Comunicação por Computador

Uma imagem com ClipArt

Descrição gerada automaticamente

Trabalho prático nº2

24 de maio de 2020

**Grupo nº 4**

Filipa Alves dos Santos (A83631)

Guilherme Pereira Martins (A70782)

Luís Miguel Arieira Ramos (A83930)

Rui Alves dos Santos (A67656)



Mestrado Integrado em Engenharia Informática

Universidade do Minho

**Índice de conteúdos**

**1. Introdução 3**

**2. Arquitetura da solução 4**

**3. Especificação do protocolo UDP 6**

**4. Implementação 7**

**5. Testes e resultados 9**

**6. Conclusões 10**

1. **Introdução**

Neste 2º trabalho prático, no âmbito da disciplina de Comunicações por Computador, foi-nos dado como desafio criar uma rede de anonimização, com o objetivo de garantir a privacidade e segurança dos utilizadores.

Resumindo o cenário descrito no enunciado, é pretendido criar uma rede de AnonGws, isto é, um conjunto de Gateways de Transporte que estabeleça a conexão de um Cliente com um certo TargetServer. O Cliente liga-se ao primeiro AnonGw através de uma conexão TCP, que depois comunica com um segundo AnonGW através um túnel UDP. Finalmente, este último estabelece outra conexão TCP com o TargetServer.

Para além disso, também foram pedidos alguns requerimentos mínimos. O programa tem que garantir a entrega ordenada dos pacotes, isto é, devem ser enviados na mesma ordem que foram recebidos. Também temos de assegurar a confidencialidade, com qualquer tipo de encriptação, e a integridade dos dados transportados, que se garante através de verificações de que os pacotes se mantiveram inalterados. Por fim, é pedido que várias conexões possam ser estabelecidas em simultâneo, isto é, uma multiplexagem de clientes.

No resto deste relatório, iremos explicar o modo de como construímos o nosso programa, abordando a sua arquitetura, o nosso raciocínio na conexão UDP, uma descrição da implementação feita, os testes realizados e, por fim, os resultados obtidos.

1. **Arquitetura da solução**

Para explicarmos a arquitetura da nossa solução, achamos adequado criar 2 esquemas que ilustram o nosso raciocínio no desenvolvimento deste programa:

(2)

(3)

(1)

Cliente

AnonGW

Primário

(Worker)

[www.youtube.com/watch?v=STyOk0u5D2c](http://www.youtube.com/watch?v=STyOk0u5D2c)

Secundário

AnonGW

Secundário

Target

Secundário

Worker 2

Secundário

Worker 1

ACK

Figura 1 - Envio do comando até ao Target

(4)

Cliente

AnonGW

Primário

(Worker)

[www.youtube.com/watch?v=STyOk0u5D2c](http://www.youtube.com/watch?v=STyOk0u5D2c)

Secundário

AnonGW

Secundário

Target

Secundário

Worker 2

Secundário

Worker 1

(5)

(6)

ACK

Figura 2 - Envio do ficheiro até ao Cliente

Iremos agora detalhar cada uma das ligações mostradas nas figuras acima:

1. É a primeira conexão a ser feita. A conexão é estabelecida pelo Primário, que ouve TCP, e de seguida cria um worker para essa sessão;
2. PrimarioWorker vai estar sempre a enviar PDUs para o peer, que são ouvidas pelo Secundário. Por cada um, cria um novo SecundarioWorker2 (as necessárias para receber a informação toda) e recebe um ACK de confirmação, antes de mandar a seguinte;
3. O pacote é redirecionado para o TargetServer. Se for o último pacote, abre a conexão TCP para receber a resposta;
4. O TargetServer passa o conteúdo requisitado pelo cliente para o SecundarioWorker2, via TCP. A resposta é transformada em PDUs;
5. Secundário Worker 2 vai estar sempre a mandar PDUs para o AnonGw original, que são ouvidas pelo Secundário, que cria um SecundarioWorker1, um para cada PDU. Novamente, repete o ciclo até enviar o conteúdo todo e recebe um ACK de confirmação para cada PDU, antes de mandar o seguinte;
6. O pacote é redirecionado para o Cliente por TCP.
7. **Especificação do protocolo UDP**

O formato final das nossas mensagens PDUs ficou de acordo com a figura seguinte:

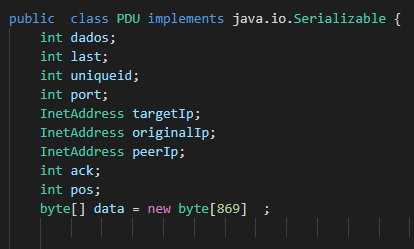


Figura 3 - Classe PDU

Iremos agora especificar cada uma das variáveis da classe:

* **dados:** inteiro que identifica quando estamos a transferir dados (1 - true) ou quando se trata de um pedido (0 - false);
* **last:** inteiro que identifica se é a última PDU (1 - true) ou não (0 - false);
* **uniqueid:** identificador único que cada sessão tem (facilitar na multiplexagem, evitar mistura de pdus de sessões diferentes). É um inteiro gerado aleatoriamente;
* **port:** porta do TCP do TargetServer;
* **targetIp:** endereço IP do TargetServer;
* **originalIp:** endereço IP do AnonGw que se conecta ao Cliente;
* **peerIp:** endereço IP do AnonGw peer;
* **ack:** inteiro que indica se se trata de um ACK (1 - true) ou não (0 - false);
* **pos:** inteiro correspondente à posição/ordem do pacote;
* **data:** array de bytes com o conteúdo do pacote;

1. **Implementação**

Para explicarmos a nossa implementação, vamos começar com um raciocínio geral que tivemos ao construir a nossa solução.

De modo a podermos lidar com multiplexagem, criámos 2 threads principais no nosso programa:

* Primário (que ouve TCP)
* Secundário (que ouve UDP)

Estas threads serão apenas listeners e sempre que recebem uma ligação apenas criam uma nova thread worker e, de seguida, voltam de novo a ouvir. De modo a simplificar a explicação iremos considerar apenas 1 cliente, 1 target e 2 anonGWs (anon1 e anon2).

**Cliente - Anon1**

O cliente irá efetuar um pedido ao anon1, que já tem o target pretendido. O Primário ouve este pedido, gera um identificador único e guarda num hashmap global ao anonGW tanto o id como o socket. Por último, cria uma nova thread de PrimarioWorker1.

**Anon 1 - Anon2**

O PrimarioWorker1 irá ser responsável por ler todo o conteúdo da conexão TCP, transformá-la em PDUs e enviá-las para o anon2. Este processo é iniciado por introduzir na hashmap de Pedidos um value de -1 com a key o nosso identificador. De seguida, a primeira PDU é enviada (já encriptada) para o anon2, ficando de seguida à espera do ACK correspondente (neste caso, consultando a tabela). O ACK é recebido pelo Secundário (visto ser UDP) que vai apenas atualizar o hashmap de pedidos, desbloqueando assim o ciclo descrito acima e permitindo que este envie o próximo PDU.

**Anon 2 – Target**

O anon2 vai receber PDUs por UDP. Ao receber uma, cria uma nova thread de SecundarioWorker2. Este será responsável por falar com o TargetServer e enviar de volta a resposta para o anon1. Por cada PDU recebida no anon2, o SecundarioWorker2 correspondente irá desencriptar o conteúdo, enviar o mesmo para o target server e enviar um ACK, descrito no parágrafo anterior, de volta.

**Target – Anon 2**

Caso o PDU recebido esteja marcado como último, este será o SecundarioWorker2 responsável por receber a resposta do TargetServer. A mesma será transformada em PDUs.

**Anon2 - Anon1**

Já com as PDUs prontas, iremos agora introduzir na hashmap de dados o valor -1 cuja key é o nosso identificador, enviando de seguida o primeiro PDU. Da mesma maneira que funcionavam os primeiros ACKs, iremos ficar à espera do ACK de confirmação enviado pelo anon1. Fazemos isto através da consulta da tabela, que vai ser alterada pelo nosso Secundário, aquando da receção do mesmo.

**Anon 1 – Cliente**

Por cada pacote que o anon1 receber, irá criar uma thread de SecundárioWorker1, sendo que os pacotes vêm marcados como dados. O SecundarioWorker1 irá apenas enviar o conteúdo da PDU para o Cliente, bem como um ACk de volta ao Anon2. De modo a saber qual o Cliente certo, iremos consultar o Socket na Hasmap criada inicialmente.

No final da conexão, as 3 hashmaps são limpas, de modo a não guardar qualquer tipo de log.

Iremos agora resumir como tratamos de satisfazer os requerimentos pedidos no enunciado:

* **Confidencialidade:** todos os PDUs são encriptados antes de ser enviados, através de uma encriptação simétrica, em que todos os anonGW sabem a key;
* **Anonimato:** aanonimização é garantida sendo que apenas o anon1 é que conhece o Cliente. O anon2 não recebe informações sobre o Cliente e, como consequência, o TargetServer também nunca conhecerá o Cliente que fez o pedido;
* **Multiplexagem:** com temos apenas listeneres e workers, garantimos que não perdemos informação. Usando o identificador único, certificamo-nos de que não há pacotes a ser enviados para o sítio errado;
* **Entrega ordenada:** Esperamos sempre por um ACK de confirmação antes de enviar o próximo, garantido a ordem de entrega das PDUs.

Em termos de bibliotecas usadas, para além das que são utilizadas normalmente num projeto Java, como as util, net e io, também utilizamos a javax.crypto para a encriptação que escolhemos.

**NOTA:** Na função pacotify do SecundarioWorker2, tivemos que adicionar um sleep de 2 segundos (e mais 0.1 por iteração) para funcionar na VM fornecida pelos docentes, caso contrário seria demasiado rápido para a máquina. No entanto, na VM que utilizamos, não houve necessidade de tal alteração.

1. **Testes e resultados**

Para garantirmos a qualidade da nossa solução, testamos transferir ficheiros de diversos formatos. Após algumas iterações e alterações ao código, conseguimos pôr o programa a funcionar independentemente do ficheiro, como pdf, mp3, txt e xml. Também efetuamos testes com 2 clientes para testar a implementação da multiplexagem e verificamos que tudo resultou como esperado.

Na figura seguinte, temos o exemplo de um ficheiro mp3 a ser entregue ao Cliente, com 2 anonGws:

Uma imagem com captura de ecrã

Descrição gerada automaticamente

Figura 4 - Programa a ser executado

1. **Conclusões**

Podemos concluir que saímos satisfeitos deste último trabalho prático pois consideramos que cumprimos todos os objetivos pedidos no enunciado. Garantimos multiplexagem, entregas ordenadas do pacote, anonimato do cliente e confidencialidade. O único aspeto a melhorar seria o controlo de perdas pois, como o trabalho se encontra no momento, se um pacote for perdido, não há nenhum mecanismo para o reenviar.

O grupo considera que saiu deste trabalho com um conhecimento mais aprofundado relativamente a redes Overlay, AnonGws e sobre a implementação de um protocolo UDP. Pôr em prática esta matéria, que já tinha sido abordada antes em Redes de Computadores e nesta mesma unidade curricular, irá facilitar qualquer trabalho futuro que tenhamos sobre estes assuntos.