígitos que formam a hora obtida. Já no reencaminhador, Figura 2, observa-se que ele ignora alguns PDUs e injeta atrasos proporcionalmente, conforme planeado. Finalmente, no cliente, Figura 3, as respostas são recebidas com sucesso. Para observar melhor nos testes quando o cliente ignora as respostas devido ao tempo máximo de atraso permitido ter sido excedido, ele não imprime a hora na consola, mas informa que a resposta foi recebida.

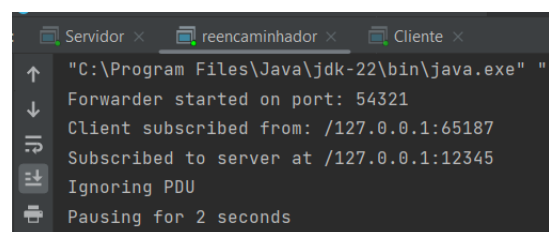


```

C:\Program Files\Java\jdk-22\bin\java.exe
Server started on port: 12345
Reading digit images to array list
Waiting for connections
Client Connected from: /127.0.0.1:65188
15:25:15
15:25:16
15:25:17
15:25:18
15:25:19
15:25:20

```

Figura 1: Resultados Servidor.



```

C:\Program Files\Java\jdk-22\bin\java.exe
Forwarder started on port: 54321
Client subscribed from: /127.0.0.1:65187
Subscribed to server at /127.0.0.1:12345
Ignoring PDU
Pausing for 2 seconds

```

Figura 2: Resultados Reencaminhador.

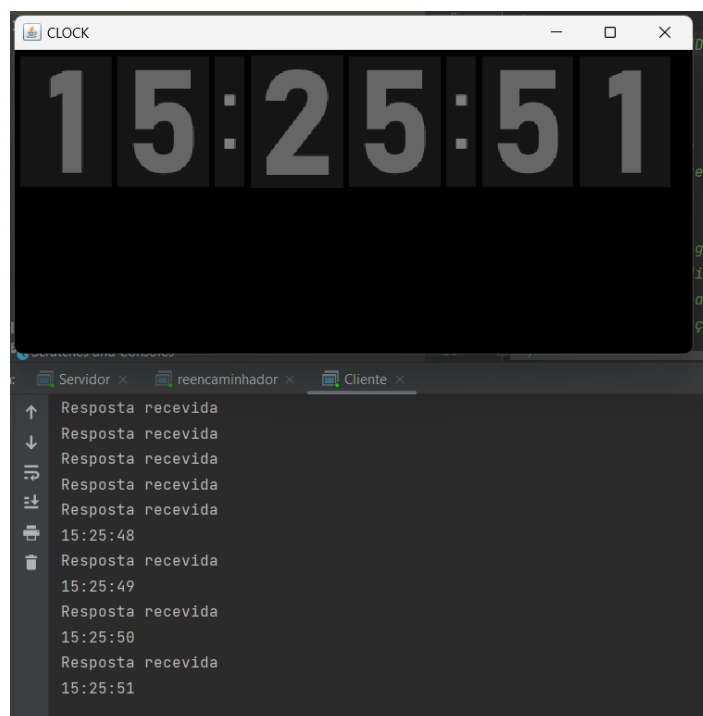


Figura 3: Resultados Cliente.

6 Conclusão

O desenvolvimento deste projeto permitiu a criação de um sistema de streaming multimédia simples, composto por um servidor, um reencaminhador e um cliente. Cada componente desempenha um papel crucial na geração, transmissão e reprodução de informações multimédia, mantendo a sincronização e respeitando o atraso máximo definido pelo servidor.

O servidor foi responsável por gerar frames de informação multimédia em intervalos regulares e enviá-las encapsuladas em PDUs para o reencaminhador. O reencaminhador, por sua vez, simulou as condições de atraso e perda de pacotes típicas da rede, repassando os dados para o cliente de forma controlada. O cliente recebeu e reproduziu as informações, gerindo um buffer de entrada para assegurar a reprodução contínua e sincronizada dos dados recebidos.

Os testes realizados confirmaram a funcionalidade e a eficiência do sistema desenvolvido. O servidor enviou com sucesso as imagens dos dígitos que formam a hora, o reencaminhador injetou atrasos e ignorou PDUs conforme planeado, e o cliente recebeu e reproduziu os dados corretamente. Além disso, o sistema conseguiu simular condições de rede não fiáveis e inseguras, demonstrando robustez ao lidar com perdas de pacotes e atrasos na transmissão.

Em suma, este projeto alcançou os objetivos propostos, fornecendo uma implementação prática dos conceitos teóricos relacionados à transmissão de informação multimédia pela Internet. A utilização de UDP como protocolo de transporte foi justificada pela menor latência e maior eficiência na transmissão contínua de dados, corroborando com a literatura revisada. Este trabalho contribui para o entendimento e aplicação dos princípios de sistemas de streaming multimédia, oferecendo uma base sólida para futuros aprimoramentos e pesquisas na área.