

UNIVERSIDADE DO MINHO

DEPARTAMENTO DE INFORMÁTICA

Comunicações por Computador
TP2-Rede Overlay de anonimização do
originador
Grupo Nº 03

Gonçalo Almeida (A84610)

Emanuel Rodrigues (A84776)

Lázaro Pinheiro (A86788)

24 de Maio de 2020

Conteúdo

1	Introdução	3
2	Arquitetura da solução	4
2.0.1	Cliente - AnonGW	4
2.0.2	AnonGW – Servidor	5
3	Especificação do protocolo UDP	6
3.0.1	Formato das mensagens protocolares (PDU)	6
3.0.2	Interações	6
4	Implementação	7
5	Como executar a aplicação?	9
5.0.1	Testes da aplicação:	11
6	Conclusões e trabalho futuro	13

Capítulo 1

Introdução

O presente trabalho prático desenvolve-se no âmbito da Unidade Curricular Comunicações por Computador, lecionada no 2º semestre do 3º ano do curso de Engenharia Informática. O objetivo deste trabalho é desenhar e implementar uma Rede Overlay de Anonimização do originador, isto é, foi desenhado e implementado um Gateway de transporte, designado por AnonGW, o qual recebe pedidos TCP vindos dos clientes e pedidos UDP vindos dos seus pares, ou seja, das outras instâncias, sendo a rede formada por múltiplas instâncias deste servidor. Por sua vez, os Gateways processam o pedido retirando o endereço IP da origem, garantindo assim que o endereço IP não é denunciado no servidor destino, mantendo o seu rasto inaudível neste.

Capítulo 2

Arquitetura da solução

A arquitetura da solução está dividida em dois subsistemas - o dos Clientes e o do Servidor, tal como se pode verificar na figura abaixo. A aplicação é constituída por várias classes, sendo que duas dessas, são muito importantes, AnonGWClientCloud e AnonGWServerCloud, pois nestas está especificada toda a lógica de negócio que modela os dados e processa as transações entre o Cliente, o servidor e os overlay-peers.

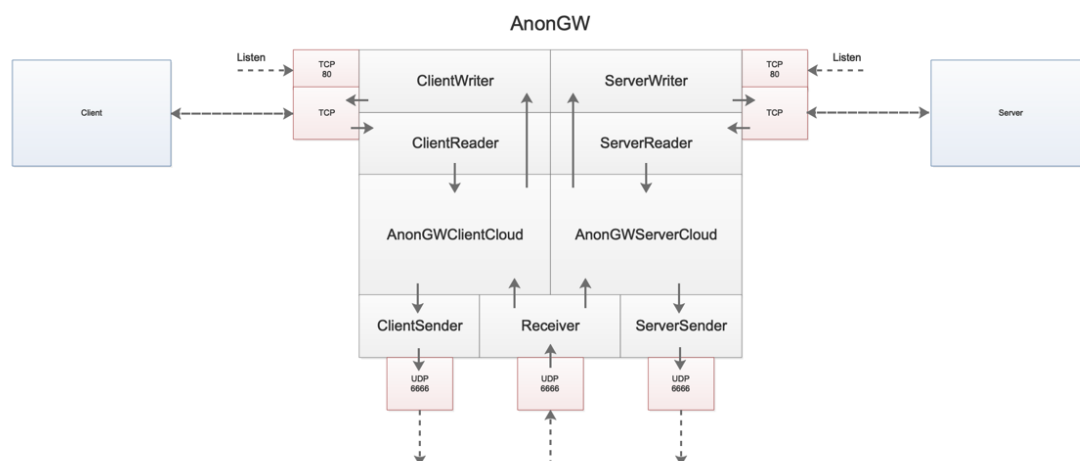


Figura 2.1: Representação da arquitetura da aplicação.

Por forma a aumentar a performance da aplicação, paralelizou-se os fluxos de dados entre o Cliente - AnonGW e AnonGW – Servidor. Dado que estes são independentes entre si, sempre que um AnonGW é contactado por um Cliente, o AnonGW reenvia os dados para um outro parceiro.

Visto que existiam requisitos a cumprir, o desenho da arquitetura foi desenvolvido em função destes (cf. Trabalho Prático No2 – Rede Overlay de anonimização do originador pp. 1).

2.0.1 Cliente - AnonGW

Com o intuito de garantir que um Cliente requer uma operação de cada vez, o seu endereço de IPv4 é guardado, sendo este mascarado, por forma a anonimizar o Cliente, por meio de um identificador que é atribuído aquando existe a sua aceitação no AnonGW, criando também uma ligação TCP para que exista permutações de dados.

Os pedidos são obtidos a partir da ligação especificada anteriormente, pedidos estes que serão transformados em pacotes, mapeados por o identificador supracitado, e por forma a garantir a sua ordem, serão armazenados numa fila de prioridades de pacotes (FIFO), sendo que o primeiro a entrar nesta fila é o primeiro a sair. Estes pacotes serão enviados por meio de UDP para um overlay-peer. De igual forma, as respostas seguem a mesma linha de pensamento. São obtidos pacotes a partir da ligação UDP e, posteriormente, as respostas(dados) que estes contem são mapeadas para o identificador que foi atribuído ao Cliente no início. Por último, utilizando a ligação TCP criada no início do fluxo, são enviadas todas as respostas.

2.0.2 AnonGW – Servidor

Por meio da ligação UDP são obtidos os pacotes que contém os pedidos endereçados pelo Cliente. No caso de ser o primeiro pacote são criados mapeamentos de um identificador de sessão para o identificador do Cliente bem como para o endereço do overlay-peer que remeteu o pedido, garantido assim a congruência dos dados, uma vez que pode existir o mesmo identificador de Cliente para overlay-peers diferentes. Não esquecer que estes pedidos são armazenados numa fila de prioridade de pacotes mapeada pelo identificador de sessão. É ainda gerada uma ligação TCP para o servidor destino para que os pedidos do Cliente lhe sejam remetidos.

A ligação TCP anteriormente referida é utilizada para obter as respostas que o Servidor envia. Com estas respostas são gerados pacotes e estes são armazenados numa fila de prioridade de pacotes, mapeada por o identificador de sessão, por forma a ser possível o envio pela ligação UDP para o Cliente que requereu.

Capítulo 3

Especificação do protocolo UDP

3.0.1 Formato das mensagens protocolares (PDU)

Para representar um pacote e enviar os dados das requests dos clientes e replies dos servidores por uma socket UDP definimos a seguinte estrutura Packet com os seguintes campos:

- **id:** Identificador do cliente. Quando o pacote se encontra no AnonGWClientCloud é o clientId, quando se encontra no AnonGWServerCloud é o sessionId do cliente para manter o anonimato deste;
- **last:** Flag que indica se o pacote é o último da request/reply;
- **sequenceNum:** Identificador do número de sequência do pacote na request/reply;
- **destination:** Flag que indica se o destino do pacote é um cliente ou um servidor;
- **data:** Array de bytes com o conteúdo da mensagem com tamanho máximo de 1024.

3.0.2 Interações

Para o envio e recepção dos pacotes pela socket UDP definimos as seguintes classes:

- **ClientSender:** Responsável por receber, sempre que existam, requests do cliente armazenadas no AnonGWClientCloud, encriptar o campo 'data' dos pacotes através do algoritmo AES e enviar os pacotes encriptados para a socket UDP;
- **ServerSender:** Semelhantemente ao ClientSender, é responsável por receber, sempre que existam, replies do servidor armazenadas no AnonGWServerCloud, encriptar o campo 'data' dos pacotes através do algoritmo AES e enviar os pacotes encriptados para a socket UDP;
- **Receiver:** Responsável por ler da socket UDP os pacotes encriptados, descriptá-los e enviar para a classe AnonGWClientCloud caso seja uma reply (campo 'destination' = 0) ou para a classe AnonGWServerCloud caso seja um request (campo 'destination' = 1);
- **Encryptor:** Responsável por encriptar/descriptar byte arrays através do algoritmo AES;
- **UDPConection:** Responsável por criar um DatagramSocket para o envio dos pacotes.

Capítulo 4

Implementação

Para a implementação da rede Overlay desenvolvemos as seguintes classes com recurso à linguagem de programação Java:

- **AnonGWMain:** A aplicação tem início nesta classe fazendo o parse da String que é passada como argumento, verificando se o target-server e os overlay-peers correspondem a um IPv4 com formato válido e se o número destes é válido. Ainda nesta classe são inicializadas as classes de modelação dos dados AnonGWClientCloud e AnonGWServerCloud e a classe de conexão UDPConnection. Por fim, é criada a socket TCP de conexão com o cliente;
- **AnonGWClientCloud:** Esta classe é responsável por, para cada cliente, inicializar as threads com as classes ClientReader, ClientWriter e ClientSender e armazenar as requests recebidas do TCP e as replies do servidor recebidas do UDP. É ainda associado a cada cliente as permissões de escrita nas sockets TCP e UDP;
- **AnonGWServerCloud:** Esta classe é responsável por, para cada sessão associada a cada cliente, inicializar as threads com as classes ServerReader, ServerWriter e ServerSender e armazenar as requests recebidas do UDP e as replies do servidor recebidas do TCP. É ainda associado a cada cliente as permissões de escrita e leitura nas socket TCP e de escrita na socket UDP;
- **ClientReader:** Esta classe é responsável por ler da socket TCP os requests do cliente e inseri-los na classe AnonGWClientCloud;
- **ClientWriter:** Esta classe é responsável por ler da classe AnonGWClientCloud, sempre que disponível, as replies do servidor e escrevê-las na socket TCP para o cliente.
- **ServerReader:** Esta classe é responsável por ler da socket TCP as replies do servidor e inseri-los na classe AnonGWServerCloud;
- **ClientSender:** Esta classe, como referida em 3.0.2, é responsável pelo envio de Packets de requests encriptados pela socket UDP.
- **ServerSender:** Esta classe, como referida em 3.0.2, é responsável pelo envio de Packets de replies encriptados pela socket UDP.
- **Receiver:** Esta classe, como referida em 3.0.2, é responsável pela leitura e descriptação dos Packets enviados pelo UDP.
- **ServerWriter:** Esta classe é responsável por ler da classe AnonGWServerCloud, sempre que disponível, as requests do cliente e escrevê-las na socket TCP para o servidor.
- **Packet:** Esta classe, como referida em 3.0.1, representa um pacote enviado pela socket UDP.

- **Packets:** Esta classe contém todos os Packets de um request/reply organizados numa Queue de modo a garantir entrega ordenada, uma flag que indica se o request/reply está completo e um indicador de quantos Packets foram inseridos.
- **Encryptor:** Esta classe, como referida em 3.0.2, é responsável pela encriptação/descriptação do campo 'data' dos Packets.
- **TCPConnection:** Esta classe é responsável por devolver o InputStream e OutputStream de uma socket, assim como fechá-la corretamente.
- **UDPConnection:** Esta classe, como referida em 3.0.2, é responsável por criar um DatagramSocket para o envio dos pacotes.
- **ClientCloudPermissions:** Esta classe é responsável por conter e tratar das permissões referidas na classe AnonGWClientCloud.
- **ServerCloudPermissions:** Esta classe é responsável por conter e tratar das permissões referidas na classe AnonGWServerCloud.
- **Constants:** Esta classe é responsável por conter as constantes MaxSizeBuffer (tamanho máximo, em bytes, do campo 'data' dos Packets), MaxSizePacket (tamanho máximo, em bytes, de um Packet), ToClient (valor do campo 'destination' dos Packets quando se trata de uma reply), ToServer (valor do campo 'destination' dos Packets quando se trata de um request), TCPPort (porta da conexão TCP), UDPPort (porta da conexão UDP), IPv4Pattern (formato de um IPv4), ALGORITHM (algoritmo de encriptação), TRANSFORMATION (transformação do algoritmo de encriptação), Key (chave usada na encriptação) e MinOverlayPeers (mínimo de Overlay Peers permitidos na aplicação).

Capítulo 5

Como executar a aplicação?

Para executar a aplicação desenvolvida, sugere-se que execute os passos indicados abaixo.

1. Iniciar a topologia CC-Topo-2020.imn no emulador CORE

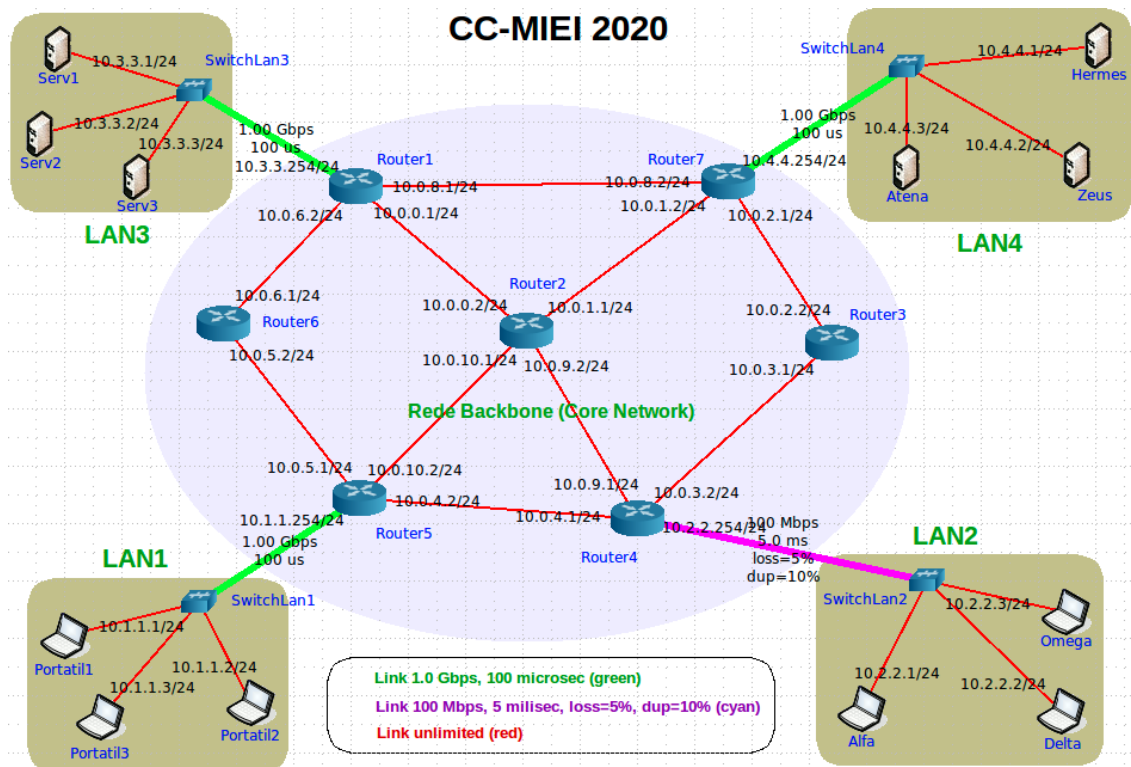
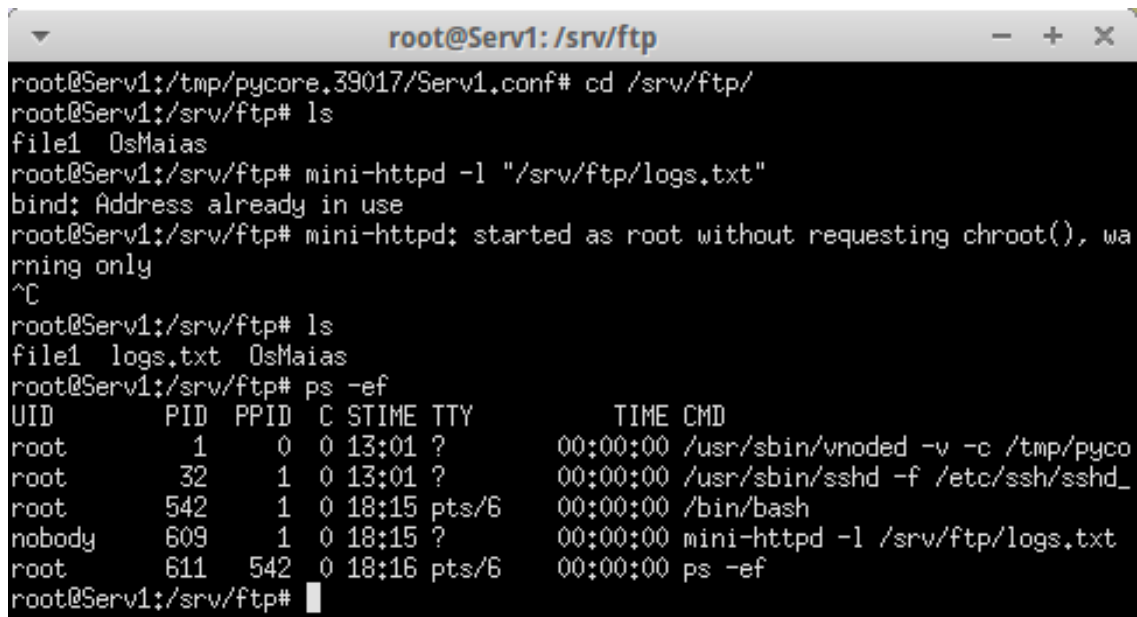


Figura 5.1: Topologia CORE disponibilizada pela equipa docente.

2. Usar o sistema Serv1 (10.3.3.0.1) como servidor HTTP “crítico” (TargetServer), com o mini-http a funcionar na porta 80

```
cd /srv/ftp
mini-httpd -l “srv/ftp/logs.txt”
```



```
root@Serv1: /srv/ftp
root@Serv1:/tmp/pycore.39017/Serv1.conf# cd /srv/ftp/
root@Serv1:/srv/ftp# ls
file1  OsMaiais
root@Serv1:/srv/ftp# mini-httpd -l "/srv/ftp/logs.txt"
bind: Address already in use
root@Serv1:/srv/ftp# mini-httpd: started as root without requesting chroot(), warning only
^C
root@Serv1:/srv/ftp# ls
file1 logs.txt OsMaiais
root@Serv1:/srv/ftp# ps -ef
UID      PID  PPID  C  STIME TTY          TIME CMD
root         1     0  0  13:01 ?           00:00:00 /usr/sbin/vnoded -v -c /tmp/pyco
root        32     1  0  13:01 ?           00:00:00 /usr/sbin/sshd -f /etc/ssh/sshd_
root       542     1  0  18:15 pts/6       00:00:00 /bin/bash
nobody     609     1  0  18:15 ?           00:00:00 mini-httpd -l /srv/ftp/logs.txt
root       611   542  0  18:16 pts/6       00:00:00 ps -ef
root@Serv1:/srv/ftp#
```

Figura 5.2: Procedimento para executar o "mini-httpd".

Nota: Devem existir ficheiros na diretoria onde o "mini-httpd" está a ser executado.

3. Executar instâncias do AnonGW nas máquinas Antena, Hermes e Zeus com os parâmetros certos para se conhecerem uns aos outros e saberem que todos protegem o servidor Serv1;

```
cd "Diretoria onde se encontra o 'AnonGW'"
javac *.java
java AnonGWMain target-server 10.3.3.0.1 overlay-peers 10.4.4.2 10.4.4.3
```

```
root@Hermes: ~/Desktop/AnonGW
root@Hermes:/tmp/pycore.39017/Hermes.conf# cd ~/Desktop/AnonGW/
root@Hermes:~/Desktop/AnonGW# ls
AnonGWClientCloud.java  ClientWriter.java  ServerCloudPermissions.java
AnonGWMain.java         Constants.java     ServerReader.java
AnonGWServerCloud.java  Encryptor.java    ServerSender.java
ClientCloudPermissions.java  Packet.java      ServerWriter.java
ClientReader.java        Packets.java      TCPConnection.java
ClientSender.java        Receiver.java     UDPConnection.java
root@Hermes:~/Desktop/AnonGW# javac *.java
root@Hermes:~/Desktop/AnonGW# java AnonGWMain target-server 10.3.3.1 overlay-peers 10.4.4.2 10.4.4.3
```

Figura 5.3: Procedimento para executar o AnonGW.

Nota: Realizar o mesmo procedimento nos overlay-peers.

4. Testar no sistema Portatil1 com `wget 10.4.4.1/OsMaias`

`wget 10.4.4.1/OsMaias`

```
root@Portatil1: /tmp/pycore.44444/Portatil1.conf
root@Portatil1:/tmp/pycore.44444/Portatil1.conf# wget 10.4.4.1/OsMaias
--2020-05-24 17:30:46-- http://10.4.4.1/OsMaias
Connecting to 10.4.4.1:80... connected.
HTTP request sent, awaiting response... 200 Ok
Length: 1360317 (1.3M) [text/plain]
Saving to: `OsMaias.13'

100%[=====>] 1,360,317 20.8K/s in 73s

2020-05-24 17:32:01 (18.1 KB/s) - `OsMaias.13' saved [1360317/1360317]
```

Figura 5.4: Procedimento para obter um ficheiro.

5.0.1 Testes da aplicação:

Os testes da aplicação passaram por testar as funcionalidades do mesmo com diferentes ficheiros, tais como músicas (.mp3), vídeos (.mp4) e ficheiros de texto (.txt). Na figura abaixo, demonstra-se

[illegible]

Capítulo 6

Conclusões e trabalho futuro

Terminado o trabalho, o grupo considera que de uma forma geral, a avaliação é positiva, visto que os requisitos mínimos foram cumpridos e o código do trabalho encontra-se suficientemente “maleável”, dado o seu elevado grau de encapsulamento, para que numa iteração futura se possa implementar os requisitos opcionais. O principal adversário do grupo foi o tempo, dado que não foi suficiente para a implementação integral que o grupo, numa fase preliminar, almejou. Ainda assim, atendendo à atual situação provocada pela SARS-CoV-2, procurou-se através de plataformas digitais fazer face aos entraves à comunicação e à partilha de ideias. Contudo o grupo superou estas dificuldades culminando o trabalho com o sentido de missão cumprida.