Internetværk og webprogrammering (IWP)

Indhold

[Lektion 1: Course introduction and introduction to web-programming 2](#_Toc40816756)

[Lektion 2: Client-side scripting: Interactive HTML 2](#_Toc40816757)

[Lektion 3: Client-side scripting: Client-server interaction 2](#_Toc40816758)

[Lektion 4: Server side programming with Node.JS: Sessions and cookies 2](#_Toc40816759)

[Lektion 5: Web-services and Web-APIs 2](#_Toc40816760)

[Lektion 6: Introduction to networks and the Internet protocol stack 2](#_Toc40816761)

[Struktur af internettet 2](#_Toc40816762)

[Packet switching 3](#_Toc40816763)

[Circuit Switching 4](#_Toc40816764)

[Packet switching vs circuit switching: 5](#_Toc40816765)

[Forsinkelse 5](#_Toc40816766)

[Trafik intensitet 5](#_Toc40816767)

[Throughput (flaske halse) 6](#_Toc40816768)

[Internet protokol stakken 6](#_Toc40816769)

[Header-payload modellen (Encapsulation) 7](#_Toc40816770)

[Lektion 7: The application layer protocols 7](#_Toc40816771)

[Domain name system (DNS) 11](#_Toc40816772)

[Peer to peer (P2P) 13](#_Toc40816773)

[Lektion 8: The transport layer protocols and reliable communication 15](#_Toc40816774)

[Lektion 9: TCP and Using the transport layer in programs: Sockets programming 15](#_Toc40816775)

[Lektion 10: The network- and link layers 15](#_Toc40816776)

[Lektion 11: Network security 15](#_Toc40816777)

[Lektion 12: Web-security 15](#_Toc40816778)

## Lektion 1: [Course introduction and introduction to web-programming](https://www.moodle.aau.dk/course/view.php?id=33287" \l "section-1)

## Lektion 2: [Client-side scripting: Interactive HTML](https://www.moodle.aau.dk/course/view.php?id=33287#section-2)

## Lektion 3: [Client-side scripting: Client-server interaction](https://www.moodle.aau.dk/course/view.php?id=33287#section-3)

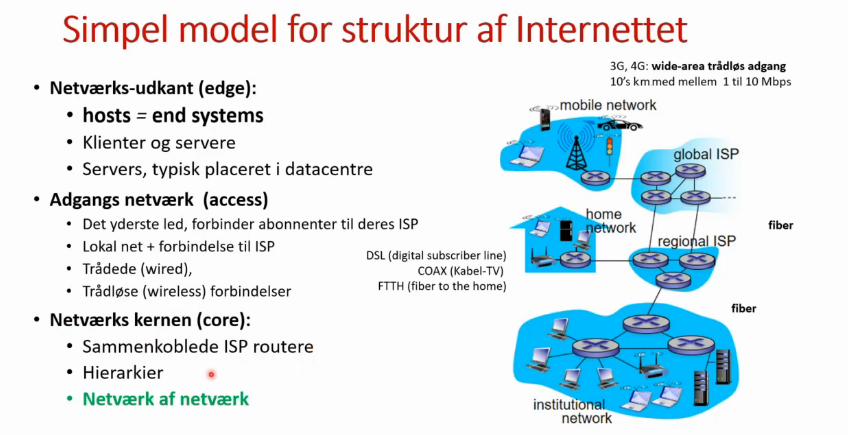
## Lektion 4: [Server side programming with Node.JS: Sessions and cookies](https://www.moodle.aau.dk/course/view.php?id=33287#section-4)

## Lektion 5: [Web-services and Web-APIs](https://www.moodle.aau.dk/course/view.php?id=33287#section-5)

## Lektion 6: [Introduction to networks and the Internet protocol stack](https://www.moodle.aau.dk/course/view.php?id=33287#section-6)

### Struktur af internettet

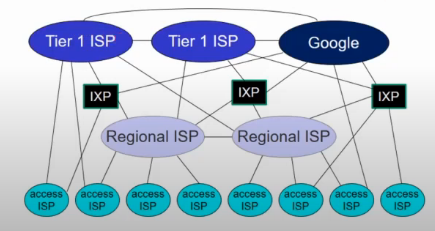
*Hosts* = end systems = PC, servers, laptops, telefoner, cyber-physical systems osv.  
*Links* = faste forbindelser (kobber, fiber) eller trådløse forbindelser (wifi).  
*Packet Switches* = en enhed der videresender data-pakker i nettet.  
*Protokol* = regler der styrer afsendelse og modtagelse af beskeder (IP, UDP, TCP osv).  
*RFC* = Internet standarder.



Keywords:  
Netværks-udkant (edge), Adgangs Netværk (Access), Netværks Kernen (Core)

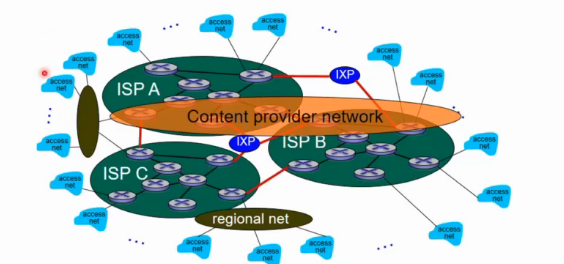
*Adgangs netværk* består (typisk) af lokale netværk brugt af størrer institutioner som typisk er forbundet med *Ethernet* (2 lag) switch.

*Netværk af netværk* består af et hirarki af ISP’er. På de laveste lag er der ”*access ISP*’er” som udbyder internetforbindelse til brugere. Disse Access ISP’er kan være forbundet med en *regional ISP* eller en *national ISP*. Access ISP’er, Regionale og nationale er forbundet til ”*Tier 1 ISP’er*” som er ”de store spillere” og er globale (f.eks Google). Alle ISP’er kan kommunikere direkte med hinanden **eller** med IXP (Internet exchange points) som er en fysisk bygning hvor man udveksler netværks-trafik(f.eks Danish Internet Exchange). Dette betyder der ikke er ét hiraki som er arbitrært. Et eksempel kan ses her:



Keywords:  
ISP, IXP, Reginal ISP, Access ISP

*Interforbundne netværk* består af forbundne *server-farme* som er tættere på end-hosts. Et eksempel kunne være en streaming tjeneste som har flere server farme tættere på brugeren. F.eks Google/Youtube. Figur:

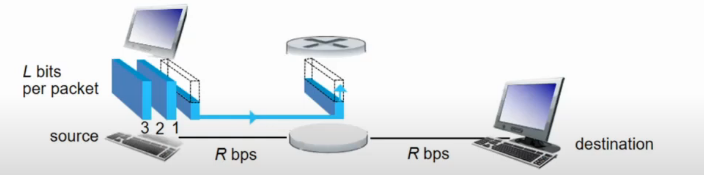


Keywords:  
Content Provider Network, Reginal Net, Access net

Formålet er at fordele trafik, så der ikke kommer en flaskehals gennem en global server.

### Packet switching

Packet switching består af at hosts deler data op I mindre dele (packets) som bliver sent til den næste router på stien (f.eks fra en host, til en router, til en anden router, til modtager host). Hver packet sendes med den fulde transmissionsrate af linket. Routeren skal modtage hele pakken og gemmes I dens hukkomelse før den bliver vidresendt (store and forward). Eksempel:



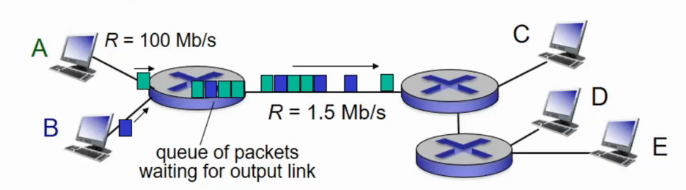
Keywords:  
Source, Destination

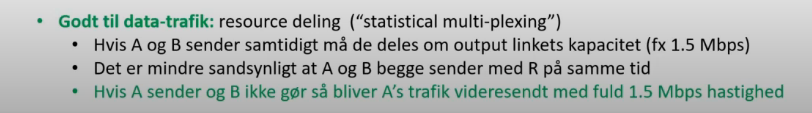
En pakke oplever forsinkelser når den er sendt igennem internettet. En pakke består af **L** *BITS* (8 bit er 1 byte) og den bliver sendt med en rate på **R** (mbps, gbps…osv).

Forsinkelser:

* En pakke der er L-Bits lang, sendt med raten R, tager sekunder at sende
* Ex. Forsinkelse:
* Ex. 2 hop til destination: Samlet forsinkelse:

*Pakke-kø* og *pakke-tab* kan ske I et netværk. En *pakke-kø* kan opstå ved at ankomst raten overstiger transmissionsraten over en kort periode, her bliver pakken gemt I en buffer med first-in-first-out princippet (en kø struktur). *Pakke-tab* sker hvis bufferen løber tør for plads, så routeren bliver nød til at droppe pakken. Forstoppelse (*congestion*) giver mulighed for pakke-tab (eksempel næste side).

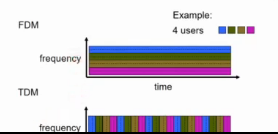




* Godt til data-trafik som optræder I stød.

### Circuit Switching

Circuit switching er en alternativ løsning hvor der oprettes en forbindelse mellem hosts og der reserveres en *fast* transmissionsrate igennem hele netværket. Måden man deler linkets kapacitet op, er ved FDM eller TDM. Eksempel:



Keywords:  
FDM, TDM

Packet switching vs circuit switching:  
Eksempel:   
1 Mb/s link, hver burger anvender 100 kbps, men er kun ”aktiv” 10% af tiden.

|  |  |
| --- | --- |
| Circuit- Switching: | Packet switching: |
| Tillader I alt: 10 brugere | P (en given burger= 0.1 |
| Dårlige resource udbyttelse | Ud af fx 35 (uafængige brugere, bliver sandsynligheden for at mere end 10 er aktivt samtidigt bliver ca. 0004 |
| Godt til kritisk data | God resource udnyttelse ved ”bursty” trafik (kommer i ”stød”) |
| Godt tilt ids-følsomt data | Simplere, ingen opsætning af kredsløbet |
|  | Forstoppelse (congestion) er muligt, giver pakketab   * (Behov for mere avancerede protokoller til pålidelig data overførsel og forhindring af forstoppelse) |

### Forsinkelse

Kilder til forsinkelse af routeren består af behandlingstid, kø-tid, transmissionstid og udbrednings delay (propagation delay). Man bestemmer forsinkelsen med følgende formel:



Keyword: dnodal

dproc = data-behandlingstid, checksum fejl, bestemme output link. Dproc er typisk < 1 millisekund.

dqueue = Ventetiden i køen. Tallet afhænger af forstoppelses-niveauet (congestion level).

dtrans = L/R – Længden af pakken (i BITS) divideret med transmissionsraten (bits per sekund).

dprop = d/s – Distancen af det fysiske link divideret med udbredselshastigheden s = 2 \* 10^8 m/s.

(traceroute.org kan bruges til at måle rigtige forsinkelser på internettet)

### Trafik intensitet

Trafik intensitet er defineret som forholdet imellem indkommende og udkommende trafik rate.

Trafik intensitet:

R – Transmissions rate (bps)  
L – Længden af pakken i bits  
a – Gennemsnitlig pakke ankomst (pakker per sekund).

Hvis trafik intensiteten overstiger 1 vil der akkumuleres pakker og der vil komme pakke-tab.  
Hvis trafik intensiteten nærmer sig 0 er der ”klar bane” og alt er godt.  
Hvis trafik intensiteten nærmer sig 1 vil hver pakke have lang ventetid.

### Throughput (flaske halse)

Throughput er den opnåelige transmissionsrate i en forbindelse mellem 2 end-hosts. Throughput bestemmes af linket med mindst rate:



Rc = Transmissions raten af clienten (eller 1 end-host)

Rs = Transmissionsraten af serveren (eller en anden end-host)

R/n – R = Transmissionsraten og n er antallet af forbindelser. Dette fordeler transmissions raten fair.

Typisk kommer der en flaskehals ved clienten eller serveren.

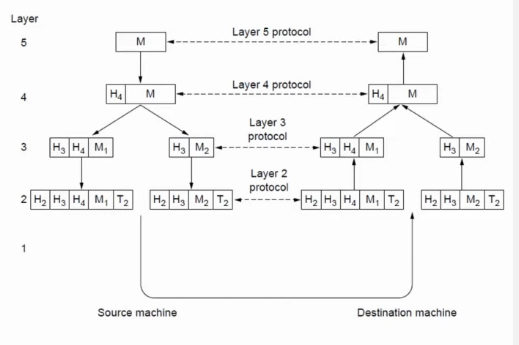
### Internet protokol stakken

|  |
| --- |
| Application |
| Transport |
| Network |
| Link |
| Physical |

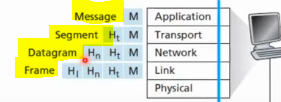
En 5 lags model, som er uddraget baseret på observationer af et konkret netværk.

* **Applikations-lag:** Protokoller som understøtter afvikling af netværks-applikationer (mail, browser, fil-transport, …)
* **Transport-lag:** Overfører data fra et kørende program (process) på end-system A til modpart på end system B (TCP, UDP)
* **Netværks-lag:** Routing og videre sendelse af pakker fra source til destination (IP)
* **Link-lag:** Data overførsel mellem direkte naboer i netværket, dvs. koblet sammen med et enkelt link. (Ethernet, IEEE-802.11 (WiFi), PPP, SONET)
* **Fysiske-lag:** Bits ”på ledningen”

### Header-payload modellen (Encapsulation)



Keywords: Layer, Protocol, Source machine, Destination machine



Keywords: Message, Segment, Datagram, Frame

## Lektion 7: [The application layer protocols](https://www.moodle.aau.dk/course/view.php?id=33287#section-7)

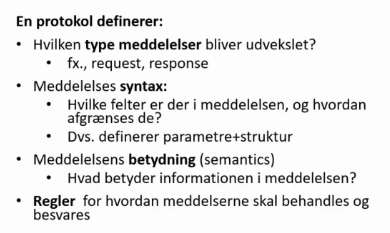
Programmer som:

* Afvikles på (flere forskellige) **end systems**
* Komunikerer over netværket
* Anvender applikations-niveau protokoller

Eks. Discord, Github, File transfer protocol(FTP), dropbox osv.

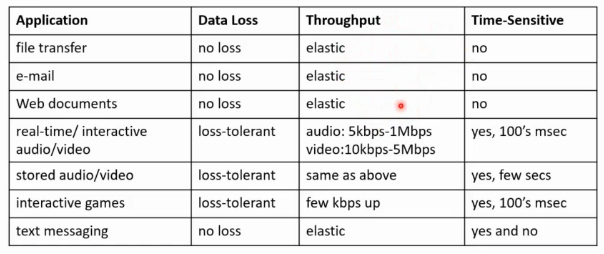
En *applikationslagsprotokol* understøtter applikationer (*HTTP, SMTP, DNS* osv...)

En protokol definerer:

* Hvilken **type meddelelser** bliver udvekslet?
  + Eks. request, response
* Meddelelses **syntax:**

Komponenter som snakker sammen kaldes *processer* (et kørende program på en end-host). Der findes roller til processer som f.eks *client/server modellen*. En process kan også have tildelt flere roller. En process kan både modtage og sende data hvilket styres gennem *sockets.* (lidt ligesom en dør til transportlaget). Processer navngives ved *IP-adressen* og et *portnummer*. Servers ”lytter” på en port den har bestemt. **Nogle portnumre er ”velkendte” eller ”reserveret” porte, f.eks HTTP som bruger port 80, HTTPS bruger 443 og DNS bruger 53. Disse velkendte porte kan ikke bruges i webapplikationer.**

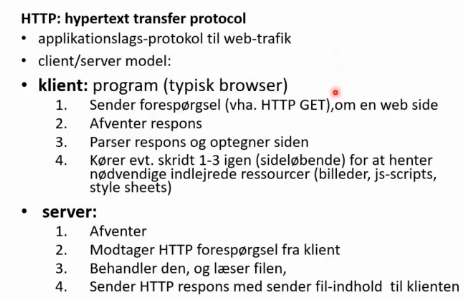
Nogle applikationer bruger kommunikationskrav:



Der er også sikkerhedskrav som: Kryptering, data integritet (der er ikke ændret på dataen) pg authenication (modparten er den som den giver sig ud for at være).

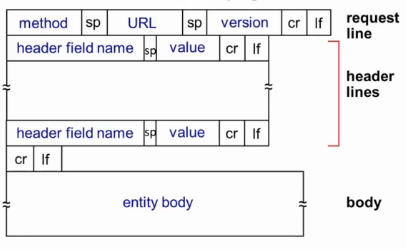
**HTTP**

Eksempel på et simpelt HTTP scenarie:



det behøves ikke være en klient, bare en eller andet end-host. Serveren kan også sende filer f.eks. et database look-up.

En *HTTP-forespørsel* består af en header(som sendes som tekst)



(Dette er et request, et repons er samme format)

Method = GET, POST, PUT osv (metode navnet)

URL = URL som metoden anvendes på

Version = Protokol versionen

**HTTP reponstid**

Her bruger vi *Round Trip Time (RTT)*, den tid det tager at sende en pakke og får et svar tilbage. HTTP repons tid kræver 1 RTT for at opsætte TCP og 1 til fil transmissionstiden.

*HTTP Responstid = 2\*RTT + File transmission time*

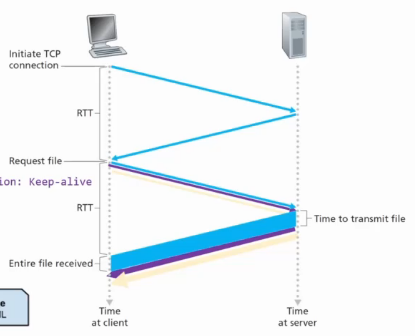
Hvis man skal indlæse en hjemmeside som har et HTML dokument, Stylesheet, scriptfil og et jpg, ville dette tage *4 \* HTTP responstid = 4 \* (2\* RTT) = >8 \* RTT.* **Der kræver et HTTP respons for hver objekt der skal hentes.** Advanceret hjemmesider kan sende parallel forbindelser (f.eks 1 til at hente HTML og en til at hente linked objekter (billeder, scripts osv).

Dette er meget hurtigere for klienten, men der er krævende for serveren.

**Persistent HTTP**

Her sparer man en RTT per HTTP respons, fordi serveren holder forbindelsen åben (altså der skal ikke oprettes en ny TCP forbindelse). Dette sker med Connection: Keep-alive. HTTP/1.1 bruger ”pipelining” hvor forespørgelserne kan blive sendt i rap, uden at vente på svar (samlebåndspricip).

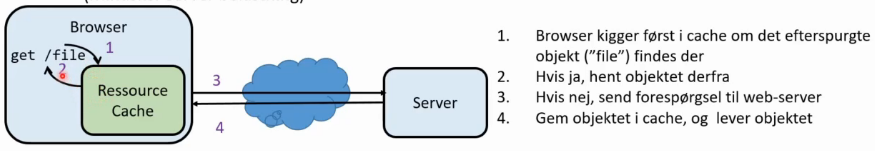
Eksemplet fra før ville være *HTTP repons pipelining = 2 RTT + File transmission + server overhead*



Dette løser problemet. Det er hurtigere for klienten og mindre krævene for serveren fordi den kun skal opretholde en TCP forbindelse per klient. HTTP/2 gør dette hurtigere ved at sende indhold til klienten uden forespørgsel.

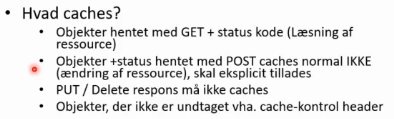
**HTTP Caching**

Her catcher man en kopi af forespurgte objekter (ligesom i en CPU med L1-L3 caches). Web-caching gør svartiden hurtigere og mindske server-belastning.



*HTTP cache control* styrer hvor længe kopien er gyldig og hvorvidt det kan caches. F.eks bliver styling ikke ændret meget, hvilket gør det godt at cache, men noget som dynamiske database opslag ændres ”tit” og er mindre godt at cache. (bliver styret af HTTP med ”*max-age=x*” hvor x er antal sekunder hvor klienten må bruge en cache, hvis det cached objekt er for gammelt skal man opdatere cachen). HTTP beskriver også hvornår objektet er blevet ændret, hvis objektet er blevet opdateret, så skal cachen også opdateres (*last-modified: dato & if-modified-since*).

Cache control kan bestemme om objekter er *no cache* (indholdet skal valideres af serveren), *no-store*(objektet må ikke caches, der skal ske en ny forespørgsel hver gang) og *private/public* cache (privat må kun gemmes i klientens cache)

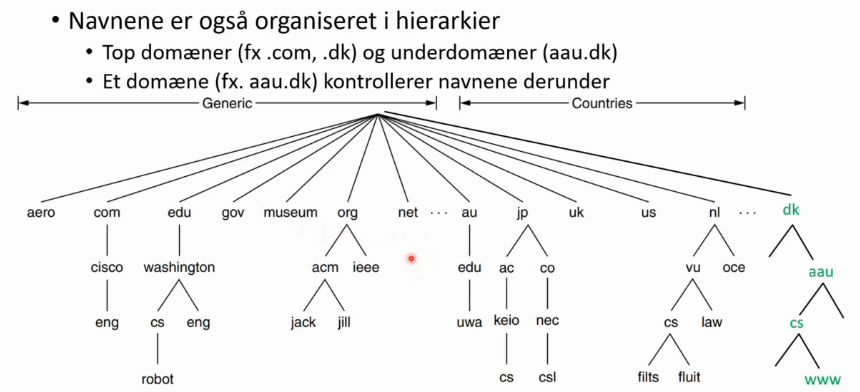


*HTTP cache control* benytter også betinget forespørgsler med *ETAGS*. Dette bruges til at validere objektet i cachen (altså er det cached objekt den samme som det objekt serveren har). Dette implementeres som en *hash-streng.* Man kan også bruge en *HTTP Proxy* som bruges som en fælles cache mellem alle clienter (placeret hos en ISP). HTTP proxies har også andre brug som: Load balancing af en server farm, indholds filter, anonymizer, compression/cryptering. **Dette virker ikke med HTTPS pga. signaturen ikke normalt gemmes.**

### Domain name system (DNS)

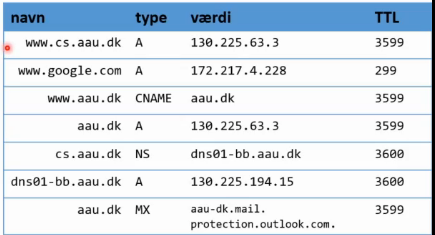
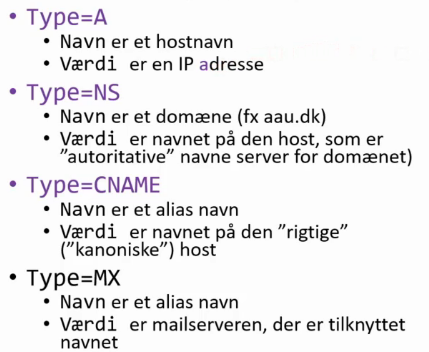
#### Formålet med DNS

DNS’ formål er at oversætte hostnavne til nummeriske værdier (fra www.whatever.dk til en IP). DNS er en *Distribueret Hierarkisk Database*, man uddeler belastning ud på mange servere i et hierarki. Dette undgår *single-point-failure* som sørger for at hvis en maskine går ned, så går hele webservicen ikke ned. DNS bruger *UDP* på port 53 (*velkendt port for DNS*). Hierarkier er opsat på følgende måde:



DNS benytter sig også af dataposer, som består af 4 ting: *Navn, type, værdi og TTL* (time to live).

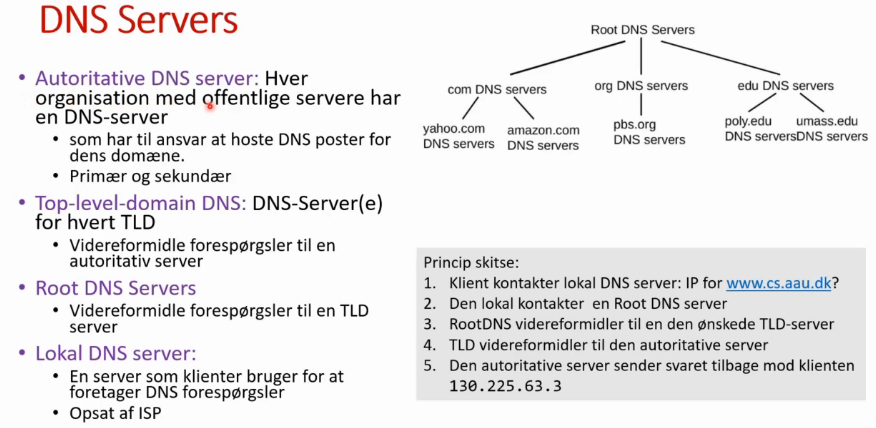
Typerne er beskrevet således:

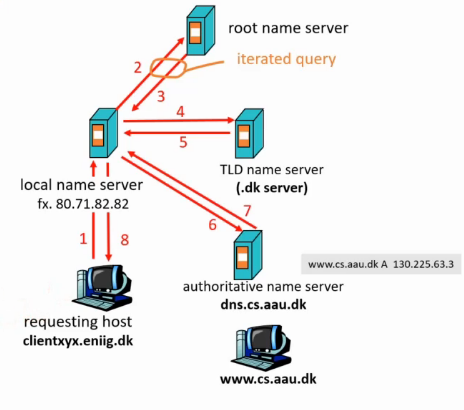
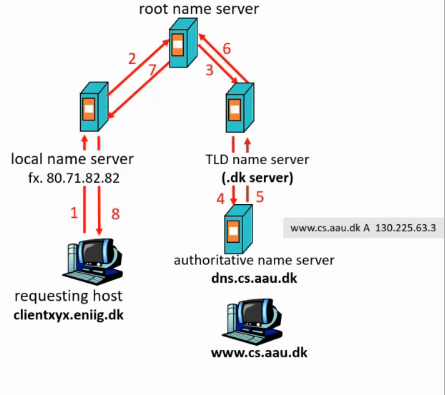


Her kan man se www.cs.aau.dk har *type A*, hvilket betyder at værdien er en *IP-adresse*, time to live er et antal sekunder. Mere om TTL i DNS-caching.

#### DNS server forespørgelser

DNS servere er også opbygget i hierakier, her er der tale om 4 forskellige ”niveauer”:

Dette procedure kald i ”princip skitse” er defineret både *rekursivt og iterativt*:



#### DNS caching

Når en name server for kendskab til en *DNS-post* gemmer den det i en *lokal cache* så der ikke skal bruges flere server kald på den samme forespørgsel. *DNS-caching* fjerner meget belastning fra serverne, men hvis hosten af hjemmesideren ændre IP-adresse er de cached adresser ikke gyldige. Her bruges Time to live (TTL) som er det antal sekunder som DNS-serveren holder på informationen i sin cache (en ”bedst før dato”).

*Root servers* består af **13 globale servers navngivet a-m.root-server.net**. Root servers er en logisk server som dækker over mange tusind fysiske servers. Der er mange fysiske servers fordi de altid bliver kaldt af *local-name-serveren*. *Flaskehalse* og *single-point-failures* bliver undgået med de mange fysiske servere og *DNS-caching mekanismen*.

#### Opdatering af DNS

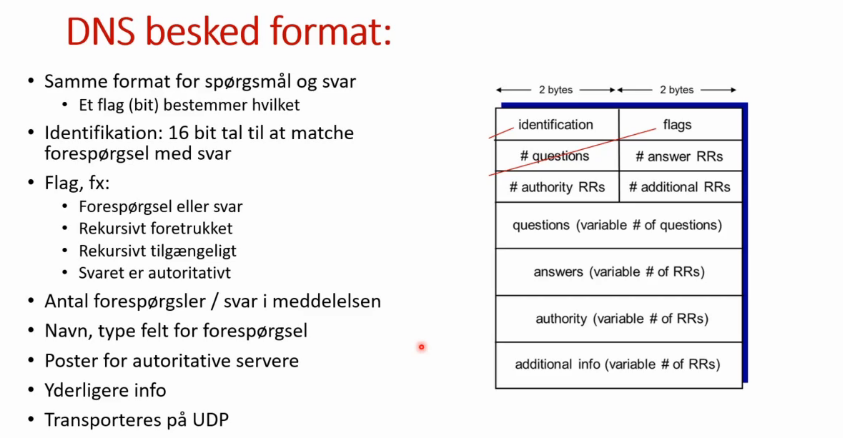
Hvis man skal registrer et domæne skal det købe det (hvis det er ledigt) og få det registreret hos en registrator (dk-hostmaster.dk). Her skal der også være minimums information som: ønsket domæne, name server og IP-adressen. Dette information bliver sat ind i et Top Level Domain server for .dk.

Opdatering af poster styres gennem et management interface hos en system admin eller en dynamisk-DNS.

#### DNS sikkerhed

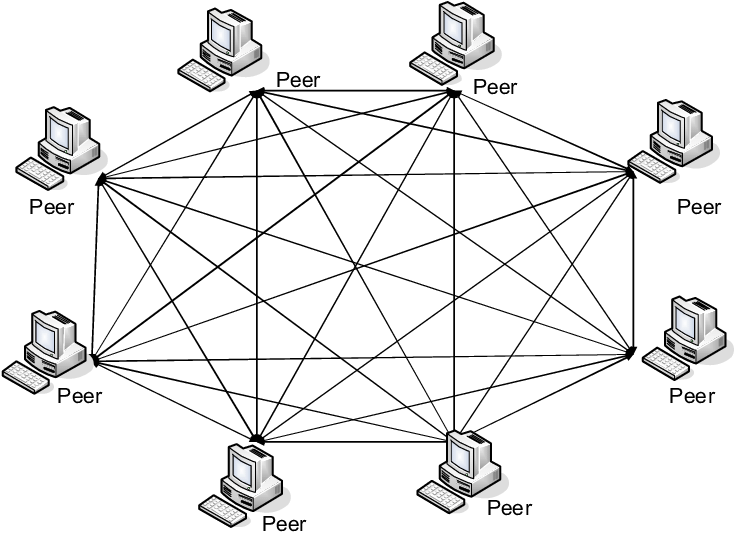
DNS kører over UDP som ukrypteret og er derfor \*lidt\* sårbar overfor diverse angreb. Spoofing, cacheforgiftning (lægge falske oplysninger i DNS-cachen og omdiregere brugere til et forkert website) desuden skal dette ske udenom caching-systemet. Dette er teknisk svært at udføre, men ikke umuligt. Dog er et DOS-angreb ikke muligt da der eksitere mange fysiske globale servers af root servers og på lavere niveauer er adresserne typisk cachet. Et DDOS angreb er kun muligt når der er snakke om store angreb. Der findes også DNSSEC (DNS Secure) hvor serverne ikke tager imod forespørgelser fra hvem som helst.

#### DNS værktøjer



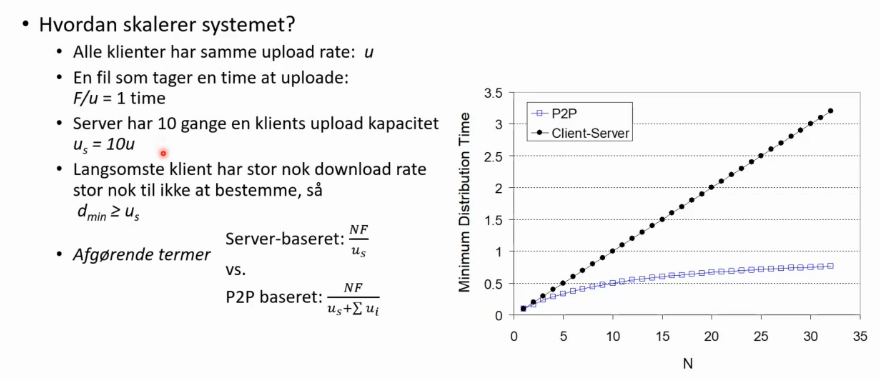
### Peer to peer (P2P)

Peer to peer er et alternativ til client-server modellen som benytter sig af end-host forbindelser. De er et system af slutbruger (dog nogle gange servers). End-hosts bidrager med resourcer som lager, beregningskraft eller netværkskapacitet og det virker derfor som et ”noget-for-noget princip” system. P2P systemer ejes ikke af nogen, så derfor er det meget svært at lukke systemet ned. På grund af der ikke er en centraliseret server er oppetiden også bedre end client/server modellen. Eksempler på P2P-systemer kan være: BitTorrent/uTorrent, Skype og TOR. Typiske anvendelser er fil-distribution, men kan også anvendes til andre ting.



P2P bliver brugt i ulovlige sammenhæng (som Napster, Piratebay og mange andre server indexe). Disse indexe er ulovlige, fordi de henviser til gratis filer med ophavsret.

P2P systemer er bedre til fildistribution end client/server modellen, fordi der ikke er mulighed for en flaskehals fra serveren. Hvis serveren skal uddele filer til mange end-hosts fordi serveren skal uploade mange kopier da clienterne skal downloade en hver. P2P skal kun uploade en fil til en end-host, som andre end-hosts kan downloade, hertil hjælper alle peers med upload-kapacitet som gør systemet bedre på stører skala (med nogle undtagelser).



Problemet med peer to peer opstår ved at end-hosts er dynamiske og melder sig til og fra konstant. Klienter vil hellere nyde end yde, hvilket ikke giver en ”fair” belastning over alle klienterne. Det er heller ikke sikkert at klienterne har en fast IP (hvis de bruger et NAT’ed netværk)

## Lektion 8: [The transport layer protocols and reliable communication](https://www.moodle.aau.dk/course/view.php?id=33287#section-8)

## Lektion 9: [TCP and Using the transport layer in programs: Sockets programming](https://www.moodle.aau.dk/course/view.php?id=33287#section-9)

## Lektion 10: [The network- and link layers](https://www.moodle.aau.dk/course/view.php?id=33287#section-10)

## Lektion 11: [Network security](https://www.moodle.aau.dk/course/view.php?id=33287#section-11)

## Lektion 12: [Web-security](https://www.moodle.aau.dk/course/view.php?id=33287#section-12)