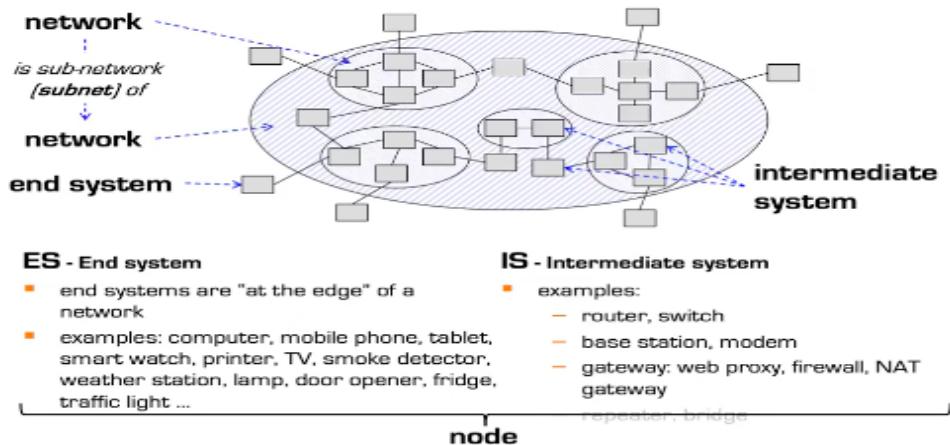


## Nettverksstrukturer

### **Nettverkskomponenter i datanettverk**

- Tjenere (Stor datamaskin som kjører en maskinlære algoritme som tolker info fra en hjemmeassistent for eksempel alexa,siri osv)
- Klienter (Dette er enheter som brukes daglig som mobil, pc osv.)
- Switcher (Brukes for å dele eller koble datamaskiner sammen)
- Routere ( Tar for seg datatrafikken og vender den mot en retning)

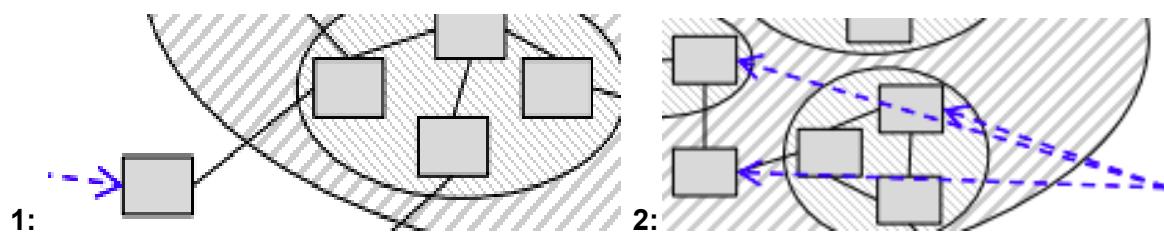
### **Network Components**



Over er et bilde som viser hvordan nettverket er satt opp. Dette er repetisjon fra IN1020.

### **Endesystemer (Den blå pilen)**

- Finnes helt ytterst i nettverket
- Eksempler: Datamaskiner, mobiltelefoner, sensorer (Altså klienter eller tjenere)
- Disse er enheter som ikke frakter datapakker mellom andre datamaskiner.
- Se bildet (1) under for bedre forståelse:

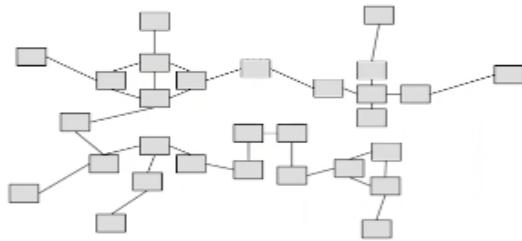


### **Intermediate system (Alle blå pilene)**

For eksempel:

- router, switch
- gateway
- bridge, repeater
- Enheter som frakter datapakker mellom nettverk
- Se bildet (2) for bedre forståelse:

## Network Structures



without the lines showing network boundaries,  
this looks like a typical graph

there are **nodes**, and **edges** connecting pairs of nodes to each other  
a specific arrangement of nodes and edges is called a **topology**

the terminology implies that edges are network connections

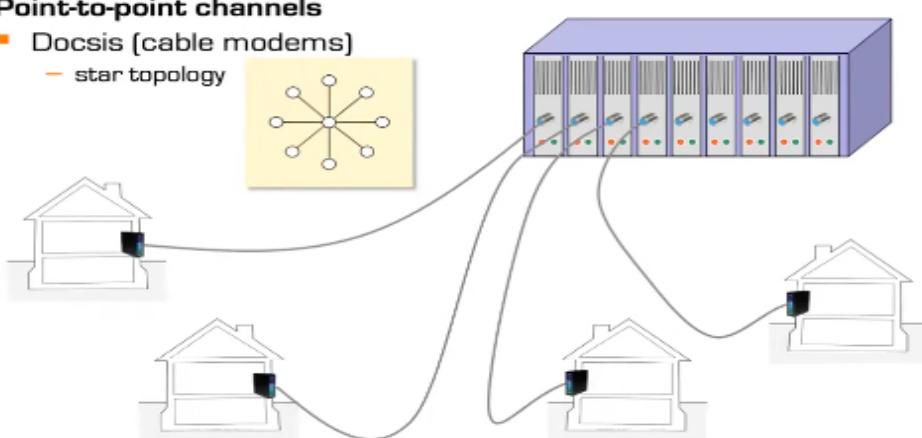
Hvis vi forestiller oss en vanlig graf, så har vi som sagt "noder" og "kanter". I dette tilfelle så vil nodene tilsvare "ende-systemer" eller "intermediate-systemer". Kantene tilsvarer nettverkstilkobling mellom ulike noder hvor vi ser at de nodene som er på kanten, ansees som endesystemer. Disse nodene får nettverkstilkobling gjennom nabonodene som finnes i grafen. Hensikten med dette avsnittet er å introdusere til de ulike topologiene, altså ulike måter å strukturere et nettverk på.

### Punkt-til-Punkt

#### Docsis(Cable modems)/Stjerne-topologi

##### Point-to-point channels

- Docsis (cable modems)
  - star topology



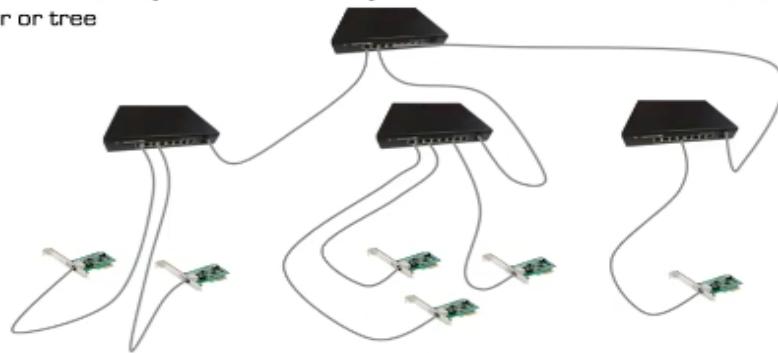
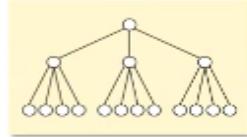
Dette er repetisjon fra IN1020 hvor vi har da punkt-til-punkt nettverk. Her benyttes ikke radiosignaler, men vi bruker kabler til å koble klient og tjener med hverandre. Dette kan gjøres gjennom "switcher" eller "modem" som da Carsten brukte i sitt eksempel. Dette går ut på at vi har såkalte "modem" som hjelper med å overføre datatrafikken fra ditt ståsted til omverdenen. "Omverdenen" kan vi omtale som internettleverandøren som hjelper oss med nettverksforbindelse. Vi ser et bilde av stjernetopologi hvor vi har at internettleverandøren er i midten og klienter rundt internettleverandøren. Slik fungerer stjernetopologi.

## Gigabit Ethernet/Tree-topologi

### Network Structures

#### Point-to-point channels

- Docsis (cable modems)
  - star topology
- Gigabit Ethernet ("1GB Ethernet")
  - star or tree



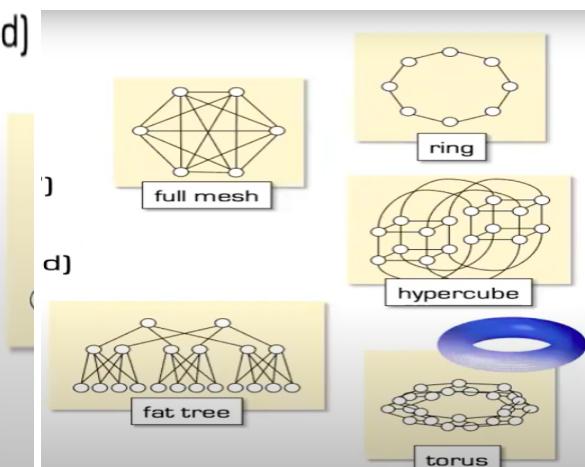
Dette var slik Carsten forklarte dette og brukte eksempel med tilkobling til IFI. Som man kan se i bildet over, spesifikt øverst til høyre, så har vi mange noder i bunnen. Disse nodene kan vi anse som klienter som kobler opp mot en "switch". Dette ville opprinnelig sett ut som en stjernetopologi, men switchen kan da videreføre tilkoblingen til en "ruter". Da kan ruten videre tilkoble alle disse klientene over til IFI-serverne som former en slags hierarki og blir en node-trestruktur. Merk at vi ikke har en "switch" som tar imot alle klientene, men vi har separate "switcher" som kobler opp til hver sitt kanal opp mot ruten.

#### ▪ IEEE 802.5 "TokenRing" (outdated)

- ring

#### ▪ Some supercomputers use

- full mesh
- hypercube
- torus
- fat tree



## Ring-topologi

Dette er en utdatert topologi som kan forklares gjennom hvordan den er satt opp. Hvis vi har at en node skal videreføre en datapakke til en annen node, så må vi traversere gjennom ulike noder som fører til at det kan ta tid å overføre en datapakke fra en annen. Derfor er ikke denne brukt så ofte.

## Full-mesh-topologi

Dette er en topologi som ikke benyttes så ofte ettersom det krever mange nettverkskort og kabler mellom de ulike serverne.

## Hyper-cube-topologi

En forbedret versjon av "full-mesh" hvor vi har dimensjoner som beskriver hvor mange noder som er koblet til hverandre. F.eks så har dimensjon "0" en node, dimensjon "1" har to noder, dimensjon "2" har 4 noder som former en kvadrat, dimensjon "3" har 8 noder som former en ternetning. Vi ser et mønster her ved at det blir noe form for  $2^x$  hvor x står for den gitte dimensjonen. Det fine med denne strukturen er at man kan finne antall utgående kanter fra en node, kan bruke for å beregne hvor mange noder som kan finnes i et nettverk ved å benytte av metoden  $2^x$ .

## Fat-tree-topologi

Brukes i mange av dagens datamaskiner hvor vi har en type tre-struktur. Hver node har da to foreldre og hensikten er at, skulle et av foreldrenes nettverkskort og tilkobling bli brutt, så kan vi bruke den andre forelderen. En ganske billig og god alternativ ettersom hver node har et begrenset antall forelder.

## Torus-topologi

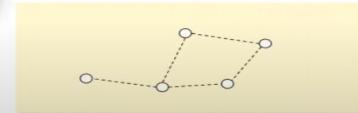
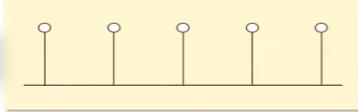
I denne topologien så vil denne vokse seg større, men hvor nodene ikke trenger mer enn "4" nabonoder. God for eskalering som egner seg for store kløstere og supercomputere.

## Trådløs-tilkobling

### Network Structures

**Broadcasting channels**

- Cable
  - old-fashioned Ethernet 
- Radio
  - Aloha (first wireless data transmission)
  - WiFi (IEEE 802.11)
  - mobile: 3G, 4G, 5G
  - satellites 
- Properties
  - when one node sends, potentially many nodes can hear it
  - when two nodes send, both messages are potentially ruined
  - error detection is important
  - coordination is desirable



Trådløs-tilkobling hvor endesystemer ikke behøver å tilkoble seg til nettverk gjennom kabler, men radiosignaler. En ruter er oftest knyttet til radiosignalene hvor den da tildeler nettverk til klienter gjennom å tilkoble til servere via radiosignalene. Lurt å vite om de ulike "properties"

som vi ser i bildet over. F.eks at hvis en node sender radiosignaler så kan potensielt andre noder høre disse radiosignalene. Eller at to noder som sender radiosignaler samtidig, kan føre til at meldinger blir potensielt ødelagt. Derfor er koordinering ønskelig for å unngå at dette skjer.

## **Nettverksoppgaver**

Det er en rekke oppgaver som nettverket må være i stand til å håndtere. Blant annet må nettverket:

- Vite veien fra en node til en annen. F.eks vite muligheter til å nå en node, skulle tilkoblingen til en annen node bli brutt.
- Involvere endesystemene for å vite hvem som er sender og mottaker i en meldingsutvikslsing. F.eks så er det viktig at nettverket vet at datamaskin av prosess "A" skal sende sin melding til datamaskin av prosess "B". Meldingen skal ikke sendes til en datamaskin av prosess "D" eller "C" f.eks.
- Vite hvilken prosess som skal kontaktes på det gitte endesystemet. Det kan være flere prosesser i et gitt endesystem f.eks mobil, datamaskiner som står i veien så det å vite at en melding skulle sendes til mobilen, så skal den sendes til mobilprosessoren, ikke datamaskinen.
- Samtidig som en sender må vite veien til en mottaker, så må en mottaker også vite veien til en sender. Skulle jeg sende en melding til en annen person gjennom nettverk, så skal det være mulig for den andre personen å finne veien tilbake til meg.
- Å kode dataene i en omfattende måte. Dvs, hvis jeg sender en melding på engelsk til en som bor i Frankrike, så skal mottaker være i stand til å motta den meldingen i fransk f.eks.
- Å håndtere nettverksproblemer. F.eks så kan det hende at ruten til å nå en mottaker som jeg ønsker å nå, ikke går ann siden ett av nodene i ruten ble brutt. Dermed må nettverket være i stand til å håndtere dette nivået ved å finne en annen rute slik at jeg når min mottaker.
- Å garantere at kommunikasjonen mellom to noder er sikkert og ivaretar konfidensialitet. En man-in-the-middle skal ikke være mulig å bli utført og meldingsutvekslingen mellom to noder skal være konfidensielt mellom kun dem, ingen andre skal lytte på denne meldingsutvekslingen.
- Å kommunisere effektivt og å unngå høy latens. Dvs at mottaker skal ikke vente så lenge for å motta en datapakke fra en sender. Det skal være en effektiv overføring mellom to noder.

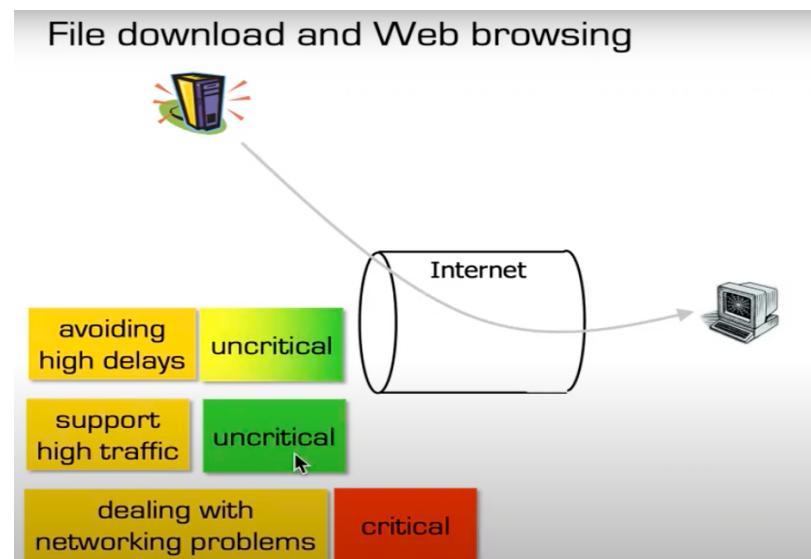
## **Brukstilfeller**

the application



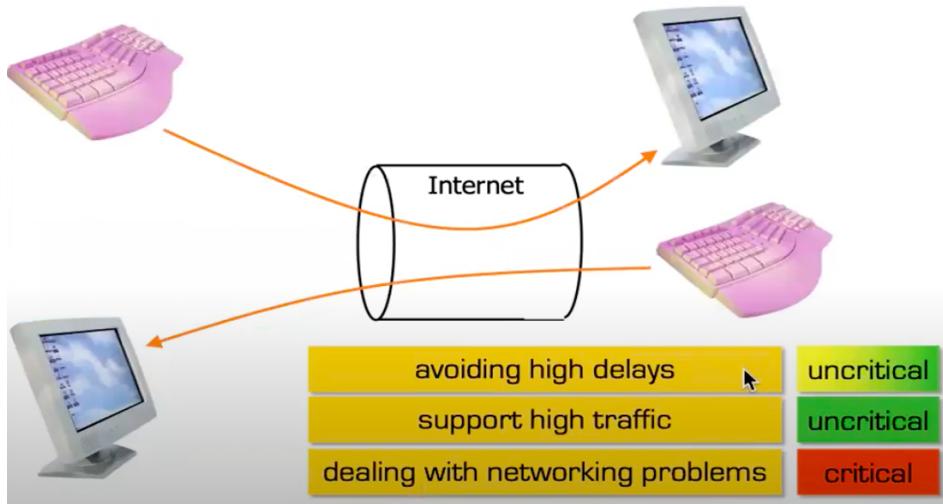
Applikasjonen har forventninger fra internettet og internettet tilbyr en sett av muligheter som applikasjoner kan oppnå. Andre internett kan gjøre bedre men poenget er at forskjellige applikasjoner har ulike forventninger fra internettet. Men generelt så er de tre punktene som vist i bildet, de punktene som man ønsker å oppnå. Applikasjonen må prøve å unngå store forsinkelser på veien gjennom nettet, støtte store mengder av data og å håndtere nettverksproblemer. Nå skal vi se på ulike tilfeller under med disse punktene.

### **File download and Web browsing**



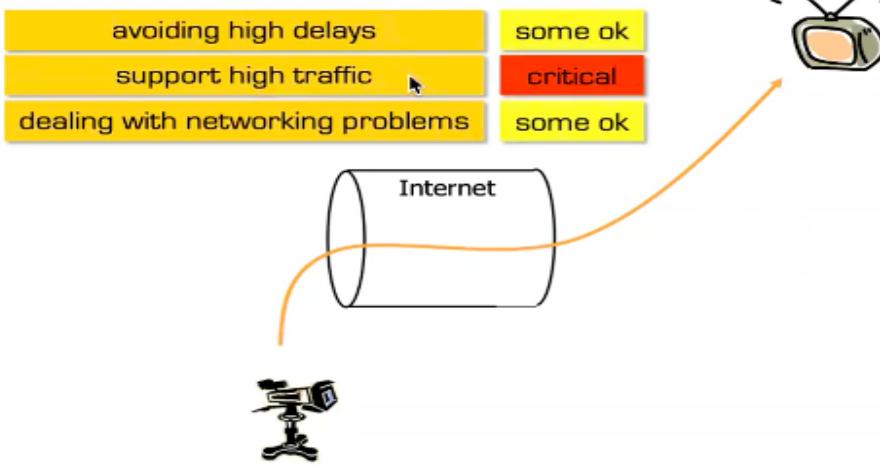
Store forsinkelser ved filnedlastning og web browsing, er ikke så prioritert. Men det er allikevel ønskelig å at slike forsinkelser ikke forekommer ofte. Å støtte høy datatrafikk er ikke særlig prioritert heller. Men det å håndtere nettverksproblemer er ønskelig og høyt prioritert siden datapakker kan gå tapt om vi ikke løser slike problemer. For hvis et endesystem skal overføre en datapakke til et annet endesystem, og et nettverksproblem oppstår, så kan det hende at pakken aldri nådde det andre endesystemet.

## Textual commands and textual chat



Ved fjerninlogging til IFI-nettverket, så er det vel ikke kritisk at tilkoblingen tar tid. En måtte å tenke på disse brukstilfellene på, er å tenke på å bygge en app med et gitt system. Vi ønsker at brukerne skal være i stand til å utføre det minimale ved appen, altså "MVP". Det kan være brukervennlig hvis alt går smooth, og kjapt, men å håndtere de problemene som kan utgjøre en fare ved appen, er det som burde prioriteres høyt. Slikter er det her også hvor det å håndtere nettverksproblemer ansees som kritisk og er viktig at de blir håndtert.

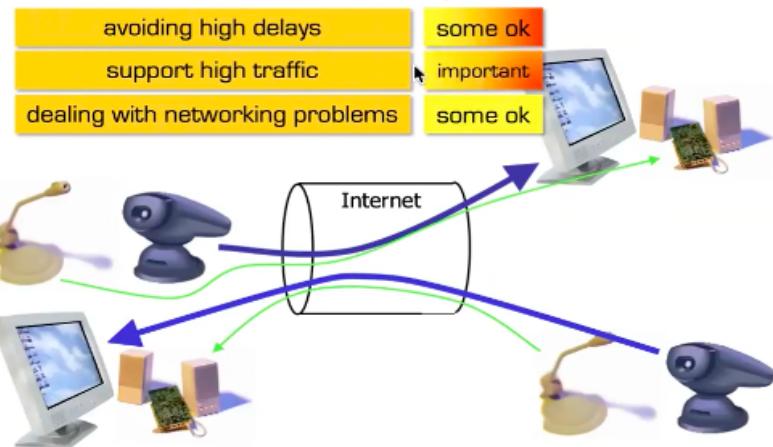
## Live and on-Demand Streaming



Om vi ønsker å se på Netflix, så kan det være ønskelig å prioritere de tilfellene som kan påvirke brukerens opplevelse. F.eks så kan det å støtte høye trafikk, være ønskelig siden vi ønsker å se netflix-videoer i høy kvalitet og at alt er bra. Å støtte høy trafikk vil si at vi kan overføre mer data i Netflix i dette tilfellet hvor vi kan da se på videoer i høy kvalitet med bra lyd osv.

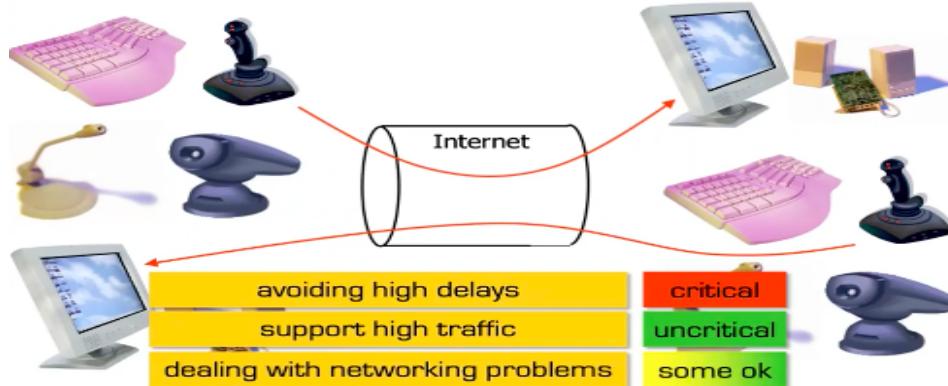
---

## AV chat and AV conferencing



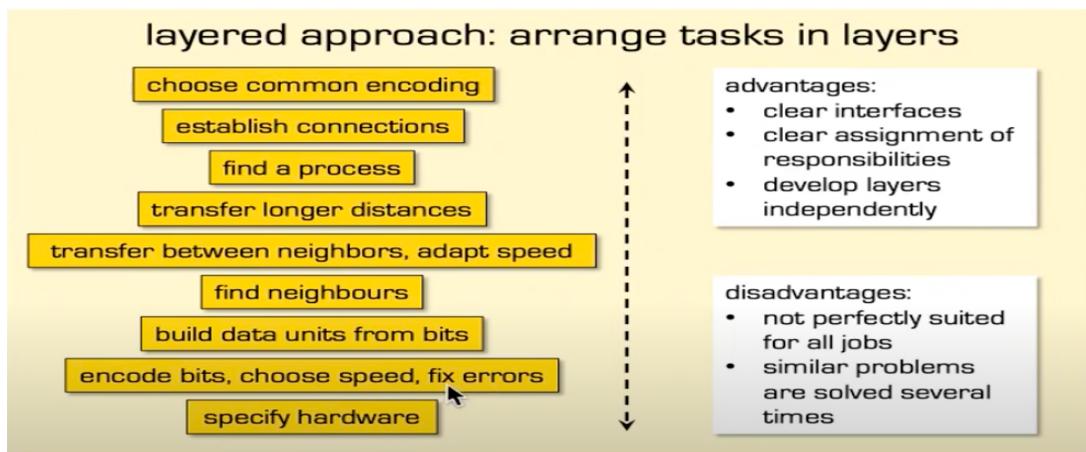
Her er det snakk om audiovisuelle komponenter og "Zoom" er et slikt eksempel. Vi har sikkert ikke stort behov for store datamengder, men å tillate at flere brukere kan få muligheten til å møte opp til zoom, så det å støtte høytrafikk kan være viktig å ha i bakhode.

## Haptic Interaction



Tilslutt har vi gaming-interaksjon hvor vi ser at å unngå høye forsinkelser er noe som burde høyest unngås og burde prioriteres. Så for å konkludere så har vi sett på ulike brukstilfeller og for hvert enkelt tilfelle så må internett kunne støtte alle disse. Det trengs en form for struktur til å løse disse oppgavene, noe vi skal se på videre.

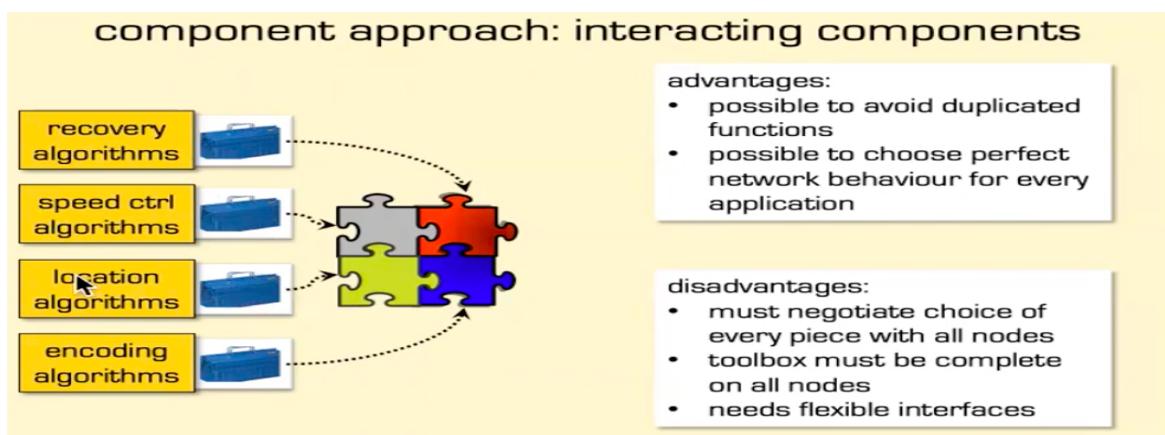
## Nettverksarkitektur



En mulig retning til å håndtere ulike oppgaver på, er å bruke et slags "lag"-vending hvor vi da løser de ulike oppgavene ved å se på lagene over. Man ser det fra bunnen og går da helt til topp hvor man kan se prosessen i bildet over(selvforsklaende).

Fordelen ved dette er at vi kan ha klare grensesnitt som har spesifikke ansvarsområder som de skal løse. I tillegg til at disse lagene kan utvikles uavhengig.

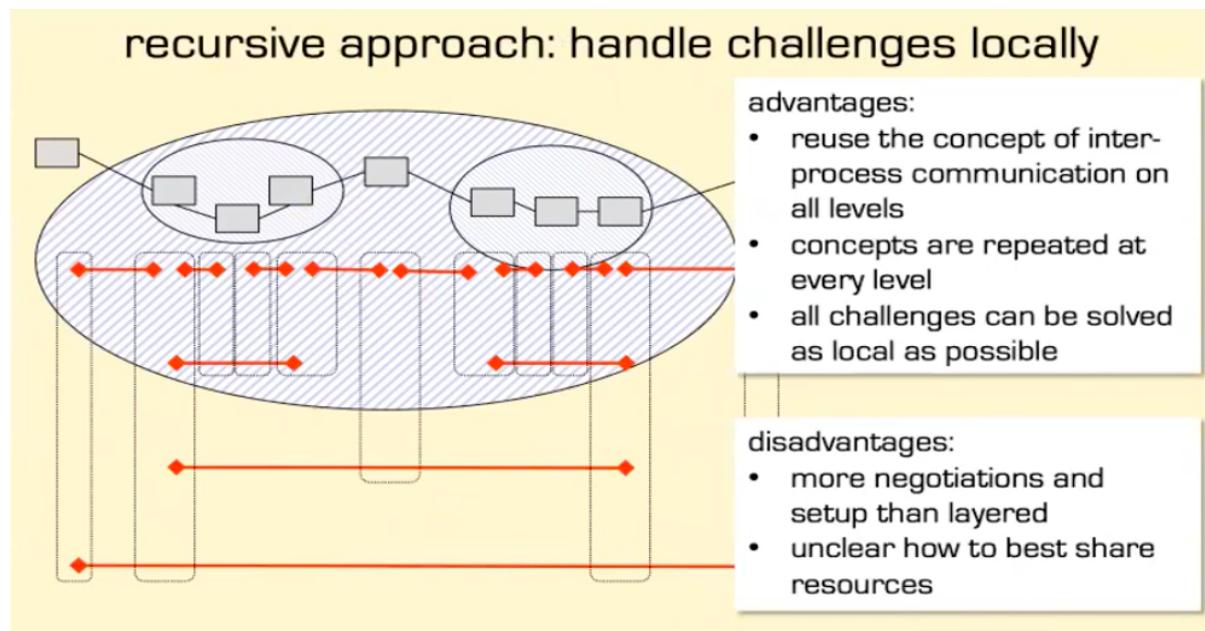
Ulempen er at disse lagene kommer til å trenge kommunikasjoner mellom hverandre for å håndtere ulike brukstilfeller i en applikasjon som vi så tidligere. Eller så kan det være at liknende problemer må kunne håndteres gjentatte ganger for å tilpasse de ulike lagene i en måte slik at, skulle det samme problemet oppstå igjen så vil problemet løses effektivt.



En annen vending som er mulig å se, er å inndele oppgavene i komponenter hvor vi har en verktøykasse som håndterer feil(nettsverksproblemer), lokasjon(veien mellom endesystem og intermediate-system), overføring av datapakker(hastighetstilpasninger) og kode-algoritmer. Fordelen er at funksjonene fra disse ulike verktøykassene, kan da benyttes en gang siden man har en funksjon som benyttes for det enkelte tilfelle. En annen fordel er at vi kan

benytte den perfekte samlingen av komponentene over, til å tilpasse en enkelt applikasjon ved å håndplukke de ulike funksjonene fra verktøykassene.

Ulempen er at alle noder må ha forhåndsinstallert disse komponentene for å kunne velge ut de rette komponentene for den gitte applikasjonen. Dermed kommer en annen ulempe om å finne ut hvilke verktøy som er best for den applikasjonen. Tilslutt, trenger man fleksible grensesnitt for å gjenbruke algoritmer for de ulike oppgavene innad ulike applikasjoner.



I denne vendingen, så håndteres alle problemer lokalt. F.eks hvordan skal kommunikasjonen foregå mellom to direkte nabonoder som du ser i bildet(intermediate system). Man kan da bruke enkoding av bit, tilpasse hastigheten, fikse feil mellom nodene. Videre kan man utvide dette ved at man går fra to noder i et lite nettverk og gå over til et større nettverk ved å gjenbruke valgene fra tidligere som ble tatt av de to nabonodene lokalt. Konseptet om kommunikasjonen er en kommunikasjon mellom to direkte noder, men desto dypere man er i rekursjonen, desto nærmere må nodene være fysisk til hverandre.

Fordelen er at man løser lokalt problemene, noe som er det beste miljøet å løse problemene i siden man vet hvordan de skal løses effektivt. Man velger da de lokale valgene gjort av to nabonoder og bygger dette til å løse de ulike brukstilfellene som vi så tidligere.

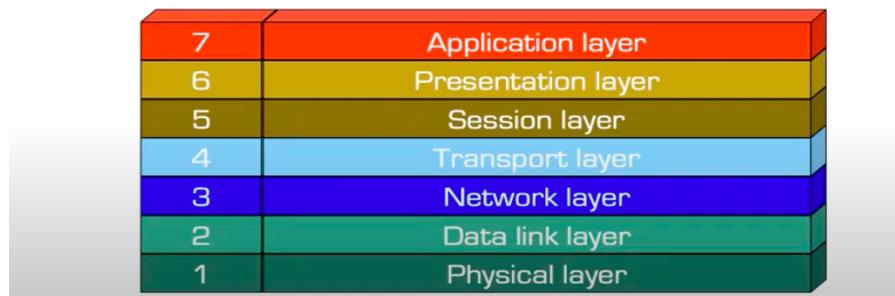
Ulempen er at man kommer til å koble de ulike nodene i bildet sammen og hvordan man best deler ressursene mellom hverandre, kan bli uklart. Det trengs også flere "samtaler" mellom noder til å vite hvilke valg som er best å ta sammenlignet med "lag"-arkitekturen.

Vi har nå sett på ulike arkitekturen, men vi har da to ulike strukturer som blir benyttet i dagens samfunn. Disse er da **“ISO OSI(Open systems interconnection) Reference-model”** og **“TCP/IP-modellen”**. “TCP” er allikevel den som brukes til nettverk i dagens samfunnen siden den er best tilpasset for dette. Allikevel så skal vi forkalre OSI-modellen siden TCP baseres seg på de ulike lagene i OSI-modellen.

## OSI-modellen

### **ISO OSI [Open Systems Interconnection] Reference Model**

- model for layered communication systems
- defines fundamental concepts and terminology
- defines 7 layers and their functionalities



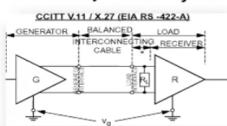
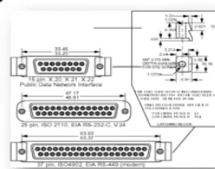
Dette er da OSI-modellen hvor vi skal se på de ulike lagene i detalj.

## Fysiske laget

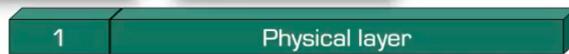
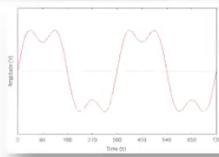
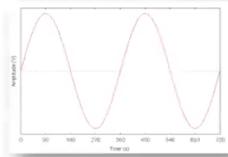
Layer functions: physical layer

### **Responsibility:** insecure bitstream between adjacent systems

- mechanics
- electronics
- procedural



encoding and decoding of bits

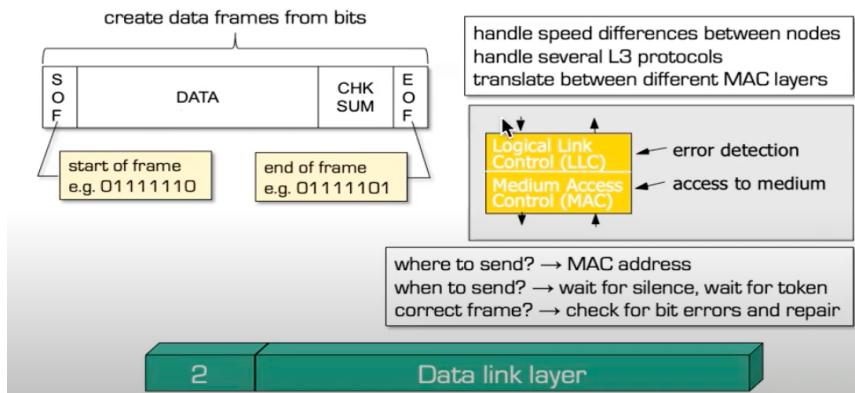


- Sender bitene mellom to direkte nabosystemer i et nettverk. Her er ansvarsområdet blitt implementert uavhengig om det er kabler som benyttes eller wifi, 3G, 4G osv.
- Signalrepresentasjon av bits
- Sørger for at 1-bit også blir mottatt som 1-bit
- Mekanikk: koblingstype, kabler/medium. Når man benytter av switchere som da kobler en gitt server til en datamaskin, så har man da løst hvordan overførselen av bitene skal gjøres.

- Elektronikk: Spennin, bit-lengde. Her ser vi mer på hardware og i bunnen i abstraksjonslaget over hvilke porter som er blitt benyttet (NOR, XNOR osv).
- Hvordan de skal sende bits(Procedural): Dette er da referert i bildet med den blå skriften som da beskriver hvordan bit-overføringen skal gjøres mellom to naboer.

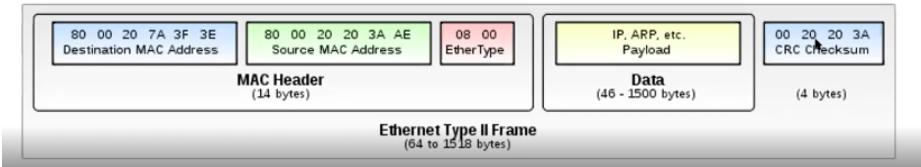
## Linklaget

**Responsibility:** error-recovering frame stream, adjacent systems  
**Reliable data transfer between adjacent stations with frames**



- Pålitelig overføring mellom to nabonoder (Overfører pakken mellom datapakken fra Oslo til endesystemet til New York f.eks)
  - Pakker som overføres i linklaget kalles "frames"
  - Feildeteksjon or retting innenfor en "frame".
- Man lager større enheter av bitene fra det fysiske laget og disse enhetene kalles for en dataramme(frame) som har en start og en slutt med kjente bitmønstre.
- Det er viktig å sikre at:
  - Mengden bits fra datapakken vil bli delt opp i "frames".
  - Disse "framesene" vil bli sendt videre til endesystemet
  - At dataen som er blitt bevart i "framesene", ikke går utapt.
- Den vil inneholde en sjekksum som forsikrer om dataen er tilkoblet riktig eller ikke. Sjekker altså at bitene stemmer overens med datapakken som skal overføres og at ikke noe har endret seg.
- En "switch" vil kun jobbe på linklaget
- Vanligste linklagene er Ethernet(kablett nettverk) og Wifi(trådløs nettverk) for å overføre datapakken videre til det fysiske laget.
- Linklaget vil kunne ha enkel flytkontroll
  - Rask sender, treg mottaker
- Medium Access Control (MAC)
  - Kontrollerer hvordan rammene skal overføres på nettverket.
- Logical Link Control (LLC)

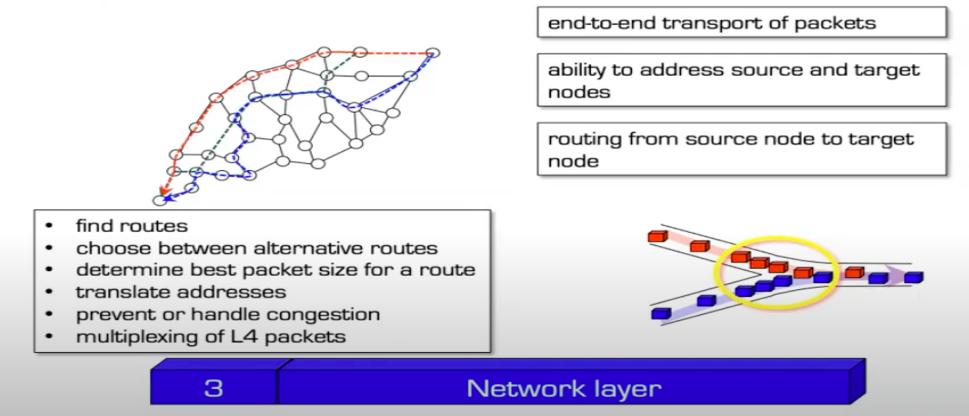
- Linklaget har en feildetekterende mekanisme, Ser til at dataen transportereres riktig.
- Vanligste linklagene er "Ethernet", og "Wifi". Disse er ganske like men noen forskjeller
- Bruker MAC-adresser, en 6-byte adresse (48-bit) som ofte er lagret i nettverkskortet
- MAC-adresse, brukes både på WiFi og Ethernet
- Hver byte representeres med en heksadesimal verdi: 07:01:02:2C:4B
- Slik er headeren i Linklaget bygd opp med Ethernet linklaget som eksempel:



## Nettverkslaget

Layer functions: network layer

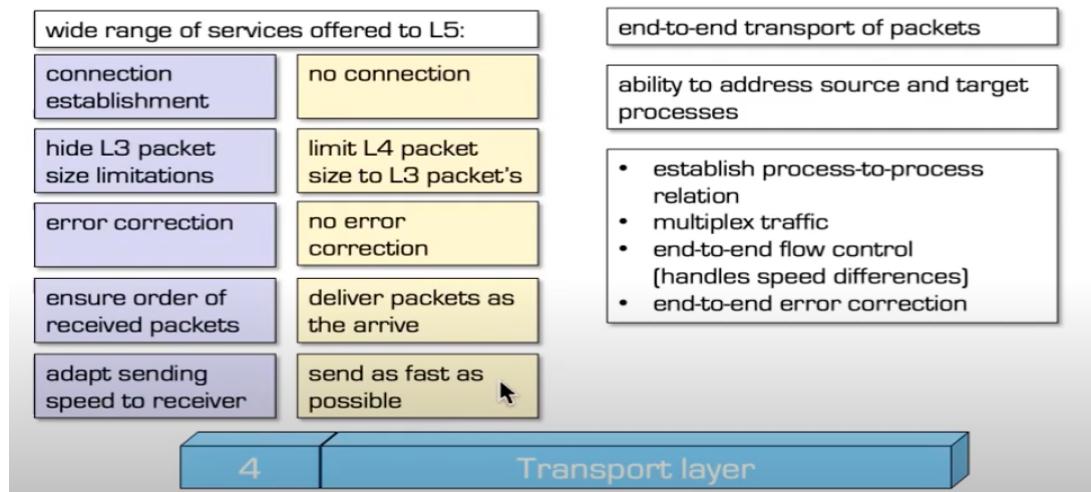
**Responsibility:** packet stream between end systems



- Kobler sammen systemene ende-til-ende. Finner da stien som skal benyttes for å overføre datapakken mellom to noder.
- Har som hensikt å transportere datapakkene gjennom den foretrukne stien fra en gitt algoritme(DFS, BFS). Rutene blir brukt siden datapakken skal transportereres til en gitt destinasjon, for eksempel at en datapakke fra Oslo skal til New York. Ruterne vil da ta for seg de smarte, korte veiene for at datapakken fra Oslo skal nås til New York, og det finnes mange stier og velge mellom.
- Man ser på nettverket til internett som en graf hvor man har da algoritmer som DFS, BFS eller lignende algoritmer som finner da den beste ruten for å overføre en datapakke mellom to nabonoder.

## Transportlaget

## Responsibility: end-to-end message stream between processes



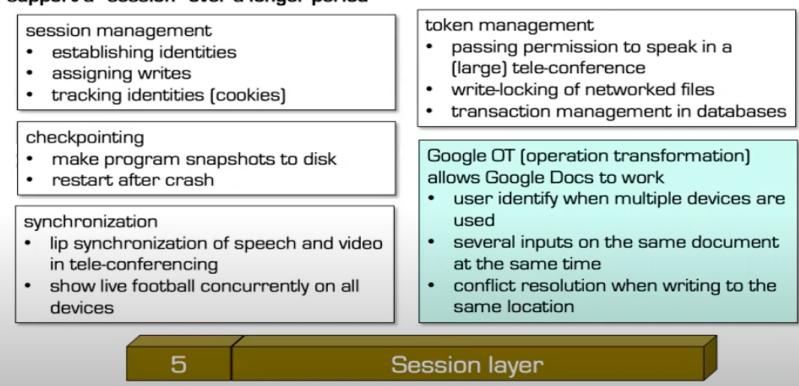
- Dette laget sender meldinger mellom prosesser mellom sender og mottaker.
- Trenger muligheten til å adressere kilder og mottakerprosesser som vi får gjennom nettverkslaget.
- Dette laget må kunne skille mellom UDP(Lyd/Video) og TCP(Epost, HTTP, Filoverføring).

Laget tilbyr også en rekke tjenester som vist under og de vist i bildet over

- Metningskontroll som vil si at man garanterer at pakkene leveres i riktig rekkefølge, hver datamengde får et sekvensnummer, disse numrene vil da mottas av en mottaker. Mottakeren vil sende en kvittering(ACK) for hvilke nummer med datapakke som skal sendes.
- Pålitelighet - Pakker sendes på nytt hvis kvitteringen (ACK) ikke kommer frem innen en gitt tid.
- Flytkontroll og metningskontroll har som mekanismer å sørge for at det ikke blir kø i datatrafikken.
- Flytkontroll: Ikke sende fortore enn mottakeren kan ta imot.
- Metningskontroll: En ruter er i utgangspunktet bare en FIFO (first-in-first-out) kø. Om det blir for mye trafikk over en gitt kø, vil det føre til stopp i trafikken. Dette kalles "congestion". Om det er så mye trafikk at det blir full stopp for alle, kalles for "congestion collapse". For å unngå dette ble det lagt mekanismer i TCP for å tilpasse senderaten til nettverksforholdene. Justerer kapasiteten for datatrafikken slik at det er mulig å få ført trafikken videre. Ressursene deles teoretisk sett likt mellom forbindelser.
- Byte-strøm og levering i rekkefølge( Sekvenser osv)

## Sesjonslaget

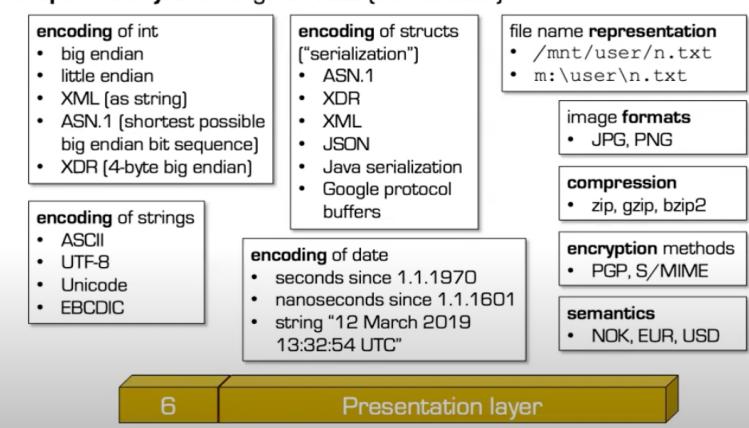
**Responsibility:** structured dialogue  
support a "session" over a longer period



- Laget er ansvarlig for å lage en strukturert dialog mellom prosessene som kjører i to endesystemer. Selvom en prosess bytter fra en datamaskin/nettverk fra en annen, så skal den sesjonen som ble opprinnelig etablert, være liggende uansett.
- Det er viktig å etablere identiteter og benytte av cookies til å følge sesjonen og håndtere et tilfellet som vi så i punktet over.
- Checkpointing vil si at man skal være i stand til å benytte av den samme sesjonen skulle datamaskinen slå av.
- Token-håndtering som vil si at man tildeler rettigheter som at hvem som skal få lov til å snakke i en tele-konferanse.
- Synkronisering er et annet mål som at når man streamer og ser på en fotballkamp så skal alle se at "Chelsea" skårte i nøyaktig samme tid.
- Google OT er et godt eksempel på lag 5, som beskriver de øvre punktene. F.eks at en bruker skal kunne identifiseres selvom man benytter flere enheter. Sesjonslaget skal også håndtere konflikter hvor to personer skriver i nøyaktig samme linje slik at begge personers input skal kunne vises frem.

## Presentasjonslag

**Responsibility:** exchange of data (semantics!)



- Ansvarlig for å løse ulikheter mellom endesystemer. F.eks så skal to datamaskiner mellom endesystemer, ha en felles forståelse av hva en “Integer” er.
- Som vi ser i bildet, så skal begge endesystemer være klare over “semantikken” bak de ulike datatypene.

## Applikasjonslaget

Layer functions: application layer

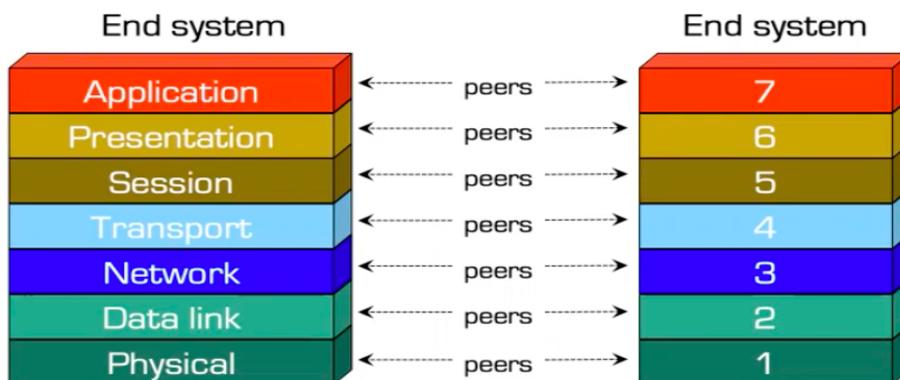
Responsibility: cooperating entities



- Beskriver hvilken applikasjonen som endesystemene skal kommunisere med hverandre. Om dette er et spill, webserver, video osv.

## Architecture

Data flow between two adjacent systems

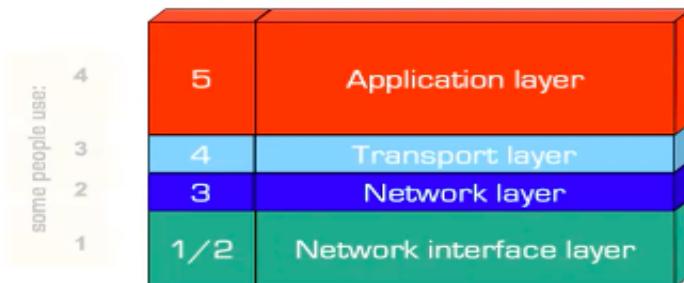


For å konkludere så har vi at “Applikasjonslaget” må vite hvilken applikasjon som tas i bruk. Videre må “Presentasjonslaget” håndtere de ulike presentasjonsutfordringer som å danne en felles forståelse for hva en “Integer” er. “Sesjonslaget” må opprette en sesjon mellom endesystemer gjennom identiteter og spore disse gjennom “cookies”. “Transportlaget” må kunne sende meldinger mellom prosessene i endesystemene ved å danne en forbindelse mellom dem. “Nettverkslaget” skal kunne overføre datapakkene mellom endesystemene.

“Linklaget” må kunne dele opp datapakken i “rammer” som skal overføres til et endesystem. “Fysiskelaget” skal da dekode “rammene” og tolke informasjonen bit-for-bit.

## TCP/IP

### Five Layer Reference, Internet Reference Model and a Comparison



### **TCP/IP Reference Model Internet Architecture**

- ISO-OSI presentation, session and application layer merged
- ISO-OSI data link layer and physical layer merged to form Network Interface

I TCP/IP har man kun 5 lag hvor vi har at “Network-interface-layer” utfører funksjonene til det “Fysiske” og “Link”-lagene. Viktig å poengtere da at “Network-interface-layer” følger terminologien for de to lagene i henhold til OSI-modellen. Denne er simplere enn OSI ettersom vi har færre lag å forholde oss til og gjør det lettere å forstå hvordan kommunikasjon mellom to endesystemer skal foregå. Dette er standard-arkitektur som benyttes til å forklare hvordan kommunikasjon i nettverk skal foregå.

#### Why no clear separation of upper layers?

- layers 1-4 are essential for co-existence on the Internet
  - e.g. different congestion control mechanisms on different hosts can lead to strong congestion
- session and presentation layer functions provide mostly application support

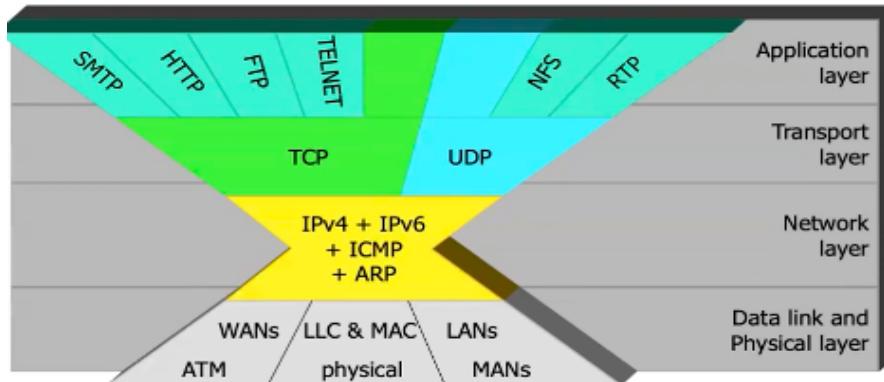
#### Layers 3 and 4 are not clearly separated

- transport protocol (TCP, UDP, others) and network protocol IP
- sometimes hard to draw a clear line where TCP ends and IP begins
- example:
  - Early Congestion Notification (ECN) capability is indicated on layer 3 and congestion is indicated on layer 3
  - sender is told about receiver's reception of congestion signal on layer 4

Grunnen til at vi mangler ulike lag, er at man kom frem til at lag (1-4), er de viktigste lagene for internett. Disse er vanligvis implementert i kjernen så da må noe være forhåndsdefinert i motsetning til om det var i user-space. En annen grunn til at presentasjon og sesjonslaget ikke er prioritert så høyt, skyldes av at disse lagene kun er ment for å støtte en applikasjon. Disse lagene er svært essensielle til å kunne overføre datapakker mellom to endesystemer, men heller løse problemene som å gjøre kommunikasjonen mellom endesystemer bedre.

Lag "3" og "4" er ikke klart separert i og med at de er nær knyttet til hverandre. F.eks så er det slik at om TCP-protokollen blir innført, så benyttes IP-protokollen på bakgrunn av dette. Derfor er lag "3" og "4" bundet til hverandre, ikke separert som i OSI-modellen.

## Internet Protocol Stack



Nickname: "Hourglass Model"

Her ser vi en rekke protokoller som blir benyttet i internett. Disse skal gås mer igjennom i neste forelesning, men dette er protokollene som benyttes mellom datakommunikasjon til to endesystemer.