F1
Nätverk och meddelande-sändning med UDP

EDA095 Nätverksprogrammering

Roger Henriksson
Datavetenskap
Lunds universitet

Nätverksprogrammering

- Tekniker för att skriva program som kommunicerar med varandra över ett nätverk.
- Särskild tonvikt på Internet/IP-baserade nätverk, men generellt tillämpbara tekniker.
- Praktisk implementering i Java. Samma principer som i andra programmeringsspråk.

Kursöversikt

- Nätverksbegrepp
- Enkel meddelandesändning: UDP, Multicast
- Uppkopplade förbindelser: TCP
- Fjärrexekvering av kod: RPC/RMI
- Trådar
- Webbtekniker:
  - HTTP
  - XML
  - CGI/Servlets/JSP/PHP
- Strömmande media

Klienter och servrar

Vanlig arkitektur för applikationer som erbjuder tjänster via nätverk. Engelska: “client/server architecture”.

Klient

Ett program som kopplar upp sig (ofta via nätverk) till ett annat program för att begära någon tjänst.

Exempel: Webbläsare, e-postprogram.

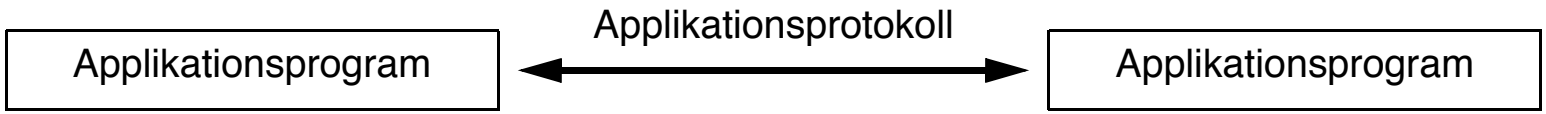
Server

Ett program som erbjuder tjänster åt andra program.

Exempel: Webbserver, mailserver.

Protokoll

För att två program ska kunna förstå varandras meddelanden behövs ett *protokoll*, dvs ett gemensamt språk.



Exempel

HTTP (HyperText Transfer Protocol) - för kommunikation mellan webbläsare och webbserver.

Transaktioner

Kommunikationen mellan en klient och en server kan ofta delas upp i en följd av *transaktioner*.

En transaktion kan omfatta sändning av ett antal relaterade meddelanden i olika riktningar.

Syftet med en transaktion är att få en primitiv – typiskt *atomär* – operation utförd av servern.

Exempel

Hämta en webbsida från en webbserver (med HTTP):

- Klienten skickar en begäran till webbservern.
- Servern skickar den begärda webbsidan.

Session

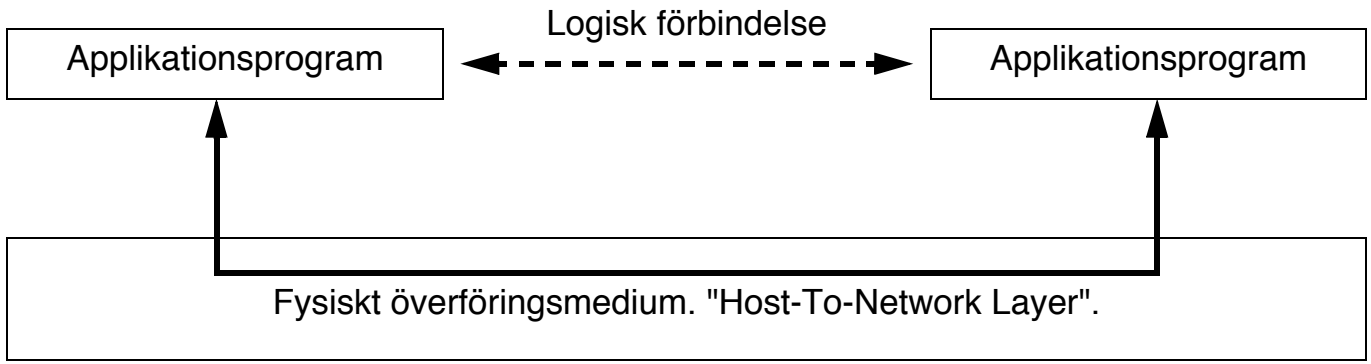
Hela "uppkopplingen", ofta flera transaktioner.

Fysiskt överföringsmedium

Applikationsprotokollet är ett protokoll för kommunikation mellan två applikationsprogram.

För överföringen behövs ett fysiskt överföringsmedium.

Exempel: Ethernet, RadioLAN.



"Host-To-Network"-lagret omvandlar mellan analoga och digitala signaler.

Datagram

Ett datagram är ett sammanhållet meddelande som sänds som en enhet över ett nätverk.

Jämför: Telegram, SMS.

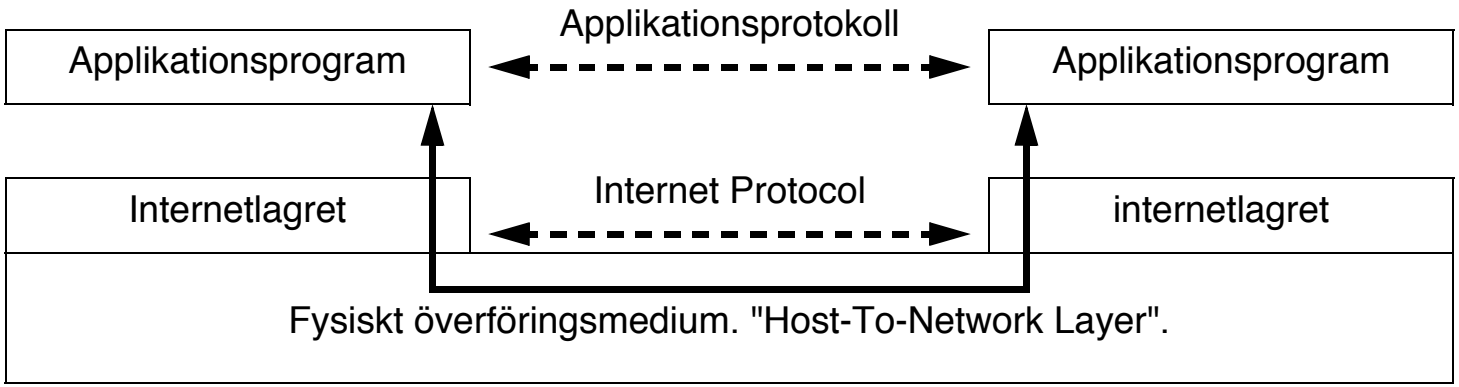
“Host-To-Network”-lagret

- Endast begränsade möjligheter att skicka datagram.
- Ingen felkontroll/omsändning.

Opraktiskt!

Internet Layer

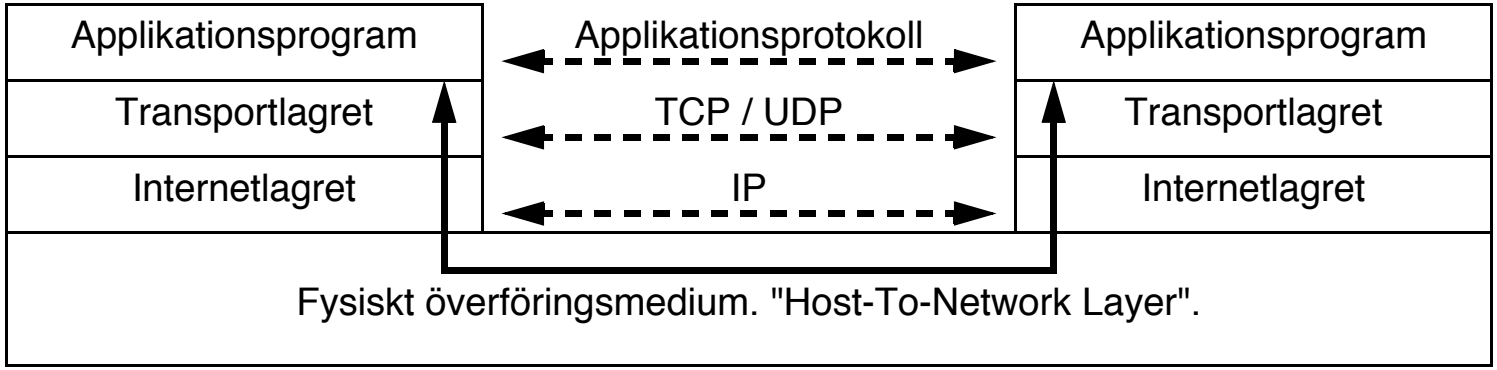
- IP - Internet Protocol
- Viss felkontroll
- Oberoende av fysiskt överföringsmedium



Bättre, men fortfarande komplicerat!

Transport Layer

Bygger vidare på “Internet Layer” / IP.



TCP/UDP

- **User Datagram Protocol (UDP)**
  - Datagram
  - Upp till 65507 byte stora datagram (IPv4)
  - Garanterat korrekta meddelanden.
  - Ej garanterad leverans eller inbördes ordning.
- **Transmission Control Protocol (TCP)**
  - Fast (logisk) uppkoppling över nätverket.
  - Dataström – data behöver inte delas upp i paket.
  - Automatisk felkontroll/omsändning. Garanterad leverans och inbördes ordning.

Adressering

För att ett program ska kunna upprätta en nätverksförbindelse behöver det ett sätt att ange vilket annat program det vill koppla upp sig mot.

En kombination av IP-nummer och portnummer!

IP-nummer

Anger vilken dator programmet kör på. Jämför: gatuadress.

Portnummer

En dator kan ha flera olika program igång som erbjuder nätverkstjänster. Varje program har ett eller flera unika logiska portnummer. Jämför: lägenhetsnummer inom en fastighet.

IP-nummer

- 32 bitar - 4 byte (IPv4)
- 128 bitar - 16 byte (IPv6)

Exempel: 130.235.16.34 (IPv4)

2001:fe0c:0000:0000:0000:0000:00db:1dc0 (IPv6)

2001:fe0c::db:1dc0 (IPv6)

Symboliska (domän-)namn istället:

www.cs.lth.se = 130.235.16.34

Översättning mellan domännamn och IP-adresser görs av en "Domain Name Server" – DNS.

Övergång till IPv6

- IP-numren i IPv4 slut!
- IPv4 / IPv6 kommer att användas parallellt ett tag
- Dubbla IP-adresser under övergångsfasen
- Moderna datorer stöder IPv6
- Idag dåligt stöd bland internetleverantörer

- Liten eller ingen betydelse för oss som applikationsprogrammerare – undvik bara att förutsätta IP-adresser på formen "X.X.X.X"!

InetAddress

Javaklass för att representera internetadresser (JNP kap 6).

Finns i paketet java.net.

Skapa ett InetAddress-objekt

```
public static InetAddress getByName(String hostname)
                                    throws UnknownHostException;
```

Även getAllByName och getLocalHost.

Hämta information

```
public String getHostName();
public byte[] getAddress();
public String getHostAddress();
```

DEMO - addressuppslagning

Portnummer

- Identifierar vilken tjänst på en dator man vill ansluta till.
- Rent logisk abstraktion – ingen fysikalisk motsvarighet.

Ett program som accepterar nätverksuppkopplingar, en server, väljer ett ledigt portnummer och väntar på att andra program ska ansluta till just detta portnummer.

Port 1-1023 är reserverade för standardtjänster.

Exempel:

Tjänst	Port	Protokoll
echo	7	TCP/UDP
FTP	21	TCP
Telnet	23	TCP
HTTP	80	TCP

Port 1024-65535 är tillgängliga för vanliga applikationer.

Internetstandarder

Standardprotokoll för kommunikation på Internet fastställs av IETF – Internet Engineering Task Force – en ganska informell organisation.

Beskrivs i form av RFC – Request For Comments.

Exempel:

- RFC768 – UDP
- RFC791 mfl – IP
- RFC1945 – HTTP version 1.0

Java och UDP

I paketen java.net och java.io finns stöd för att sända och ta emot meddelanden (paket) mha UDP.

DatagramPacket

Representerar ett meddelande som kan skickas med UDP.

DatagramSocket

Fungerar som en sändare/mottagare för meddelanden.
"socket" = uttag / "hål-i-väggen". Jämför: telefonjack/fax.

DatagramPacket

Konstruktorer

DatagramPacket-objekt som ska användas för att *ta emot* meddelanden:

```
public DatagramPacket(byte[] buffer, int length);
```

DatagramPacket-objekt som ska användas för att sända meddelanden:

```
public DatagramPacket(byte[] buffer, int length,
                      InetAddress destination, int port);
```

Vi måste tillhandahålla en vektor av typen byte[] som är tillräckligt stor för att rymma det meddelande som ska sändas/tas emot.

Ett meddelande består alltså av ett antal *bytes*.

DatagramPacket, fortsättning

Get-metoder

```
public InetAddress getAddress();
public int getPort();
public byte[] getData();
public int getLength();
```

Set-metoder

```
public void setData(byte[] data);
public void setAddress(InetAddress remote);
public void setPort(int port);
public void setLength(int length);
```

DatagramPacket, exempel

Skapa ett DatagramPacket-objekt avsett att skickas till port 2000 på login.cs.lth.se och som innehåller texten “Network Programming”:

```
// Create an InetAddress object
InetAddress dest = null;
try {
   dest = InetAddress.getByName("login.cs.lth.se");
} catch(UnknownHostException e) { System.exit(1); }

// Create message buffer
String s = "Network Programming";
byte[] data = s.getBytes(); // Default character encoding

// Create the DatagramPacket object
DatagramPacket packet =
    new DatagramPacket(data,data.length,dest,2000);
```

DatagramSocket

Klassen DatagramSocket ansluts till en port och kan:

- Sända UDP-paket från denna port till en annan port på en annan dator.
- Ta emot meddelanden som sänds till denna port på denna dator.

Jämför med en fax (DatagramSocket) som ansluts till ett telefonjack (porten).

Datagramsocket, fortsättning

Konstruktorer

Skapa en DatagramSocket och anslut den till angiven port:

```
public DatagramSocket(int port) throws SocketException;
```

Skapa en DatagramSocket på en för tillfället ledig port:

```
public DatagramSocket() throws SocketException;
```

Konstruktorerna genererar ett SocketException om det inte gick att skapa socketen, t.ex. därför att angiven port var upptagen.

DatagramSocket, fortsättning

Skicka ett datagram

```
public void send(DatagramPacket dp) throws IOException;
```

Glöm inte att skapa ett DatagramPacket med tillhörande byte-vektor först!

Glöm inte heller att fylla i byte-vektorn med ditt meddelande!

Ta emot ett datagram

```
public void receive(DatagramPacket dp) throws IOException;
```

Anrop av receive blockerar tills ett meddelande anländer.

Glöm inte att skapa ett tomt DatagramPacket med en tillhörande byte-vektor som är tillräckligt stor för att rymma meddelandet!

Vid både mottagning och sändning genereras ett IOException om det uppstod ett fel av något slag, tex om man försöker sända ett för stort UDP-paket.

DatagramSocket, fortsättning

Frigöra portar

När man inte har behov av en port längre skall man frigöra den så att den kan återanvändas till andra ändamål. Detta görs genom att man anropar "close" på motsvarande DatagramSocket-objekt:

```
public void close();
```

Extra inställningar

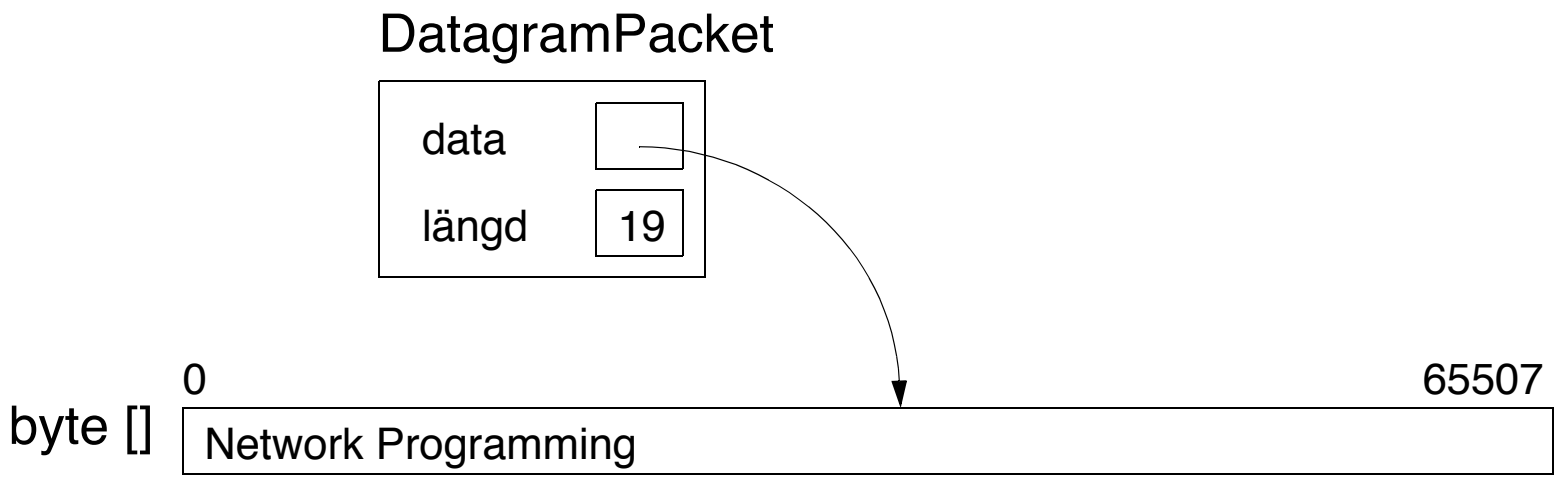
Anrop av receive blockerar ända tills ett meddelande anländer. Vill man att receive ska vänta på ett meddelande högst en viss tid kan man sätta en timeout för förbindelsen. Har inget meddelande mottagits inom angiven tid (i millisekunder) genereras ett InterruptedIOException.

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

DatagramSocket, fortsättning

DEMO - Sändare/mottagare

Längd för DatagramPacket



Omvandla mottaget meddelande till en sträng:

```
String s = new String(dp.getData(),0,dp.getLength());
```

Längden i ett DatagramPacket styr hur stort meddelande som kan tas emot. Kom ihåg att återställa om du vill återanvända ett DatagramPacket:

```
dp.setLength(dp.getData().length);
```

UDP-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);
}
```

Multicast

En variant av UDP.

Unicast

Ett meddelande sänds från en avsändare till EN mottagare.

Flera mottagare - flera kopior av meddelandet sänds.

Multicast

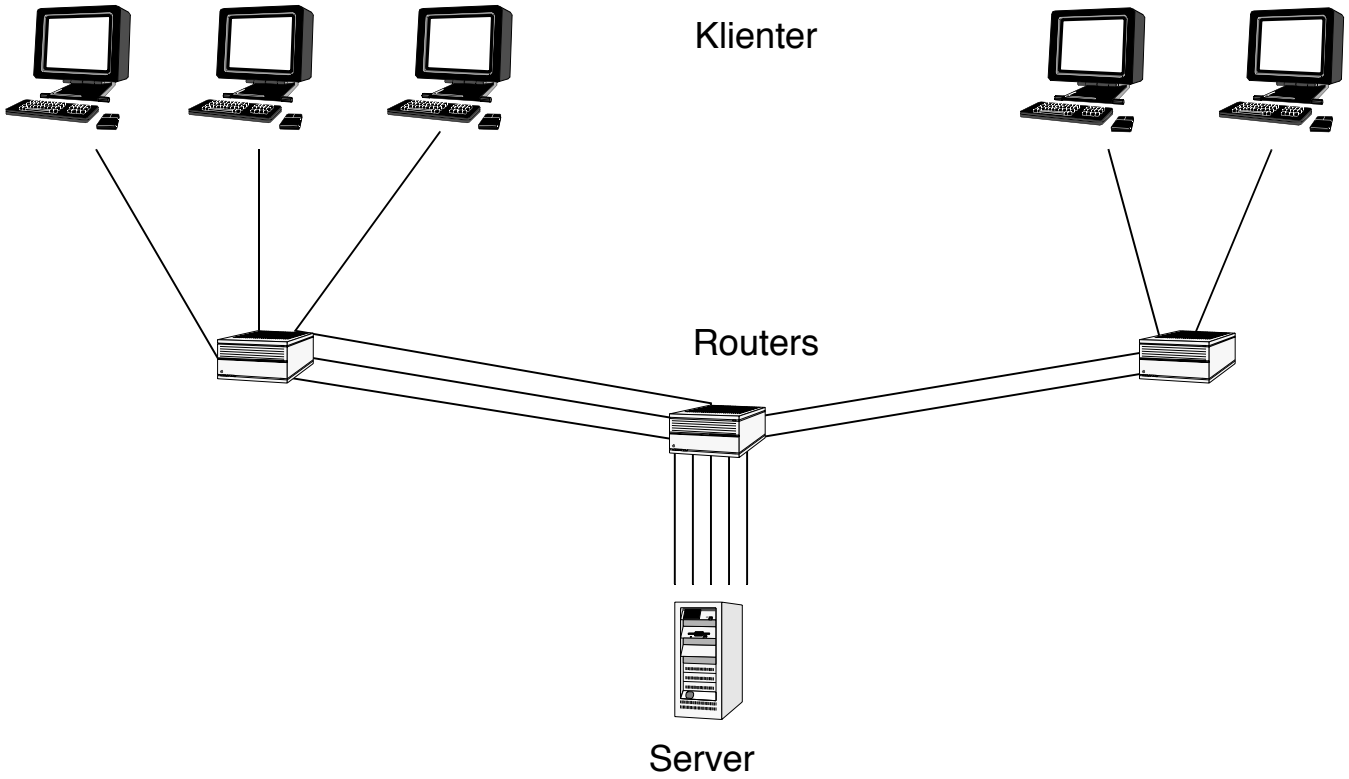
Ett meddelande sänds från en avsändare till FLERA mottagare – alla som är intresserade av att mottaga det.

Endast EN kopia av meddelandet så långt som möjligt.

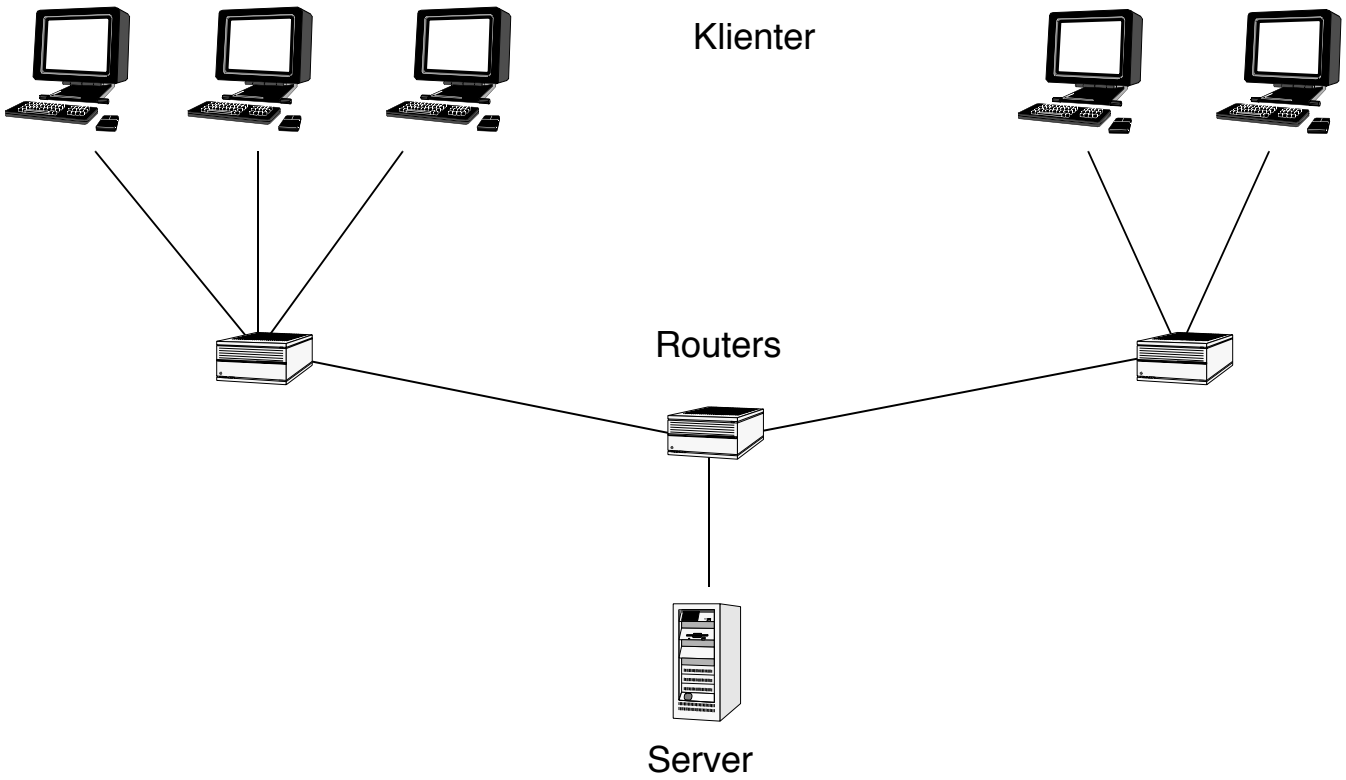
Kräver stöd av routrar.

Exempel: Live-utsändning av videodata.

Utan multicast



Med multicast



Multicastgrupper

En multicastgrupp omfattar alla datorer som är intresserade av att mottaga en viss typ av meddelanden.

Exempel: En videoutsändning av en live-konsert.

Multicastadresser

Varje multicastgrupp motsvaras av ett speciellt IP-nummer i serien 224.0.0.0 - 239.255.255.255 (IPv4).

Man kan säga att alla datorer i en multicastgrupp "delar på" detta IP-nummer.

Datagram som sänds till IP-numret går ut till alla datorer i multicastgruppen – avsändaren behöver inte veta vilka dessa är.

Att välja multicastadress

Permanenta multicastadresser

IANA - Internet Assigned Numbers Authority - delar ut fasta multicastadresser. Börjar med 224.0, 224.1, 224.2 eller 239.

Domännamn	IP-adress	Syfte
all-systems.mcast.net	224.0.0.1	Alla datorer på det lokala subnätet.
experiment.mcast.net	224.0.1.20	Experiment som inte går utanför det lokala subnätet.
ntp.mcast.net	224.0.1.1	Network Time Protocol
ietf-1-video.mcast.net	224.0.1.12	Video från IETF-möten. Kanal 1.

Tillfälliga multicastadresser

Vem som helst kan välja vilken adress som helst som inte är reserverad av IANA.

Time To Live – TTL

Till för att undvika överdrivna trafikvolymer och begränsa spridningen av multicastpaket.

Varje paket förses med ett "bäst-före-datum", TTL, i form av en räknare som räknas ned varje gång paketet passerar en router.

När paketets räknare blir noll dör paketet.

Java och multicast

Meddelanden är av typen DatagramPacket - som för UDP.

MulticastSocket

Ersätter DatagramSocket, men är likartad.
Subklass till DatagramSocket.

En MulticastSocket kan:

- Ansluta sig till en multicastgrupp.
- Skicka meddelanden till andra datorer i gruppen.
- Mottaga meddelanden från andra datorer i gruppen.
- Lämna en multicastgrupp.

MulticastSocket

Konstruktorer

Som för DatagramSocket:

```
public MulticastSocket() throws SocketException;
public MulticastSocket(int port) throws SocketException;
```

Ansluta till en multicastgrupp

Behövs bara för att ta emot meddelanden.

```
public void joinGroup(InetAddress address)
                                    throws IOException;
```

Lämna en multicastgrupp

```
public void leaveGroup(InetAddress address)
                                    throws IOException;
```

MulticastSocket, fortsättning

Ange Time-To-Live

```
public void setTimeToLive(int ttl) throws IOException;
```

Sända paket

```
public void send(DatagramPacket packet) throws IOException;
```

Ta emot paket

```
public void receive(DatagramPacket dp) throws IOException;
```

Frigöra portar

```
public void close();
```

MulticastSocket, fortsättning

DEMO – Sändare/mottagare