Tentamen
Nätverksprogrammering
Lösningsförslag

2010–06–01, 08.00–13.00

---

1. a) TCP och UDP
   b) TCP
   c) UDP
   d) TCP

2. Real-time transport protocol (RTP) är ett protocol ovanpå TCP/UDP som identifierar innehållet samt lägger till tidsstämpel och sekvensnummer. Real-time control protocol (RTCP) är en del av RTP-protokolet. Det skickar periodiskt paket att ge statistik för överföringskvaliteten.

   Real-time streaming protocol (RTSP) är ett HTTP-orienterat protokoll för att övervaka strömmande media. Det har funktionalit som en sorts fjärrkontroll. Det finns ingen direkt länk till RTP. För att fungera måste RTP ha ett övervakningsprotokoll. Det kan vara RTSP, SIP eller något annat.

3. XHTML är en omformulering av HTML i XML och huvudskillnaden är att XHTML-dokument ställer krav på en välformad struktur: öppnande och slutande element måste vara balancerade.

4. Marshalling innebär att objekt som ska skickas över nätverket kodas om till en seriell form som en serie av bytes. Dessa kan sedan skickas över nätverksförbindelsen varvid objekten kan återskapas på den mottagande sidan (demarshalling).

5. Både GET och POST är kommandon för att hämta ner en webbsida eller annan typ av data från webbservern. Eventuellt kan en så kallad query-sträng skickas med kommandot, t.ex. efter att ett webbformulär fyllts i. Överföringen av denna query-sträng skiljer sig åt mellan GET och POST. I GET inkluderas strängen i själva URL:en, medan för POST skickas den efter eventuella metadatarader som följer på raden innehålland POST-kommandot. Det finns en semantisk skillnad mellan GET och POST i det att GET bör användas endast då kommandot inte förväntas få några bieffekter på servern. POST bör däremot förväntas kunna ha sidoeffekter, t.ex. att data ändras i en databas på serversidan. HEAD fungerar likadant som GET fast innehållet i det efterfrågade dokumentet skickas inte över. Endast den metadata som skickas i headerraderna i svaret inkluderas.

6. a) Sant
   b) Falskt
   c) Falskt
   d) Sant
   e) Falskt (vår data delas visserligen upp i mindre paket, men inte i UDP-paket)
   f) Sant

7. a) Busy-wait

   b) Den tar väldigt mycket CPU-tid och kan faktiskt även i värsta fall förhindra andra trådar att köra.

   c)
```
public class BinarySemaphore {
    private boolean locked = false;

    public synchronized void take() {
        while(locked) {
            try {
                wait();
            } catch(InterruptedException e} { }
        }
        locked = true;
    }

    public synchronized void give() {
        locked = false;
        notifyAll();
    }
}
```

8. a) Det finns ingen garanti att anropet faktiskt läser 100 bytes. Det beror på hur mycket data som har anlänt.

   b)
```
int bytesRead = 0;
while(bytesRead<100) {
    bytesRead += input.read(buffer,bytesRead,100-bytesRead);
}
```

1. Överföring av sensordata med UDP

Vi börjar med att fundera över vad uppgiften går ut på på en lite högre nivå. Det vi har är en eller flera klienter (styrdatorer) som ska kommunicera via UDP (multicast och unicast) med en eller flera servrar (sensordatorer). På klientsidan ska vi bara skriva kod för att kommunicera med en enda enskild server. Klienten känner inte till adressen till servern, utan förväntas använda multicast för att ta reda på detta. Förutom hanteringen av försvunna meddelanden (timeout vid ej svar/bortsortering av gamla meddelanden) är problemställningen likvärdig med det som gjordes på laboration 1 i kursen.

Ett designval vi behöver göra är om vi vill använda separata sockets för multicast och unicast. Detta påverkar komplexiteten i lösningen en del. Man kan tycka att det kan vara en god idé att hålla isär multicast och unicast och därför använda två olika sockets och två olika portnummer. Lösningen kan anses bli klarare på det sättet. Det är dock inte nödvändigt att göra så. Det går alldeles utmärkt att skicka ett UDP-paket med hjälp av unicast till en multicastsocket. Det går också bra att skicka ett unicastpaket från en multicastsocket om man skulle vilja. Det som avgör om det blir ett multicastpaket eller unicastpaket är vilken adress paketet är adresserat till. Ett möjligt extra problem som kan uppstå är att mottagaren kanske måste ha ett sätt att ta reda på om ett mottaget paket var sänt med multicast eller unicast. Men måsta man verkligen det i vårt fall? Vi får se...

Nedan ges två olika förslag till lösning. Först ges en lösning baserad på separata sockets för unicast och multicast. Designvalet får inte så stor effekt på klientsidan mer än att man naturligtvis blir tvungen att skapa två olika sockets. På serversidan uppstår dock problemet att vi måste bevaka båda våra sockets samtidigt eftersom multicast och unicast kan anlända oberoende av varandra. På laboration 1 löstes detta genom att ha två olika program körandes på samma dator. Dock lärde vi oss hur trådar fungerar redan till laboration 2 och genom att låta två olika trådar hantera våra sockets kommer vi enkelt runt problemet.

Det andra lösningsförslaget nedan använder sig av en multicastsocket på båda sidor, vilket förenklar koden en hel del. Eftersom det bara finns en socket på serversidan att bevaka behöver vi inte införa extra trådar. Vi slipper även att dubblera en mängd kod på båda sidor av nätverksförbindelsen. Det visar sig faktiskt även att man inte ens behöver särbehandla multicast och unicast i servern. I båda fallen skickar vi tillbaka aktuellt sensorvärde i ett unicastpaket – förutsatt att det sensorid som finns i paketet överensstämmer med serverns id. Var det frågan om ett multicastmeddelande kommer det skickade sensorvärdet visserligen inte att användas av klienten, men det gör ingen skada. Omvänt, om det var frågan om ett unicastpaket är sensorid:et i paketet överflödigt, men vi behåller det för enkelhetens skull.

I båda lösningarna ville man för absolut full poäng sortera bort eventuella gamla meddelanden som var väldigt fördröjda (med rimlig säkerhet). Ett enkelt sätt är att förse varje begäran som skickas till servern med ett serienummer. Detta serienummer returnerar sedan servern i svaret. Stämmer inte serienumret i svaret med det skickade numret kan meddelandet sorteras bort.

Det stod ingenting i uppgiften om hur många sensorer som kunde finnas, men i lösningarna nedan har vi antagit att det kan vara maximalt 255 stycken så att det räcker med en byte för att representera id-numret. Det är inte svårt att använda fler bytes för att representera större tal om så skulle vara nödvändigt.

Lösning 1: separata sockets för unicast och multicast

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

class SensorCommunication extends Thread {

    private static final int REPLY_SIZE = 256;
    private static final int CLIENT_UNICAST_PORT = 4098;
    private static final int SERVER_MULTICAST_PORT = 4099;
    private static final int SERVER_UNICAST_PORT = 4098;

    private int sensorId;
```

```
/* Create a thread that communicates with sensor id. */
public SensorCommunication(int id) {
    sensorId = id;
}

public void run() {
    byte[] request = new byte[2];
    byte[] reply = new byte[REPLY_SIZE];
    int serialNo = 0;

    try {
        /* Create unicast socket. */
        DatagramSocket socket = new DatagramSocket(CLIENT_UNICAST_PORT);

        /* Obtain address of sensor computer. */
        socket.setSoTimeout(1000);
        MulticastSocket ms = new MulticastSocket();
        ms.setTimeToLive(1);
        byte[] address = {(byte) 224,(byte) 0,(byte) 1,(byte) 20};
        InetAddress ia = InetAddress.getByAddress("experiment.mcast.net",address);
        request[0] = (byte) sensorId;
        request[1] = (byte) serialNo;
        DatagramPacket rq = new DatagramPacket(request,request.length,ia,
                                               SERVER_MULTICAST_PORT);
        DatagramPacket rp = new DatagramPacket(reply,reply.length);
        boolean addressFound = false;
        for(int retries = 0;retries<10 && !addressFound;retries++) {
            ms.send(rq);
            addressFound = true;
            try {
                socket.receive(rp);
            } catch(SocketTimeoutException e) {
                addressFound = false;
            }
        }
        ms.close();
        if (!addressFound) {
            System.err.println("Sensor computer not found. terminating.");
            System.exit(1);
        }
        InetAddress server = rp.getAddress();

        /* Perform normal operation. */
        socket.setSoTimeout(500);
        while(true) {
            request[0] = (byte) sensorId;
            serialNo = (serialNo+1)%256;
            request[1] = (byte) serialNo;
            rq = new DatagramPacket(request,request.length,
                                    server,SERVER_UNICAST_PORT);
            socket.send(rq);
            rp = new DatagramPacket(reply,reply.length);
            boolean receivedReply = true;
            do {
                try {
                    socket.receive(rp);
                } catch(SocketTimeoutException e) {
                    receivedReply = false;
                }
            } while(receivedReply && reply[0]!=serialNo);
            if (receivedReply) {
                String valueString = new String(rp.getData(),1,rp.getLength()-1);
```

```
                    Control.newValue(sensorId,Integer.valueOf(valueString));
                } else {
                    Control.noResponse(sensorId);
                }
                try {
                    sleep(1000);
                } catch(InterruptedException e) {
                    System.err.println("Sleep interrupted! Should not happen!");
                    System.exit(1);
                }
            }

        } catch(IOException e) {
            System.out.println("Exception:"+e);
            System.exit(1);
        }
    }
}

---------------------------------

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

public class SensorComputer {

    public static void main(String[] args) {
        if (args.length!=1) {
            System.out.println("Syntax: java SensorComputer <id>");
            System.exit(1);
        }
        int id = Integer.parseInt(args[1]);
        new MulticastServer(id).start();
        new UnicastServer().start();
    }
}


class MulticastServer extends Thread {

    private static final int REQUEST_SIZE = 256;
    private static final int CLIENT_UNICAST_PORT = 4098;
    private static final int SERVER_MULTICAST_PORT = 4099;
    private int sensorId;

    public MulticastServer(int id) {
        sensorId = id;
    }

    public void run() {
        try {
            MulticastSocket ms = new MulticastSocket(SERVER_MULTICAST_PORT);
            byte[] address = {(byte) 224,(byte) 0,(byte) 1,(byte) 20};
            InetAddress ia = InetAddress.getByAddress("experiment.mcast.net",address);
            ms.joinGroup(ia);
            DatagramSocket socket = new DatagramSocket();
            byte[] buf = new byte[REQUEST_SIZE];
            byte[] reply = {(byte) 0};
            while(true) {
                DatagramPacket dp = new DatagramPacket(buf,buf.length);
                ms.receive(dp);
                if (buf[0]==sensorId) {
```

```
                    dp = new DatagramPacket(reply,reply.length,dp.getAddress(),
                                            CLIENT_UNICAST_PORT);
                    socket.send(dp);
                }
            }
        } catch(IOException e) {
            System.out.println("Exception:"+e);
            System.exit(1);
        }
    }
}


class UnicastServer extends Thread {

    private static final int REPLY_SIZE = 256;
    private static final int SERVER_UNICAST_PORT = 4098;

    public void run() {
        try {
            DatagramSocket socket = new DatagramSocket(SERVER_UNICAST_PORT);
            byte[] buf = new byte[REPLY_SIZE];
            while(true) {
                DatagramPacket dp = new DatagramPacket(buf,buf.length);
                socket.receive(dp);
                buf[0] = buf[1]; // Return serial number
                String sensorValue = ""+Sensor.readValue();
                byte[] sensorBytes = sensorValue.getBytes();
                for(int i=0;i<sensorBytes.length;i++) {
                    buf[i+1] = sensorBytes[i];
                }
                dp = new DatagramPacket(buf,sensorBytes.length+1,
                                        dp.getAddress(),dp.getPort());
                socket.send(dp);
            }
        } catch(IOException e) {
            System.out.println("Exception:"+e);
            System.exit(1);
        }
    }
}
```

Lösning 2: samma socket för både unicast och multicast

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

class SensorCommunication extends Thread {

    private static final int REPLY_SIZE = 256;
    private static final int SERVER_PORT = 4099;

    private int sensorId;

    /* Create a thread that communicates with sensor id. */
    public SensorCommunication(int id) {
        sensorId = id;
    }

    public void run() {
        byte[] request = new byte[2];
```

```
byte[] reply = new byte[REPLY_SIZE];
int serialNo = 0;

try {
    /* Create socket. */
    MulticastSocket ms = new MulticastSocket(SERVER_PORT);
    ms.setTimeToLive(1);

    /* Obtain address of sensor computer. */
    ms.setSoTimeout(1000);
    byte[] address = {(byte) 224,(byte) 0,(byte) 1,(byte) 20};
    InetAddress ia = InetAddress.getByAddress("experiment.mcast.net",address);
    request[0] = (byte) sensorId;
    request[1] = (byte) serialNo;
    DatagramPacket rq = new DatagramPacket(request,request.length,ia,
                                           SERVER_PORT);
    DatagramPacket rp = new DatagramPacket(reply,reply.length);
    boolean addressFound = false;
    for(int retries = 0;retries<10 && !addressFound;retries++) {
        ms.send(rq);
        addressFound = true;
        try {
            ms.receive(rp);
        } catch(SocketTimeoutException e) {
            addressFound = false;
        }
    }
    if (!addressFound) {
        System.err.println("Sensor computer not found. terminating.");
        System.exit(1);
    }
    InetAddress server = rp.getAddress();

    /* Perform normal operation. */
    ms.setSoTimeout(500);
    while(true) {
        request[0] = (byte) sensorId;
        serialNo = (serialNo+1)%256;
        request[1] = (byte) serialNo;
        rq = new DatagramPacket(request,request.length,
                                server,SERVER_PORT);
        ms.send(rq);
        rp = new DatagramPacket(reply,reply.length);
        boolean receivedReply = true;
        do {
            try {
                ms.receive(rp);
            } catch(SocketTimeoutException e) {
                receivedReply = false;
            }
        } while(receivedReply && reply[0]!=serialNo);
        if (receivedReply) {
            String valueString = new String(rp.getData(),1,rp.getLength()-1);
            Control.newValue(sensorId,Integer.valueOf(valueString));
        } else {
            Control.noResponse(sensorId);
        }
        try {
            sleep(1000);
        } catch(InterruptedException e) {
            System.err.println("Sleep interrupted! Should not happen!");
            System.exit(1);
```

```
                }
            }
        } catch(IOException e) {
            System.out.println("Exception:"+e);
            System.exit(1);
        }
    }
}

---------------------------------

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

public class SensorComputer {

    private static final int MESSAGE_SIZE = 256;
    private static final int SERVER_PORT = 4099;

    public static void main(String[] args) {
        if (args.length!=1) {
            System.out.println("Syntax: java SensorComputer <id>");
            System.exit(1);
        }
        int sensorId = Integer.parseInt(args[1]);

        try {
            MulticastSocket ms = new MulticastSocket(SERVER_PORT);
            byte[] address = {(byte) 224,(byte) 0,(byte) 1,(byte) 20};
            InetAddress ia = InetAddress.getByAddress("experiment.mcast.net",address);
            ms.joinGroup(ia);

            byte[] buf = new byte[MESSAGE_SIZE];

            while(true) {
                DatagramPacket dp = new DatagramPacket(buf,buf.length);
                ms.receive(dp);
                if (buf[0]==sensorId) {
                    buf[0] = buf[1]; // Return serial number
                    String sensorValue = ""+Sensor.readValue();
                    byte[] sensorBytes = sensorValue.getBytes();
                    for(int i=0;i<sensorBytes.length;i++) {
                        buf[i+1] = sensorBytes[i];
                    }
                    dp = new DatagramPacket(buf,sensorBytes.length+1,
                                            dp.getAddress(),dp.getPort());
                    ms.send(dp);
                }
            }
        } catch(IOException e) {
            System.out.println("Exception:"+e);
            System.exit(1);
        }
    }
}
```

2. Första uppdraget som civilingenjör inom IT

a) Skjut in följande rader mellan rad 273 och 274 i bilagan:

```
} else if (name.endsWith(".mpg") || name.endsWith(".mpeg")) {
    return "video/mpeg";
} else if (name.endsWith(".qt") || name.endsWith(".mov")) {
    return "video/quicktime";
} else if (name.endsWith(".wrl")) {
    return "model/vrml";
```

b) Ersätt rad 191 med:

```
if (method.equals("GET") || method.equals("HEAD")) {
```

Ersätt raderna 216 och 217 med:

```
if (method.equals("GET")) {
    raw.write(theData);
    raw.flush();
}
```

c) I)

```
public void processQuery(String path, String query, Writer out)
                                                    throws IOException {
    path = "." + path;
    Process process = Runtime.getRuntime().exec(path + " " + query);
    out.write("HTTP/1.0 200 OK\r\n");
    InputStream in = process.getInputStream();
    int c;
    while ((c = in.read()) != -1) {
        out.write((char) c);
    }
    out.flush();
}
```

II) Förutom att man vill ha tillgång till metoden `processQuery()` ovan, föreslås följande ändringar:
Skjut in följande kod mellan rad 190 och rad 191:

```
URI requestURI = null;
byte[] messageBody = null;

if (method.equals("POST")) {
    // Let's extract the request
    try {
        requestURI = new URI(filename);
    } catch (Exception e) { }

    // Let's read the header
    String header = readHeader(in);

    // Let's read the Content-Length
    int contentLength = getContentLength(header);
    if (contentLength != 0) {
        messageBody = new byte[contentLength + 1];
        int c = 0;
        inx = 0;
        while ((c != -1) && (inx < contentLength)) {
            c = in.read();
            messageBody[inx++] = (byte) c;
```

```
                }
                messageBody[inx] = '\0';
            }
            processQuery(requestURI.getPath(), URLDecoder.decode(new String(messageBod
            //We are done with the query
            try {
                connection.close();
            } catch (Exception e) { }
            continue;
        }
```

Vi behöver dessutom följande hjälpmetoder. Sökningen i getContentLength kan varieras om man vill undvika att använda så kallade reguljära uttryck eller inte är bekant med begreppet.

```
public String readHeader(Reader in) throws IOException {
    StringBuffer headerLines = new StringBuffer();
    int inx, cur, prev;
    while (true) {
        inx = 0;
        prev = 0;
        while (true) {
            cur = in.read();
            inx++;
            headerLines.append((char) cur);
            if (prev == '\r' && cur == '\n') {
                break;
            }
            prev = cur;
        }
        if (inx == 2) {
            break; // Empty line
        }
    }
    String header = headerLines.toString();
    return header;
}

public int getContentLength(String header) {
    Pattern contentLength = Pattern.compile(".*Content-Length: *(\\d+).*", Pattern.CAS
    Matcher matcher = contentLength.matcher(header);
    if (matcher.matches()) {
        return new Integer(matcher.group(1));
    } else {
        return 0;
    }
}
```