



Universidade do Minho
Escola de Engenharia

Rede Overlay de anonimização do originador TP2 Grupo 3



Ana Margarida Campos
A85166



Ana Catarina Gil A85266



Filipe Oliveira A80330

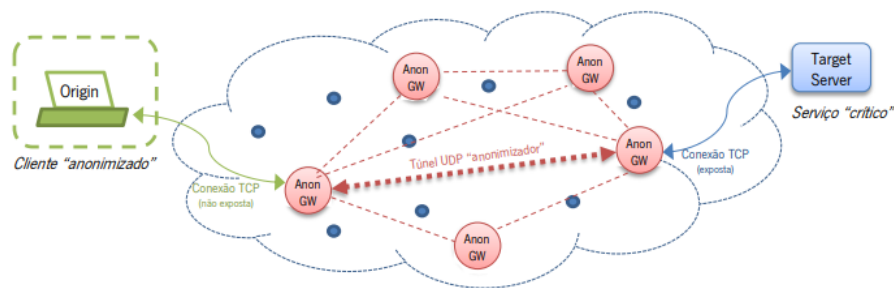
Contents

1	Introdução	3
1.1	Contextualização	3
1.2	Especificação do Problema	3
1.3	Organização do Relatório	3
2	Parte I	4
3	Parte 2	6
3.1	Arquitetura da Solução	6
3.2	Especificação do Protocolo	6
4	Implementação	7
5	Melhorias futuras	8
5.1	Dificuldades no desenvolvimento	8
5.2	Aspetos a melhorar	8
6	Conclusão	9

1 Introdução

1.1 Contextualização

Para este trabalho prático temos como cenário um cliente (Origin) que pretende interrogar um servidor (TargetServer). Porém, em vez de efetuar uma conexão de transporte TCP diretamente a esse servidor, denunciando assim o seu endereço IP de origem e deixando um rasto auditável nesse servidor, o cliente opta por usar uma Rede Overlay de Anonimização (Anon).



1.2 Especificação do Problema

O principal objetivo deste trabalho consiste em desenhar e implementar a rede *Overlay* especificada em cima. Esta rede é formada por um conjunto de *Gateways de Transporte* espalhados pela rede bem como o Cliente e o Servidor. O que o Cliente tem de fazer é dirigir a sua conexão para qualquer um dos *AnonGW* disponíveis (via TCP). O primeiro *AnonGW* contactado tem de aceitar a conexão e enviar para outros *AnonGW* o pedido mas desta vez a partir de conexões UDP. O último *Gateway de Transporte* recorre novamente ao TCP para se ligar com o Servidor. O percurso inverso é usado da mesma maneira nas respostas do servidor. A construção desta rede permite a camuflagem dos endereços trazendo por consequência mais segurança ao sistema.

1.3 Organização do Relatório

Numa fase inicial do relatório é mostrado o processo desenvolvido para a primeira parte do trabalho. Numa segunda parte é explicado o procedimento da segunda fase bem como todas as implicações que isso trouxe na nossa abordagem. Por último, encontra-se a conclusão onde é resumido o trabalho desenvolvido bem como as dificuldades sentidas ao longo do seu desenvolvimento.

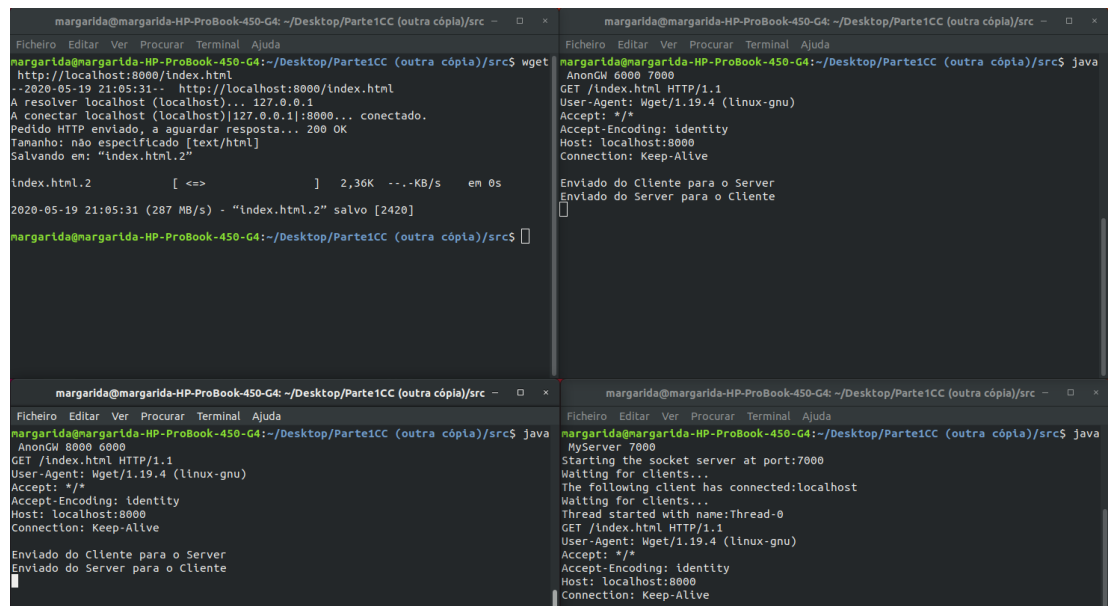
2 Parte I

Esta primeira fase consistiu na elaboração de um programa que garante a conexão da rede apresentada anteriormente apenas via TCP. Para tal foram criadas duas classes principais: uma com o objetivo de representar cada nodo *AnonGW* e outra que representa o servidor. O cliente utilizado foi gerado a partir do comando *wget* onde era indicado o *host* e a porta correspondente.

Uma das vantagens da nossa implementação é que tanto o servidor como os *AnonGWs* são multithread, ou seja é possível correr vários em simultâneo.

Os dados enviados pelo cliente são recebidos pelo primeiro *AnonGW* e caso existam mais *AnonGW* disponíveis são enviados para os mesmos através de uma conexão TCP. O último *AnonGW* na linha de envio faz a conexão com o *TargetServer* que, posteriormente, envia os dados de volta para o cliente utilizando o caminho inverso.

Na seguinte figura é possível verificar as conexões existentes entre o cliente, diferentes *AnonGWs* e o servidor, com a passagem de um ficheiro html. Nesta implementação é necessário a passagem das portas de cada um dos intervenientes.



```
margarida@margarida-HP-ProBook-450-G4: ~/Desktop/Parte1CC (outra cópia)/src - □ ×
Ficheiro Editar Ver Procurar Terminal Ajuda
margarida@margarida-HP-ProBook-450-G4:~/Desktop/Parte1CC (outra cópia)/src$ wget
http://localhost:8000/index.html
2020-05-19 21:05:31-- http://localhost:8000/index.html
A resolver localhost (localhost)... 127.0.0.1
A conectar localhost (localhost)[127.0.0.1]:8000... conectado.
Pedido HTTP enviado, a aguardar resposta... 200 OK
Tamanho: não especificado [text/html]
Salvando em: "index.html.2"

index.html.2 [ <=> ] 2,36K --.KB/s em 0s
2020-05-19 21:05:31 (287 MB/s) - "index.html.2" salvo [2420]

margarida@margarida-HP-ProBook-450-G4:~/Desktop/Parte1CC (outra cópia)/src$

margarida@margarida-HP-ProBook-450-G4:~/Desktop/Parte1CC (outra cópia)/src$ java
AnonGW 6000 7000
GET /index.html HTTP/1.1
User-Agent: Wget/1.19.4 (linux-gnu)
Accept: */*
Accept-Encoding: identity
Host: localhost:8000
Connection: Keep-Alive

Enviado do Cliente para o Server
Enviado do Server para o Cliente

margarida@margarida-HP-ProBook-450-G4:~/Desktop/Parte1CC (outra cópia)/src$

margarida@margarida-HP-ProBook-450-G4:~/Desktop/Parte1CC (outra cópia)/src$ java
MyServer 7000
Starting the socket server at port:7000
Waiting for clients...
The following client has connected:localhost
Waiting for clients...
Thread started with name:Thread-0
GET /index.html HTTP/1.1
User-Agent: Wget/1.19.4 (linux-gnu)
Accept: */*
Accept-Encoding: identity
Host: localhost:8000
Connection: Keep-Alive
```

Figure 1: Conexão via TCP com 2 *AnonGW*

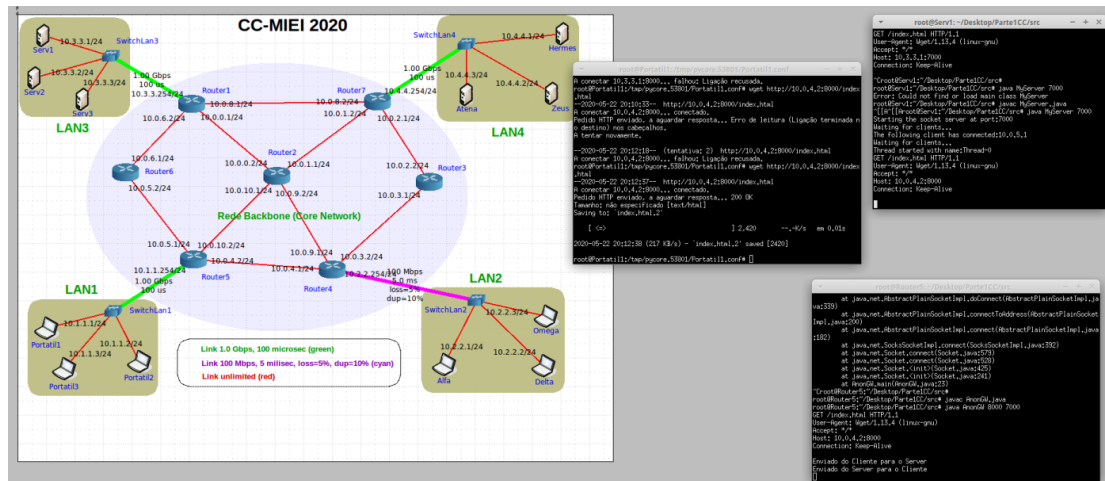


Figure 2: Teste da implementação Parte 1 na topologia Core

3 Parte 2

O objetivo principal desta fase consiste na elaboração de um protocolo para funcionar sobre UDP que garanta a entrega ordenada e confidencial dos dados de uma ou mais conexões de transporte TCP. Todos os dados recebidos de uma extremidade TCP devem ser reenviados pelo túnel UDP e vice-versa.

3.1 Arquitetura da Solução

Como podemos verificar no esquema, a nossa solução consistiu em desenvolver dois *AnonGW* e as respectivas conexões entre eles, o Cliente e o Servidor. As ligações entre os *AnonGW* seguem o protocolo UDP e as restantes o protocolo TCP.

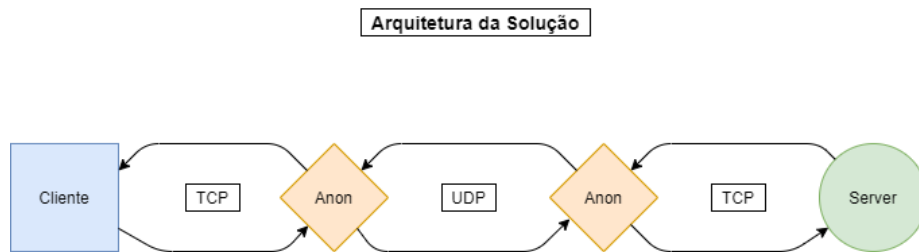


Figure 3: Arquitetura da Solução

3.2 Especificação do Protocolo

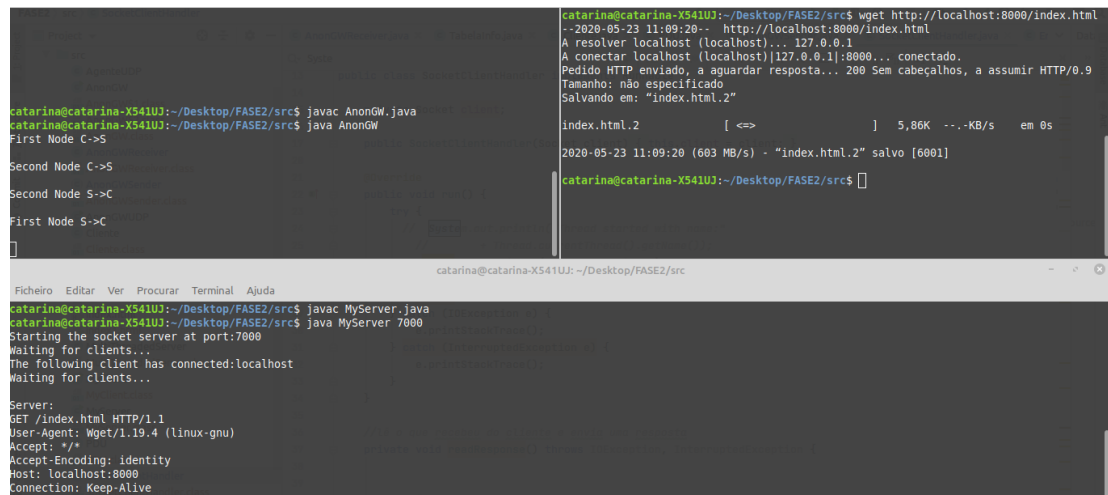
Primitivas de comunicação: Sockets TCP e DatagramSockets UDP.

Formato do PDU: o formato do PDU desenvolvido é formado pelo número de sequência, que vai ser incrementado sequencialmente de acordo com o número de fragmentos do pacote inicial, número do ACK do próximo pacote para garantir a integridade do pacote total, ou seja, para garantir que todos os fragmentos são recebidos e um valor de checksum para verificar a existência de erros.

4 Implementação

1. **MySever:** Nesta classe foi desenvolvida a *main* de um Servidor Http genérico. Este possui a vantagem de ser Multithread. É responsável por receber o *request* do último *AnonGW* da rede bem como enviar a *response*. Utiliza como classes auxiliares a *MultiThreadServer*, responsável pela multiprogramação, e a classe *SocketClientHandler* responsável pelos pedidos e respostas.
2. **AnonGW:** Nesta classe vão ser executados o *AnonGWReceiver* bem como o *AnonGWSender* através de threads.
3. **AnonGWReceiver:** Esta classe é responsável pela receção do request do Cliente através de TCP bem como o envio da response para o próximo *AnonGW* através de UDP.
4. **AnonGWSender:** Esta classe é responsável pela receção do request por parte do outro *AnonGW* a partir de conexão UDP bem como o envio da response para o Servidor a partir de TCP.

Na nossa implementação está funcional o processo de envio do pedido desde o Cliente até ao Servidor e o respetivo regresso passando apenas por dois *AnonGW*.



```
catarina@catarina-X541UJ:~/Desktop/FASE2/src$ wget http://localhost:8000/index.html
--2020-05-23 11:09:20-- http://localhost:8000/index.html
A resolver localhost (localhost)... 127.0.0.1
A conectar localhost (localhost)[127.0.0.1]:8000... conectado.
Pedido HTTP enviado, a aguardar resposta... 200 Sem cabeçalhos, a assumir HTTP/0.9
Tamanho: não especificado
Salvando em: "index.html.2"

index.html.2      [ <> ]  5,86K  --.-KB/s  em 0s

2020-05-23 11:09:20 (603 MB/s) - "index.html.2" salvo [6001]

catarina@catarina-X541UJ:~/Desktop/FASE2/src$

catarina@catarina-X541UJ:~/Desktop/FASE2/src$ javac AnonGW.java
catarina@catarina-X541UJ:~/Desktop/FASE2/src$ java AnonGW
First Node C->S
Second Node C->S
Second Node S->C
First Node S->C

catarina@catarina-X541UJ:~/Desktop/FASE2/src$ javac MyServer.java
catarina@catarina-X541UJ:~/Desktop/FASE2/src$ java MyServer 7000
Starting the socket server at port:7000
Waiting for clients...
The following client has connected:localhost
waiting for clients...

Server:
GET /index.html HTTP/1.1
User-Agent: Wget/1.19.4 (linux-gnu)
Accept: */*
Accept-Encoding: identity
Host: localhost:8000
Connection: Keep-Alive
```

Figure 4: Conexão final - Fase 2

5 Melhorias futuras

5.1 Dificuldades no desenvolvimento

À medida do desenvolvimento do trabalho foram sentidas diversas dificuldades, dentro das quais se destacam a conversão do request de TCP para UDP, a utilização de DatagramSockets uma vez que nunca teria sido implementado por nós nos restantes trabalhos e o envio de regresso dos dados do Servidor para o Cliente. Devido a estes conflitos foi bastante difícil a implementação do PDU no projeto.

5.2 Aspetos a melhorar

Devido às dificuldades sentidas e especificadas em cima, não nos foi possível aplicar todos os conhecimentos obtidos sobre os protocolos de comunicação TCP e UDP bem como grande parte dos seus constituintes.

De maneira a aplicar alguns conhecimentos, embora não usadas, foram desenvolvidas as classes PDU, TabelaInfo e Encriptação. O objetivo destas era armazenar informação sobre os pacotes que são enviados por cada *AnonGW* de forma a anonimizar os endereços IP relativos ao cliente. Na TabelaInfo iríamos guardar todos os PDU's associados ao endereço do anonGW anterior, de forma a garantir que no percurso inverso (Servidor -> Cliente) a response passe pelos mesmos nodos e assim garantir que irá ser recebido pelo Cliente certo. O objetivo da classe Encriptação é encriptar os dados que são transferidos através do canal UDP. A nossa solução não funcionou devido a um erro na desincriptação.

6 Conclusão

Na realização deste trabalho, sentimos dificuldades no desenvolvimento de algumas tarefas, principalmente na realização da conexão entre o protocolo TCP e UDP.

O nosso maior problema foi abordar a passagem de informação entre o cliente, os vários anons e o servidor, sem a perda de referência do cliente (saber o ip de origem do pedido) que fez esse mesmo request.

Para facilitar o desenvolvimento do programa, realizamos um modelo da arquitetura da solução o que nos ajudou a compreender melhor o projeto. Assim sendo, todas as nossas classes foram criadas seguindo o modelo da arquitetura solução.