Rede Overlay de anonimização do originador

Ana Rosendo, Gonçalo Esteves, and Rui Oliveira

Universidade do Minho, Departamento de Informática, 4710-057 Braga, Portugal
e-mail: {a84475, a85731, a83610}@alunos.uminho.pt

Resumo Este relatório visa a documentação do trabalho prático n°2 na unidade curricular de Comunicações por Computador que se baseia numa rede overlay de anonimazação do originado, desenvolvido na linguagem Java.
Serão abordadas as decisões tomadas ao longo do desenvolvimento do projeto, bem como a sua estruração.

1 Introdução

No âmbito da unidade curricular de Comunicações por Computador foi-nos proposto desenhar e implementar uma rede overlay de anonimização do originador. O trabalho desenvolvido dividiu-se em duas parte, sendo que na primeira foi necessário:

1. Criar o programa principal com os parâmetros adequados
2. Implementar as componentes TCP (cliente/servidor)
3. Definir as estruturas de dados internas
4. Implementar a lógica de gateway de transporte genérico

Na segunda fase do trabalho foi necessário:

1. Especificar o protocolo (formato PDU: sintaxe e semântica)
2. Desenhar e implementar os PDU Anon Protocol
3. Implementação das componentes UDP e Anon Protocol

Para ambas as fases, foi necessário recorrer a vários testes de maneira a podermos comprovar as funcionalidades anteriores, sendo que além da linguagem Java, utilizamos o CORE como ferramenta de testes.

2 Arquitetura da solução

Inicialmente, idealizamos o que seria necessário para chegar ao resultado pretendido e delineamos a arquitetura do mesmo:

- Uma classe para tratar do Cliente.
- Uma classe para tratar do Servidor.
- Uma classe para tratar do AnonGW.
- Uma classe para tratar das informaçãos relativas ao PDU.

3 Especificação do protocolo UDP

As nossas mensagens protocolares são construídas a partir de strings de informação. Numa primeira fase, transforma-se a string recebida diretamente num array de bytes. Em seguida, divide-se esse mesmo array num conjunto de arrays menores, designados por blocos, adicionando no inicio do array o valor da posição relativa do bloco, por forma a posteriormente ser possível ordenar os blocos. Todos os blocos têm o mesmo tamanho (valor fixo, definido por nós como sendo 64). Assim, a informação fica pronta para ser enviada em partes mais pequenas.

Utilizamos este tipo de mensagens na troca de informação entre o segundo e o primeiro anon. Deste modo, quando o segundo anon recebe a resposta do comando passado pelo cliente e executado pelo servidor, este divide-a em diversos blocos, que serão passados ao primeiro anon.

4 Implementação

Para seguir a arquitetura delineada, criamos as seguintes classes:

4.1 Cliente

Na classe **Cliente** é tratada a ligação entre o mesmo e um dos anons passados como argumento (a escolha do anon a qual ligar é aleatória).

Posteriormente, recebe pelo terminal os comandos a executar e, de seguida, recorrendo a um socket TCP com porta de conexão 80, envia o mesmo ao anon escolhido.

4.2 AnonGW

Esta classe é responsável pela execução do anon, sendo inicializada com o endereço do servidor, porta de conexão e lista dos restantes anons ativos.

Feita a inicialização, são lançadas duas threads, uma para tratar das ligações com clientes via TCP e outra para lidar com a ligação ao servidor, também recorrendo a TCP. As threads comunicam uma com a outra através de UDP.

4.3 AnonGWThread2Cliente

Nesta classe, é passado como argumento a lista dos anons existentes para a sua inicialização. Tem a responsabilidade de aceitar ligações de Clientes via TCP pela porta de conexão 80 e, posteriormente, selecionando um anon aleatoriamente, lança uma nova thread para comunicar com o mesmo.

4.4 AnonGWThreadAux

Esta classe é utilizada para criar a thread invocada em *AnonGWThread2Cliente*, sendo que tem como pârametros o socket que ligou o anon ao cliente e o endereço IP do segundo anon com o qual vai comunicar.

Criamos um *InetAddress* a partir do endereço do segundo anon e tratamos da conexão UDP com que irá enviar as mensagens para o mesmo, pela porta 6666, com auxílio da classe *DatagramPacket*, mensagens estas que representam os comandos vindos do cliente. Depois, fica a espera da resposta do segundo anon, para reencaminhar as mensagens recebidas para o cliente.

4.5 AnonGWThread2Servidor

Esta classe é responsável pela ligação do anon ao servidor. Ela trata de lidar com a receção de mensagens via UDP, oriundas de outro anon, e, após isso, reencaminhar as mesmas ao servidor. Por fim, deverá reencaminhar as respostas que recebe do servidor para o anon que enviou o comando.

4.6 Servidor

A classe **Servidor** é a responsável pela execução do mesmo, sendo que para a sua criação é passado como argumento a porta pela qual deve "escutar"os comandos que irá receber. Além disso, trata de receber ligações dos anons através de uma conexão TCP e inicializa a thread responsável por processar os comandos enviados por cada um deles.

4.7 ServThread

Esta classe foi desenvolvida para lidar com os comandos recebidos pelo servidor, oriundos de um anon.

Optamos por dotar o nosso servidor da capacidade de ler ficheiros. Neste caso, os comandos recebidos são o nome dos ficheiros a ler, respondendo o servidor com o conteúdo do ficheiro.

4.8 Pdu

Esta última classe desempenha a responsabilidade de representar as informações relativas ao PDU utilizado na conexão UDP entre anons.

5 Testes e resultados

Para testar o funcionamento do programa realizado, foi utilizada a topologia de rede fornecida pelos docentes para a realização dos restantes trabalhos práticos, recorrendo ao emulador core.

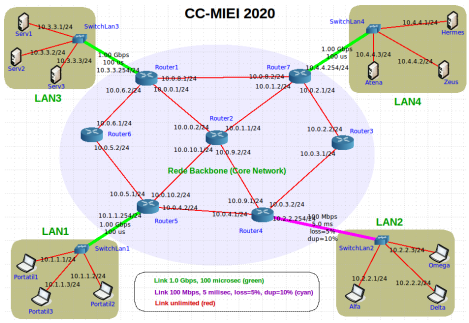

Figura 1. Topologia da rede desenhada no core

Para testarmos o programa, utilizamos o sistema *Serv1* como servidor. De seguida, utilizamos o sistema *Portatil1* como cliente anonimizado, que indicará os comandos a executar. Executamos também 4 instâncias do AnonGW nas máquinas *Portatil3*, *Atena*, *Zeus* e *Serv3* com os parâmetros corretos para se conhecerem uns aos outros e saberem que todos protegem o servidor *Serv1* na porta 80.

Listing 1.1. Comandos para inicializar o Servidor em Serv1

```
javac Servidor.java
java Servidor 80
```

Listing 1.2. Comandos para inicializar o AnonGW em Atena

```
javac AnonGW.java
java AnonGW 10.3.3.1 80 10.1.1.3 10.3.3.3 10.4.4.2
```

Listing 1.3. Comandos para inicializar o Cliente em Portatil1

```
javac Cliente.java
java Cliente 10.1.1.3 10.3.3.3 10.4.4.2 10.4.4.4
```

6 Conclusões e trabalho futuro

Desenvolvido o projeto, concluímos que o nosso programa é eficiente, já que é possível a comunicação entre cliente e servidor através de uma rede overlay de anonimzação do originador.

Como trabalho futuro, poderia ser implementado o controlo de perdas(retransmissão de pacotes perdidos), uma vez que seria algo interessante para as funcionalidades do programa. Em suma, consideramos que a realização deste trabalho foi relativamente bem sucedido, visto que pensamos ter implementado as funcionalidades propostas.