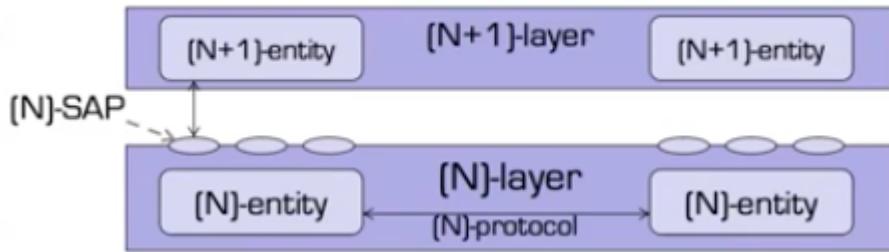


Terminologi knyttet til lag innenfor OSI-modellen

Layers in General (OSI terminology)



(N)-Layer

Et abstraksjonsnivå som har definerte arbeidsoppgaver for et visst lag. Lag vil da bestå av diverse lag fra OSI-modellene f.eks at 3-layer beskriver da 3 lag satt sammen hvor disse lagene kan være Applikasjon, presentasjon og sesjonslaget f.eks, eller bare nettverkslaget siden dette er det tredje laget innenfor OSI-modellen.

(N)-Entity

Aktive elementer innen et lag, f.eks at endesystemer blir involvert i (N)-layers som er da sant i forhold til TCP/IP hvor da klient og tjener blir involvert i alle de 5 lagene. I hovedsak så blir prosesser i endesystemene eller en såkalt “intelligent I/O module” som blir involvert.

(N)-Service Access Point, (N)-SAP

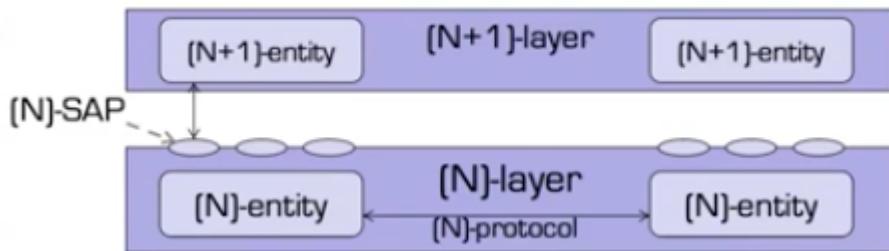
- Det kan hende som oftest at et lag kommer til å alltid til å benytte seg av tjenestene som er i lagene over og under. For å vite hvilke tjenester det er snakk om, så har man såkalte (N)-SAP(Service Access Point) som beskriver hvordan tjenesten ser ut mellom overførsel av den tjenesten fra et lag til et annet. Et lag kan flere slike (N)-SAP. Dette sees bedre i lag-4(transportlaget) siden man velger mellom protokollene TCP eller UDP.

(N)-Protocol

- For å utføre disse tjenestene fra hvert lag, så vil entitetene fra lag “N” snakke med hverandre med hjelp med tjenestene fra “N-1” hvor disse entitetene former et regelverk som beskriver hvordan overførsel skal foregås. Dette kalles for “protokoller”.

Protokoll: Communication mellom entiteter i lagene

Layers in General (OSI terminology)



En protokoll definerer mellom entitetene mellom lag:

- Formatet, hvordan skal meldingsutvekslingen foregås
- Rekkefølge for hvordan meldingene skal sendes, slik at funksjonaliteten skal utføres.
- Overføring mellom to eller flere kommuniserende entiteter. Man forhandler i forbindelsen mellom entiteter hvor mye man skal overføre av datapakker og derfor så er det nødvendigvis ikke slik at enhetene kan bare sende alt av gangen, men man har også regler hvor en regel kan være "hvor kan man overføre av pakker". Man beskriver også hvilke meldinger som skal overføres og hvordan denne utvekslingen skal gjøres
- Må definere handlinger som utføres ved overføring av datapakker, mottakelse av datapakker eller hva som skal skje når datapakken ikke blir overført

Protokollen definerer ikke:

- Tjenester som leveres til laget over, også "N+1". Er vi i transportlaget så skal ikke en protokoll overføre tjenestene fra dette laget, over til applikasjonslaget.
- Tjenester som blir brukt, også beskrivelse av hva denne tjenesten går ut som er da "(N-1)-SAP".

Informasjon om protokollen selv

- Protokolls syntaks: regler for formatering, hvordan meldinger skal se ut
- Protokolls semantikk: Regler for handlinger i tilfeller hvor man sender melding, mottar melding osv. Alle semantikkene må være like for alle entiteter som skal snakke med hverandre.

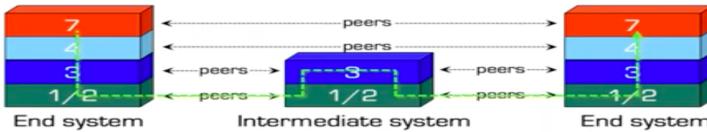
Eventuelle navn som forbinder med meldinger

- Protocol data unit (PDU)
- Frame(lag 3), Packet, message(lag 4), datagram(lag 4)

- Symbol

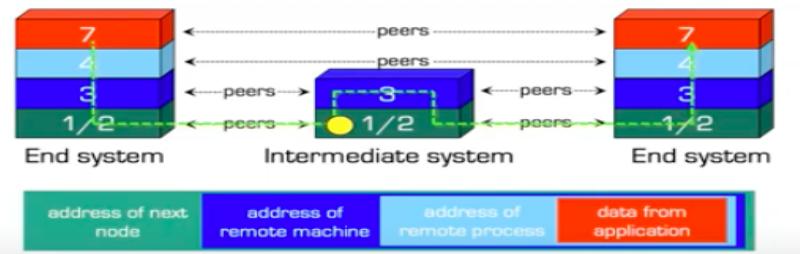
Data flow gjennom nettverk

Data flow through the network

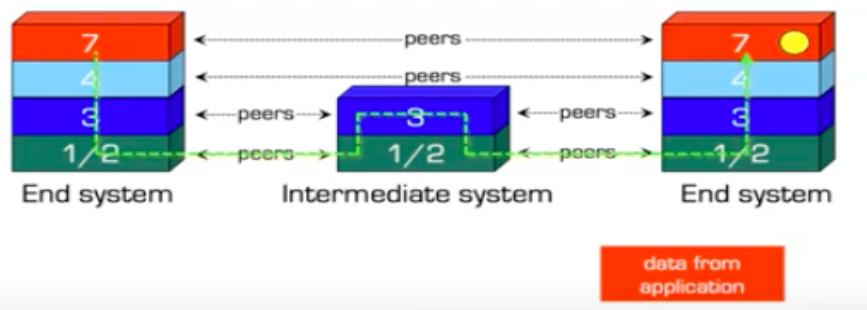


- Each sending N-entity at layer N adds N-protocol information
- ... which is important for its peer N-entity
- ... and the receiving N-entity removes it before passing the data to layer N+1

I "IN1020" og kanskje "IN2120", så husker du det at ved TCP/IP-laget, så har de ulike lagene har "headere" hvor man pakker opp datapakken i "headers" som er da forskjellig fra lag til lag. F.eks så er "headeren" til transportlaget ulik fra "headeren" til nettverkslaget siden disse lagene har forskjellige protokollen. "Innkapsling" er begrepet som Carsten bruker til å beskrive "Header"-prosessen. Bildet handler om denne prosessen som kan da sees i bildeteksten hvor hver entitetet i et gitt lag "N", legger til protokoll-informasjon fra det "N"-te laget. Dette er da viktig for de andre entitetene(Intermediate systems, mottaker(server)) slik at de vet om hvilke tjenester som føres inn i datapakken via protokollene.

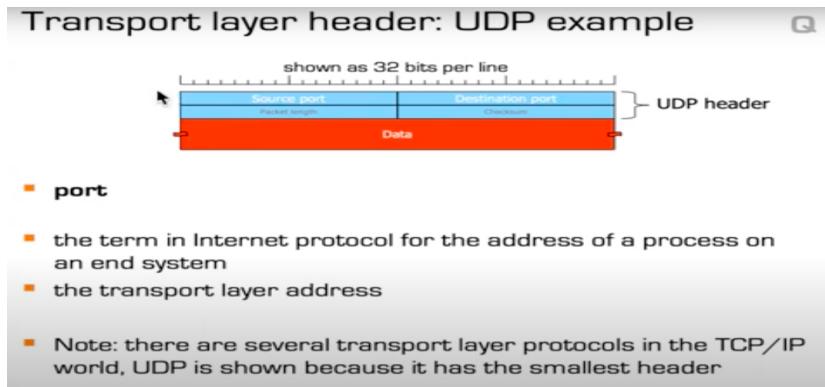


Her ser vi hvordan datapakken sendes fra endesystemet fra venstre og de ulike protokollene som legges opp på datapakken over til intermediate system. Intermediate system må da vite hvilke adresse som datapakken skal overføres til og det er på en måte dette som er viktig å få med seg i forbindelse med data-overføring mellom entiteter i internett. Nemlig at vi har slike protokoller som gir en strukturert og oversiktlig måte å overføre datapakker på.



Når mottaker(server), mottar datapakken så vil de ulike "headerne" pakkes ut, en etter en, som den når applikasjonslaget.

Transport-lagets header



En datapakke vil få en header basert på hva applikasjonen ønsker å sender, enten TCP eller UDP, men hensikten er som sagt å ivareta datapakken i ett av disse headerene (konvolutt). I bildet over, så ser vi et eksempel av UDP-header. I denne headeren så har man plass til 8 bytes, eller 64 bits hvor hver av disse bitene representerer datapakken og noen porter inkludert. UDP er den minste protokollen i bytes som man kan tilby sammenlignet med TCP, som er større i byte-størrelse. Dette kan skyldes av at TCP har tross alt, mange ekstra protokoller som metningskontroll, flytkontroll osv. Se på bitene som når man leser en bok hvor man starter med å lese siden fra øverst til venstre, og slutter på øverst til høyre. Slik er det med datapakken også hvor hver linje blir vist som 32 bit. Men viktig å merke at man setter av 16 bits for hver av portene: "Source-port" og "Destination-port". "Source-port" kan tenkes som lokasjonen fra senderen imens "Destination" er da mottakerens lokasjon altså hvor datapakken skal sendes til. Kan se på porten som en adresse, hvor den adressen skal lagene transportereres til.

Header: Har funksjonen til en konvolutt, hvor protokollen utifra hvilket transportlagprotkoll(TCP eller UDP) som benyttes, vil få den headeren. Under kan du se oversikt over headerne til TCP og UDP.

TCP Segment Header Format						
Bit #	0	7 8	15	16	23 24	31
0		Source Port			Destination Port	
32			Sequence Number			
64			Acknowledgment Number			
96	Data Offset	Res	Flags		Window Size	
128					Urgent Pointer	
160...			Header and Data Checksum			
			Options			

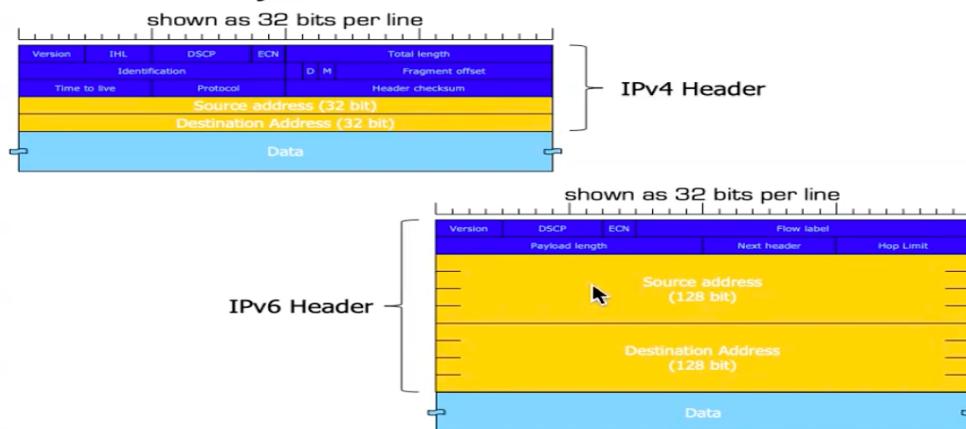
UDP Datagram Header Format						
Bit #	0	7 8	15	16	23 24	31
0		Source Port			Destination Port	
32		Length		Header and Data Checksum		

Man kan se at den har en Source port og destinasjonsport, portene som leder datapakken til det angitte stedet.

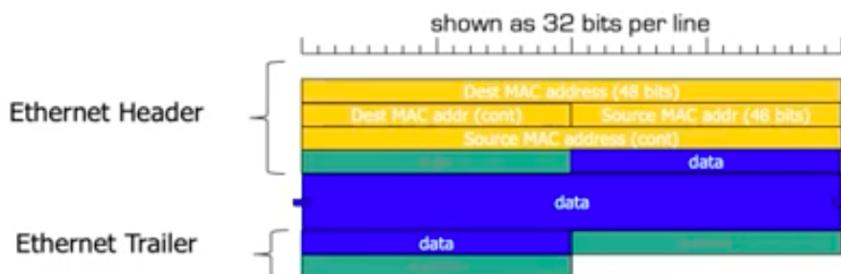
Nettverks-lagets header

Kobler sammen systemene ende-til-ende. Har som hensikt å koble sammen datapakkene med intermediate systemer for å kunne transportere datapakkene gjennom den foretrukne stien valgt av intermediate systemet. Rutene blir brukt siden datapakken skal transporteres 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. IP (Internet protocol, tilkoblingsløst), vil få en såkalt IP-header hvor datapakken fra transportlaget blir lagt i en IP-header. Denne headeren har en IP-adresse som fungerer som en port ved at IP-adressen er unik og gjør det lettere å transportere datapakken til dets destinasjon. Den mest brukte nettverkslagsprotokollen i dag er internet protocol (IP), mest brukte versjonen er IPv4. IPv4 er en 32-bits adresse og problemet ved dette, var at man begynner å gå tom for muligheter å adressere nye IP-adresser siden det er 32-bit. Derfor vil den nye versjonen IPv6 med 128-bit adresser, være den mer foretrukne versjonen siden dette øker muligheten for å lage flere nye adresser. Under vil du se hvordan IP-Headeren ser ut:

Network layer headers: IPv4 and IPv6



Link-lagets header



Pålitelig overføring mellom to direkte nabonoder, hvor disse nodene kan være intermediate-system eller endesystem. Husk "data-flow"-bildet og hvordan datapakken har headeren "address of next node".

Network byte order, Hvorfor bruker vi "Big Endian" nummeret i lagene

2-4(Linklaget, Nettverkslaget, Transportlaget)?

Network byte order(Big endian), er nødvendig etter måten bitmønstrene i protokollene som vi så over(sjekk info om UDB og dette med øverste side av bok). Big endian er den såkalte "network byte order-formatet" som benyttes i lagene 2-4. Skal vi se litt nærmere på dette som er da repetisjon fra IN1020 og binære/heksadesimal-notasjon

Big vs LittleEndian

- Representing numbers
 - the decimal number 36
 - can be seen as $1 \cdot 32 + 0 \cdot 16 + 0 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 0 \cdot 1$
 - is identical to binary 100100
 - we prefer to think in whole bytes, and may write 00100100
 - is identical to hexadecimal $24 = 2 \cdot 0x10 + 4 \cdot 0x1$
 - for clarity we write 0x24
 - it is hard to transform directly from decimal to binary
but easy to transform from hexadecimal to binary
 - 00100100
 - 0010 : 0100
 - $\Leftrightarrow 0 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 : 0 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 0 \cdot 1$
 - $\Leftrightarrow 2 : 4$
 - $\Leftrightarrow 0x24$

sufficient to
think about
4 bits at a time!

Dette er repetisjon for hvordan nummeret blir representert i byteform, hvor man har da ulikt for binærform og heksform.

Hvorfor bruke "Big Endian"?

Argument for Big Endian

■ compatible with western-world writing direction

```
but when we use memory like this:
unsigned char byte[8];
byte[0] = 0x81;
for( int i=1; i<8; i++ ) byte[i] = 0;

unsigned char* ptr1 = (unsigned char*)&byte[0];
printf("%x\n", 1 + *ptr1);
unsigned short* ptr2 = (unsigned short*)&byte[0];
printf("%x\n", 1 + *ptr2);
unsigned int* ptr3 = (unsigned int*)&byte[0];
printf("%x\n", 1 + *ptr3);
unsigned long long* ptr4 = (unsigned long long*)&byte[0];
printf("%llx\n", 1 + *ptr4);
```

81 00 00 00 00 00 00 00
Big Endian
82
8101
81000001
8100000000000001

Hvis man kommer fra den vestlige verden hvor man skrive bokstaver eller tall fra venstre til høyre hvor tallet til venstre har den høyeste verdien. Dette er grunnen til å bruke Big-endian fremfor little-endian.

Hvorfor bruke “Little Endian”?

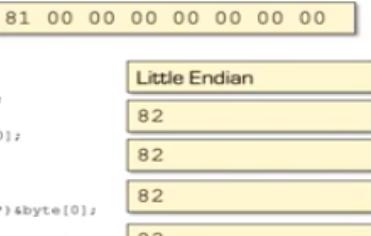
Argument for LittleEndian

- easy to transform

- when we use memory like this:

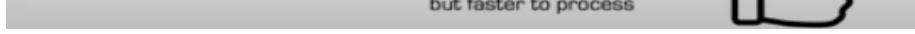
```
unsigned char byte[8];
byte[0] = 0x81;
for( int i=1; i<8; i++ ) byte[i] = 0;

unsigned char* ptr1 = (unsigned char*)&byte[0];
printf("%x\n", 1 + *ptr1);
unsigned short* ptr2 = (unsigned short*)&byte[0];
printf("%x\n", 1 + *ptr2);
unsigned int* ptr3 = (unsigned int*)&byte[0];
printf("%x\n", 1 + *ptr3);
unsigned long long* ptr4 = (unsigned long long*)&byte[0];
printf("%llx\n", 1 + *ptr4);
```



81 00 00 00 00 00 00 00
Little Endian
82
82
82
82

- cheap and easy to change the number of bytes used for an integer value



harder for the human mind
but faster to process

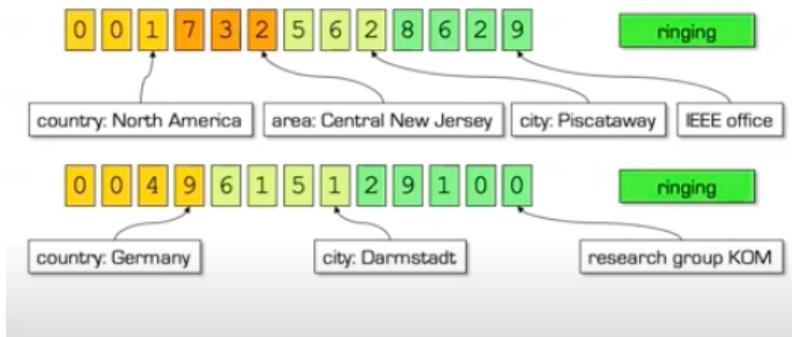
Typecasting i little-endian maskiner er mer effektiv enn ved big-endian maskiner. Er altså billig og enkelt å endre et nummer av bytes brukt for en integer verdi, sammenlignet med Big-endian.

Bonus for Big Endian

- L5 sends bytes to L4
 - L4 passes packets to L3
 - L3 adds a header for routing (and more)
 - L3 passes frame content to L2
 - L2 adds frame header for addressing (and more)
 - L2 passes bits to L1
 - L1 transfers bits
-
- L1 transfer starts at low memory addresses
then continuing to high memory addresses
 - speed matters
 - headers are **in front** to process before all bits have arrived

De laveste adressene vil sendes først fra det fysiske laget “L1” også fortsetter man til de adressene i høyminnet. Dette har en betydning som vi skal se på nå.

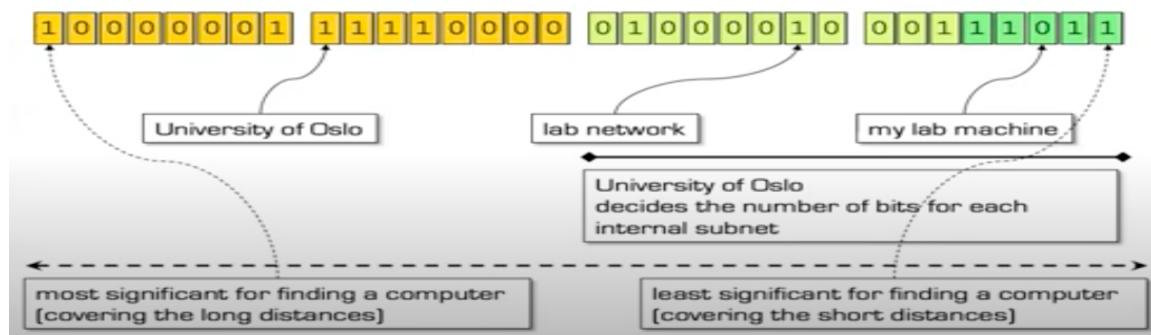
- analogue in telephone numbers



Ved å ha de laveste adressene, også ha de høyere adressene etter disse, så gjør man jobben enda enklere for intermediate system for å koble sammen to ulike entiteter med lang avstand fra hverandre. Slik som i bildet over, så blir det enkelt å oppbevare bytene i den rekkefølgen for å oppnå god hastighet for tiden det tar før sender ringer og mottaker svarer.

Bonus for Big Endian

- my lab machine in our lab network
129.240.66.59
0x81 F0 42 3B



Slik kan vi da forme nettverks-byte-orderen for å oppnå det mest optimale måten. Nemlig å sette de mest signifikante bitene til venstre i forbindelse med å finne endesystemer i lange avstander. Imens de minst signifikante bitene blir lagt til høyre for å finne endesystemer i korte distanser.

Bonus for Big Endian

- my lab machine in our lab network
129.240.66.59
0x81 F0 42 3B

these are **4 bytes**, they are often represented as a **long** in programs

most significant byte least significant byte

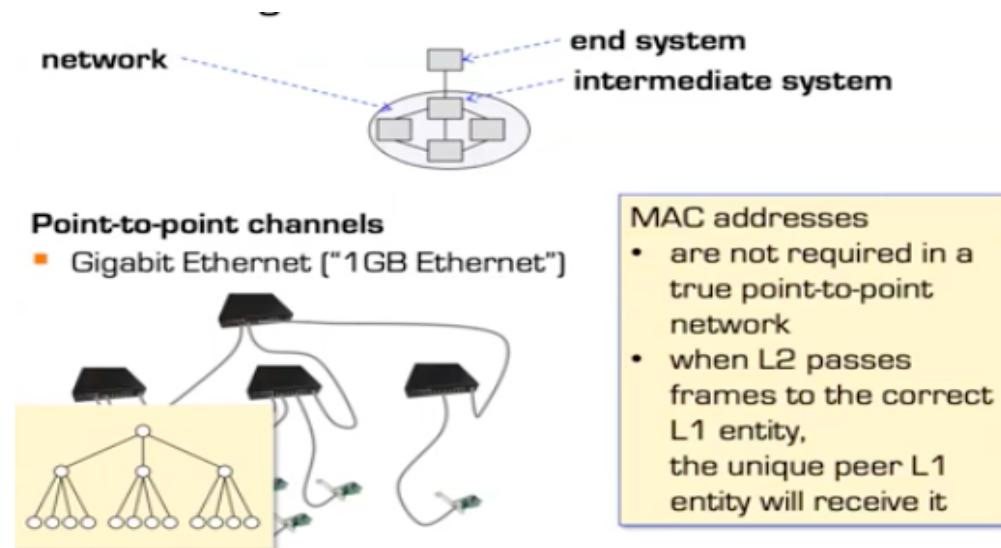
but building this address on a Little Endian machine is dangerous

I little-endian maskiner så kan dette være farlig siden det ikke vil være nok plass å oppbevare slike "byte-formater" med tanke på hvor store de kan bli. F.eks så kan vi ha at 0X81, representeres med en verdi som er større enn little-endian maskiner kan tåle. Dermed så burde man være forsiktig å bygge slike bit-formater i disse type maskiner.

Addressering

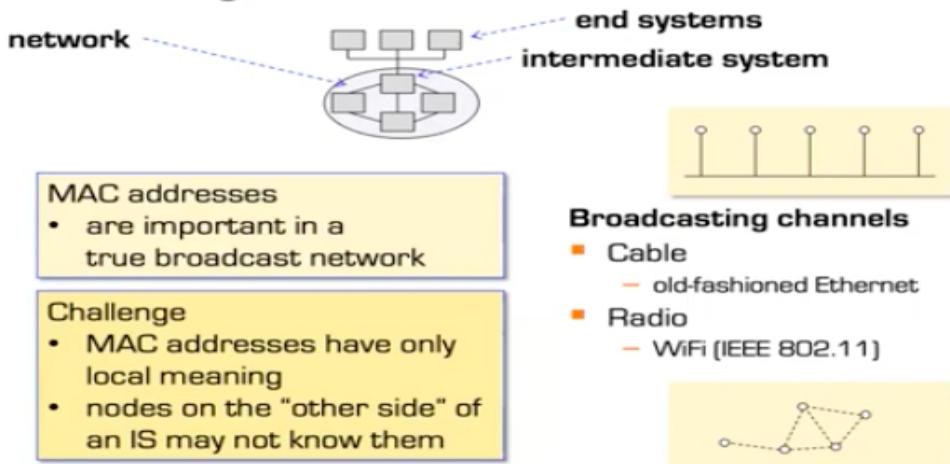
Her skal vi se nærmere på "MAC-adresser", som vil si adressene på linklaget/lag-2 og hvordan denne funker i TCP/IP-laget.

Siden vi jobber med "MAC-adresser" så trenger vi kun å forholde til et lite nettverk av nabonoder siden vi jobber innenfor linklaget. Intermediate systems ved linklaget kobler sammen nabonoder så disse vil da være nærmere i hverandre i forhold til nettverk, som bildet under.



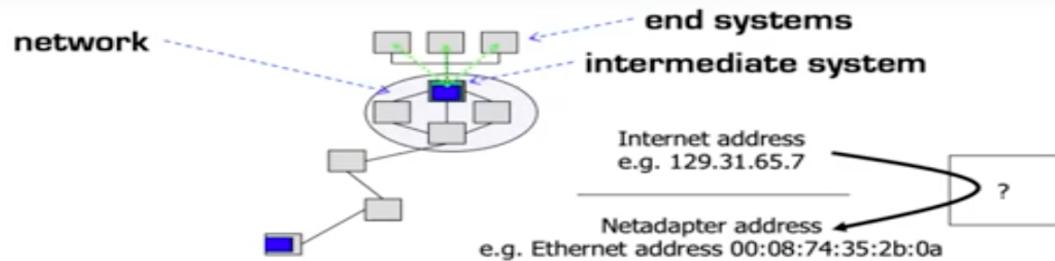
Hvis vi lager et nett i det fysiske laget, som er bare koblet med punkt-til-punkt forbindelse mellom to direkte naboer, slik at vi har en stjerne/tre-topologi, men en node har en nettverkskort som kobler til en mottaker, så har vi en punkt-til-punkt kanal. Da behøver vi nødvendigvis ikke å kjenne adressen til nodene for å finne den mottakeren vi er ute etter, så lenge vi kjenner identiteten til vår egen port, som fører til at vi kan sende datapakker som mottas fra nettverkslaget, direkte til mottaker-naboen. Dvs, om en sender ønsker å sende en datapakke innenfor denne kanalen, så behøver den kun å vite porten som kobler seg opp mot mottakeren slik at hvis senderen må gå igjennom andre noder, så er ikke dette farlig siden vi ønsker å finne den "porten" som kobler seg mot mottakeren. Så ethernet-kablene vil kobles fra nettverkskortet til sender og videre gjennom switch, direkte til mottakeren vi skal sende til siden porten kobler sender og mottaker direkte.

Addressing



I motsetning til en punkt-til-punkt-kanal, så er det i broadcast-kanaler, en “must” å vite mottakeradressen på grunn av følgende: Hvis en sender skal sende en datapakke i et slikt nettverk, så kan potensielt alle høre og derfor kan det være viktig å vite hvem som pakken skal sendes til.

Address resolution



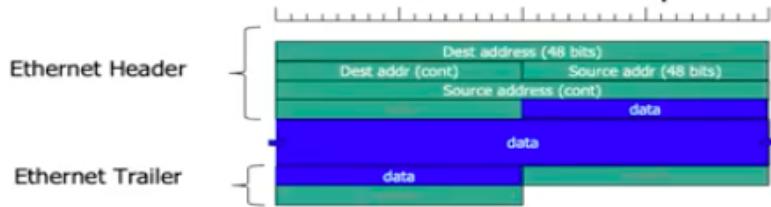
Problem

- Potentially every link can use a different L2 protocol
- | | | | | | | | | |
|------------|-----------------|---------|-----|-----------|---------|-------------|------|----------|
| NRK server | 10GB Eth router | Telenor | DSL | DSL modem | 1GB Eth | WiFi router | WiFi | desk-top |
| ES | IS | | | IS | | IS | | ES |
- Different L2 protocols have different address styles
 - IP address must be mapped onto the MAC address
48 bit for Ethernet and WiFi, DSL may use 20 or 48 bits

Et problem vedvarende nettverkslaget, er å finne den korrekte linklaget-adressen som kobler seg mot mottaker i pakken, raskt. Vi vil ikke at mottaker skal vente skikkelig lenge bare, fordi at nettverkslaget ikke greide å finne den korrekte adressen iks, så dette må vi ha en løsning for. Men dette byr på flere problemer som vi ser i bildet over. F.eks, hvordan skal IP-adressen oversettes til Netadapter adressen. I tillegg, hvordan skal vi håndtere dette med at ulike switchere benytter av forskjellige linklagsprotokoller. F.eks er NRK-serveren sin protokoll, ulikt fra Telenor sin ettersom NRK bruker en server imens Telenor bruker en ruter. Dette vil på forskjellige addreseringstiler som enhver switcher må kunne håndtere, hvor

da telenor må håndtere den forskjellige adresseringen den får fra NRK. Det siste punktet beskriver at IP-adressen må være mappet oppå mac-adressen siden de ulike switcherene kan oppbevare forskjellig mengde bit. Derfor, må disse switcherne lagre mulige bits for hvordan IP-adressen kan være og velger da den mest hensiktsmessige "bits-antallet" i henhold til IP-adressen som ble valgt.

Address resolution: Ethernet example



- MAC address structure
 - Ethernet and WiFi are L2 layers using "EUI-48": Extended Unique Identifier with 48 bits
 - 6 bytes, written like this: f2 : 18 : 98 : 3a : b8 : 97
 - to recognize easily that the text is supposed to mean a MAC address
- Ethernet MAC addresses should be globally unique

Dette er bare viktig forhåndsinformasjon til å vite hvordan vi skal håndtere nettverksproblemene med MAC-adresser som vi så i forrige avsnitt. Ved å formtere adressen på denne måten, så skiller dette fra IP-adresse og gjør det enklere å identifisere hva en MAC-adresse er. Altså måten adressen er skrevet over, er måten MAC-adresser blir representert.

- IANA and IEEE decide how to split the address space

- first 3 bytes explain whether an address is special

OR

- first 3 bytes determine who owns the address range

e.g.:

- F0:18:98 : Apple, Inc.
 - 78:45:C4: Dell Inc.
 - 00:50:56: VMWare, Inc.
 - B8:AC:6F: Dell Inc.

IANA - Internet Assigned Number Authority
IEEE - Institute of Electrical and Electronics
Engineers

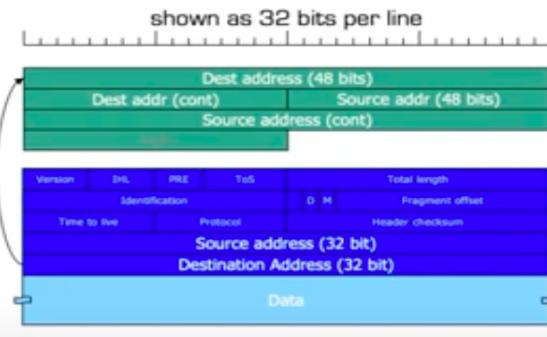
IANA og IEEE har delt opp de 48-bitene i MAC-adressen slik at de 3 første byteene enten er en "spesiell" adresse eller hvem som eier adresseområdet. Er kun nyttig å vite at dette finnes, at man kan dele opp bitene til å tilpasse det man ønsker etter sitt formål.

1st idea: direct mapping

the 32 bit destination IP address would fit into the 48 bit destination MAC address

but:

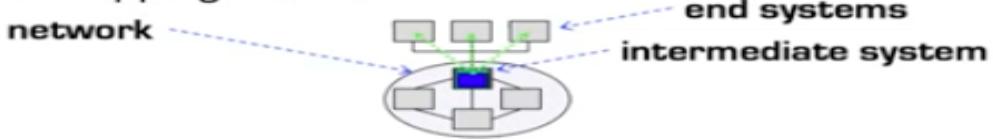
- there is a new MAC address for every pair of direct neighbours
 - need to re-write the destination IP address on every IS
 - but IP addresses are globally unique
- ⇒ does not work for the Internet



En mulig løsning for å finne den korrekte mottager-adressen, er å kopiere IP-adressen inn i MAC-adressen, såkalt "direct mapping". Merk at IP-adressen er av versjon IPv4 så den har 32-bit, og vil da få plass i en MAC-adresse med 48-bit. Da vil man trenge en mellomnode som tar imot ip-pakker ved å sette den lokale prefiksen til mottakers lokale nett slik at pakken sendes direkte til mottaker. Dette vil muligens funke i nettverket som er knyttet til mottakeren, men dette vil ikke funke i internettet siden internett består av mange nettverk. IP-pakken vil inneholde mottakeradressen i lag-3, altså adressen til den mottakeren som vi skal sende datapakken til. Men det er utrolig mange intermediate systemer som muligens må gås igjennom for å finne den "stien" som fører til mottaker. Hvor skal vi oppbevare disse adressene, for hvis vi ikke gjør dette så vil vi aldri nå mottaker. Løsningene som å skrive om på destinasjonen til IP-adresser, er umulig å gjøre så derfor er ikke denne ideen mulig å benytte.

Address Resolution

2nd idea: mapping table



every node maintains a table that maps
IP address ↔ MAC address
for every network interface and
for every directly reachable node [L2 neighbour]

idea 2.1: manually maintained by people

- a lot of work, but not unrealistic – IFI allows only well-known MAC addresses in well-known network plugs – **could be used for this but is not**

idea 2.2: established by broadcasts from stations

En annen ide er å lage en oppslagstabell. Dvs at hver mellomnode har en tabell for alle

IP-adresser til alle endesystemer som den kan nå direkte. Også kan denne tabellen oppdateres etter behov også slår man opp hvilken MAC-adresse som passer for den gitte IP-adressen som er ute etter den spesifikke endesystemet. Vi har da to ulike varianter av å lage denne tabellen, hvor man kan manuelt lage den, men er ikke brukt så ofte. Den andre varianten er at oppslagstabellene lages når man kobler opp mot nettverkene gjennom wifi. Den tredje varianten er den beste varianten av alle 3 som vi skal se på nå.

node with a packet to deliver:

```

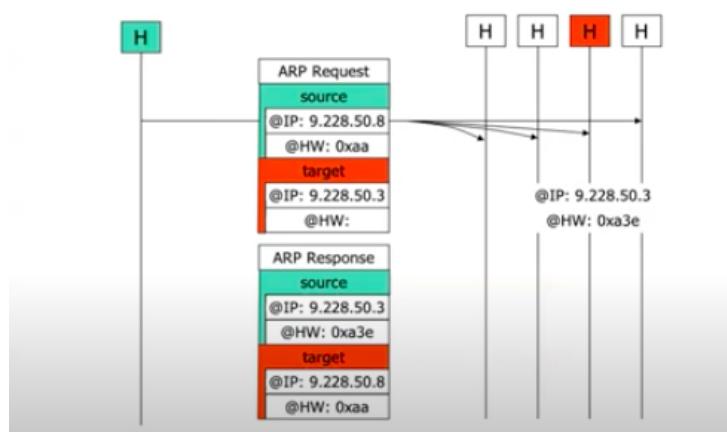
if a local cache contains IP address ↔ MAC address
    send packet & update cache removal timeout
else
    send broadcast to all stations
    "Who has IP address?"
    if one node responds
        add IP address ↔ MAC address mapping to cache
        set timeout for removal from cache to some minutes
        send packet
    else
        drop packet

```

Denne varianten er en oppslagsvariant hvor hver eneste maskin i et lokalt nettverk vil vedlikeholde en cache som består av alle direkte naboer. For alle disse direkte naboene så vil kantene representere IP-adressen til MAC-adressen for den samme maskinen. Hvis vi ikke finner den tilsvarende MAC-adressen for den IP-adressen, altså mellomnoder vi ønsker å koble oss mot for å komme nærmere til mottaker, så kobler man opp mot alle stasjoner og spør "hvem er det som er knyttet med meg(IP-adresse(Sender))". Hvis en node responderer og finner ut at "det er jeg som er knyttet med deg(IP-adresse(Mellomnode))", så vil pakken overføres mellom disse to.

ARP - Address Resolution Protocol (ARP)

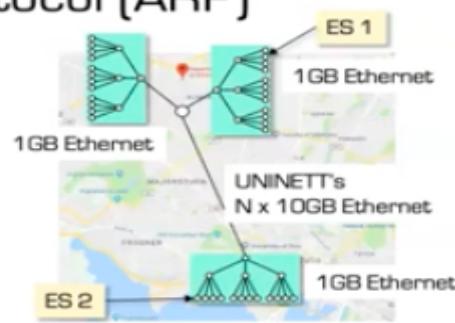
Address Resolution Protocol [ARP]



Nettverkskortene har en 6 byte lang media access control (MAC)-adresse som brukes til å identifisere maskinen innenfor et kringkastingsdomene. For at IP skal fungere, må avsenderen vite hvilken MAC-adresse pakken skal sendes til. ARP(Address Resolution Protocol) kobler IP(Internett) og MAC(Linklaget). Fungerer på samme måte som DHCP, bortsett fra at det er MAC-adresse som benyttes og at neste gang en enhet skal koble til ARP, vil tilkoblingen lagres i ARP-Cache. I bildet over så har vi da en “ARP-request” som sender en forespørsel i domenet om noen har “target”-IP-adressen. Da fant man den “røde-hosten” som har den “target”-adressen slik at datapakker vil da kun overføres mellom de to. De andre “hostene” vil ikke være i stand til å finne informasjon om denne pakken, kun hosten med fargen med “rød” siden den hadde “target”-adressen.

Address Resolution Protocol (ARP)

- End system not directly available by broadcast
- Example: ES 1 to ES 2
 - ARP would not receive a response
 - Ethernet broadcast is not rerouted over a router
- L2 solution: proxy ARP
 - the local router knows all remote networks with their respective routers
 - responds to local ARP
 - local ES 1 sends data for ES 2 always to the local router, this router forwards the data [by interpreting the IP address contained in the data]
- L3 solution: remote network address is known
 - local ES 1 sends data to the appropriate remote router
 - local router forwards packets

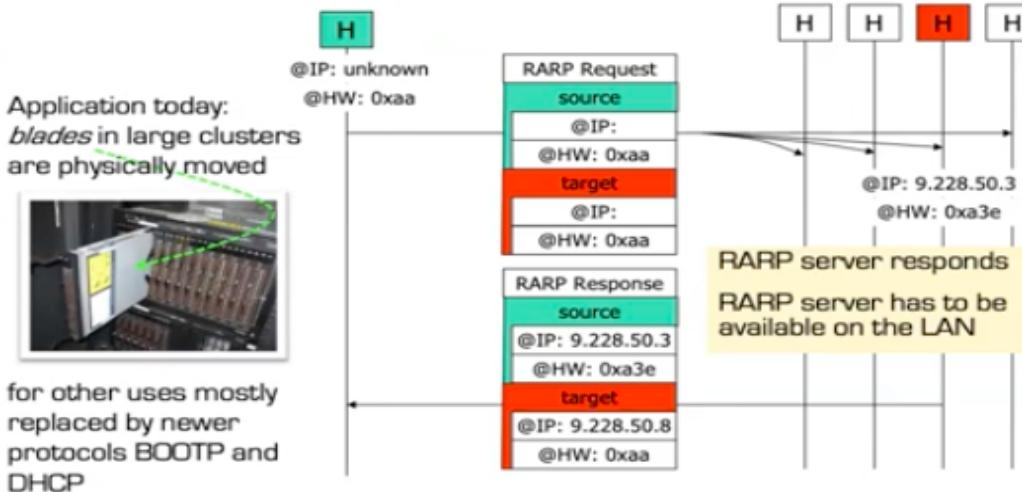


Et problem ved ARP, er om vi skal overføre datapakker mellom to noder siden den andre noden ikke er tilgjengelig gjennom broadcast. En mulig løsning er derfor å installere en “proxy-ARP” som er en lokal ruter med en stor cache som kjenner til alle nettverkene som kobles til ruteren. Denne ruteren vil da håndtere ARP-forespørslene som sendes ut til broadcast hvor den da sjekker om “target” finnes i cachen til ruteren. Hvis den finnes i ruteren, så vil denne ruteren håndtere det slik at man slipper å måtte utføre prosessen på nytt igjen, men man henter tilkoblingen mellom de to nodene fra “proxy-ARP” istedenfor. “Proxy-ARP’en” vil da være den eneste “hvite” noden som er i kartet for seg selv.

RARP- Reverse Address Resolution Protocol

Reverse Address Resolution Protocol (RARP)

Retrieve Internet address from knowledge of hardware address

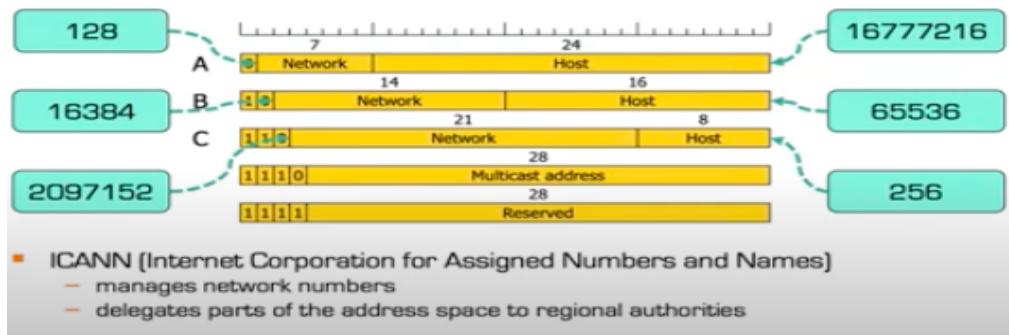


Er en protokoll som brukes til å finne en nodes egen IP-adresse. Dette gjøres ved å koble sende forespørsel til en broadcast-server hvor man spør "Hvem er det som er koblet seg til meg". Den røde-hosten som er markert, sier da "det er meg, og din IP-adresse er da xxxx".

IP adresser i TCP/IP modellen

Internet Addresses and Internet Subnetworks

- Original global addressing concept for the Internet
 - For addressing end systems and intermediate systems
 - each network interface (not ES) has its own unique address
 - 5 classes



Originalt så tenkte man at IPV4(32-bits adresser) skulle holde for hele verden utover i god tid. Men vi vet at vi begynner å gå tom for IP-adresser og derfor kommer IPV6 som vil inneholde flere bits. Men dette er ikke poenget ved denne delen, denne delen er ment til å forklare hvordan IP-adressene er satt opp.

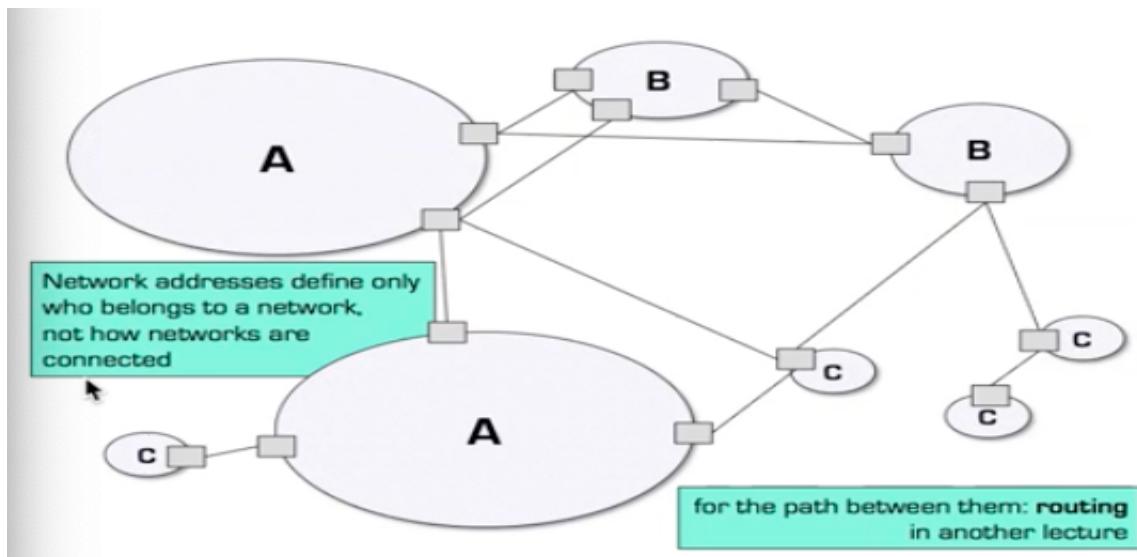
Vi har da 2^{32} adresser som er da fordelt i klassene over i bildet. Vi har de vanlige nettverkene som er delt opp i klassene "A,B og C". Videre har vi multicast adresser som er adresser som mange datamaskiner kan dele ved forhandling av datapakker. Altså hvis en

klient sender en datapakket til en slik adresse, så vil det alle andre endesystemer som benytter av denne adressen, være i stand til å motta datapakken. Denne type adresser ansees som et problem med tanke på sikkerheten og derfor blir ikke disse benyttes ofte. Tilslutt har vi reserverte adresser. De viktigste vi skal se på, er av klassene "A, B og C".

Klasse "A" vil ha en "0"-bit i første posisjon, så 7-bits for å identifisere nettverket og klarer man å finne nettverksinngangen gjennom et raskt oppslag også har man mange "hosts" som er da mange adresser tildelt for endesystemer. Hvis vi ser i bildet over, så har vi ved klassen at man kan ha 16 millioner adresser og 128 nettverk.

Klasse "B" nettverk bruker flere bits til å beskrive nettverk, men har færre "hosts" altså endesystemer som kan kobles innen dette nettverket. Hvis vi ser i bildet over, så har vi ved klassen at man kan ha 65 tusen adresser og 16 tusen nettverk.

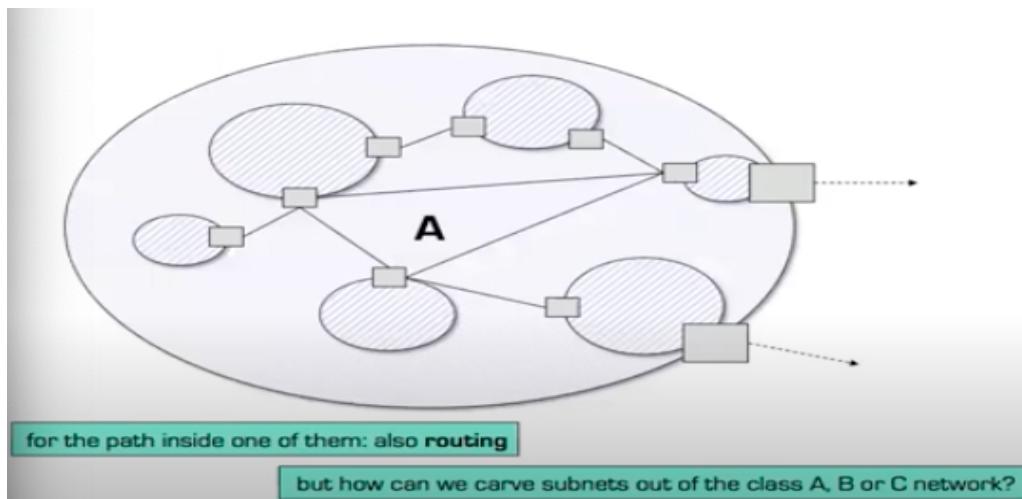
Klasse "C" nettverk har mange bits for å beskrive nettverk, men mye færre "hosts". Hvis vi ser i bildet over, så har vi ved klassen at man kan ha 20 millioner nettverk og 256 adresser.



Her ser vi hvordan nettverkene blir representert av de ulike klassene. Nettverkadressene definerer bare hvem som tilhører et gitt nettverk og undergrupperer basert på IP-adressene i en måte slik at man raskt og effektivt kan slå opp ved å se på første bit og prefiksen, hvem som eier denne IP-adressen i disse nettverkene og benytter av "routing" som vi skal se på senere. Men vi ser at man trenger en form for oppslagsverk slik at nettverk av type "C" skal kunne nå et endesystem i klasse "A".

- Networks grow and should be somehow structured
 - several networks instead of one preferable
 - but getting several address areas is hard
 - since address space is limited
 - e.g., university may have started with class B address, doesn't get second one
 - Problem
 - class A, B, C refer to one network, not collection of LANs
- ⇒ Allow a network to be split into several parts
- for internal use
 - still looks like single network to outside world

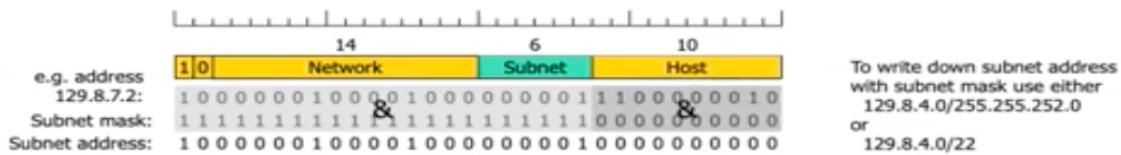
Hvis du lurer på hva dette bildet har å si for nettverket, så er dette knyttet til de klassene som vi så tidligere(A, B og C). For nettverk i noen steder vil jo benytte av ett av disse klassene og siden nettverket vokser og blir større, så vil det være begrenset hvor mye disse klassene kan tilordne. Man vil altså nå antallet adresser eller nettverk i de ulike klassene. F.eks som det står i eksempelet med universitetet, så kan universitetet ha startet med et klasse B. Her så har man 16 tusen nettverk og antallet man starter ved universitetet, vil eventuelt vokse. La oss anta at antall hosts var 10 tusen og universitetet har 11 tusen studenter. Hvis universitetet begynner å vokse og har da 11 tusen studenter, så vil det være noen som ikke vil bli adressert på grunn av den begrensede antall av hosts. Derfor, burde man strukturere nettverket ved universitetet slik at man ikke kun har et klasse B nettverk, men en samling av nettverk som består av de ulike nettverkene sammen.



Løsning blir å dele opp nettverket i flere deler hvor de store nettverkene som klasse A, kan brukes for “internt use” altså å koble sammen enkelte små nettverk sammen gjennom routing. Det kan se ut som et “single network”, men i realiteten så er det faktisk et “single network” satt sammen av mange andre nettverk som vi ser over. Problemet er nå, hvordan skal man skille subnett ut av slike fordeling, altså de små nettverkene inne i nettverk “A”? Dette skal vi se på nå.

- Idea

- local decision for subdividing host share into subnetwork portion and end system portion



- Use "subnet mask" to distinguish network and subnet part from host part
- Computers inside the network 129.8.4.0/22 can make 3 checks
 - Algorithm in router [by masking bits: AND between address and subnet mask]:
 - packet to another network [& with Network mask are different]
 - packet to local end system [& with Network and Subnet masks are the same]
 - packet to other subnetwork [& with Network mask is the same, but Subnet is not]

Dette er kanskje kjent fra IN1020 om dette med nettverksmasker. Ved å inndele IP-adressen på følgende måte, så får vi muligheten til å adressere subnett også. Subnett er bare et lite nettverk av intermediate og endesystemer. Så dermed vil vi få muligheten å adressere et annet subnett som hjelper med ruting.

CIDR: Classless InterDomain Routing

- Subnetting not good enough
 - Too many organizations require addresses
 - in principle many addresses due to 32-bit address space
 - but inefficient allocation due to class-based organization
 - class A network with 16 million addresses too big for most cases
 - class C network with 256 addresses is too small
 - most organizations are interested in class B network, but there are only 16384 (in reality, class B too large for many organizations)

⇒ Introduction of CIDR (Classless InterDomain Routing) (RFC1519)

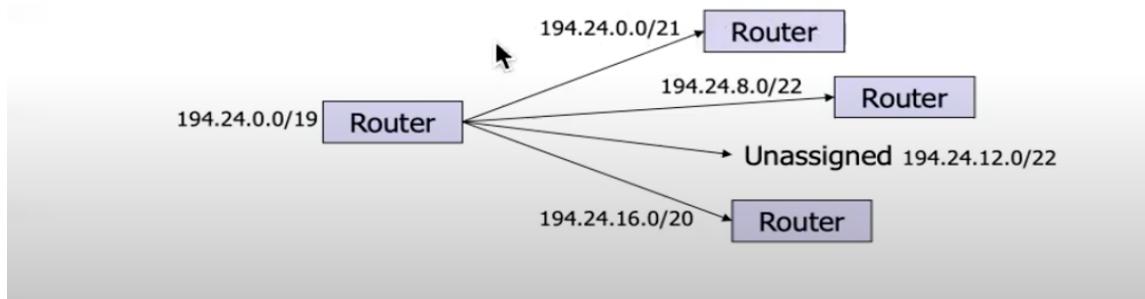
- CIDR Principle
 - to allocate IP addresses in variable-sized blocks
 - [without regard to classes]
 - e.g., a request for 2000 addresses would lead to
 - assignment of 2048 address block starting on 2048 byte boundary
- but, dropping classes makes forwarding more complicated
- Large number of networks leads to large routing tables

Selvom løsningen i forrige avsnitt var en god løsning, så er ikke dette godt nok. Grunnen er at det store nettverket har fremdeles en eier, nemlig en klasse "A" f.eks. Så står det informasjon om at det blir ineffektiv allokering gjennom den klasse-baserte versjonen. Klasse "A" er uegnet siden det blir for stor i de fleste tilfeller for organisasjoner. Klasse "C" er det motsatte hvor det er for lite. Klasse "B" egner seg derfor mest av disse klassene.

CIDR prinsippet vil hjelpe oss med å løse blokkallokering av IP-adresser ved å alllokere IP-adresser i variabel-størrelsers blokker. Slik som det står så vil en forespørsel på 2000 adresser føre til at man allokerer 2048 adresser i blokkene.

Men ved å droppe klasse-prinsipper så vil det føre til at ruting innen et nettverk, vil muligens bli komplisert. Store nummeret av nettverk vil kreve store ruting-tabeller siden man ikke er fast begrenset til klasse-tildelingen av nettverk, subnet og hosts som vi har sett tidligere. Men denne løsningen er fremdeles god med tanke på fleksibilitet og for å oppfylle forespørsler fra klienter i nettverk.

- Search for longest matching prefix
 - if several entries with different subnet mask length may match
 - then use the one with the **longest mask**
 - i.e., AND operation for address & mask must be done for each table entry
- Entries may be aggregated to reduce routing tables



Når man skal rute en datapakke gjennom ulike subnett som i bildet over, så skal man bruke den subnet-masken som er lengst av dem alle. Dette vil sørge for at man vil hente hosten som oppfylles av nettverket som sendes av ruteren til venstre i bildet.