Verteilte Systeme im Sommersemester 2021

Steffen Herweg, Matr. Nr. 873475
Luca Fabio Kock, Matr. Nr. 879534

Osnabrück, 04.05.2021

Aufgabenblatt 3

Was ist speziell bei Verwendung von UDP als Transportprotokoll zu berücksichtigen?

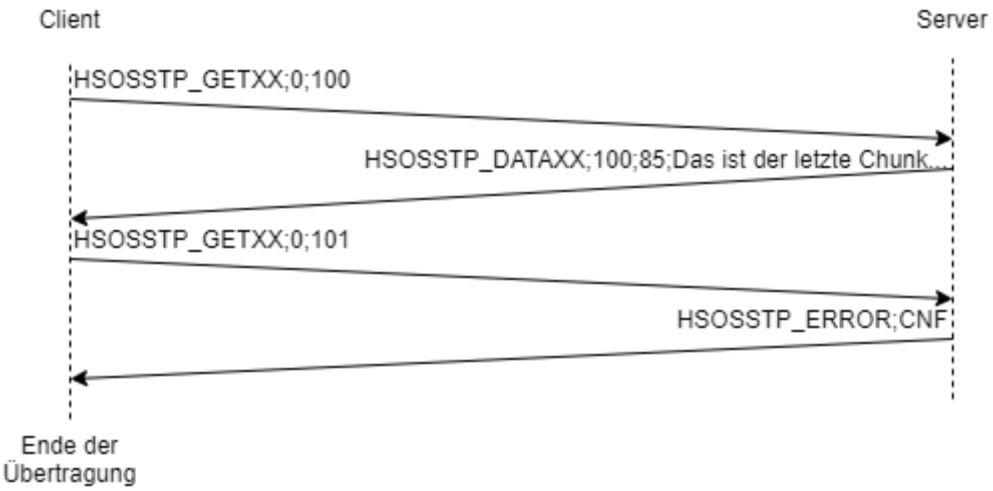
-UDP garantiert keine zuverlässige Paketauslieferung bzw. der Sender erhält keine Fehlermeldung

Warum sollte für die Umsetzung des Protokolls UDP und nicht TCP verwendet werden?

-Iterative Server werden meistens mit UDP realisiert + UDP ist deutlich schneller

- Bei kleinen Dateien wird der Overhead von TCP nicht benötigt

Wie kann man das Ende einer Datentransfer-Sitzung erkennen? Beschreiben Sie die bei Ihrer Lösung ausgetauschten Nachrichten durch ein Sequenzdiagramm.



• Wie werden Datentransfer-Sitzungen im Server terminiert?

Der Client fragt der Reihe nach die Blöcke ab, sobald das Ende der Datei erreicht ist sendet der Server den CNF-Error. Daran erkennt der Client, dass die Datei keine weiteren Blöcke enthält.

• Was passiert bei Wiederaufnahme einer Sitzung?

Das Wiederaufnehmen einer Sitzung wird vom Server nicht von jeder anderen Anfrage unterschieden.

• Was passiert beim Zugriff auf eine nichtexistierende Sitzung?

Der Server sendet den NOS-Error

• Wie groß kann die Chunk-Size maximal eingestellt werden?

Mit UDP können maximal 65527 Bytes übertragen werden, davon muss man aber noch die Größe des Headers (als HSOSSTP…) abziehen. Da im Header jedoch manchmal die Größe des Chunks oder die

Session-ID übermittelt wird kann man seine Größe nicht genau voraussagen. Deshalb lässt sich die maximale Chunkgröße nicht genau bestimmen.

Client.java:

```
package hsosstp.client;

import hsosstp.Main;

import java.io.*;
import java.net.*;


public class Client {

    public static final int CHUNKSIZE = 1024;
    public static final int HEADSPACE = 100;
    public static final int BUFFSIZE = CHUNKSIZE + HEADSPACE;

    public static void run(String address, String filename,int port) throws
IOException {

        // get a datagram socket
        DatagramSocket socket = new DatagramSocket();

        // Create Initial message
        String msg = Main.INITX + ";" + CHUNKSIZE + ";" + filename;
        byte[] msgBuf = msg.getBytes(Main.CHARSET);

        // Resolve Address
        InetAddress inetAddress = InetAddress.getByName(address);

        // Create Packet
        DatagramPacket packet = new DatagramPacket(msgBuf, msgBuf.length,
inetAddress, port);

        // Send Packet
        socket.send(packet);

        // Receive SessionId

        byte[] responseBuffer = new byte[BUFFSIZE];
        DatagramPacket responsePacket = new DatagramPacket(responseBuffer,
responseBuffer.length);
        System.out.println("Waiting for SessionId:");
        socket.receive(responsePacket);
        System.out.println("Got response:");
        System.out.println(new String(responseBuffer));
        String responseString = new String(responseBuffer).trim();
        String[] responseStrings = responseString.split(";");
        if(responseStrings.length <= 1){
            malformedResponse(responseString);
        }
        if(responseStrings[0].equals(Main.ERROR) &&
responseStrings[1].equals("FNF")){
            System.out.println("File Not Found on hsosstp.server!");
            System.exit(0);
        }
        try {
            int sessionId =
```

```
Integer.parseInt(responseString.split(";")[1]);System.out.println("SessionI
d: " + sessionId);

            int chunkNr = 0;
            RandomAccessFile out = new RandomAccessFile(new
File(filename),"rw");
            while (true) {

                String request = Main.GETXX + ";" + sessionId + ";" +
chunkNr;
                byte[] requestBytes = request.getBytes(Main.CHARSET);
                DatagramPacket requestPacket = new
DatagramPacket(requestBytes, requestBytes.length, inetAddress, port);
                socket.send(requestPacket);


                byte[] responseBytes = new byte[BUFFSIZE];
                responsePacket = new DatagramPacket(responseBytes,
responseBytes.length);
                socket.receive(responsePacket);
                responseString = new String(responseBytes,
Main.CHARSET).trim();
                responseStrings = responseString.split(";");
                if(responseStrings.length <= 1) {
                    malformedResponse(responseString);
                }
                else if (responseStrings[0].equals(Main.ERROR) &&
responseStrings[1].equals("CNF")) {
                    break;
                }
                else if (responseStrings[0].equals(Main.DATAX) &&
responseStrings.length >= 4) {
                        int chunkNumber =
Integer.parseInt(responseStrings[1]);
                        int chunkSize =
Integer.parseInt(responseStrings[2]);
                        int offset = responseStrings[0].length() +
responseStrings[1].length() + responseStrings[2].length() + 3;
                        byte[] data = new byte[chunkSize];
                        System.arraycopy(responseBytes, offset, data, 0,
data.length);
                        out.seek(CHUNKSIZE * chunkNumber);
                        out.write(data);
                } else {
                    malformedResponse(responseString);
                }


                chunkNr++;
            }
            out.close();
        }catch (NumberFormatException e){
            malformedResponse(responseString);
        }

        socket.close();
    }

    private static void malformedResponse(String responseString){

        System.err.println("Malformed Response: " + responseString);
```

Server.java:

```
package hsosstp.server;

import hsosstp.Main;

import java.io.*;
import java.net.*;
import java.util.*;

public class Server {

    private static DatagramSocket socket = null;

    private static ArrayList<Session> sessions = new ArrayList<>();


    public static void start(int port) {
        try {
            socket = new DatagramSocket(port);
            run();
        } catch (SocketException e) {
            System.err.println("Could not Create Socket!");
            System.exit(-1);
        }
    }


    private static void run() {

        while (true) {
            try {
                byte[] buf = new byte[256];

                // receive request
                DatagramPacket packet = new DatagramPacket(buf,
buf.length);
                System.out.println("Waiting for request:");
                socket.receive(packet);

                String requestString = new String(buf,
hsosstp.Main.CHARSET).trim();
                System.out.println("Got request: " + requestString);
                String[] args = requestString.split(";");

                if (args.length > 1) {
                    switch (args[0]) {
                        case Main.INITX:
                            initSession(socket, packet);
                            break;
                        case Main.GETXX:
                            getData(socket, packet);
                            break;
                        default:
                            System.out.println("Malformed Request; Not
Responding");
                    }
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
```

```
        }
    }

    private static void initSession(DatagramSocket socket, DatagramPacket
packet) throws IOException {
        String[] args = new String(packet.getData()).trim().split(";");
        if (args.length == 3) {
            try {
                int chunkSize = Integer.parseInt(args[1]);
                String filename = args[2];
                RandomAccessFile in = new RandomAccessFile(new
File(filename), "r");
                Session newSession = new Session(in, chunkSize);
                sessions.add(newSession);
                sendSIDXX(socket, packet.getAddress(), packet.getPort(),
newSession.sessionId);

            } catch (NumberFormatException e) {
                System.out.println("Malformed Request; Not Responding");
            } catch (FileNotFoundException e) {
                sendFNF(socket, packet.getAddress(), packet.getPort());
            }
        }
    }

    private static void getData(DatagramSocket socket, DatagramPacket
packet) throws IOException {
        String[] args = new String(packet.getData()).trim().split(";");
        try {
            int sessionId = Integer.parseInt(args[1]);
            int chunkNumber = Integer.parseInt(args[2]);
            for (Session session : sessions) {
                if (session.sessionId == sessionId) {
                    System.out.println("Getting chunk " + chunkNumber + "
for " + sessionId);
                    byte[] chunk = session.getChunk(chunkNumber);
                    //System.out.println("Chunk: " + new String(chunk));
                    if (chunk.length == 0) {
                        sendCNF(socket, packet.getAddress(),
packet.getPort());
                    }
                    sendDATAX(socket, packet.getAddress(),
packet.getPort(), chunkNumber, chunk);
                    return;
                }
            }
            sendNOS(socket, packet.getAddress(), packet.getPort());
        } catch (NumberFormatException e) {
            System.out.println("Malformed Request; Not Responding");
        }

    }

    private static void sendStr(DatagramSocket socket, InetAddress address,
int port, String message) throws IOException {
        byte[] msgBuff = message.getBytes(Main.CHARSET);
        DatagramPacket packet = new DatagramPacket(msgBuff, msgBuff.length,
address, port);
        socket.send(packet);
    }

    private static void sendFNF(DatagramSocket socket, InetAddress address,
```

```
int port) throws IOException {
        sendStr(socket, address, port, Main.ERROR + ";FNF");
    }

    private static void sendCNF(DatagramSocket socket, InetAddress address,
int port) throws IOException {
        sendStr(socket, address, port, Main.ERROR + ";CNF");
    }

    private static void sendNOS(DatagramSocket socket, InetAddress address,
int port) throws IOException {
        sendStr(socket, address, port,Main.ERROR + ";NOS");
    }

    private static void sendSIDXX(DatagramSocket socket, InetAddress
address, int port, int sessionId) throws IOException {
        sendStr(socket, address, port, Main.SIDXX + ";" + sessionId);
    }

    private static void sendDATAX(DatagramSocket socket, InetAddress
address, int port, int chunkNumber, byte[] data) throws IOException {
        byte[] prefix = (Main.DATAX + ";" + chunkNumber + ";" + data.length
+ ";").getBytes(Main.CHARSET);
        byte[] msgBuff = new byte[prefix.length + data.length];

        System.arraycopy(prefix, 0, msgBuff, 0, prefix.length);
        System.arraycopy(data, 0, msgBuff, prefix.length, data.length);

        DatagramPacket packet = new DatagramPacket(msgBuff, msgBuff.length,
address, port);
        socket.send(packet);
    }


}
```

Session.java:

```
package hsosstp.server;

import java.io.*;

public class Session {

    private static int nextSessionId = 0;

    public final int sessionId;
    private final int chunkSize;
    private final RandomAccessFile in;

    public Session(RandomAccessFile inFile, int chunkSize) {
        sessionId = nextSessionId++;
        this.chunkSize = chunkSize;
        in = inFile;
    }

    public byte[] getChunk(int chunkNumber) throws IOException {

        byte[] buff = new byte[chunkSize];
        in.seek(chunkNumber * chunkSize);
        int n = in.read(buff);
```

```
        if (n == -1) {
            return new byte[0];
        }
        byte[] data = new byte[n];
        System.arraycopy(buff, 0, data, 0, n);
        return data;
    }
}
```

Test Übertragung der Sample.txt:

```
Waiting for SessionId:
Got response:
HSOSSTP_SIDXX;0
SessionId: 0
```

Test Übertragung der Sample.bin:

```
Waiting for SessionId:
Got response:
HSOSSTP_SIDXX;0
SessionId: 0
```

Test Übertragung von nicht existierender Datei:

```
Waiting for SessionId:
Got response:
HSOSSTP_ERROR;FNF
File Not Found on hsosstp.server!
```

Server Rückmeldung von nicht existierender Datei:

```
Waiting for request:
Got request: HSOSSTP_INITX;1024;SampleNotFound.txt
Waiting for request:
```