

Rede Overlay de Anonimização do Originador

Adriana Meireles (A82582), Mariana Pereira (A81146), and Helena Martins (A82500)

¹ Universidade do Minho

² Mestrado Integrado em Engenharia Informática

³ Comunicações por Computador

Resumo No âmbito da Unidade Curricular de Comunicações por Computador, foi-nos pedida a implementação de uma rede overlay que conseguisse a anonimização do originador.

Keywords: Conexões UDP · Conexões TCP · anonimização · Protocolo PDU

1 Introdução

Para a unidade curricular de Comunicações por Computador, foi-nos proposta a realização do trabalho prático número dois –Rede Overlay de anonimização do originador – que consiste em desenhar um protocolo que funciona sobre UDP e que garanta a entrega ordenada e confidencial de informação de várias conexões de transporte TCP.

Neste trabalho pretende-se incluir todos os conhecimentos adquiridos ao longo das aulas, nomeadamente, o mecanismo de camada de Transporte tanto UDP como TCP.

Neste relatório iremos apresentar a implementação para dar resposta ao problema proposto. Para tal utilizamos Java como linguagem e o CORE como ferramenta de testes.

2 Arquitetura da Solução

Para a resolução do problema apresentado, começamos por implementar todas as ligações da sequência de transferência, tanto as ligações TCP como as UDP.

Primeiro implementamos as ligações TCP. Estas ligações são feitas apenas entre o Cliente - **Origin** - e o primeiro nodo da rede Overlay de anonimização e entre o Servidor - **TargetServer** - e o último nodo dessa rede.

Só posteriormente é que foi implementado o túnel UDP. Este túnel é constituído por várias instâncias do gateway de transporte, por nós designado de **AnonGW**. Aqui, o conteúdo é transferido para um dos 3 *peers* existentes, que é escolhido de forma aleatória, de modo a que o servidor do destino não conheça o cliente que fez o pedido.

Depois de implementadas as conexões, o objetivo seria conseguir a transferência de dados e a multiplexagem de clientes.

3 Especificação do Protocolo UDP

O protocolo UDP - *User Datagram Protocol* - é um protocolo simples da camada de transporte que permite que uma aplicação envie datagramas encapsulados em pacotes para um determinado destino. Um dos objetivos deste projeto seria implementar este protocolo de modo a conseguirmos transferir dados entre o cliente e o servidor.

3.1 Implementação da PDU

Para que fosse possível a transferência de dados entre clientes e servidores, incorporamos a PDU - *Protocol Data Unit* - de modo a que a informação fosse dividida em pacotes. Cada PDU é constituída por um número de sequência e um array que vai conter a informação transferida convertida em bytes.

Resumidamente, toda a informação que um *AnonPeerT* pretender enviar, vai ser dividida em diversos blocos com um tamanho fixo de 100. Cada bloco de informação vai ser convertido num array de bytes e posteriormente encaminhado para o *AnonGW* através da rede de anonimização. No final da sequência de transferência, quando estes pacotes chegarem ao destino, que é o *AnonGW*, vão ser convertidos ao estado inicial e reencaminhados ordenadamente para o *Cliente*, considerando para tal o número de sequência.

4 Implementação

A aplicação pode ser dividida em três partes.

- A ligação TCP entre a *Origin* e o primeiro nodo da rede(*AnonGW*).
- O túnel UDP
- A ligação TCP entre o último nodo da rede(Um dos peers) e o *TargetServer*

Para o desenvolvimento desta aplicação foi necessária a junção de vários mecanismos que constituem estas três partes. Estes são apresentados nas diversas classes que apresentamos de seguida:

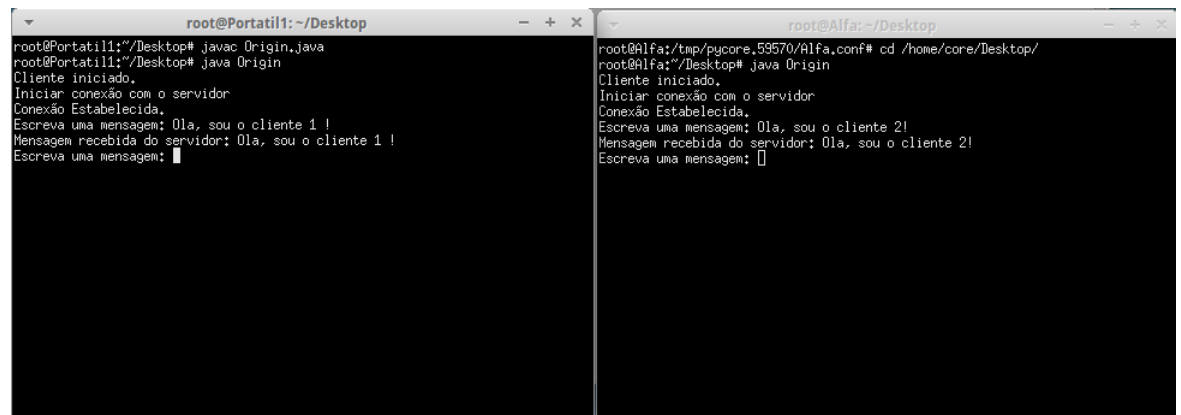
- **Origin** : A classe *Origin* é responsável pela ligação entre um cliente e o *AnonGW* cujo endereço IP é "10.1.1.3". Para tal, recorre a um socket TCP com porta de conexão 80.
- **AnonGW** : A classe *AnonGW* é responsável por executar o anon, também na porta 80. Aqui é inicializada a thread que trata das ligações com clientes via TCP, sendo possível a comunicação de vários clientes.
- **AnonGWT** : A classe *AnonGWT* deve aceitar ligações de clientes via TCP pela porta 80 e reencaminhar os pedidos de cada um para o respetivo *peer* que é selecionado aleatoriamente de um array com os 3 peers existentes cujo endereço IP pode ser "10.1.1.2", "10.4.4.2" ou "10.4.4.3". Foi criado um *InetAddress* a partir de um dos endereços anteriores, pela porta 6666. Aqui tratamos da conexão UDP que envia mensagens (comandos oriundos do cliente) para o peer recorrendo à classe *DatagramPacket*. Posteriormente, tratamos as mensagens via UDP provenientes do peer, de modo a que sejam entregues com a sua ordem inicial ao cliente.
- **AnonPeerT** : Esta classe é responsável pela ligação do peer ao servidor. Aqui são tratadas as mensagens via UDP provenientes do anon. De seguida, reencaminha toda a informação para o servidor cujo endereço IP é "10.3.3.1". Por último, recebe a resposta do servidor que envia para o anon.
- **TargetServer** : A classe *TargetServer* representa um servidor que se liga pela porta 80 de modo a "ouvir" os comandos recebidos. Este pode responder ao cliente, enviando a resposta pelo mesmo processo que o cliente lhe envia informação. Também recebe a ligação do peer escolhido através de uma conexão TCP e inicializa a thread responsável por processar os comandos enviados por ele.
- **TargetServerT** : A *TargetServerT* é uma classe idêntica à classe *AnonGWT*. É responsável por tratar dos comandos enviados pelo peer. O nosso servidor é capaz de ler o ficheiro inteiro sendo a sua função responder com o conteúdo do mesmo.

- **PDU** : A classe *PDU* tem como objetivo converter a informação que se pretende transferir desde o cliente até ao servidor em arrays de bytes, de modo a facilitar a sua transferência. Isto implica que os 'blocos' de informação não percam a sua ordem, ou seja, são entregues com a sua ordem inicial.

5 Testes e Resultados

De modo a testar o funcionamento da aplicação foi utilizada,mais uma vez, a topologia core.

Para testar o programa, foi utilizado o **Serv1** como servidor. De seguida, foram executadas 3 instâncias dos AnonPeer nas máquinas **Portátil2, Zeus** e **Atena**. Depois, executamos também a instância do AnonGW na máquina **Portátil3**. Por último, utilizamos as máquinas **Portátil1** e **Alfa** como clientes anonimizados (de notar que podem ser utilizados quantos clientes forem desejados).



```
root@Portatil1:~/Desktop
root@Portatil1:~/Desktop# javac Origin.java
root@Portatil1:~/Desktop# java Origin
Cliente iniciado.
Iniciar conexão com o servidor
Conexão Estabelecida.
Escreva uma mensagem: Ola, sou o cliente 1 !
Mensagem recebida do servidor: Ola, sou o cliente 1 !
Escreva uma mensagem:

root@Alfa:~/Desktop
root@Alfa:~/Desktop# cd /home/core/Desktop/
root@Alfa:~/Desktop# java Origin
Cliente iniciado.
Iniciar conexão com o servidor
Conexão Estabelecida.
Escreva uma mensagem: Ola, sou o cliente 2!
Mensagem recebida do servidor: Ola, sou o cliente 2!
Escreva uma mensagem:
```

Figura 1. Clientes Portátil1 e Alfa

Como podemos observar pela figura em baixo, o anon ligou o cliente1 ao peer Portátil2 e o cliente2 ao peer Zeus.

```

root@Portatil3: ~/Desktop
root@Portatil3:~/Desktop# javac AnonGW.java
root@Portatil3:~/Desktop# java AnonGW
Anon iniciado.
Iniciar conexão com o servidor
Conexão estabelecida
Número random: 0
Número do peer a qual vai ligar: 10.1.1.2
Mensagem recebida do cliente [10.1.1.1]: Ola, sou o cliente 1 !
numero0
Mensagem recebida do peer []: Ola, sou o cliente 1 !
Conexão estabelecida
Número random: 1
Número do peer a qual vai ligar: 10.4.4.2
Mensagem recebida do cliente [10.2.2.1]: Ola, sou o cliente 2!
numero0
Mensagem recebida do peer []: Ola, sou o cliente 2!

```

Figura 2. AnonGW

<pre> root@Atena:~/Desktop# java AnonPeerT.java root@Atena:~/Desktop# java AnonPeerT </pre>	<pre> root@Zeus:~/Desktop# java AnonPeerT Mensagem recebida do anonGW []: Ola, sou o cliente 2! cuseje Mensagem recebida do server [Serv1]: </pre>	<pre> root@Portatil2:~/Desktop# java AnonPeerT Mensagem recebida do anonGW []: Ola, sou o cliente 1 ! cuseje Mensagem recebida do server [Serv1]: </pre>
---	--	--

Figura 3. Peers: Atena,Zeus,Portátil2

```

root@Serv1:~/Desktop
root@Serv1:~/Desktop# javac TargetServer.java
root@Serv1:~/Desktop# java TargetServer
Servidor iniciado.
Aguardar Conexão de um cliente.
Conexão estabelecida
Conexão estabelecida
Conexão estabelecida
Mensagem recebida do cliente [10.1.1.2]: Ola, sou o cliente 1 !
Tamanho da mensagem do cliente [10.1.1.2]: 22
Mensagem enviada do cliente [10.1.1.2]: Ola, sou o cliente 1 !
Tamanho da mensagem do cliente [10.1.1.2]: 22
Mensagem recebida do cliente [10.4.4.2]: Ola, sou o cliente 2!
Tamanho da mensagem do cliente [10.4.4.2]: 21
Mensagem enviada do cliente [10.4.4.2]: Ola, sou o cliente 2!
Tamanho da mensagem do cliente [10.4.4.2]: 21

```

Figura 4. Servidor

6 Conclusões e Trabalho Futuro

Após a finalização deste projeto, que achamos bastante complexo, podemos concluir que não só foi necessário utilizar todo o conhecimento adquirido nesta unidade curricular ao longo do semestre, como também o diverso conhecimento que adquirimos ao longo do curso até ao momento. Apesar disto, consideramos que contribuiu para assimilar vários conceitos que podem ser úteis para o futuro.

No entanto, o resultado final ficou aquém das nossas expectativas, uma vez que não conseguimos implementar tudo o que nos era pedido, tal como a cifragem do conteúdo que se pretende transferir.

No que toca a trabalho futuro, poderíamos implementar a parte da cifragem do conteúdo e alguns dos pontos opcionais pedidos no enunciado, tais como o controlo de perdas e o controlo de congestão.