Programska potpora komunikacijskim sustavima

10. predavanje, 24. svibnja 2023.

doc. dr. sc. Jelena Božek

Protokoli UDP i SCTP

SVEUČILIŠTE U ZAGREBU
Fakultet elektrotehnike i računarstva

- Predavanja izradio prof. dr. sc. Krešimir Pripužić, 2022.
- Predavanja doradila doc. dr. sc. Jelena Božek, 2023.

Sadržaj predavanja

- Protokol UDP
- Primjer klijenta i poslužitelja koji komuniciraju protokolom UDP
- Protokol SCTP
- Primjer klijenta i poslužitelja koji komuniciraju protokolom SCTP

Protokol UDP

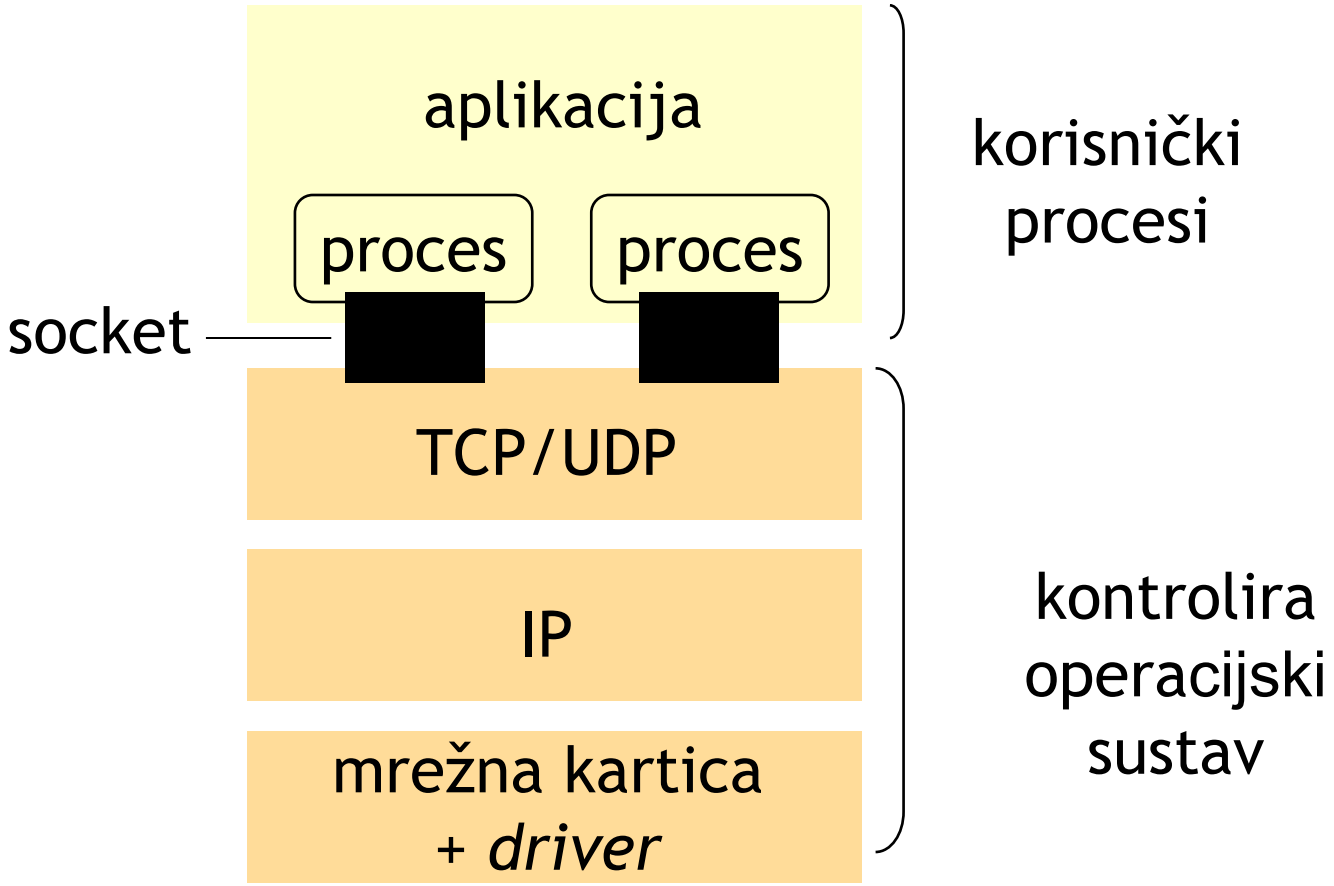
SVEUČILIŠTE U ZAGREBU
Fakultet elektrotehnike i računarstva

Komunikacija korištenjem priključnica

Socket API

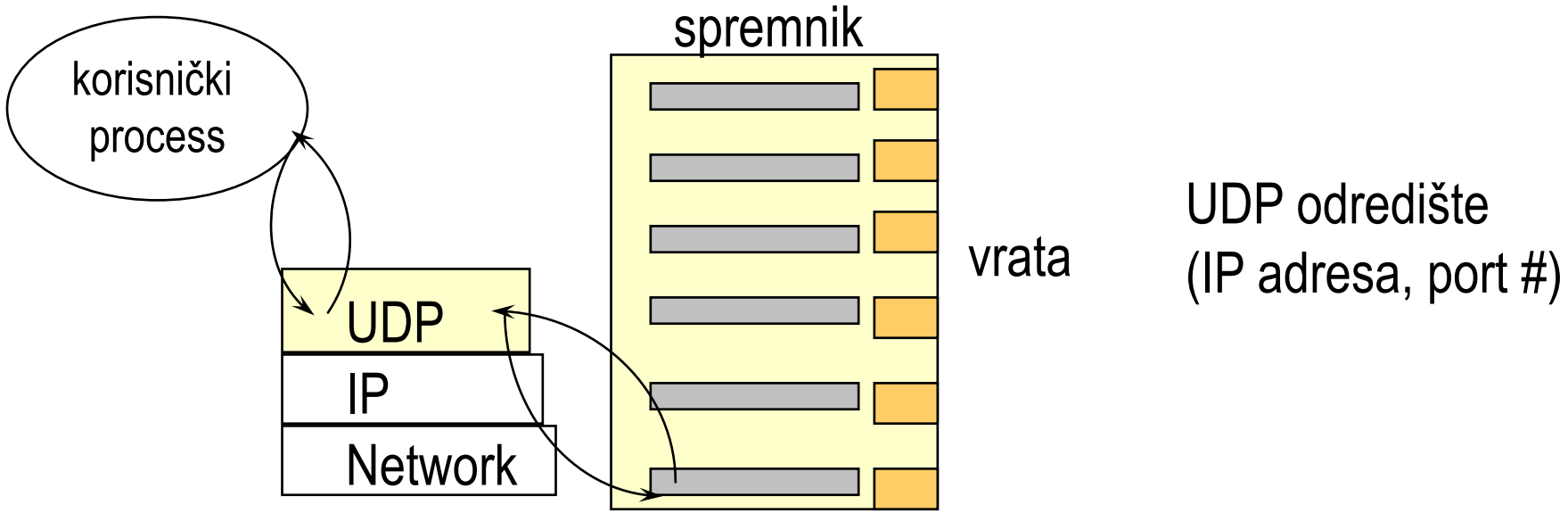
- koristi funkcionalnost transportnog sloja
  - TCP – konekcijski protokol, pouzdan prijenos podataka
  - UDP – prijenos nezavisnih paketa (*datagrami*), nepouzdan prijenos
- priključnica (*socket*)
  - pristupna točka preko koje aplikacija šalje podatke u mrežu i iz koje čita primljene podatke
  - viši nivo apstrakcije nad komunikacijskom točkom koju operacijski sustav koristi za pristup transportnom sloju
  - veže se uz vrata (*port*) koja jednoznačno određuju aplikaciju kojoj su poruke namijenjene

Komunikacija pomoću priključnica

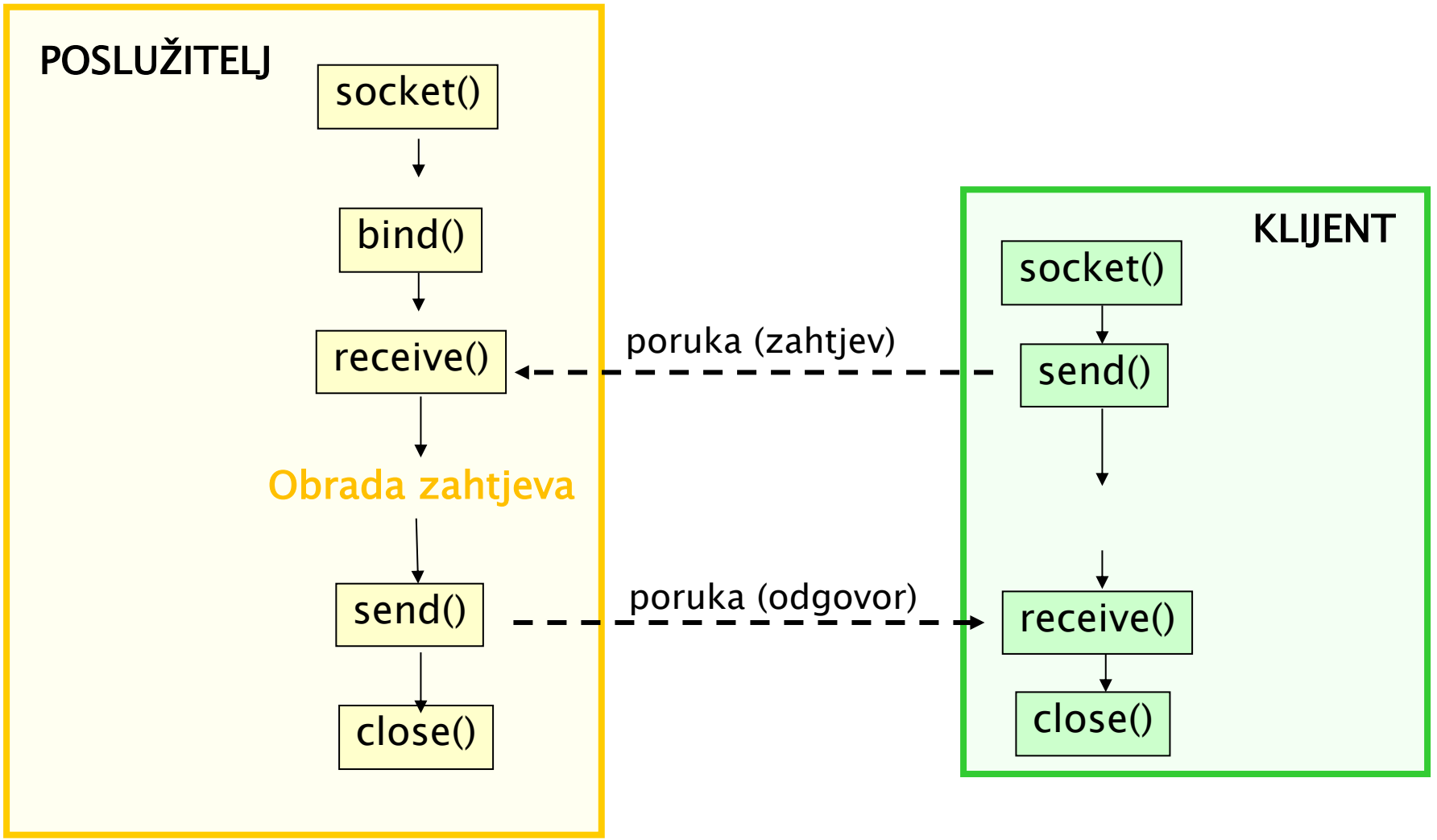


Transportni protokol UDP

- User Datagram Protocol (UDP)
  - komunikacija se odvija preko vrata (engl. portova) koje dodjeljuje operacijski sustav na strani klijenta, na strani poslužitelja se koriste “dobro poznata vrata”



Komunikacija pomoću *socketa UDP*



Obilježja *socketa* UDP

- model klijent-poslužitelj
- vremenska ovisnost procesa
  - poslužitelj mora biti aktivan za primanje datagrama
- klijent mora znati identifikator poslužitelja
- tranzijentna komunikacija
- asinkrona komunikacija
  - klijent šalje datagram i nastavlja obradu, nema blokiranja pošiljatelja
- nepouzdana komunikacija
- može se koristiti za implementaciju komunikacije na načelu *pull* i *push*

UDP: implementacija klijenta

1. Kreirati socket:

```
DatagramSocket socket = new DatagramSocket();
```

2. Kreirati paket i napuniti ga podacima:

```
byte[] sndBuffer = new byte[256];

DatagramPacket packet = new DatagramPacket(sndBuffer,
  sndBuffer.length, dstAddress, dstPort);
```

3. Slanje paketa:

```
socket.send( packet );
```

4. Po potrebi obrada i čekanje odgovora

5. Zatvoriti socket:

```
socket.close();
```

UDP: implementacija poslužitelja

1. Kreirati socket poslužitelja:

```
DatagramSocket socket = new DatagramSocket( PORT );
```

2. Kreirati paket (prazan, priprema za primanje):

```
byte[] rcvBuffer = new byte[256];
DatagramPacket packet = new DatagramPacket(rcvBuffer,
rcvBuffer.length);
```

3. Čekati korisnički paket (blokira proces do klijentskog zahtjeva!):

```
socket.receive(packet);
```

4. Obrada pristiglog paketa i po potrebi odgovor klijentu
5. Zatvoriti socket (gasi poslužitelja!):

```
serverSocket.close();
```

Primjer UDP klijenta i poslužitelja

- Klijent šalje senzorska očitanja u JSON formatu
- Server parsira primljena senzorska očitanja i ispisuje ih na konzolu

Klasa UDPClient [1]

```
public class UDPClient {
    public static void main(String[] args) {
        try ( DatagramSocket socket = new DatagramSocket()) { //SOCKET
            while (true) {
                //send a reading
                Reading reading = new Reading("id_5", "temperature", (Math.random() * 60) - 20, "C");
                byte[] buffer = reading.toJson().getBytes();

                DatagramPacket packet = new DatagramPacket(buffer, buffer.length,
InetAddress.getByName("localhost"), 11111);
                socket.send(packet); //SEND
                System.out.println("Sent: " + reading);
```

Klasa UDPClient [2]

```
                //receive a confirmation
                buffer = new byte[1024];
                packet = new DatagramPacket(buffer, buffer.length);
                socket.receive(packet); //RECEIVE
                String confirmation = new String(packet.getData(), 0, packet.getLength());
                System.out.println("Received: " + confirmation);

                Thread.sleep(3000);
            }

        } catch (IOException | InterruptedException ex) {
            ex.printStackTrace();
        } //CLOSE
    }
}
```

Klasa UDPServer [1]

```
public class UDPServer {
    public static void main(String[] args) {
        try ( DatagramSocket socket = new DatagramSocket(11111)) {//SOCKET -> BIND
            while (true) {
                //receive a reading
                byte[] buffer = new byte[1024];
                DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
                socket.receive(packet); //RECEIVE
                String json = new String(packet.getData(), 0, packet.getLength());
                Reading reading = Reading.fromJson(json);
                System.out.println("Received: " + reading);
```

Klasa UDPServer [2]

```
                //send a confirmation
                String confirmation = "OK";
                buffer = confirmation.getBytes();

                packet = new DatagramPacket(buffer, buffer.length,
packet.getAddress(), packet.getPort());
                socket.send(packet); //SEND
                System.out.println("Sent: " + confirmation);
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        } //CLOSE
    }
}
```

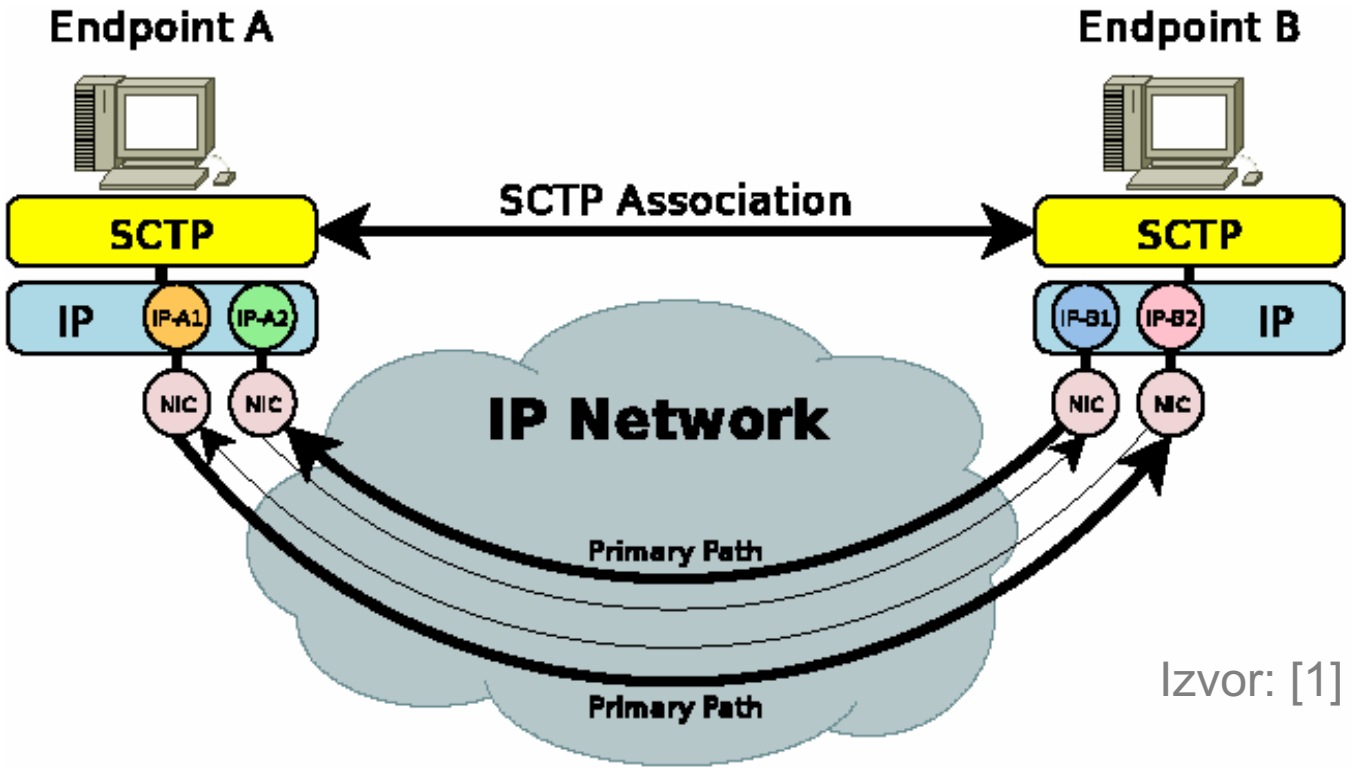
Protokol SCTP

SVEUČILIŠTE U ZAGREBU
Fakultet elektrotehnike i računarstva

Stream Control Transport Protocol (SCTP)

- Pouzdani protokol transportnog sloja
- Sličan je TCP-u po tome što je pouzdan
- Sličan je UDP-u po tome što se prijenos podataka odvija porukama, a ne korištenjem tokova okteta kao kod TCP-a
- *Multihoming*
  - Omogućava (mrežnu) redundanciju jer krajnja točka može biti predstavljena s više adresa za slanje i primanje podataka
  - Pri tome je port isti na svim redundantnim adresama
- Svaka asocijacija podržava više logičkih tokova poruka

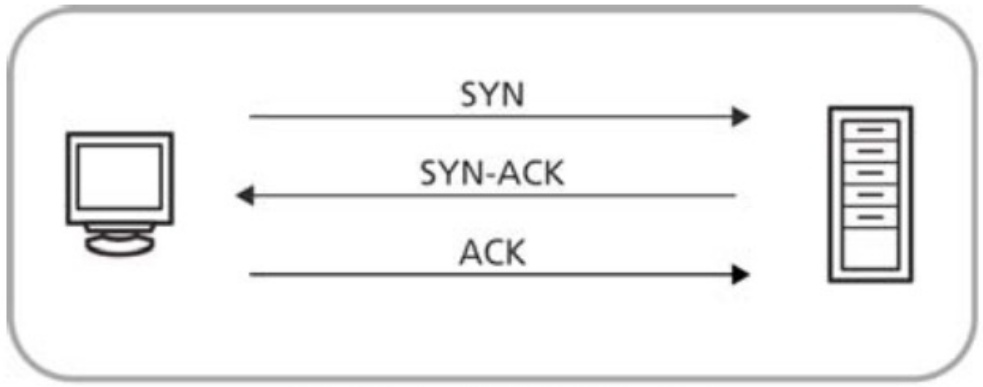
Multihoming kod SCTP-a



Izvor: [1]

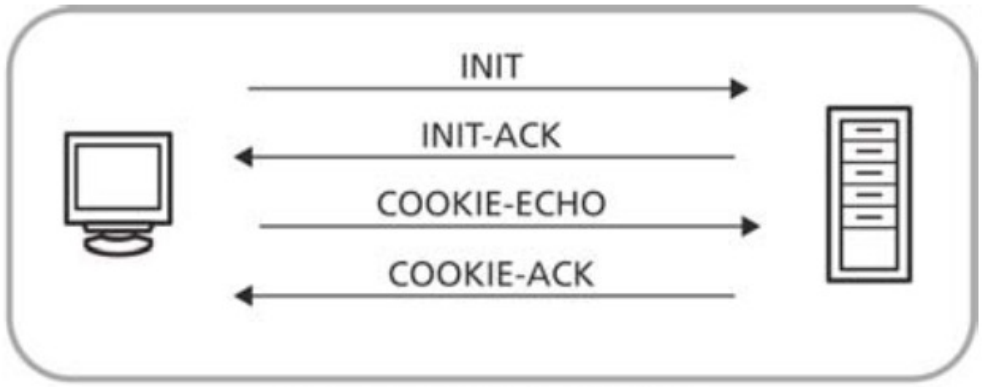
Uspostava i prekidanje konekcije kod TCP-a i SCTP-a

TCP: three-way handshake

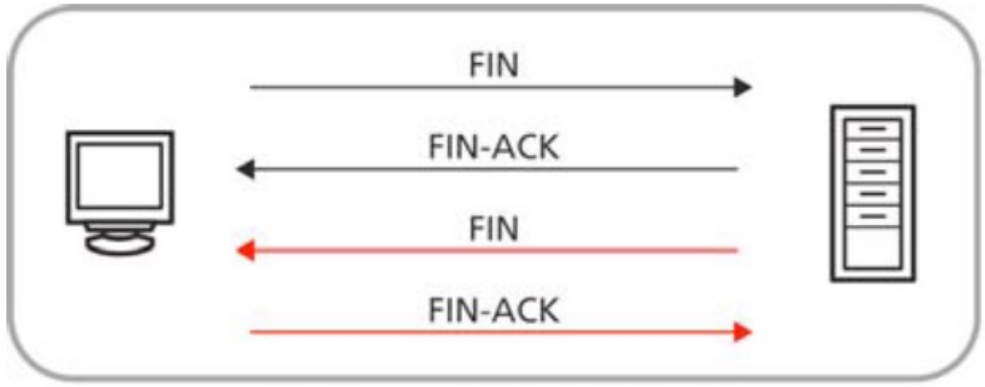


TCP Connection Initiation

SCTP: four-way handshake



SCTP Connection Initiation



TCP Connection Termination
(Half-open shown by red)

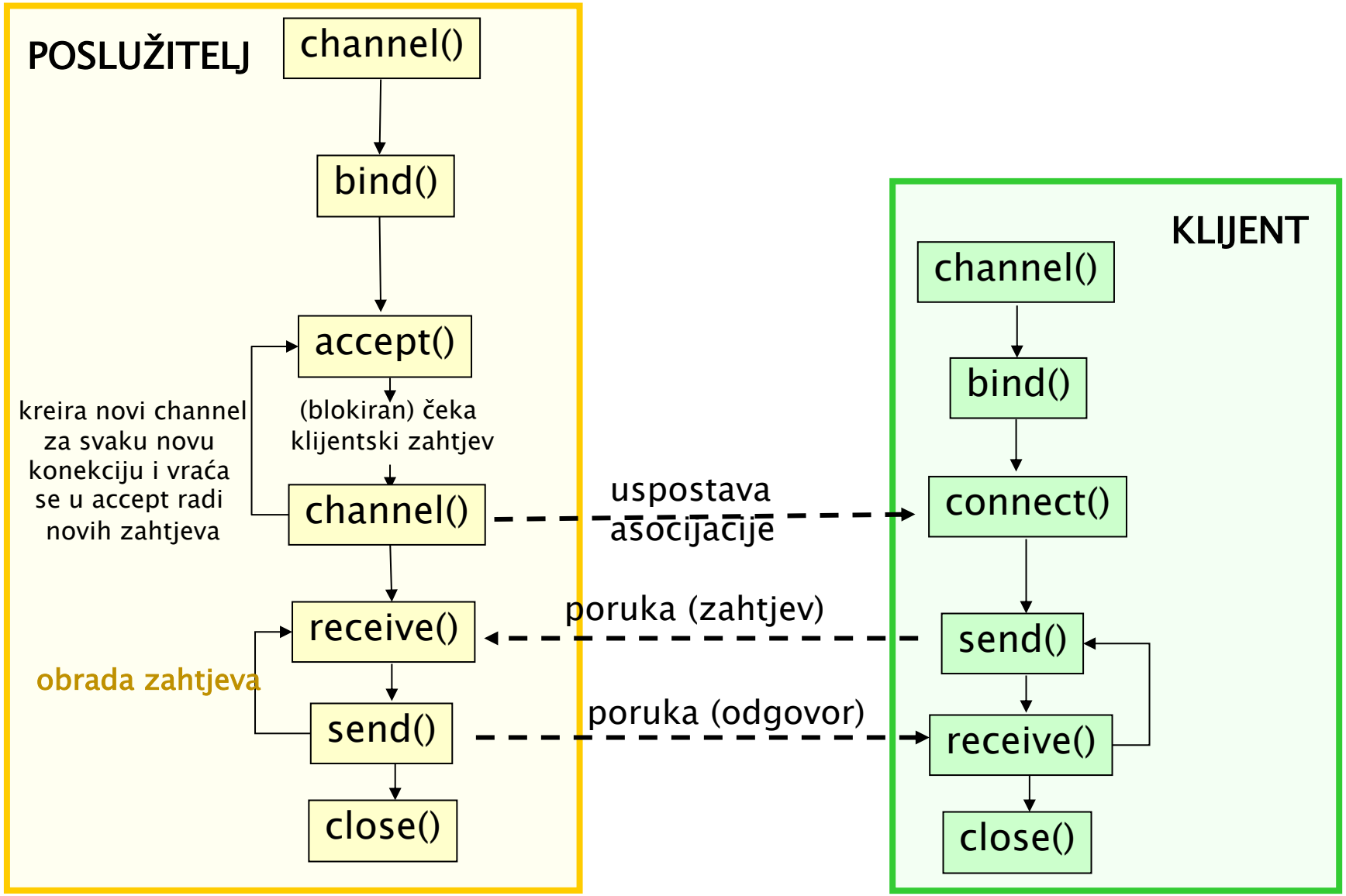
SCTP Close Connection

Izvor: [2]

Razlike između protokola TCP, UDP i SCTP

	TCP	UDP	SCTP
Pouzdanost	Da	Ne	Da
Konekcijska komunikacija	Da	Ne	Da
Prijenos podataka	Tok okteta	Poruke	Poruke
Kontrola toka	Da	Ne	Da
Kontrola zagušenja	Da	Ne	Da
Tolerancija na ispade	Ne	Ne	Da (*Multihoming*)
Redoslijed	Uređen	Neuređen	Djelomično uređen
Sigurnost	Da	Da	Poboljšana

Konekcijska komunikacija pomoću kanala SCTP



Obilježja protokola SCTP

- model klijent-poslužitelj
- vremenska ovisnost procesa
  - poslužitelj mora biti aktivan za primanje datagrama
- klijent mora znati identifikator poslužitelja
- tranzijentna komunikacija
- asinkrona komunikacija
  - klijent šalje datagram i nastavlja obradu, nema blokiranja pošiljatelja
- pouzdana komunikacija

SCTP: implementacija klijenta

1. Kreirati klijentski kanal:

```
SctpChannel clientSctpChannel =
SctpChannel.open().bind(clientSocketAddress));
```

2. Povezati se s poslužiteljem:

```
clientSctpChannel.connect(serverSocketAddress, 1, 1);
```

3. Komunikacija porukama s poslužiteljem

```
clientSctpChannel.send(byteBuffer, messageInfo);
clientSctpChannel.receive(byteBuffer, null, null);
```

4. Zatvoriti kanal:

```
clientSctpChannel.close();
```

SCTP: implementacija poslužitelja

1. Kreirati poslužiteljski kanal:

```
SctpServerChannel sctpServerChannel;
sctpServerChannel = SctpServerChannel.open().bind(new
InetSocketAddress(PORT)));
```

2. Čekati korisnički zahtjev (blokira proces do klijentskog zahtjeva!!!) i kreirati kopiju originalnog kanala:

```
SctpChannel clientSctpChannel = sctpServerChannel.accept()
```

3. Komunikacija porukama s klijentom

```
clientSctpChannel.receive(byteBuffer, null, null);
clientSctpChannel.send(byteBuffer, messageInfo);
```

4. Zatvoriti kopiju kanala:

```
clientSctpChannel.close();
```

5. Zatvoriti poslužiteljski kanal:

```
sctpServerChannel.close();
```

Primjer SCTP klijenta i poslužitelja

- Klijent šalje senzorska očitanja u JSON formatu
- Server parsira primljena senzorska očitanja i ispisuje ih na konzolu

Klasa `SCTPClient` [1]

```
public class SCTPClient {

    public static void main(String[] args) {
        InetSocketAddress serverSocketAddress = new InetSocketAddress("localhost", 54321);
        InetSocketAddress clientSocketAddress = new InetSocketAddress(54325);

        try ( SctpChannel clientSctpChannel = SctpChannel.open().bind(clientSocketAddress)) { //CHANNEL -> BIND

            clientSctpChannel.connect(serverSocketAddress, 1, 1); //CONNECT

            while (true) {
                ...
            }

        } catch (IOException | InterruptedException ex) {
            ex.printStackTrace();
        } //CLOSE
    }
}
```

Klasa `SCTPClient` [1]

```
//start sending readings
while (true) {
    //send a reading
    Reading reading = new Reading("id_4", "temperature", (Math.random() * 60) - 20, "C");
    byte[] message = reading.toJson().getBytes("UTF-8");
    ByteBuffer byteBuffer = ByteBuffer.wrap(message);
    System.out.println("Sent: " + reading);
    MessageInfo messageInfo = MessageInfo.createOutgoing(null, 0);
    clientSctpChannel.send(byteBuffer, messageInfo); //SEND

    //receive a confirmation
    byteBuffer = ByteBuffer.allocate(1024);
    messageInfo = clientSctpChannel.receive(byteBuffer, null, null); //RECEIVE
    message = Arrays.copyOfRange(byteBuffer.array(), 0, messageInfo.bytes());
    String confirmation = new String(message, "UTF-8");
    System.out.println("Received: " + confirmation);

    Thread.sleep(2000);
}
```

Klasa SingleClientSCTPServer [1]

```
public class SingleClientSCTPServer {

    private int port;

    public SingleClientSCTPServer(int port) {
        this.port = port;
    }

    public static void main(String[] args) {
        SingleClientSCTPServer server = new SingleClientSCTPServer(12345);
        server.start();
    }
```

Klasa `SingleClientSCTPServer` [2]

```
public void start() {
        SocketAddress serverSocketAddress = new InetSocketAddress(54321);

        try ( SctpServerChannel stcpServerChannel = SctpServerChannel.open().bind(serverSocketAddress)) { //CHANNEL -> BIND

            SctpChannel clientSctpChannel = stcpServerChannel.accept(); //ACCEPT -> CHANNEL


            while (true) {
                try {
                    //receive a reading
                    ByteBuffer byteBuffer = ByteBuffer.allocate(1024);
                    MessageInfo messageInfo = clientSctpChannel.receive(byteBuffer, null, null); //RECEIVE
                    byte[] message = Arrays.copyOfRange(byteBuffer.array(), 0, messageInfo.bytes());
                    String jsonReading = new String(message, "UTF-8");
                    Reading reading = Reading.fromJson(jsonReading);
                    System.out.println("Received: " + reading);
```

Klasa `SingleClientSCTPServer` [3]

```
                    //send a confirmation
                    String confirmation = "OK";
                    message = confirmation.getBytes("UTF-8");
                    byteBuffer = ByteBuffer.wrap(message);
                    System.out.println("Sent: " + confirmation);
                    messageInfo = MessageInfo.createOutgoing(null, 0);
                    clientSctpChannel.send(byteBuffer, messageInfo); //SEND
                } catch (IOException ex) {
                    ex.printStackTrace();
                }
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        } //CLOSE
    }
```

Klasa SCTPServer [1]

```
public class SCTPServer {
    private int port;

    public SCTPServer(int port) {
        this.port = port;
    }


    public void start() {
        SocketAddress serverSocketAddress = new InetSocketAddress(54321);

        try ( SctpServerChannel stcpServerChannel = SctpServerChannel.open().bind(serverSocketAddress)) { //CHANNEL -> BIND

            while (true) {
                SctpChannel clientSctpChannel = stcpServerChannel.accept(); //ACCEPT -> CHANNEL
                new Thread(new ServerTask(clientSctpChannel)).start();
            }

        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    public static void main(String[] args) {
        SCTPServer server = new SCTPServer(12345);
        server.start();
    }
```

Klasa SCTPServer [2]

```
private class ServerTask implements Runnable {

        private SctpChannel sctpChannel;

        public ServerTask(SctpChannel sctpChannel) {
            this.sctpChannel = sctpChannel;
        }

        @Override
        public void run() {
            while (true) {
                try {
                    //receive a reading
                    ByteBuffer byteBuffer = ByteBuffer.allocate(1024);
                    MessageInfo messageInfo = sctpChannel.receive(byteBuffer, null, null); //RECEIVE
                    byte[] message = Arrays.copyOfRange(byteBuffer.array(), 0, messageInfo.bytes());
                    String jsonReading = new String(message, "UTF-8");
                    Reading reading = Reading.fromJson(jsonReading);
                    System.out.println("Received: " + reading);
```

Klasa SCTPServer [2]

```
                    //send a confirmation
                    String confirmation = "OK";
                    message = confirmation.getBytes("UTF-8");
                    byteBuffer = ByteBuffer.wrap(message);
                    System.out.println("Sent: " + confirmation);
                    messageInfo = MessageInfo.createOutgoing(null, 0);
                    sctpChannel.send(byteBuffer, messageInfo); //SEND
                } catch (IOException ex) {
                    ex.printStackTrace();
                }
            }
        }
    }
}
```

Literatura

[1] Dreibholz, T., Zhou, X., Becke, M., Pulinthanath, J., Rathgeb, E. and Du, W., 2010. On the security of reliable server pooling systems. International Journal of Intelligent Information and Database Systems, 4(6), pp.552-578.

[2] Introduction to the Stream Control Transmission Protocol (SCTP): The next generation of the Transmission Control Protocol (TCP)