Universidade do Porto

Peer-to-peer backup service for the Internet

Bruno Sousa **up201604145**@fe.up.pt

Catarina Figueiredo **up210606334**@fe.up.pt

Francisco Filipe **up201604601**@fe.up.ptt

Pedro Fernandes **up201603846**@fe.up.pt

Unidade Curricular: Sistemas Distribuídos

Docente: Diana Guimarães

Turma: MIEIC02

Grupo: 8

31 de Maio de 2019

Conteúdo

1.1 Introdução

Este relatório foi desenvolvido no âmbito do segundo trabalho prático da Unidade Curricular de Sistemas Distribuídos. O seu objetivo é clarificar alguns dos aspetos principais da implementação do projeto *"Peer-to-peer backup service for the Internet"*. A sua estrutura é a seguinte:

- **Overview:** As instruções necessárias para compilar e executar corretamente o programa desenvolvido, tanto em Windows como em Linux. Também está presente nesta secção a descrição dos scripts implementados, que ajudam na demonstração do trabalho.
- **Protocolos:** Descrição detalhada dos mecanismos e estruturas de dados utilizadas no desenvolvimento deste trabalho, que permitem a execução concorrente dos diferentes protocolos. Esta descrição é acompanhada de alguns excertos do código fonte para ajudar a compreender a implementação desenvolvida.
- **Design de Concorrência:** Descrição de cada uma das melhorias propostas, da solução pensada e da implementação desenvolvida.
- **JSSE:** As instruções necessárias para compilar e executar corretamente o programa desenvolvido, tanto em Windows como em Linux. Também está presente nesta secção a descrição dos scripts implementados, que ajudam na demonstração do trabalho.
- **Escalabilidade:** Descrição detalhada dos mecanismos e estruturas de dados utilizadas no desenvolvimento deste trabalho, que permitem a execução concorrente dos diferentes protocolos. Esta descrição é acompanhada de alguns excertos do código fonte para ajudar a compreender a implementação desenvolvida.
- **Tolerância a Falhas:** Descrição de cada uma das melhorias propostas, da solução pensada e da implementação desenvolvida.

1.2 Overview

1.3 Protocolos

1.4 Design da Concorrência

1.5 JSSE

Nesta implementação está também presente a utilização de JSSE. Este é utilizado em qualquer tipo de comunicação entre os nós de forma a garantir uma troca de mensagens segura entre estes (desde pedidos e respostas de peers a transferências de ficheiros). Desta forma é protege-se a identificação de cada peer (IP e porta), tornando a comunicação mais segura. Por este motivo, não é possível indicar um protocolo específico em que se use este mecanismo pois ele está presente em todos.

Na implementação de JSSE optou-se pela utilização da classe javax.net.ssl.SSLSocket que adiciona uma camada de segurança à comunicação. Esta classe é instanciada sempre que é necessário o envio de uma nova mensagem entre nós, ou seja:

- **Comunicação entre Peers:** através de mensagens que
- **Comunicação entre TestApp e Peers:** através da realização do "handshake" e da troca de mensagens que indicam o protocolo a realizar.

```
SSLSocketFactory factory = (SSLSocketFactory)SSLSocketFactory.getDefault();
SSLSocket socket = (SSLSocket) factory.createSocket(ip, port);

DataOutputStream out = new DataOutputStream(socket.getOutputStream());
BufferedReader in = new BufferedReader(new
    InputStreamReader(socket.getInputStream()));

String message = "FINDSUCCESSOR " + requestId + " " + id + " \n";
System.out.println("[Node " + requestId + "] " + message);
out.writeBytes(message);

String response = in.readLine().trim();
socket.close();
```

O excerto de código acima apresenta a implementação deste socket do lado do emissor (aquele que envia a mensagem). A implementação é relativamente simples e muito semelhante à de um socket desprovido de SSL. Apenas é necessário instanciar uma SSLFactory contendo a "default factory" e criar um SSLSocket com o ip e porta respetivos. De seguida é instanciada a stream de output (DataOutputStream) , encarregue de enviar a mensagem para outro nó. É também neste momento que se instancia um BufferedReader responsável pela leitura da resposta. De seguida constrói-se a mensagem a enviar pela stream de output. Na penúltima

instrução, bloqueia-se a thread atual, que fica assim à espera de uma resposta por parte do recetor. Uma vez recebida esta resposta é fechado o socket.

```
SSLServerSocketFactory ssf = (SSLServerSocketFactory)
    SSLServerSocketFactory.getDefault();
SSLServerSocket listenSocket = (SSLServerSocket) ssf.createServerSocket(node.port);

while (true) {
  SSLSocket connection = (SSLSocket) listenSocket.accept();
  node.executor.execute(new Runnable() {
  public void run() {
       try {
          interpretMessage(connection);
       } catch (IOException e) {
       }
        }
   });
 }
```

Já este último pedaço de código apresenta a implementação deste mesmo socket do lado do recetor (aquele que recebe a mensagem). Aqui é instanciada uma SSLServerFactory onde vai estar presente um SSLServerSocket. Este último, a cada pedido que recebe, cria uma nova conexão através de um SSLSocket que vai estar responsável por esse pedido. Por último é feita uma chamada à função interpretMessage onde, de acordo com o pedido executado, realiza as operações necessárias e envia uma resposta ao emissor através de uma DataOutputStream.

Em relação às funcionalidades do JSSE que foram utilizadas,

1.6 Escalabilidade

Para assegurar que o sistema desenvolvido é escalável foi implementado o Chord, um protocolo escalável para pesquisas num sistema peer-to-peer dinâmico (com entradas e saídas frequentes de nós e com número altamente variável de nós). Este fornece suporte apenas para uma operação: dada uma chave, mapeia essa chave num nó. Esse nó pode ser responsável por armazenar um valor associado à chave.

A utilização do Chord como protocolo torna o sistema escalável pois o custo de uma pesquisa cresce logaritmicamente com o aumento do número de nós (log(N) em que N é o número de nós). Como tal, até mesmo sistemas muito grandes são alcançáveis.

1.6.1 Implementação do Chord

1.7 Tolerância a Falhas