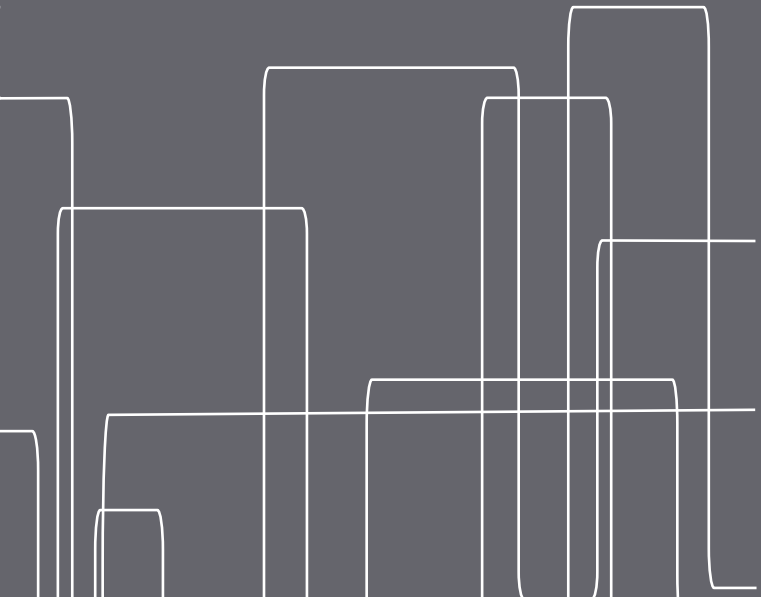




Objektorienterad Programkonstruktion

Föreläsning 10
7 dec 2015





Nätverksprotokoll - OSI

- Open Systems Interconnection model (ISO standard) definierar 7 lager för nätverkskommunikation:
 - 1 - **Physical Layer**: kablar, kontakter, bärvågor, bitdefinitioner
 - 2 - **Data Link Layer**: fysisk adressering och kommunikation mellan två noder (t.ex Ethernet)
 - 3 - **Network Layer**: logisk adressering (t.ex IPv6)
 - 4 - **Transport Layer**: uppkopplingar, tillförlitlighet (t.ex TCP/UDP)
 - 5 - **Session Layer**: autentisering, sessioner (NetBIOS, PPTP)
 - 6 - **Presentation layer**: teckenkodning, kryptering (TLS, SSL)
 - 7 - **Application Layer**: Syntax för själva dataöverföringen (t.ex HTTP, POP3, FTP, SSH)



Nätverksprotokoll – Internet Protocol Suite

- Kallas ibland även TCP/IP efter de mest använda protokollen

(Physical)	1 - Physical Layer : kablar, kontakter, bärvågor, bitdefinitioner
Link	2 - Data Link Layer : fysisk adressering och kommunikation mellan två noder (t.ex Ethernet)
Internet	3 - Network Layer : logisk adressering (t.ex IPv6)
Transport	4 - Transport Layer : uppkopplingar, tillförlitlighet (t.ex TCP/UDP)
Application	5 - Session Layer : autentisering, sessioner (NetBIOS, PPTP)
	6 - Presentation layer : teckenkodning, kryptering (TLS, SSL)
	7 - Application Layer : Syntax för själva dataöverföringen (t.ex HTTP, POP3, FTP, SSH)



HTTP

- Application Layer
- Beskriver att antal olika kommandon som en klient kan skicka till en server, och hur servern ska svara



Kommunikationsordning för webbläsare

- Skickar en begäran till en server
- Får ett svar från servern, som kan innehålla HTML
- Tolkar HTML, genererar en sidvisning
- Tar in signaler från användaren, utifrån dessa kan den skicka en ny förfrågan till en server, osv...



HTTP - Exempel

```
GET /helloworld.html HTTP/1.1
```

```
Host: www.hello.net
```

```
-----
```

```
HTTP/1.1 200 OK
```

```
Date: Mon, 05 Dec 2011 09:15:00 GMT
```

```
Server: Apache/1.2.3.4 (Unix) (Debian/Linux)
```

```
Last-Modified: Mon, 05 Dec 2011 06:07:08 GMT
```

```
Etag: "31337-123-1337b03f"
```

```
Accept-Ranges: bytes
```

```
Content-Length: 1337
```

```
Connection: close
```

```
Content-Type: text/html; charset=UTF-8
```

```
<html> ....
```



HTML

- HyperText Markup Language
- Standard som definieras av W3C
- Är en applikation av SGML (Standard Generalized Markup Language)
- Det finns också XHTML, som är en XML-applikation, och lättare att tolka
- Innehåller bland annat taggar för textformatering, rubriker, tabeller, bilder, mm
- Innehåller taggar för hyperlänkar, så att man kan koppla ihop ett dokument med ett annat (eller en annan del av sig själv)

```
<a href="http://www.google.com">Leta h&auml;r!</a>
```



HTML

- Minimalt giltigt HTML-dokument

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <p>Hello world!</p>
  </body>
</html>
```



Nätverksadresser

- **URL** - Uniform Resource locator - adresserar resurser på internet

`scheme://domain:port/path?
query_string#fragment_id`

- **IP-adress** - ett nummer för varje apparat ansluten till ett nätverk. En apparat anslutet till ett nätverk kan ha samma nummer som en annan apparat i ett annat nätverk, t.ex

`192.168.1.1`

- **Portnummer**: Ett sätt att adressera olika processer i en maskin. T.ex kan en webserver ta emot anslutningar på port 80, medan port 25 tar emot SMTP-anslutningar. Portar anges som 16 bitars heltal. De första 1024 portarna är reserverade för well known ports och kräver root-rättigheter för att använda på UNIX-system.



URL:er i Java

- Klassen URL har en konstruktor som tar en sträng, t.ex
`"http://www.cas.kth.se:80/~ccs/index.html"`
- Klassen kan plocka ut alla intressanta delar ur URL:en automatiskt:
 - `URL.getProtocol()` -> `http`
 - `URL.getAuthority()` -> `www.cas.kth.se:80`
 - `URL.getHost()` -> `www.cas.kth.se`
 - `URL.getPort()` -> `80`
 - `URL.getPath()` -> `~ccs/index.html`
- Kan anges till en JEditorPane för att hämta HTML-kod



TCP

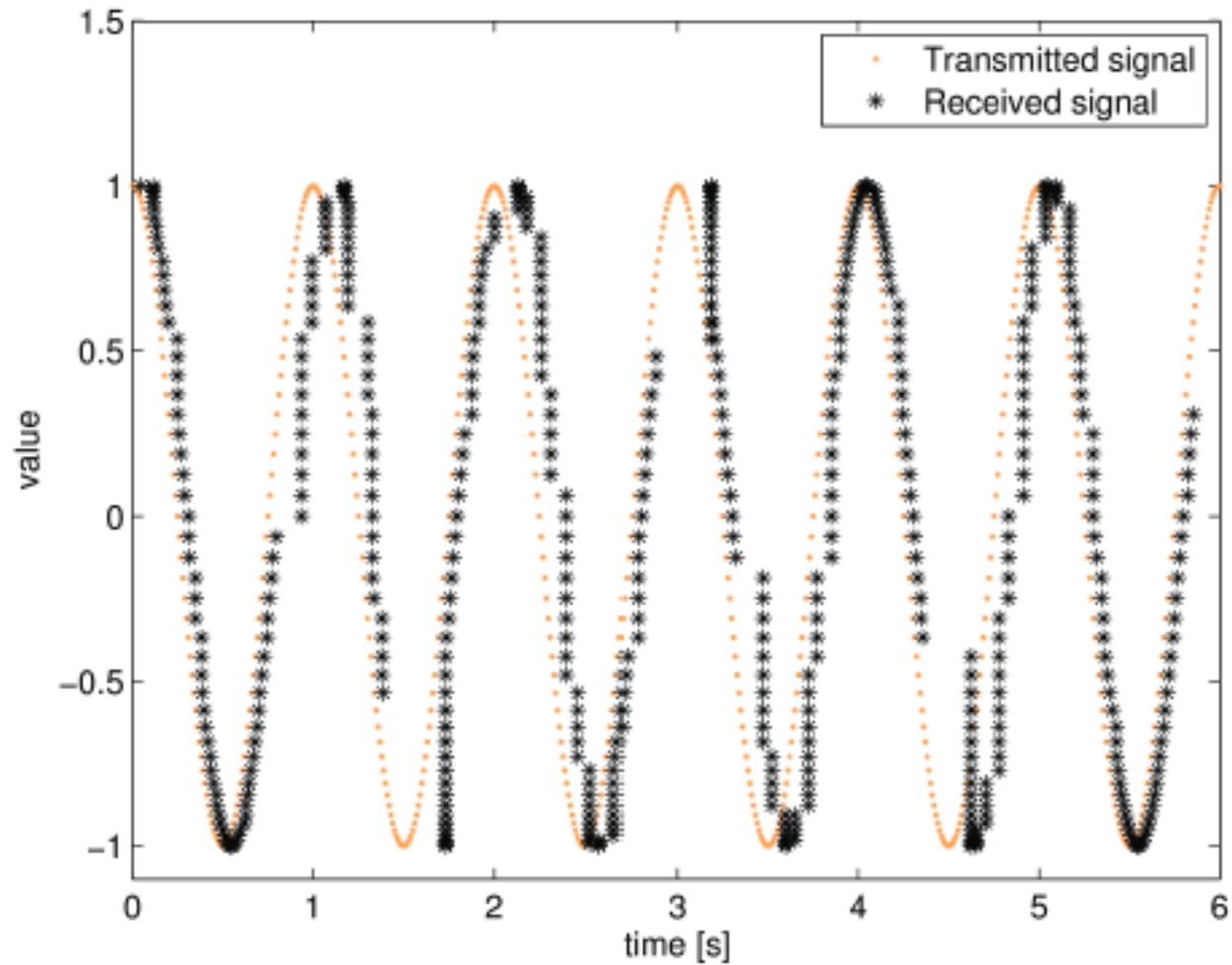
- **T**ransmission **C**ontrol **P**rotocol
- Kontrollerar att alla datapaketer kommer fram, skickar nya paket om inget svarsmeddelande kommer tillbaka
- Garanterar ordningen, dvs, alla paket kommer fram i samma ordning som de skickades
- Plockar bort dubletter
- Lämplig för t.ex filöverföring
- Kan ibland få fördröjningar på flera sekunder om borttappade paket behöver skickas om flera gånger

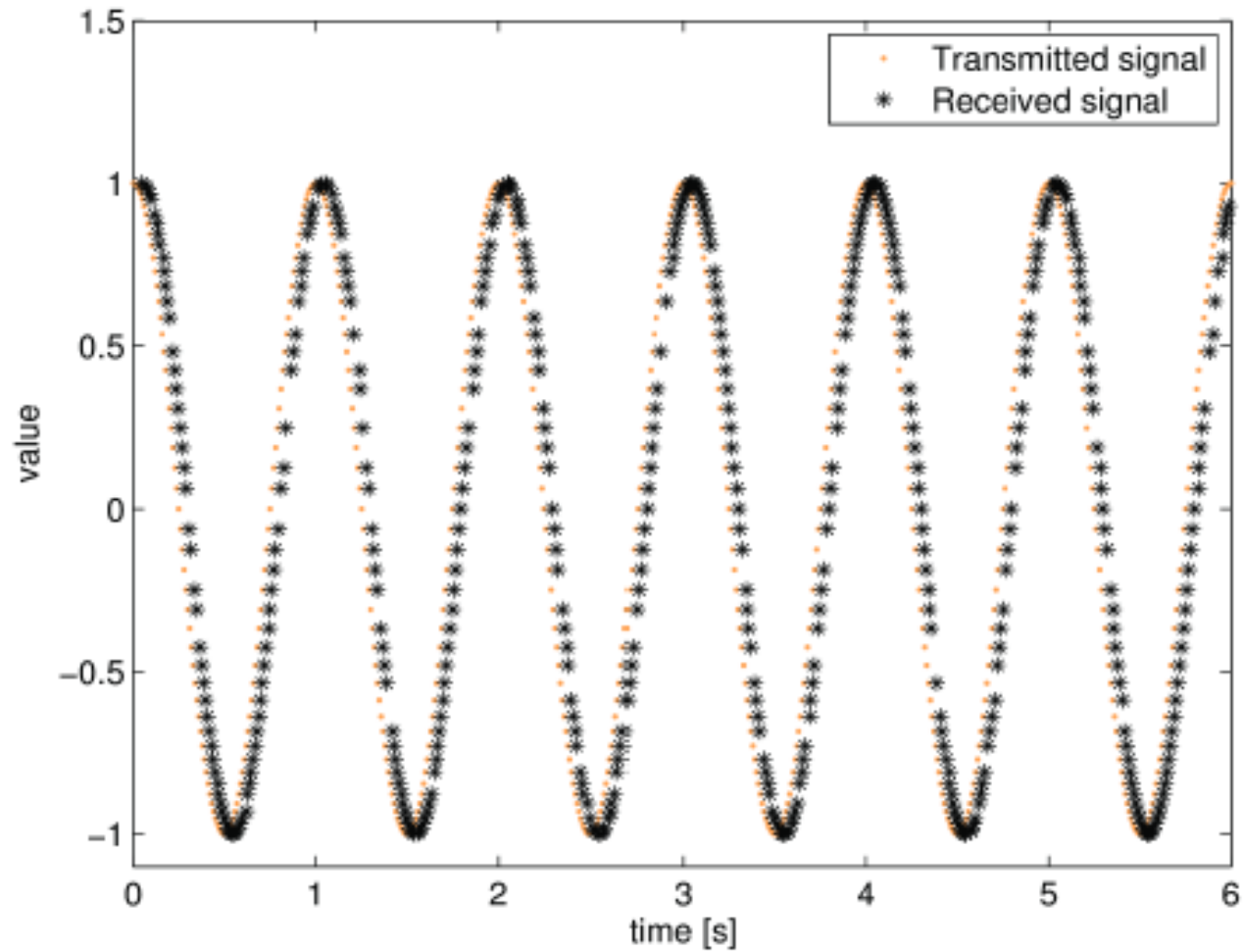


UDP

- **U**ser **D**atagram **P**rotocol
- Enkelt protokoll utan felkontroller
- Garanterar inte att alla paket kommer fram
- Meddelar inte om paket kommit bort
- Garanterar inte att alla paket kommer i rätt ordning
- Plockar inte bort dubletter
- Antar att applikationen gör alla felkollar som behövs
- Lämplig för t.ex realtidsapplikationer









Socket

- En slutpunkt för en internetanslutning
- Kan användas av processer för att skriva till/läsa från nätverket
- Initieras av användarprocesser, administreras av operativsystemet
- Identifieras unikt genom
 - Local Socket Address (lokal IP och port)
 - Remote Socket Address (TCP: motpartens IP och port)
 - Protocol (t.ex TCP, UDP)
- När kontakt har etablerats kan man skriva och läsa mellan två processer (på två olika datorer)



Socket i Java

- Två olika klasser: `Socket` och `ServerSocket`
- `Socket` används för att ansluta till en annan dator
- `ServerSocket` används för att lyssna efter och ta emot anslutningar från en annan dator
- När en anslutning har etablerats kan man skriva och läsa till dem med t.ex. en `PrintWriter` och en `BufferedReader` på ungefär samma sätt som man läser/skriver till en fil
- Man kan läsa/skriva otolkade (råa) bytes, eller hålla koll på teckenkodningar och skicka text. Det förra är bättre för överföring av binära data, tex filer, det senare för överföring av textinformation, t.ex i chat-program



Socket

```
try{
    mySocket = new Socket("myhost", 1025);
    out = new PrintWriter(
        mySocket.getOutputStream(), true);
    in = new BufferedReader(new InputStreamReader(
        mySocket.getInputStream()));
} catch (UnknownHostException | IOException e) {
    System.out.println("Error: " + e);
}
```

Om uppkopplingen fungerade kan man nu skriva:

```
out.println("Text!");
String myString = in.readLine();
out.close();
in.close();
mySocket.close();
```



ServerSocket

```
try {
    serverSocket = new ServerSocket(1025);
} catch (IOException e) {
    System.out.println("listen failed on port: 1025");
}
try {
    clientSocket = serverSocket.accept();
} catch (IOException e) {
    System.out.println("Accept failed: 1025");
}

PrintWriter out = new PrintWriter(
    clientSocket.getOutputStream(), true);
BufferedReader in = new BufferedReader(
    new InputStreamReader(
        clientSocket.getInputStream()));
```