Alapvető adattovábbítási protokollok

TCP (Transmission Control Protocol)

- **Megbízható**: A TCP biztosítja az adatok pontos, sorrendben történő kézbesítését, visszaigazolások és újraküldések segítségével.
- **Kapcsolatorientált**: A kommunikáció megkezdése előtt kapcsolatot kell létesíteni a két fél között.
- **Áramlásszabályozás és zsúfoltságkezelés**: Szabályozza az adatátvitel sebességét a hálózat és a végpontok aktuális állapota alapján.
- **Alkalmazások**: Webböngészés (HTTP/HTTPS), e-mail (SMTP, IMAP/POP3), fájlátvitel (FTP), és más, a megbízható adatátvitelt igénylő alkalmazások.

UDP (User Datagram Protocol)

- **Nem megbízható**: Nem garantálja az adatok sorrendjét vagy sikeres kézbesítését; nincs csomag újraküldés vagy csomagsorrend helyreállítás.
- **Kapcsolatmentes**: Nem igényel előzetes kapcsolatfelépítést az adatok küldése előtt, lehetővé téve a gyors adattovábbítást. E miatt valós idejű adattovábbítást biztosít.
- **Könnyűsúlyú**: Kevesebb fejlécinformációt használ, ami kevesebb hálózati terhelést jelent. 8 byte a fejléc mindössze.
- **Alkalmazások**: Streaming média (videó, audio), online játékok, VoIP (Voice over Internet Protocol), IOT (Internet of Things) és más időkritikus alkalmazások, ahol a sebesség fontosabb, mint a megbízhatóság.

QUIC (Quick UDP Internet Connections) (2021-es szabvány)

- **Párhuzamos, Multiplexált adatfolyam**: Egyetlen QUIC-kapcsolat több adatfolyamot is képes kezelni, ezáltal csökkentve az úgynevezett “fejlécblokkolást”, ami a TCP kapcsolatokban előfordulhat.
- **Független hiba- és áramlásszabályozás**: Minden QUIC-adatfolyam saját hiba- és áramlásszabályozással rendelkezik, ami azt jelenti, hogy egy folyam problémái nem befolyásolják a többi adatfolyam teljesítményét.
- **Dinamikus prioritások**: A QUIC lehetővé teszi az adatfolyamok prioritásának dinamikus módosítását, amely segít optimalizálni az erőforrások felhasználását és javítja az alkalmazások válaszidejét.
- **Titkosítás**: A QUIC alapértelmezés szerint biztosítja az adatok végponttól végpontig történő titkosítását, használva a TLS (Transport Layer Security) legújabb verzióit, ezáltal javítva az adatbiztonságot.
- **Kapcsolat migráció**: A QUIC képes fenntartani egy aktív kapcsolatot még akkor is, ha a felhasználó eszköze hálózatot vált (például Wi-Fi-ről mobil adatra), ami folyamatosabb élményt nyújt a mobil felhasználók számára.
- **Alkalmazások**: A QUIC-t széles körben használható (a normál böngészésen kívül) videó streaminghez, online játékokhoz, IOT (Internet of Things) és más, nagy sebességű és megbízhatóságot igénylő internetes alkalmazásokhoz.

további részletesebb információ

Gyakorló feladat

Készítsen egy egyszerűsített FTP (file transport) klienst és szervert, amelynél a kliens elküldhet vagy letölthet szöveges file-okat a szerverről. Általános funkció leírás:

1.) Kliens becsatlakozik a szerverhez és küld egy listázás üzenetet
2.) Szerver visszaküldi a tárolt file-ok listáját (vagy előzőleg feltöltött file-ok listáját)
3.) Kliens kilistázza a fileokat, és bekéri a felhasználótól, hogy milyen műveletet szeretne végezni? Feltöltés vagy letöltés? ('u' vagy 'd')
4.) Mindkét esetben be kell írni a file nevét kiterjesztéssel együtt
5.) A kliens elküldi a szerverre a kiválasztott file-t, vagy letölti a kiválasztott file-t egy adott könyvtárba.

** Szerver nézőpont: **

1.) Becsatlakozás után felolvassa a file-okat a /store alkönyvtárból és a listázás üzenet megérkezése után a fájlneveket elküldi a kliensnek.
2.) Várakozunk a kliens 'u' vagy 'd' műveletére
3.) Klienstől kapunk egy filenevet és ha 'd' (download) a művelet, akkor felolvassuk a file-t és visszaküldjük a tartalmát
4.) Ha a művelet 'u' (feltöltés), akkor nyitunk egy új file-t a megadott néven és várjuk az adatokat, amiket kiírunk a file-ba.

Kliens nézőpont

1.) A kliens becsatlakozik és várja a visszajövő fájlok listáját, majd ha megjön akkor kiírjuk a konzolra
2.) Bekérjük a “u” vagy “d” billentyűt
3.) Majd kérjük a file-nevet is.
4.) a kliens a /files könvytárából olvassa a file-okat, vagy a letöltött file-t is ide hozza létre
5.) “d” billentyű esetén létrehozza a /files/<filename> állományt és a szerverről jövő adatokat beleírja
6.) “u” billentyű esetén a /files/<filename> állományt elküldi a szervernek

Kiinduló minták

1.) Hagyományos blokkolt TCP alapú socket szerver

Socket szerver kód

```
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
```

```
    import java.net.Socket;
    public class Server {
        ServerSocket providerSocket;
        Socket connection = null;
        ObjectOutputStream out;
        ObjectInputStream in;
        String message;
        Server() {
        }
        void run() {
            try {
                // 1. szerver socket létrehozása
                providerSocket = new ServerSocket(8080);
                // 2. kapcsolódásra várakozás
                connection = providerSocket.accept();
                // 3. Input és Output streamek megadása
                out = new ObjectOutputStream(connection.getOutputStream());
                in = new ObjectInputStream(connection.getInputStream());
                // 4. socket kommunikáció
                do {
                    try {
                        message = (String) in.readObject();
                        System.out.println("client>" + message);
                        if (message.equals("bye")) {
                            sendMessage("bye");
                        }
                    } catch (ClassNotFoundException classnot) {
                        System.err.println("Data received in unknown
format");
                    }
                } while (!message.equals("bye"));
            } catch (IOException ioException) {
                ioException.printStackTrace();
            } finally {
                // 4: kapcsolat lezárása
                try {
                    in.close();
                    out.close();
                    providerSocket.close();
                } catch (IOException ioException) {
                    ioException.printStackTrace();
                }
            }
        }
        void sendMessage(String msg) {
            try {
                out.writeObject(msg);
                out.flush();
                System.out.println("server>" + msg);
            } catch (IOException ioException) {
                ioException.printStackTrace();
```

```
            }
        }
        public static void main(String args[]) {
            Server server = new Server();
            while (true) {
                server.run();
            }
        }
    }
```

Socket kliens kód

```
    import java.io.IOException;
    import java.io.ObjectInputStream;
    import java.io.ObjectOutputStream;
    import java.net.Socket;
    import java.net.UnknownHostException;
    public class Client {
        Socket requestSocket;
        ObjectOutputStream out;
        ObjectInputStream in;
        String message;
        Client() {
        }
        void run() {
            try {
                // 1. socket kapcsolat létrehozása
                requestSocket = new Socket("localhost", 8080);
                // 2. Input and Output streamek
                out = new
ObjectOutputStream(requestSocket.getOutputStream());
                in = new ObjectInputStream(requestSocket.getInputStream());
                // 3: Kommunikáció
                do {
                    try {
                        sendMessage("Hello szerver");
                        sendMessage("bye");
                        message = (String) in.readObject();
                    } catch (Exception e) {
                        System.err.println("data received in unknown
format");
                    }
                } while (!message.equals("bye"));
            } catch (UnknownHostException unknownHost) {
                System.err.println("You are trying to connect to an unknown
host!");
```

```
            } catch (IOException ioException) {
                ioException.printStackTrace();
            } finally {
                // 4: Kapcsolat zárása
                try {
                    in.close();
                    out.close();
                    requestSocket.close();
                } catch (IOException ioException) {
                    ioException.printStackTrace();
                }
            }
        }
        void sendMessage(String msg) {
            try {
                out.writeObject(msg);
                out.flush();
                System.out.println("client>" + msg);
            } catch (IOException ioException) {
                ioException.printStackTrace();
            }
        }
        public static void main(String args[]) {
            Client client = new Client();
            client.run();
        }
    }
```

2.) Hagyományos UDP alapú kommunikáció

a) Az alábbi Ágens küld egy üzenetet és a 8080-as porton várja a választ rá, ugyancsak UDP-vel.
Megjegyzés: az Eclipse fejlesztőkörnyezetben a consolon beírt szöveget ctrl+z leütésével lehet elküldeni.

Feladat: módosítsuk a kódot, hogy át tudjon küldeni egy beégetett nevű, és létező, 2 kbyte-nál nagyobb szöveges vagy kép állományt és ellenőrizzük a sikeres küldést.

```
  package org.ait;
  import java.io.BufferedReader;
  import java.io.InputStreamReader;
  import java.net.DatagramPacket;
  import java.net.DatagramSocket;
  import java.net.InetAddress;
  public class Component1 {
    public static void main(String args[]) throws Exception {
        BufferedReader inFromUser = new BufferedReader(new
InputStreamReader(System.in));
        DatagramSocket clientSocket = new DatagramSocket();
```

```
        InetAddress IPAddress = InetAddress.getByName("localhost");
        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];
        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes();
        DatagramPacket sendPacket = new DatagramPacket(sendData,
sendData.length, IPAddress, 8080);
        clientSocket.send(sendPacket);
        DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
        clientSocket.receive(receivePacket);
        String modifiedSentence = new String(receivePacket.getData());
        System.out.println("átalakítva:" + modifiedSentence);
        clientSocket.close();
    }
    }
```

b) A másik UDP komponens (Component2) a 8080-as porton várja az ágens üzeneteit és nagybetűre konvertálva visszaküldi a Component1 UDP socket-ére.

```
  package org.ait;
  import java.net.DatagramPacket;
  import java.net.DatagramSocket;
  import java.net.InetAddress;
  public class Component2 {
    public static void main(String args[]) throws Exception {
        DatagramSocket serverSocket = new DatagramSocket(8080);
        byte[] bytesReceived = new byte[1024];
        byte[] bytesSent = new byte[1024];
        DatagramPacket receivePacket = new DatagramPacket(bytesReceived,
bytesReceived.length);
        // itt várakozik ameddig adat jön a 8080-as porton
        serverSocket.receive(receivePacket);
        String szoveg = new String(receivePacket.getData());
        System.out.println("kaptam: " + szoveg);
        InetAddress IPAddress = receivePacket.getAddress();
        int port = receivePacket.getPort();
        String nagybetűsSzöveg = szoveg.toUpperCase();
        bytesSent = nagybetűsSzöveg.getBytes();
        // visszaküldi
        DatagramPacket sendPacket = new DatagramPacket(bytesSent,
bytesSent.length, IPAddress, port);
        serverSocket.send(sendPacket);
        serverSocket.close();
    }
  }
```

Ha egy nagyméretű adatot szeretnénk elküldeni, egyben nem tudjuk megtenni, mert nem fér bele egy UDP csomagba. Ilyenkor egy ciklusban fel kell tördelni pl. 1024 byte hosszú darabokra és ezt kell átküldeni a szervernek. A szerveren ugyanígy egy ciklusban kell beolvasni a byte-okat és összefűzni az 1024 byte-os darabokat.

De ekkor még egy problémát meg kell oldani: honnan tudjuk hogy mekkora a tényleges méret?

From:
https://edu.iit.uni-miskolc.hu/ - **Institute of Information Science - University of Miskolc**

Permanent link:
https://edu.iit.uni-miskolc.hu/tanszek:oktatas:informacios_rendszerek_integralasa:java_socket

Last update: **2024/03/06 08:34**