Universidade do Minho

Comunicação por Computador

Departamento de Informática

Trabalho Prático 2

Grupo 09:
João Abreu
Hugo Matias

Número:
A84802
A85370

24 de Maio de 2020

Conteúdo

1. Introdução

Este projeto surge no âmbito da Unidade Curricular de Comunicações por Computador, cujo objetivo é criar um serviço de transferência de dados sobre TCP e UDP de modo a manter o anonimato da origem do pedido de transferência. No projeto desenvolvido existem 4 componentes essenciais, sendo estas o cliente, o servidor e os 2 anonimizadores (AnonGW) escolhidos aleatoriamente. O servidor será um servidor HTTP **mini-http** de modo que só aceitará ligações TCP na porta 80, e o cliente enviará um pedido de transferência de ficheiro usando, por exemplo, o comando **wget**, que também requer uma ligação TCP com o target. Ou seja, tanto a ligação do cliente com o seu target (AnonGW) e a ligação da entidade aleatória (AnonGW) com o servidor target tem de ser TCP. Mas a ligação entre AnonGW's não necessita de ser TCP, na verdade o projeto foi desenvolvido usando UDP pois apresenta uma maior versatilidade e eficiência. Como, ao contrário do TCP, este não garante a ordem de chegada dos pacotes, tivemos de implementar essa propriedade no projeto.

2. Especificação do Protocolo

2.1 Estrutura do PDU

Para a estrutura do PDU, foi implementada a classe PDU com as seguintes variáveis de instância e métodos:

```
private int seqNumber;              /* Para sabermos a ordem de pacotes         */
private int isResposta;             /* Para sabermos se é uma resposta          */
private int isLast;                 /* Para sabermos se é o último pacote       */
private String target_response;     /* Para sabermos a quem temos de responder  */
private byte[] fileData;            /* Para sabermos o conteúdo do pacote       */

public byte[] toBytes() {
    byte[] utf8_host = this.target_response.getBytes(StandardCharsets.UTF_8);
    ByteBuffer packet = ByteBuffer.allocate(20 + this.fileData.length);
    packet.put(ByteBuffer.allocate(4).putInt(this.seqNumber).array());
    packet.put(ByteBuffer.allocate(4).putInt(this.isResposta).array());
    packet.put(ByteBuffer.allocate(4).putInt(this.isLast).array());
    packet.put(utf8_host); // sempre 8 bytes
    packet.put(this.fileData);

    return packet.array();
}

public void fromBytes(byte[] pdu, int length){
    seqNumber = ByteBuffer.wrap(Arrays.copyOfRange(pdu, 0, 4)).getInt();
    isResposta = ByteBuffer.wrap(Arrays.copyOfRange(pdu, 4, 8)).getInt();
    isLast = ByteBuffer.wrap(Arrays.copyOfRange(pdu, 8, 12)).getInt();
    target_response = new String(ByteBuffer.wrap(Arrays.copyOfRange(pdu, 12,  20)).array());
    fileData = ByteBuffer.wrap(Arrays.copyOfRange(pdu, 20, length)).array();
}
```

O primeiro método transforma os valores das variáveis de instância para o tipo byte, aloca espaço para estes num ByteBuffer criado neste mesmo método, e por fim mete no espaço alocado corresponde o valor em bytes.

O segundo método é o oposto do primeiro, retira os valores, em bytes, correspondentes às variáveis de instância do ByteBuffer tendo em conta o espaço ocupado bem como a posição destes valores no array, deste modo é possível saber a que variável estes correspondem.

3. Implementação

3.1 Arquitetura da solução

O nosso problema está estruturado pela comunicação Cliente-Servidor. Deste modo, o Cliente irá indicar a porta e ip destino com a qual pretender iniciar uma troca de informação, conectando-se assim a um AnonGW. Por exemplo, sendo o Portátil 2 um dos AnonGW's e tendo este o IP **10.1.1.2**, como podemos ver na figura, esta ligação ia ser feita da seguinte maneira: **wget http://10.1.1.2/ficheiro_a_ser_transferido**.

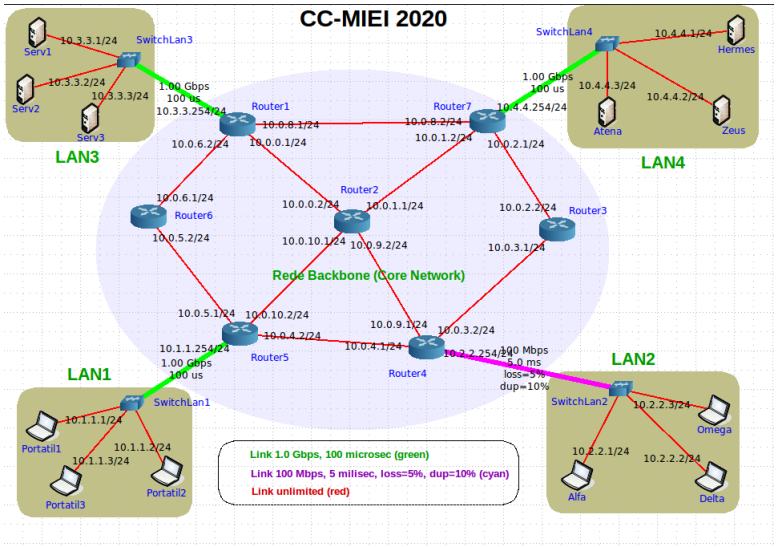

Figura 3.1: Topologia Core.

3.2 Estrutura de dados

A implementação dos conceitos referidos acima foi feita com recurso a linguagem orientada aos objectos Java. Recorrendo a esta linguagem foram implementadas as seguintes classes repartidas pela sua função:

3.2.1 AnonGW

- AnonGW - Classe principal onde recebe os argumentos.
- AnonGWWorker - Classe que cria 2 threads, uma tratará das conexões TCP e a outra das UDP, tendo em conta os argumentos recebidos na classe AnonGW.

3.2.2 TCP

- TcpReceiver - Está à escuta de ligações TCP. Dá inicio ao processo de comunicação entre o cliente e o AnonGW escolhido.
- TcpProxy - Recebe a informação do cliente e envia para o AnonGW (escolhido aleatoriamente).

3.2.3 UDP

- PDU - Classe que implementa a estrutura PDU que será usada para transferir os pacotes.
- UcpReceiver - Está à escuta de ligações UDP. Recebe informação de um Anon e dá início ao processo de comunicação entre Anon e servidor
- UdpProxy - Envia pacotes no sentido Anon –> Server e Server –> Anon.

3.3 Encriptação

Os dois principais e mais simples algoritmos da encripatção em bloco são o DES e o AES. Optamos por escolher o DES pois entendemos que seria mais fácil de implementar. Sabemos que o algoritmo DES é hoje considerado inseguro pois foi desenvolvido há mais de 20 anos. Apesar disso, nestes 20 anos não foi encontrada uma forma de o quebrar, exceto pela força bruta. AES também superou DES e é escolhido em várias aplicações onde o DES não se mostra adequadamente seguro.

A informação sobre criptografia acima e como a implementamos que está referida em baixo pode-se encontrar no capítulo 5.

Inicialmente criamos uma instância de Cipher e especificamos o nome do algoritmo, o modo e o esquema de padding que é opcional, tudo separado por uma barra "/"conforme o código abaixo:

```
Cipher cifraDES;
// Cria a cifra
cifraDES = Cipher.getInstance("DES/ECB/PKCS5Padding");
```

Para gerarmos a chave que vai ser utilizada na encriptação e decifragem conforme o algoritmo de encriptação escolhido, utilizamos o método KeyGenerator.getInstance("DES") e generateKey():

```
KeyGenerator keygenerator = KeyGenerator.getInstance("DES");
SecretKey chaveDES = keygenerator.generateKey();
```

Para criptografar a informação utilizamos o método Cipher.doFinal(), conforme mostra o código abaixo:

```
// Inicializa a cifra para o processo de encriptação
cifraDES.init(Cipher.ENCRYPT_MODE, chaveDES);
// Texto encriptado
byte[] textoEncriptado = cifraDES.doFinal(informacao);
```

Por fim, utilizamos o método Cipher.doFinal() em modo de decriptação para descriptografar a informação conforme o código destacado abaixo:

```
// Inicializa a cifra também para o processo de decriptação
cifraDES.init(Cipher.DECRYPT_MODE, chaveDES);
// Decriptografa o texto
byte[] textoDecriptografado = cifraDES.doFinal(textoEncriptado);
```

De modo a ser possível a encriptação e decifragem em múltiplos locais optamos por enviar a chave também pelo PDU. Esta é gerada inicialmente na ligação TCP cliente - AnonGW juntamente com a cifra. A mensagem é desincriptada nas ligações com o cliente e servidor, e encriptada entre ligações UDP entre Anon's.

4. Testes e Resultados

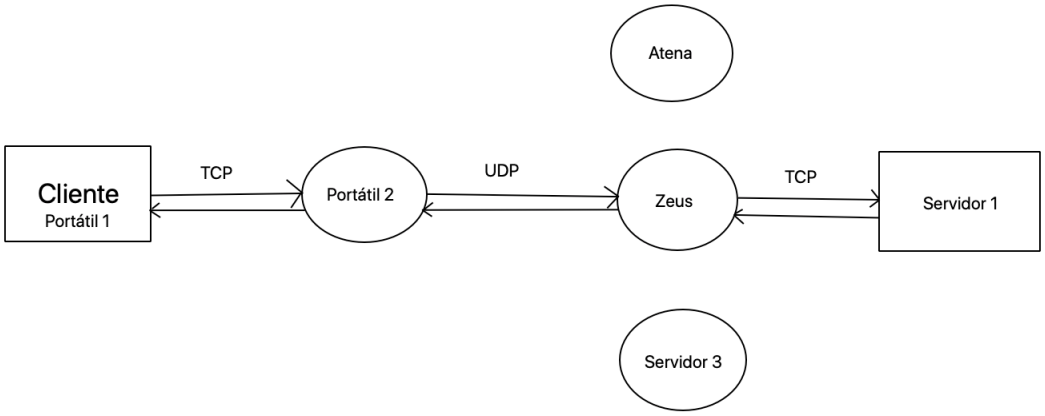

Figura 4.1: Visualização da demonstração.

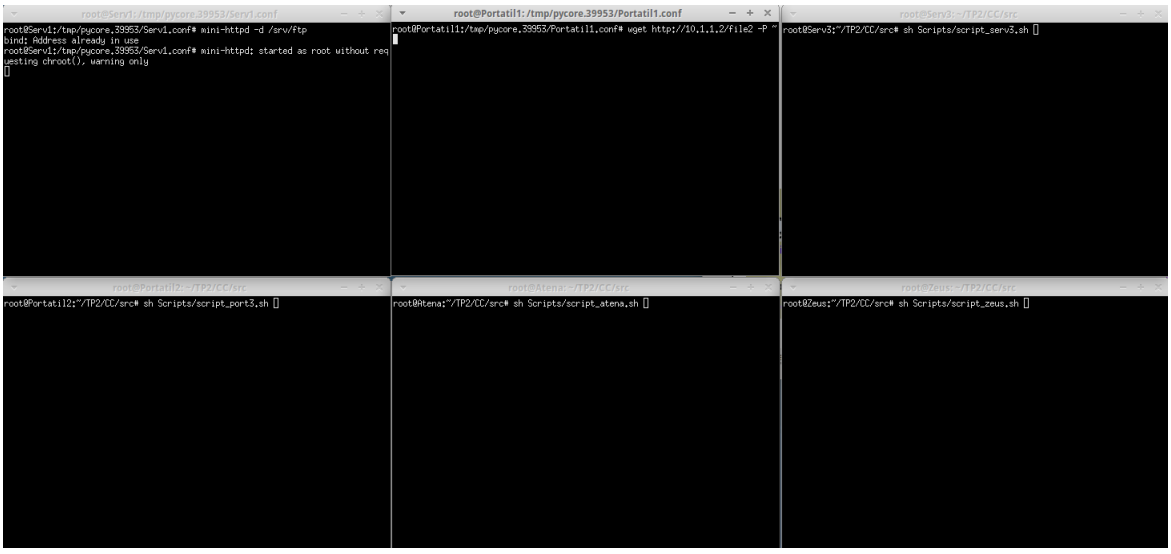

Figura 4.2: Antes da execução.

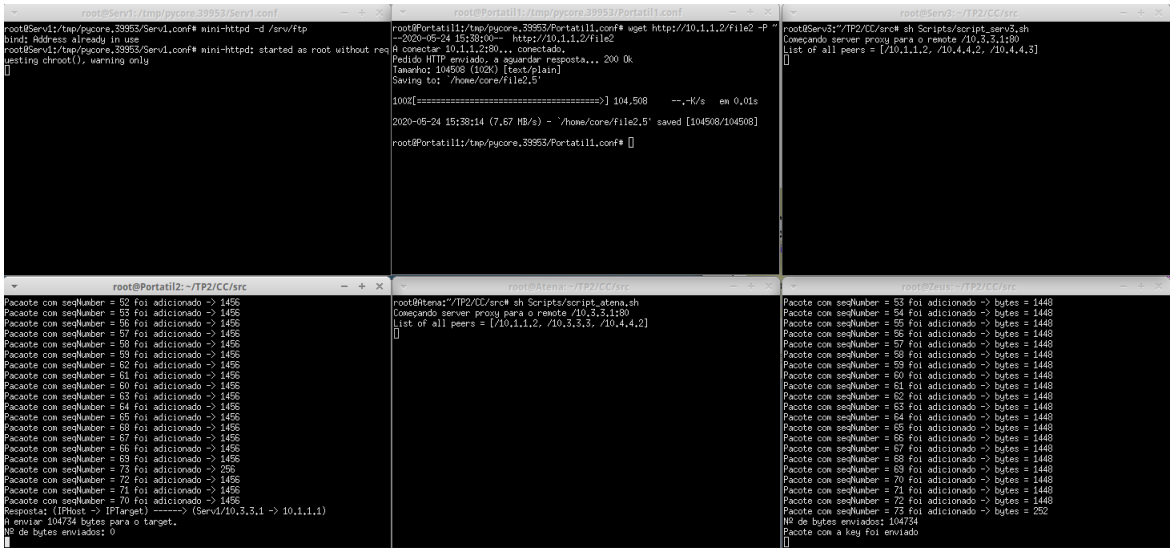

Figura 4.3: Depois da execução.

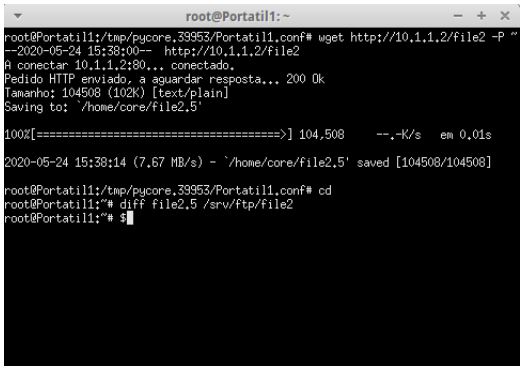

Figura 4.4: Demonstração bytes foram bem transferidos.

5. Referências

Encriptação: https://www.devmedia.com.br/utilizando-criptografia-simetrica-em-java/31170

6. Conclusão

Este foi um projeto acessível apesar da dificuldade de inicialização e estruturação do método e modo de implementação do que era pedido. Assim que esta fase inicial foi ultrapassada, a dificuldade diminui significativamente.

Implementamos as funcionalidades propostas para o bom funcionamento da troca de informação mantendo o anonimato da origem.

Consideramos assim que a estrutura do nosso projeto é razoavelmente positiva, tanto no método de encriptação como na troca de informação.

À medida que fomos fazendo este projeto, conseguimos entender melhor as grandes diferenças entre TCP e UDP, tanto como as vantagens e limitações de cada um. Também conseguimos aprofundar os nossos conhecimentos de encriptação tal como na estruturação e envio de pacotes com dados. Para uma futura melhoria ao projeto desenvolvido gostaríamos de implementar o um algoritmo de encriptação mais seguro e o controlo de perda de pacotes.