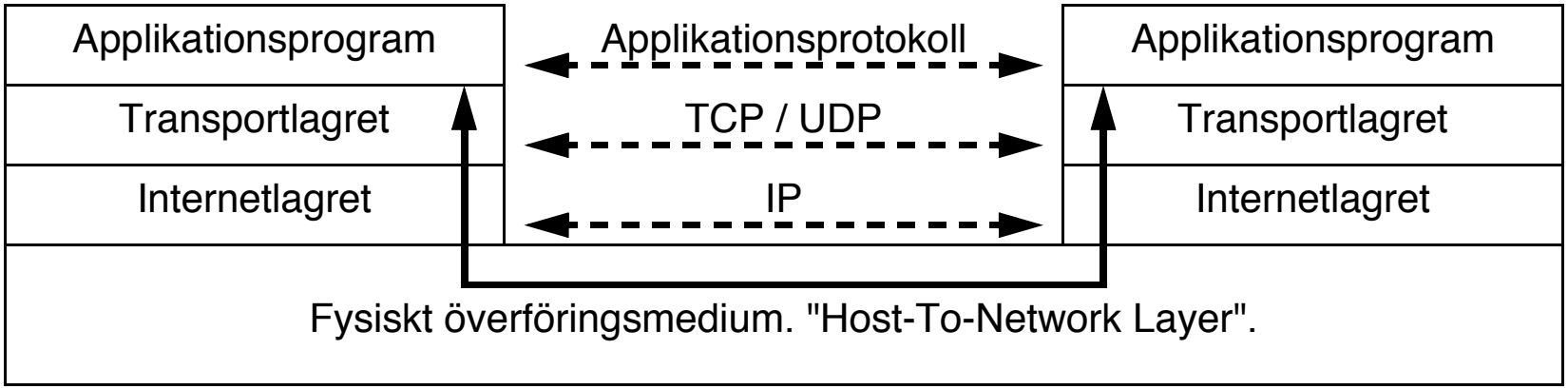
F5
Meddelandesändning med TCP

EDA095 Nätverksprogrammering

Roger Henriksson
Datavetenskap
Lunds universitet

Transmission Control Protocol – TCP

En del av transportlagret.

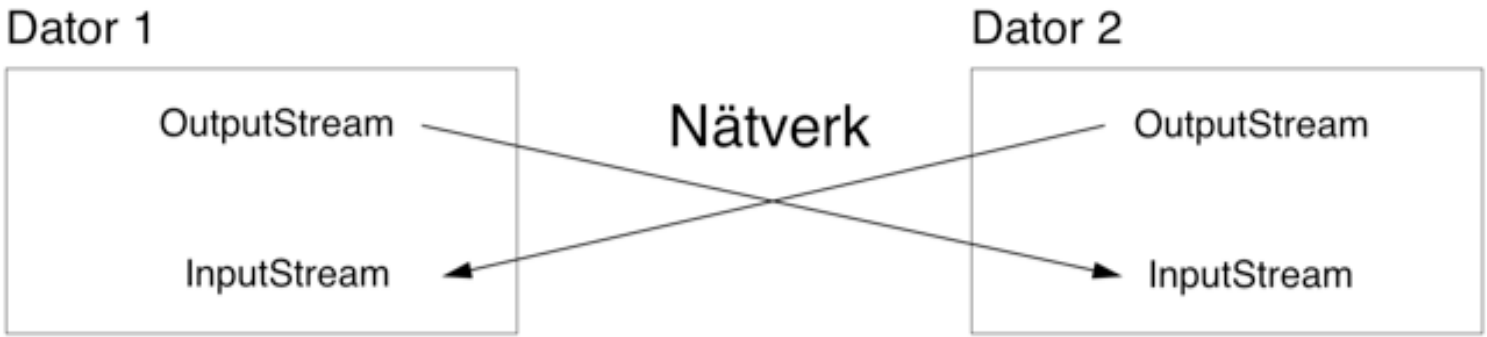


TCP

"Fast" uppkoppling mellan två portar på (vanligtvis) två olika datorer. Fast uppkoppling (connected protocol)!

- Automatisk omsändning / sortering av meddelanden.
- Ingen storleksbegränsning.

Ur applikationsprogrammets synvinkel fungerar en TCP-förbindelse som två strömmar (streams), en inkommande (InputStream) och en utgående (OutputStream):



Java och TCP

Socket

Motsvarar en upprättad förbindelse till vilken man kan skriva utgående bytes och läsa inkommande bytes.

ServerSocket

Används av en server för att vänta på uppkopplingar.

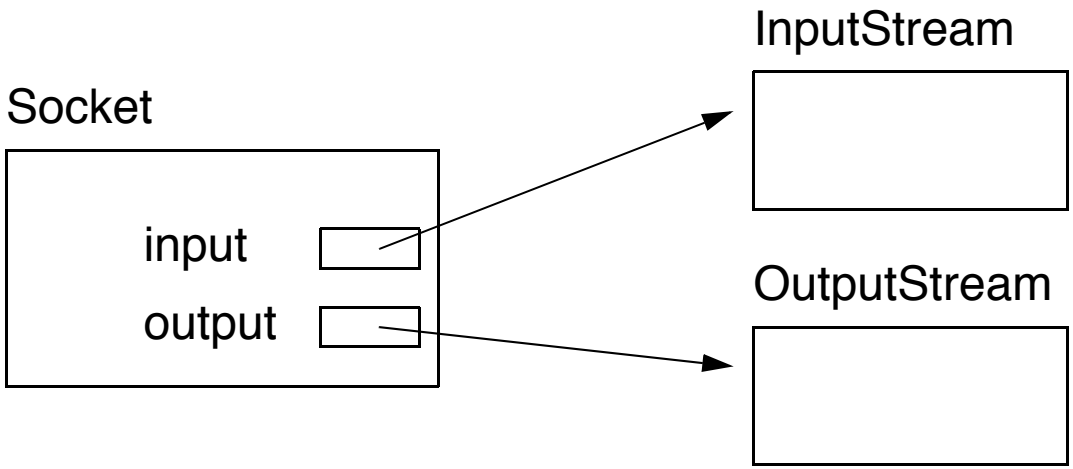
Stream I/O

InputStream och OutputStream från java.io.

Paketen java.net och java.io.

Socket – modell

Modell



Socket-objektet sköter övergripande egenskaper för uppkopplingen.

Stream-objekten har hand om in-/utmatning av bytes.

Stream-objekten skapas automatiskt.

Socket

Konstruktorer

```
public Socket(String host, int port)
                    throws UnknownHostException, IOException;
public Socket(InetAddress host,int port)
                    throws IOException;
```

Get-metoder

```
public InputStream getInputStream() throws IOException;
public OutputStream getOutputStream()throws IOException;
public InetAddress getInetAddress();
public int getPort();
public InetAddress getLocalAddress();
public int getLocalPort();
```

Socket, fortsättning

Inställningar

```
public void setTcpNoDelay(boolean on)
                                    throws SocketException;
public boolean getTcpNoDelay() throws
SocketException;
public void setSoTimeout(int milliseconds)
                                    throws SocketException;
public int getSoTimeout() throws SocketException;
...
```

Koppla ned

```
public void close() throws IOException;
```

Skapa en socket (klient)

```
Socket socket = null;
InputStream input = null;
OutputStream output = null;
try {
    socket = new Socket("cs.lth.se",80);
    input = socket.getInputStream();
    output = socket.getOutputStream();
} catch (UnknownHostException e) {
    System.out.println(e);
    System.exit(1);
} catch (IOException e) {
    System.out.println(e);
    System.exit(1);
}
```

DEMO - SimpleTelnet

Att läsa stora block - repetition

Att läsa en byte i taget med read() är mycket ineffektivt.
Lösning:

```
public int read(byte[] input,int offset,int length)
                                                throws IOException;
```

Varning! Ingen garanti att angivet antal bytes läses!
Exempel: Vi vill läsa exakt 100 byte.

```
byte[] buffer = new byte[100];
int read = 0;
int result = 0;
while (read<100 && result!=-1) {
   result = input.read(buffer,read,100-read);
   if (result!=-1)
      read = read+result;
}
```

TCP-server – skelett

```
while (true) {
    receive(client, command, parameters);
    switch (command) {
        case commandA:
            result = doCommandA(parameters);
            break;
        case commandB:
            result = doCommandB(parameters);
            break;
        case commandC:
            result = doCommandC(parameters);
            break;
        ...
        default: ...
    }
    send(client, result);
}
```

ServerSocket

Fungerar som en telefonist - tar emot uppkopplingar på serversidan och skapar ett motsvarande Socket-objekt.

Klienten och servern kommunicerar via denna Socket.

En servers livscykel

1. Skapa en ServerSocket.
2. Vänta på uppkoppling (accept()). Socket skapas.
3. Hämta strömmar med getInputStream() respektive getOutputStream().
4. Servern och klienten kommunicerar med varandra.
5. Förbindelsen kopplas ned.
6. Gå till steg 2 och vänta på ny uppkoppling.

ServerSocket, fortsättning

Konstruktorer

```
public ServerSocket(int port) throws IOException;
public ServerSocket(int port, int queueLength)
                                        throws IOException;
...
```

Vänta på en uppkoppling

```
public Socket accept() throws IOException;
...
```

Koppla ned servern

```
public void close() throws IOException;
```

ServerSocket, fortsättning

Get-metoder

```
public InetAddress getInetAddress();
public int getLocalPort();
public int getSoTimeout() throws IOException;
```

Set-metoder

```
public void setSoTimeout(int timeout)throws IOException;
```

DEMO - ToUpperServer

Skicka/ta emot tecken

Ibland vill man skicka tecken istället för bytes och vill slippa att själv göra omvandlingen i sitt program.

Reader/Writer - en parallell klasshierarki till InputStream/OutputStream som hanterar tecken enligt en given teckenkodning.

Vi gör om strömmar till en reader/writer genom att kapsla in den (precis som med FilterInput/OutputStream):

```
OutputStreamWriter out = new
OutputStreamWriter(socket.getOutputStream());
InputStreamReader in = new
                  InputStreamReader(socket.getInputStream());
```

Se kursboken och Javas klassdokumentation!

Skicka/ta emot strängar

Klasserna BufferedReader och PrintWriter representerar sträng- och textorienterade strömmar.
De är buffrade och hanterar rader av tecken effektivt via metoden `readLine()`:

```
InputStream is = ...
OutputStream os = ...
BufferedReader in = new
                BufferedReader(new InputStreamReader(is));
PrintWriter out = new PrintWriter(os,true);
```

Se kursboken och Javas klassdokumentation!

Hantera flera TCP-anslutningar

Vad händer om en andra klient kopplar upp sig medan servern är upptagen med att betjäna den första klienten?

Blockering eller förvägrad uppkoppling!

DEMO - ToUpperServer

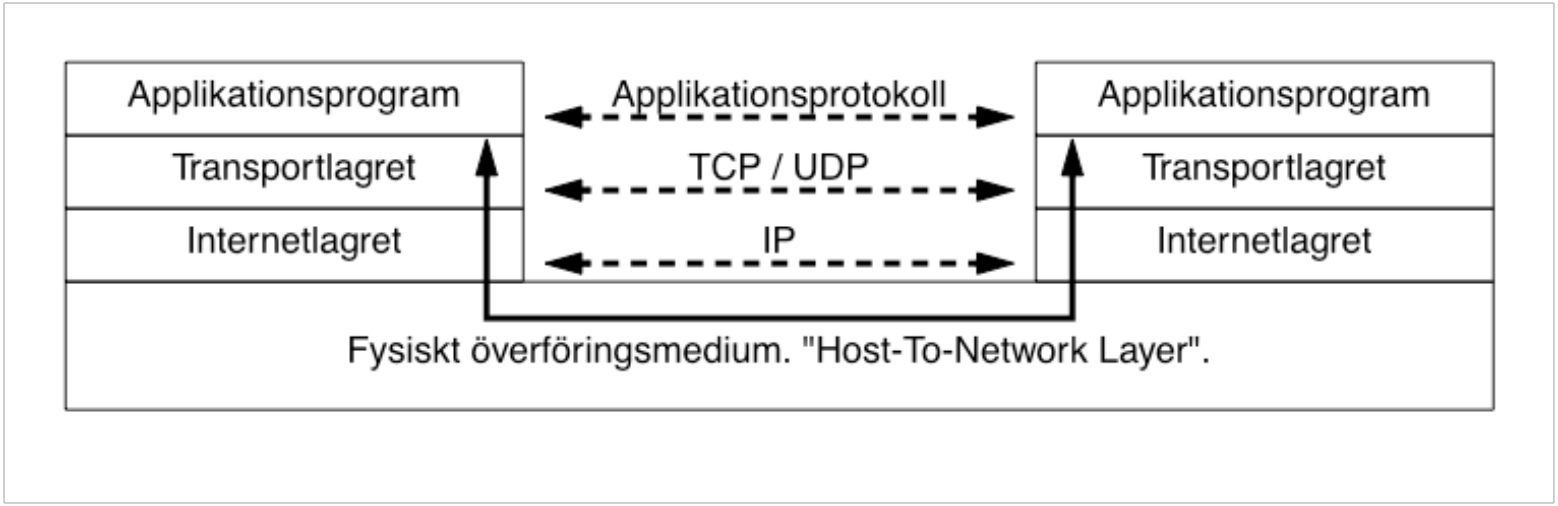
Visst hade det varit bra om ert serverprogram kunde göra flera saker samtidigt? Tex betjäna flera olika klienter?

Lösningar:

- Trådar...
- "Non-blocking I/O" - java.nio, java.nio.channels
  Introducerat i Java 1.4.

Design av applikationsprotokoll

Hittills stort fokus på gränssnittet mellan applikationsprogram och transportlagret.



Applikationsprotokoll och TCP

- Avgränsning av fält/meddelanden
- Kodning av datatyper som en sekvens av bytes
- Teckenkodning – radslut
- ”Big-endian/little-endian”

Protokolldesign – exempel

I kursen Realtidsprogrammering behöver man överföra en ström av JPEG-bilder tillsammans med tidsstämplar. För varje bild:

- Tidsstämpel (heltal, long)
- JPEG-bild (ett variabelt antal sekvensiella bytes)

Problem

- Hur representerar man en tidsstämpel? Binärt? ASCII? Ordning?
- Hur vet man när alla bytes i själva bilden sänts?
- När börjar nästa meddelande (bild)?

Exempel, fortsättning

Protokollförslag:

1. Tidsstämpel – en long i binär form, 8 bytes, big-endian
2. Antal bytes i den efterföljande JPEG-bilden – long, 8 bytes, big-endian
3. JPEG-bilden – den sekvens av bytes som utgör bilden

HTTP

Protokoll för kommunikation mellan webbläsare och webbserver.

- RFC1945 (HTTP 1.0) och RFC2616 (HTTP 1.1).
- Textbaserat
- Radbaserat, radslut: CR + LF
- HTTP 1.0:
  - Anslutning via TCP
  - Request
  - Response
  - Nedkoppling

HTTP, fortsättning

Request (avslutas med dubbla radslut, CR+LF+CR+LF):

```
GET /index.html HTTP/1.1
Accept: text/html, text/plain, image/gif, image/jpeg
User-Agent: Mozilla/4.0
Host: ygg.cs.lth.se
```

Response:

```
HTTP/1.1 200 OK
Date: Wed, 29 Mar 2006 08:20:32 GMT
Server: Apache/2.0.40 (Red Hat Linux)
Last-Modified: Mon, 21 Mar 2006 18:17:07 GMT
Content-type: text/html; charset=ISO-8859-1
Content-length: 107
<html>
<head>
...
```

Design: enkel HTTP-server

```
while (true) {
    socket = serversocket.accept();
    parseAndResponse(socket);
    socket.close();
}
```

Nackdelar

- Kan endast hantera en uppkoppling åt gången.
- Utnyttjar tillgänglig bandbredd dåligt.

Design: enkel multitrådad server

Main

```
while (true) {
    socket = serversocket.accept();
    new RequestHandler(socket).start();
}
```

RequestHandler

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

Fördelar

- enkel design
- klarar flera uppkopplingar parallellt

Nackdel

- Skalar inte upp. Starta nya trådar dyrt.

Design: trådpool och jobblista

En kö med ankomna anslutningar och en pool av servertrådar.

Main

```
while (true) {
    socket = serversocket.accept();
    jobs.insert(socket);
}
```

RequestHandler

```
while (true) {
    socket = jobs.getJob();
    parseAndResponse(socket);
    socket.close();
}
```

Variant av föregående lösning.

Design: Executors

Implementera en trådpool mha Executors i Java.

Main

```
ExecutorService pool = Executors.newFixedThreadPool(10);
while (true) {
    socket = serversocket.accept();
    pool.submit(new RequestHandler(socket));
}
```

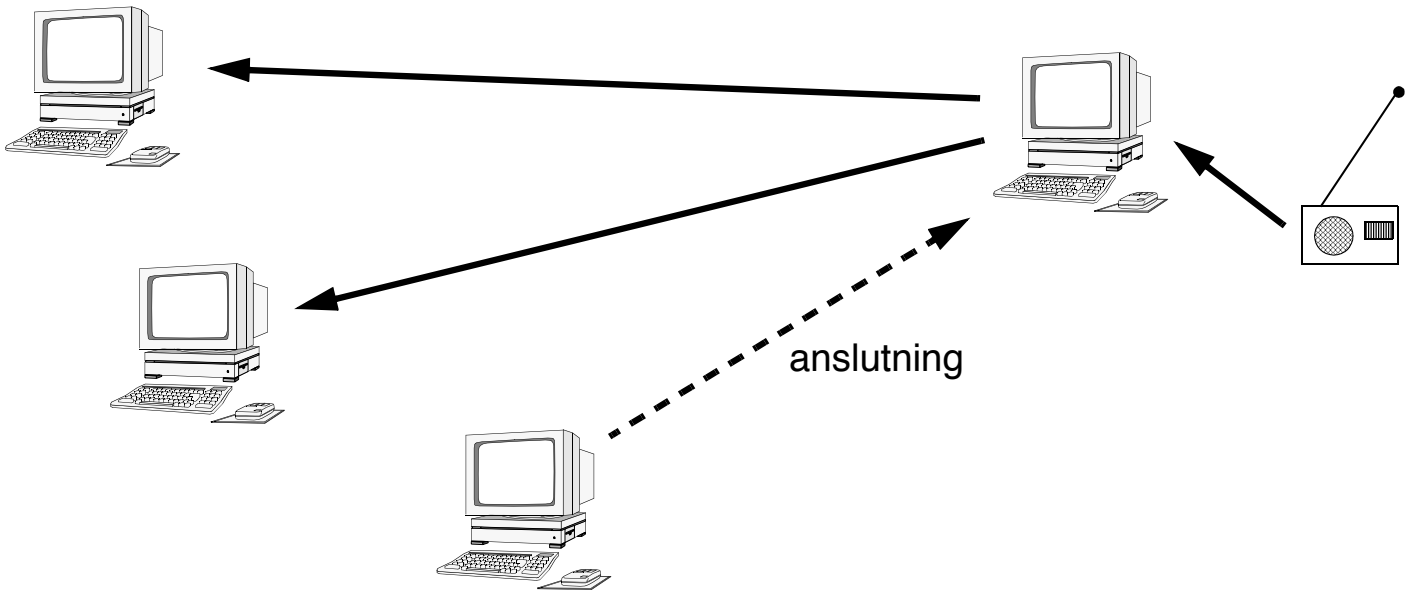
RequestHandler

```
while (true) {
    parseAndResponse(socket);
    socket.close();
}
```

Notera: Ingen `pool.shutdown()`!

Icke-blockerande I/O i C

Internetradio anno 1992 på Datavetenskap.



- raclient.c
- raserver.c