Protokol: Et sprog som forskellige computere bruger, for at kende "spillereglerne" når de skal agere med hinanden. De bruger begge samme protokol, så de kan snakke sammen. Findes flere.

Alt kommunikation over internettet foregår med protokoller.

Network edge: De ting der befinder sig i begge ender af netværket, nemlig applikationer og hosts.¨

Netword core: Midten af netværket. Routere der er sluttet sammen. Et netværk af netværk.

Client/Server model: Klient spørger søger en service hos en server som altid er tændt.

Peer-Peer model: Minimal brug af dedikerede servers. Er en computer til en anden.

Circuit-switching: Sikrer en lovet forbindelse mellem to parter, men kræver at man "caller" først.

Forbindelsen bliver først lukket når begge er done.

FDM og TDM: Frequency-division multiplexing og time-division multiplexing er noget der finder sted i et circuit-switched netværk. Det fortæller hvor båndbredden skal fordeles mellem brugerne på linien. FDM = lidt til alle hele tiden. TDM = alt til en af gangen.

Packet-switching: Sender pakken hurtigere afsted, men hvis pakken kommer igennem et congested netværk, så kan den risikere at skulle vente i en buffer, før den sendes videre.

Internet nu om dage er packet-switched.

Packet switching er godt, da det lader flere brugere til på én gang, da det ikke er alle der har fået dedikeret forbindelsen i circuitswitching der bruger den hele tiden.

Store and forward: Hele pakken skal ankomme til routeren, før den sender den videre til næste link.

Hierarkisk ISP system: Nogle i midten der har hele netværket. (Tier 1) Mindre ISP'er der køber sig adgang til T1 netværket. (Tier 2) lavere og lavere endnu tiers der betaler sig ind.

En pakke går ofte igennem mange af lagene.

4 slags delay: Nodal processing: Den del hvor pakken skal igennem routeren.

Queueing: Tid pakken venter ved output. Er afhængigt af congestion.

Transmission delay: Den mængde tid det tager at skubbe hele pakken ind i output linket, når der er plads

Propagation delay: Den tid det tager at sende den fysiske afstand til den næste router.

Buffer: En router har en buffer der samler pakker før de bliver sendt. Hvis bufferen er fyldt, så bliver pakker smidt væk.

TCP/IP stack model: Application -> Transport -> Network -> Link -> Physical

Message -> Segment -> Datagram-> Frame -> Bit

Application architectures: Client-server, Peer-to-Peer(P2P), Hybrid af de to.

Client-server: Host: Always-on host. Permanent IP-adresse på host.

Client: Kommunikerer med server, er kun connected nogle gange. Kan have dynamisk IP.

Pure P2P: Ingen always-on host. Endesystemerne kommunikerer direkte. Peers er connected nogle gange, og kan begge skifte IP.

Hybrid: Skype: Er i teorien P2P, men kontakter også servere og har mellemmænd.

IM: Chatte igennem to brugere, snakke med server for at finde IP på peers.

Processes: Der kører processer på end-systems. Disse processer kan snakke sammen med andre processer på andre end-systems vha. Messages. En sende-proces sender messages via application-lage til andre modtage-processer.

Der er klient og host processer, som kører i par. En for hver. Så ens proces i f.eks. en browser, snakker sammen med en web-server proces.

I P2P der bliver filer sendt fra en klient-proces til en anden. Men den der uploader er sat som server-proces.

Socket: Messages bliver sendt igennem sockets, som er døre.

Addressing processes: For at processer kan modtage beskeder, så er det nødvendigt at de har en identifier.

Hosten har en 32-bit IP adresse og et port nummer associeret med den proces.

HTTP: Hjemmeside består af objekter. (HTML filer, billeder, java applet, lyd osv) Har en HTML-side med referencer til de forskellige objekter. Hvert objekt har en URL.

Kører klient/server model.

Persistant og non-persistant HTTP: Persistant tillader flere objekter at blive sendt over én forbindelse, i modsætning til non-persistant.

Non-persistant eksempel: Brugere laver forbindelse til serverens Socket på port 80, serveren accepterer, klienten sender en request message med URL'en på det objekt den vil have, serveren modtager request og laver en response message med objektet i, serveren lukker forbindelsen, klienten modtager beskeden og lukker selv.

RTT er en for at starte forbindelsen med serveren, og en til request, plus transmit time.

2RTT + Transmit time per objekt

Persistnt eksempel:

Serveren lader forbindelsen være åben efter den sender response tilbage, og klienten sender request lige så snart den har html filen. Kun RTT + transmit time per objekt.

Typer af HTTP messages:

Request: Er lavet af almindelig ASCII.

Først command'en i message **GET /somedir/page.html HTTP/1.1**

**Host: www.someschool.edu**

**Så header lines User-agent: Mozilla/4.0**

**Connection: close**

**Accept-language:fr**

Til sidst en tom linie der fortæller at message'en slutter.

Typer: GET, POST, HEAD

PUT: Uploader til path specificeret

DELETE: Sletter fil på path specificeret

Reponse:

**Status linje, har status kode HTTP/1.1 200 OK**

**Connection close**

**Header lines Date: Thu, 06 Aug 1998 12:00:15 GMT**

**Server: Apache/1.3.0 (Unix)**

**Last-Modified: Mon, 22 Jun 1998 …...**

**Content-Length: 6821**

**Content-Type: text/html**

**Data som blev requested Data**

HTTP Response codes (et par eks): **200 OK**

request succeeded, requested object later in this message

**301 Moved Permanently**

requested object moved, new location specified later in this message (Location:)

**400 Bad Request**

request message not understood by server

**404 Not Found**

requested document not found on this server

**505 HTTP Version Not Supported**

Conditional GET: Send ikke fil hvis den ikke er blevet ændre siden jeg sidst har været her.

HTTP request **If-modified-since: <date>**

**HTTP/1.0 304 Not Modified eller** HTTP response **HTTP/1.0 200 OK**

**<data>**

User-server state: Cookies:

HTTP servers er jo stateless, de gemmer altså ikke noget om deres brugere, hvilket er

årsagen til at de kan behandler så mange TCP connections på én gang. Men det kan være brugbart for en webside at identificere deres brugere, både til reklamer, men også bruger- venlighed.

Består af 4 dele: Cookie header line af HTTP response message

Cookie header line af HTTP request message.

En fil på userens end system, som bliver håndteret af serveren.

En backend database på serveren.

Men går ind på en hjemmeside første gang, så opretter den et ID, og for at huske det ID så gemmer den det i en Cookie. Næste gang man skal ind på hjemmesiden, så leder den efter cookien i stedet og sender det ID der står i den, så hosten ikke skal lave endnu et ID. På den måde ved den hvem man er. Ens bruger er gemt i en database på host end-system.

Web Cache Proxy servers: En måde at sende og hente information igennem en 3. part server, så end- system ikke bliver gjort til kende. En slags mellemmand. En cache kan have en masse af objekterne fra en server liggende, en bruger kan så bede dem om serverens objekter, og hvis cachen ikke har dem, så beder den serveren om selv at få dem, og så sender den dem vider klienten.

Cache er både en server og en klient.

Hvorfor bruge Cache? Reducerer reponsetime for klient request.

Reducerer traffik på serverens links, da det kun er cachen der skal spørge.

Giver servers med dårlige forbindelser mulighed for at sende data til brugeren med høj hastighed, hvis cachen allerede har filen, så sender den, den.

FTP: File-Transfer-Protocol. Overfører fil til/fra remote host.

Client/Server model. Port 21. Klient kontakter FTP server via TCP transport protocol. Klient autoriserers over port 21 connection. (Control connection) Klient browser hosts delte drev.

Når klienten så skal modtage en fil, så åbnes der endnu en TCP forbindelse på port 20. Efter fil er overført, så lukkes port 20, men 21 er stadig åben. Åbner og lukker port 21 for hver fil.

Control connection er "out-of-band" hvilket er fordi den sender ting på en anden seperat forbindelse.

Alle beskeder er via ASCII.

FTP commands (eks): USER username

PASSWORD pass

LIST viser alle filer i directory

RETR filename retriever fil

STOR filename gemmer fil i dir

FTP return codes(eks): Statuskoder ligesom med HTTP.

**331 Username OK, password required**

**125 data connection already open; transfer starting**

**425 Can’t open data connection**

**452 Error writing file**

Mail: 3 store komponenter: User agents, Mail servers, Simple mail transfer protocol (SMTP)

User agent: Den der læser mailen. Opretter, skriver, redigerer, læser mail beskeder. F.eks. outlook.

Udgående og indgående beskeder bliver gemt på en server.

Mail servers: Har en mailbox der indeholder indkommende mails. Har en kø af udgående beskeder der venter på at blive sendt. Har en SMTP protocol til at snakke med andre mail-servers.

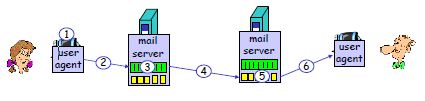
Client/Server model.

SMTP: Bruger TCP til at overføre mails fra klient til server via port 25.

3 faser af overførslen: Handshaking, Transfer, Closure.

Commands er i ASCII og responses er i status koder og phrases.

Message er i 7-Bit ASCII.



HTTP = Pull - SMTP = Push

Mail format: En mail har en header og en body. Header har: Til, fra, subject, og forskellige SMTP commands. Body har: Selve mailbeskeden i ASCII.

Mail Access protocols:

SMTP sender til modtagers server.

Mail access protocol: Henter fra server

POP: Post Office Protocol - Sørger for authorization.

IMAP: Internet Mail Access Protocol - Flere features, manipulerer gemte beskeder på servere.

POP3: Authorization phase:

**S: +OK POP3 server ready**

**C: user bob**

**S: +OK**

**C: pass hungry**

**S: +OK user successfully logged on**

Transaction phase:

list: Viser message numre, retr henter message med givent nummer, dele: delete, quit

**C: list**

**S: 1 498**

**S: 2 912**

**S: .**

**C: retr 1**

**S: <message 1 contents>**

**S: .**

**C: dele 1**

**C: retr 2**

**S: <message 1 contents>**

**S: .**

**C: dele 2**

**C: quit**

**S: +OK POP3 server signing off**

POP3 er "download and delete", så den sletter fil fra server når den er hentet.

IMAP: Holder alle filer på serveren. Tillader at brugeren arrangerer beskederne i mapper.

DNS: Domain Name System. En person har flere forskellige måder at blive identificeret på, nemlig navn/cpr-nummer f.eks. Men det er ikke altid at man kan bruge det ene, nogle gange er det bedre at bruge det ene over det andet.

En host har også identifiers, nemlig et Hostname. (f.eks. www.google.com) Disser er designet til at de skal kunne huskes og skrives, så de er simple og giver mening. Men disse fortæller ikke hvor serveren befinder sig. Derfor har de også en IP-adresse.

En ip-adresse er hierarkisk, dsv. hvis man læser den fra venstre mod højre, så bliver den mere og mere specifik, ligesom hvis man læser en normal adresse fra bunden og op.

DNS laver hostnames om til IP-adresser. Det er en distribueret database implementeret i et hierarki af DNS servers og en application-lags protocol der tillader hosts at spørge de distribueret database. De kan bliver queried af mange forskellige slags protocoller, f.eks. HTTP, SMTP og FTP, at oversætte et hostname til en IP-adresse.

Hvorfor har man lavet den distribueret? Fordi ellers ville der være single-point of failure, og DNS serveren styrer hele internettet. Fordi den ville bliver overbelastet hvis der kun var en. Fordi hvis den ligger langt væk, så vil nogle brugere skulle meget langt for at spørge den.

Den ville skulle blive opdateret meget ofte, da internettet vokser hele tiden.

Når man gå ind på en hjemmeside, så tager browseren hostnames, giver det til klientens DNS-klient, som spørger en DNS server, og får så til sidst et svar med en IP adresse.

Host aliasing: Da et hostname har mange forskellige hostnames, har man givet dem et sandt hostname.

DNS kan bliver spurgt om at hente det sande hostname udover bare IP adressen.

Mail Server aliasing: Det er at foretrække at e-mail adresser er simple, da de skal huskes af folk. F.eks. en person der har en mail hos yahoo kan være så simpelt at den hedder something@yahoo.com

Men dette hostname, yahoo.com har også et sandt hostname, som han blive hentet af en

DNS klient.

Load distribution: DNS sørger også for at fordele trafik på kopierede servere. Store sider som f.eks. google, har deres server kopieret i flere servers, med forskellige IP-adresser. Hvis der bliver spurgt efter IP-adressen på et hostname med mange af den samme servers, så vil de få en liste af IP-adresser tilbage.

Da HTTP ofte sender til den første på listen, så roterer DNS serveren disse IP-adresser rundt, fra gang til anden, så brugerne bliver fordelt jævnt på alle servers.

DNS hierarki: 3 lag: Root DNS server, Top Level Domain (TLD) DNS servers til hver slags domæne: (com, org, dk, osv.), og så autoritative DNS servers.

Først spørges root DNS server, der giver et hint om hvor man skal hen.

Derefter spørges TLD dns server, som giver et mere specifikt hint.

Derfter spørges autoritativ DNS server, som giver svaret.

Alle hosts med offentligt-tilgængelige webservers, skal have offentligt-tilgængelige DNS oplystninger. De kan enten hoste deres egne, eller betale sig fra det.

Local Name Server: Hører ikke under tidligere nævnte hierarki. Hver ISP (ISP, company og universistet/større ting) har en. Hedder også Default Name Server.

Når host laver en DNS query så bliver Default Name Server spurgt, som sender query videre til resten af hierarkiet.

Default name server cacher name servers, så hvis de bliver spurgt om igen, så kender den allerede svaret. Sparer rigtig meget tid og traffic. Et hostname bliver timed out efter ikke at have været blevet queried i et stykke tid, for ikke at løbe tør for plads.

TLD servers er typisk cached i default name servers, så den kender dem allerede. Det er derfor ikke altid nødvendigt at spørge root servers, om TLD servers.

DNS records: Resource Records format: Navn, Value, Type, Time To Live.

DNS record typer: Type=A: Name er hostname, Value er IP-adresse

Type=NS: Name er domæne (f.eks. foo.com), Value er hostname af den autoritative name server for dette domæne.

Type=CNAME: Name er et alias for et sandt hostname, value er det sande hostname.

Type=MX: Value er navnet på mailserver associeret med Name.

DNS protocol messages: Query og reply messages, som begge har samme format. 

MSG header: Identification: 16-bit field er et slags identifikationsnummer så de ved hvad de svarer eller modtager svar på.

Flags: Query eller reply, recursion desired, recursion avaible, reply is authoritative.

Questions: Har information om det Query der er ved at blive lavet.

Indeholder hostnamet, hvilken type det er, altså hvilket spørgsmål det er.

Answers: Indeholder de Resource Records der blev efterspurgt. Et svar kan have flere RR's, da et hostname jo kan have flere IP-adresser.

Authority: Indeholder records af andre authoritative servers.

Additional information: Kan f.eks. indeholde f.eks. ip adresse hvis der alligevel bliver spurgt om navnet på en mailserver. Sparer tid.

Hvis man skal starte en server og upload RR om ens server til DNS:

Det første man gør er at registrere sit domænenavn hos en registrar, som er en offentlig enhed der sørger for at alle domæner er unikke. Når man gør dette, så skal man også give nave og IP-adresser på ens primære og sekundære autoritative DNS servere.

Nu skal registraren sørge for at oprette RR records af typerne A og NS. Så brugerne både kan lave hostnamet om til en IP adresse, men også finde den autoritative server på den IP.

Og hvis der også er en mail-server, så skal der også laves en Type MX RR.

DHT: Distribueret P2P Database. Er nødvendig for mange P2P programmer, da de skal kunne søge og opdatere tabellen over brugere. DHT står får Distributed Hash Tables, en måde at gøre dette på.

Alle brugere er gemt som et Key, Value par. Key kunne være CPR nummer og Value navn.

Eller Key kunne være navne på indhold der er gemt på Valuen som er IP-adresser.

Så en peer kan querie en fil, og få parret tilbage hvis det eksisterer.

**TRANSPORT LAYER:**

Sørger for end-to-end kommunikation mellem programmer på forskellige hosts.

Deler information op i bidder som sendes. Disse samles på den anden side, og sendes videre til application layer. Kan bruge TCP eller UDP til at sende med.

Transport vs network layer: Network - Logisk kommunikation imellem hosts.

Transport - Logisk kommunikation imellem processer.

12 kids sending letters

to 12 kids

􀂈processes = kids

􀂈app messages = letters

in envelopes

􀂈hosts = houses

􀂈transport protocol =

Ann and Bill

􀂈network-layerprotocol = postal service

Multiplexing and demultiplexing: Det er en ting som netværk laget laver. Der kan køre mange processer på samme host, og det er vigtigt at vide hvilken host der skal have hvilke pakker.

Derfor bruger processer sockets, som er unikke. Multiplexing er det at samle og pakke data chunks, og smide en header på. Demultiplexing er det at åbne pakken igen, og kigge på headeren samt sende data til dem der skal have den.

Demultiplexing: Host modtager IP datagram. Hvert datagram har en source of destination IP. Hvert datagram har 1 tranport-layer sekement. Hvert sekment har source og destination port nummer. Host bruger så IP og port til at sende sekmentet videre til det rigtig socket.

UDP bruger kun destination port og ip

TCP bruger alle 4.

Webservers har et socket per tilsluttet klient.

UDP: Til streaming media eller andet loss-tolerabelt data.

Connection-less, Sender bare løs med fuld hastighed (ingen flow control), kan bruge reliability på application niveau istedet.

Bruger checksum til at tjekke om data er intakt når det når frem. (Flipped bits)

Først udregner afsenderen summen af alle 16-bit ord i segmentet. Dette er checksum og bliver sendt med i checksum fielded af UDP segmentet.

Modtager gør så det samme, og kontrollerer om disse er ens. Hvis de ikke er ens, så droppes.

RDT (Reliable Data Transfer): Bruger ACKS og timeouts. Sender pakke igen hvis intet ACK hørt. I tilfælde af tabt ack og ikke pakke, så slettes pakken når den når frem for anden gang, og der sendes den manglende ack.

**TCP:**

point-to-point:

􀂁one sender, one receiver

􀂈reliable, in-order byte

stream:

􀂁no “message boundaries”

􀂈pipelined:

􀂁TCP congestion and flow

control set window size

􀂈send & receive buffers

full duplex data:

􀂁bi-directional data flow

in same connection

􀂁MSS: maximum segment

size

􀂈connection-oriented:

􀂁handshaking (exchange

of control msgs) init’s

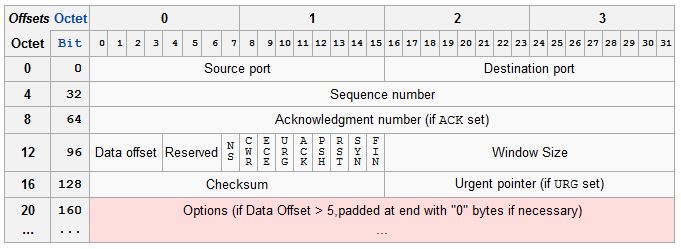
sender, receiver state

before data exchange

􀂈flow controlled:

􀂁sender will not

overwhelm receiver

****

TCP timeout value: Skal være længere end RTT, men RTT varierer. For kort kan betyde unødige retransmissioner af data, for lang tid kan betyde lange ventetider for ingen verdens nytte.

RTT: Man udregner en SampleRTT ud fra en sendt pakke, som ikke bliver gensendt. Der bliver lavet en af gangen med lidt mellemrum, og bruger så en formel til at udregne EstimatedRTT.

TCP holder et gennemsnit af RTT i EstimatedRTT, og opdaterer så EstimatedRTT når der kommer nye SampleRTT ud fra denne formel:

EstimatedRTT = (1 - α) x EstimatedRTT + α x SampleRTT

Fast retransmit: Timeout periode er ofte forholdsvis lang. Senderen sender ofte mange pakker lige efter hinanden, og hvis den får flere acks på den samme pakke, fordi modtageren ikke fik en af dem, (den sender bare den samme ack for dem den mangler når den får nye pakker, så gensender den pakken efter den der bliver ack'ed.

Flow Control: TCP sørger for at sender ikke overrumpler modtager med data, hvis han ikke kan bearbejde data han får ind så hurtigt. Han har en buffer hvor alt data han modtager ligger i indtil han bearbejder det. Ved konstant at rapportere tilbage hvor meget bufferplads han har tilbage, så kan afsender justere hastigheden. Dette gøres igennem Recieve Window field i headeren.

Da overflow = tabte pakker.

Three-way handshake: Client host sender TCP SYN pakke med et seq # og ingen data.

Server modtager SYN og svarer med SYN ACK.

Client host modtager SYN ACK.

Closing a TCP connection: Client sender TCP FIN pakke

Server modtager FIN pakke, sender ACK, lukker forbindelsen og sender derefter en FIN pakke.

Client modtager FIN pakke og svarer med en ACK. Starter "timed wait", hvor den vil svare flere FIN pakker med ACK, (hvis nu serveren ikke fik pakken).

Server modtager ACK og lukker forbindelsen helt.

Congestion: For mange brugere der sender for meget data over en for lille linie. BOTTLENECK.

Giver tabte pakker pga. overfyldte buffers og meget kø pga. mange pakker i router buffer.

Plus hvis pakker går tabt, så bliver der sendt flere pakker, hvilket giver endnu mere congestion.

TCP Congestion Control: Idéen er at sende pakker så hurtigt som muligt uden congestion, så er det også nødvendigt at finde det punkt hvor congestion begynder og sende lige under det.

Hver pakke bliver sendt med sin egen rate, som styres af feedback.

ACKs modtaged betyder ikke congested netværk.

Tabt segment betyder congestion et eller andet sted.

"Probing for Bandwidth": Forøger hastigheden indtil punktet hvor der kommer tabte pakker, hvor efter man sænker farten igen. TCP sænker farten ved at sænke mængden af pakker der bliver sendt på samme tid. CWND = Congestion Window.

Bruger formel til at udregne sendehastighed.

**cwnd >= LastByteSent-LastByteAcked**

congestion fomula.JPG

MSS = Maximum Segment Size

Hvis der er pakketab, så følges forskellige regler:

Timeout: Sæt MSS til 1.

Duplicate ACKS: Sæt cwnd til det halve.

ACK modtaget: Slowstart, derefter linær.

Slowstart: Start med 1 MSS. Forøger derefter eksponentielt indtil det første pakketab eller max.

Fordobler cwnd hver RTT.

Derefter fortsætter den lineært ved at forøge MSS med 1 hver RTT så længe den der under der hvor den døde sidst.

TCP er fair: Fordi flere klienter følger samme procedure, så de sænkes og hæves med hinanden.

**NETWORK LAYER:**

Transporterer segmenter fra sender til modtager. Indkapsler på senderens side segmenter i datagrammer. Leverer datagrammer på modtagerens side i transportlaget.

Network layer procols findes i alle hosts og routers.

Routere undersøger header fields i alle IP datagrammer der kører igennem.

Understøtter både connectionless (UDP) og connection (TCP) baseret transport.

Netværklaget har også connection based services, men som fungerer lidt anderledes fra transport lagets. Disse er host-til-host istedet for process-til-process som på transportlaget.

Disse foregår på routers i network core også, i modsætning til transport som fungerer

på end-hosts.

Forwarding: Processen at flytte pakken fra indput til det rigtige output og omvendt.

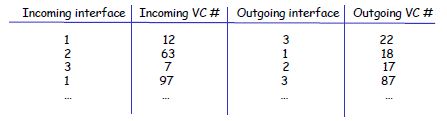
Routing: Processen at planlægge hvilken rute datagrammet skal tage fra sender til modtager.

Datagram network: Netværk der kun understøtter connection-less services.

Virtual-Circuit (VC) network: Netværk der kun understøtter connection-based services bruger VC.

VC er en måde at emulere end fast forbindelse på internettet, som er et datagram netvæk.

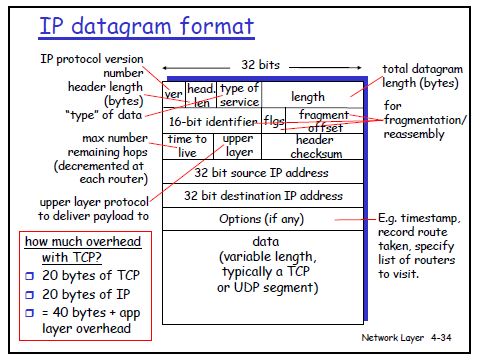
Routers har en forwarding tabel, når en pakke så skal igennem, så kontrollerer den VC nummeret i pakkens header for at finde ud af hvilket interface den skal sende den ud af, giver den et nyt nummer, (da alle routers kan have forskellige VC numre til samme pakke, afhængigt af linket), og sender den videre. Det samme sker i den næste router, indtil den når den anden ende.



Datagram netværk: Ting bliver smidt ud på netværket, de finder selv vej til modtager af de veje de nu vil,

og ting modtages.

MTU (Max Transfer size): Alle links har en max transfer size, og hvis datagrammerne er for store, så bliver de fragmenteret til mange mindre stykker, og samles så igen ved modtageren.



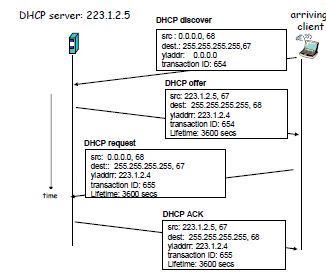
DHCP: Dynamic Host Configuration Protocol henter automatisk IP adresse fra server. Tillader at man dynamisk får en ip adresse når man tilslutter sig et netværk.

DHCP MSG: Host broadcaster "DHCP discover" message.

DHCP server svarer med "DHCP offer".

Host anmoder en IP adresse "DHCP request".

Server sender adresse med "DHCP ACK".



DHCP server: Kan også give adressen på first hop router. Navn og ip på DNS server og network mask.

Sådan for man IP: Computer slutter sig til netværket. Skal bruge IP, First-hop IP, og DNS add.

Sender en DHCP request indkapslet i UDP, som er indkapslet i IP som er indkapslet i Ethernet.

Ethernet frame broadcaster (Destination: FFFFFFFFFFFF), på LAN som bliver fanget af routeren der kører DNS server.

Requesten bliver så demuxed hele vejen op til DHCP MSG igen i routeren.

Serveren laver så en DHCP ACK der indeholder info.

Denne bliver bliver så Muxed ned igen og demuxed ved klienten.

Klienten kender nu info!

NAT: Network Adress Translation. Alle netværk bruger kun en IP adresse. Dette er navnet på routerens interface udadtil. Måde at spare på IP adresserne. Alle ting på indersiden af routeren er ikke synlige for omverdenen, før de er igennem routeren, hvilket er godt sikkerhedsmæssigt.

Routeren får ofte dens IP på samme måde som en computer på netværket gør, nemlig via DHCP. Den broadcaster også ligesom den enkelte computer.

NAT translation table: Routeren udadtil har en tabel. Når en computer på indersiden sender et datagram ud, så har den en source port, som kommer fra den computer der sendte den, når den så går igennem routeren, så udskifter den, den lokale ipadresse ud med den offentlige, finder en port udadtil der ikke bliver brugt, og skriver den på i stedet for den anden. Routeren gemmer så det gamle portnummer og IP adresse sammen med det, som det blev skiftet til, så når den får svar tilbage, så bytter den bare rundt igen, hvor efter pakken bliver sendt tilbage til den specifikke computer på netværket.

ICMP: Bruges af hosts og routere til at kommunikere netværk-level information, som fejl f.eks, echo pings. ICMP messeges bliver båret i IP datagrammer.

ICMP message: Type, kode plus første 8 bytes af IP datagrammet der skabte problemer.

Type Code description

0 0 echo reply (ping)

3 0 dest. network unreachable

3 1 dest host unreachable

3 2 dest protocol unreachable

3 3 dest port unreachable

3 6 dest network unknown

3 7 dest host unknown

4 0 source quench (congestioncontrol - not used)

8 0 echo request (ping)

9 0 route advertisement

10 0 router discovery

11 0 TTL expired

12 0 bad IP header

Traceroute (ICMP): Sender en række UDP segmenter til destinationen, første har TTL = 1, næste har 2 osv. indtil den har så høj TTL at den når frem, hvor serveren så smider pakken ud og sender en ICMP message med Type 11, kode 0, hvor den inkluderer sit navn og IP.

IPv6: Man vil gerne skifte til IPv6 da der er mange, mange flere adresser at give ud af, da den bruger 128 bit address spaces istedet.

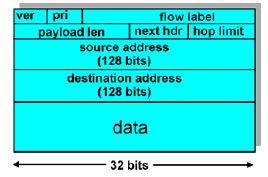
Er mere effektive pga. deres nye header.

Har altid en 40byte header. Bliver ikke fragmented.

IPv6 header: Priority: identify priority among datagrams in flow

Flow Label: identify datagrams in same “flow.”(concept of “flow” not well defined).

Next header: identify upper layer protocol for data



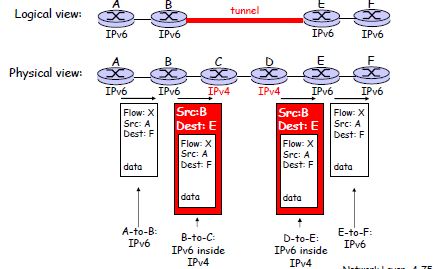
Forskelle på IPv4 og IPv6: Checksum er helt fjernet for at reducere tiden det tager per hop.

Options: Tilladt, men udenfor headeren indikeret af "next header" feltet, for at sørge for den altid er 40byte.

ICMPv6: Ny version af ICMP - Nye message typer (f.eks. packet too big)

- Multicast group management functions

Overgang til IPv6: Man kan ikke opdatere det hele på en gang. Umuligt med den store mængde routers, plus at alt ville gå ned. Skifter den lidt efter lidt, og så vil IPv6 datagrammer bliver pakket ned i IPv4. "Tunneling".

****

**Data link layer:**

Data-link laget har til ansvar at overføre datagrammer fra et punkt (node) til en anden over et link. Såsom trådet forbindelser, trådløse og lokale netværk.

Hvert link kan være forskelligt. Det kan være det først bliver overført på et LAN, derefter trådet, så trådløst, så trådet igen, før den når modtager.

Hvert link kan forskellige ting.

trip from Princeton to

Lausanne

􀂁limo: Princeton to JFK

􀂁plane: JFK to Geneva

􀂁train: Geneva to Lausanne

􀂈tourist = datagram

􀂈transport segment =

communication link

􀂈transportation mode =

link layer protocol

􀂈travel agent = routing algorithm

Framing: Indkapsler datagrammer i en frame, tilføjer en header, trailer.

Channel acces hvis det er et delt medie.

Frames bruger MAC adresser som source of destination istedet for port/ip.

Link layer services: Flow control: Hastighed mellem modtagende og sendende node.

Error detection: Fejl der sker pga. støj eller svækkelse af signalet. Modtager opdager dette og beder sender om at sende igen, og dropper selv framen.

Error correction: Modtager kan selv rette fejlen i bits, og behøver ikke får gensendt frame.

Half-duplex and full-duplex: Med halv-duplex så kan nodes i begge ender af linket sende, men ikke på samme tid.

Data Link generelt:

Er implementeret i hver host, i deres netværks adapter, altså netkort. Sidder på hostens system busser. En kombination af hardware, software og firmware.

Senderen indkapsler datagram i frame, tilføjer error control bits, RDT, flow control osv.

Modtager leder efter fejl, rdt, flow, osv. Udpakker datagrammet og giver videre til næste lag.

EDC: Error detection and correction. Noget data er beskyttet af EDC, ikke nødvendigvis alt, kan være bare headeren.

EDC er ikke 100% sikkert. Protokollen kan misse fejl, men det er sjældent. Større EDC fields giver mere protection. Disse kræver dog mere plads, og mere tid at udregne.

EDC metoder: Parity bit: Afsender smider en ekstra bit på, som sørger for at der er en lige mængde 1'ere. Så hvis der er 7 1'ere, så er parity bitten 1. Er der 4 1'ere så er den 0.

Ved at lave parity bits i 2 dimensioner, så har man mulighed for både finde fejlen, men også rette den, da der vil være en fejl i en lodret og en vandret række.

MAC: En 32-bit IP adresse som bliver brugt som network-lags adressen.

Bliver brugt til at få datagrammer til destinationen.

MAC adressen bruges til at få en frame fra en interface til et andet fysisk-tilsluttet interface (samme netværk).

48 bit MAC adresse: Brændt in i NIC ROM (Network interface card).

LAN-adresse of ARP: Hver adapter på et netværk har sin egen LAN adresse.

IEEE (Institute of Electrical and Electronics Engineers), står for uddeling af MAC adresser. Udvikleren af netværks adapteren køber en masse MAC adresser, hvilket gør at de alle er unikke.

En MAC adresse er altid den samme, lige meget hvilket netværk adapteren er sluttet til.

ARP: Address Resolution Protocol.

Alle IP nodes, altså hosts og routers, har en ARP tabel.

ARP tabel: Holder styr på IP og MAC adresserne. Så når en computer skal sende et datagram til en anden på netværket, så sender han en frame til routeren, som kigger i sin ARP tabel, finder den tilsvarende MAC adresse, og sender den videre.

ARP tabellen indeholder også en TTL værdi, der dedikerer hvor lang tid hvert sæt i tabellen er der. En typisk TTL er 20min.

**Hvad sker der hvis adressen ikke står i ARP tabellen?** Noden laver en speciel pakke der hedder en ARP pakke. En ARP pakke har nogle fields, som modtagerens og afsenderens IP of MAC adresser. Både ARP Query og ARP response har samme format. Formålet med en ARP query er at spørge alle de andre nodes på netværket om de har den MAC adresse der passer til IP adressen.

Noden der spørger, sender så pakken til broadcast MAC adressen FF-FF-FF-FF-FF-FF. Adapteren pakker så ARP pakken ind i en link-layer frame, og bruger broadcast adressen som pakkens destinations-adresse, og smider den ud på subnettet.

Computeren som skulle modtage den oprindelige pakke, gør sig så til kende, da den kan se at der bliver spurgt om dens MAC adresse, hvor den svarer tilbage til noden, hvor den så gemmer MAC, IP, TTL parret i dens ARP tabel.

Routeren opretter selv sin egen ARP tabel når den bliver sluttet til, det skal ikke gøres selv.

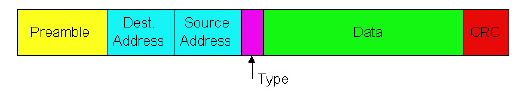
**Hvordan sendes et IP datagram fra computer A i ét subnet, til B i et andet subnet?**

A har B's IP-adresse, og den sender derfor et IP datagram med A som afsender og B som modtager. A bruger så ARP til at få MAC adressen på den node der sammenslutter de to subnet. A laver derefter en link-layer frame med R's MAC adresse som destination, som indeholder IP datagrammet. A's NIC sender frame, node R modtager frame. R fjerner IP datagrammet fra framen, ser hvor den skal hen, nemlig B. R bruger så ARP til at finde B's MAC adresse, og pakker nu IP datagrammet ind igen, med B's MAC adresse som destination.

Ethernet: En god og billig LAN teknologi, da Ethernet NIC's er billige at producere. Kan klare gode hastigheder. (10Mbps - 10 Gbps)

Ethernet netværk er bygget op i stjerneform med en switch eller router i midten, hvor hver host kører sin egen Ethernet protocol.

Ethernet Frame: Den sendende adapter indkapsler IP datagrammet( eller andre slags netværk lags pakker), i en Ethernet frame.



Preample (8 bytes): Hver af de første 7 bytes har en værdi af: "10101010", og den sidste byte er: "10101011". De første 7 bytes er designet til at "vække" de modtagne adaptere, og synkronisere dem. De synkroniseres for at sætte dem til den hastighed som der bliver overført med, da den aldrig vil være nøjagtig max. De sidste to "11" bruges til at informere modtageren om at den skal være klar til at modtage data.

Dest. Address (6 bytes): Det her field indeholder MAC adressen af den modtagende adapter. Når modtageren så modtager en frame med enten sin egen MAC, eller broadcast adresse, så giver den indeholdet af framen videre til netværk-laget. Hvis det ikke er til den, så dropper den.

Source address (6 bytes): Inderholder MAC adressen på den der sendte framen.

Type field (2 bytes): Dette field bruges til at fortælle modtageren hvad der ligger i pakken, af hvad type. Hvis det f.eks. ikke nødvendigvis er et IP datagram der ligger i pakken, så skal indeholdet måske sendes til et andet netværklag.

Data field (46 - 1500 bytes): Indeholder IP datagrammet. Maximum MTU for Ethernet er 1500 bytes, så hvis den overskrider dette, så bliver den nødt til at fragmentere framen. Der skal altid være mindst 46 bytes i data field, så hvis der ikke er det, så bliver resten bare fyldt ud til det fylder 46 bytes. Når den ankommer hos modtager, så kigger modtager på length fielded i IP datagrammet for at finde ud af hvor meget af det er ekstra fyld.

Cyclic redundancy check (CRC) (4 bytes): Den der bruges til at finde bit-fejl. (EDC)

Mere Ethernet: Ethernet er unreliable og connectionless. (Ingen handshaking) Bliver ikke sendt ACKS over Ethernet. Hullerne fra de manglende frames vil blive fyldt ud hvis applikationen bruger TCP, så bliver de fyldt der, ellers så vil applikationen bare se hullerne.

Hubs: Dumme gentagere. De bits der kommer ind igennem det ene interface, bliver gentaget ud i alle interfaces på hubben. Alle nodes der er tilsluttet en hub, kan kollidere med hinanden. Ingen buffer.

Switch: Er aktiv, og meget bedre end en hub. Laver "store and forward" af Ethernet frames. Læser indkommende frames' MAC adresser og sender framen ud på en eller flere interfaces.

Er transparent, de kan ikke ses af hosts. De er plug and play, da de selv konfigurerer sig selv.

Hver host tilsluttet switchen har sin egen dedikerede forbindelse.

Switch table: Bruges til at vide hvilke hosts der er på hvilke interfaces. Gemmer MAC adresse, interface for at kunne nå den, og et timestamp.

Ligner en routing tabel.

Lærer selv hvilke MAC adresser der er på hvilke interfaces, ved at kigge på de frames der kommer ind af interfacesne. Gemmer dem herefter i switch tabellen.

Hvis den ikke har MAC adressen i sin tabel, så "flooder" den. Den sender altså framen videre til alle interfaces undtagen der hvor den kom fra.

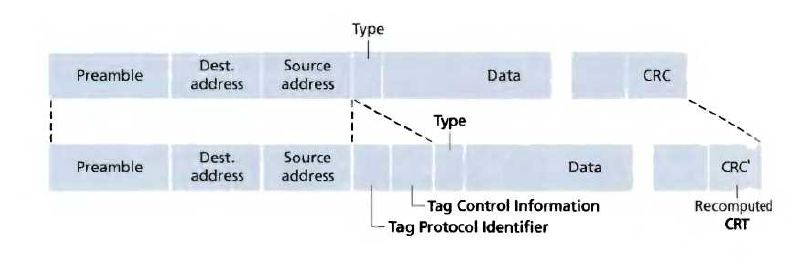
Switches vs Routers: Begge "store and forward". Routers arbejder på Network layer og switches på link layer.

Virtual LAN (VLAN): Switches der tillader VLAN kan lave et virtuelt LAN netværk, så selvom en person er tilsluttet en switch i et andet subnet, så virker der stadig som om han er i et andet. Man kan have en switch som fungerer som to virtuelle switches.

Traffic isolation: Frames til/fra porte i et VLAN kan ikke modtages af andre VLAN på samme switch. Umuligt. (Kan både defineres på switch-porte men også MAC adresser.

Forwarding imellem VLANS: Fungerer på sammen måde som hvis det var to seperate switches.

Trunk port: En port i en VLAN som sidder i en port i en anden seperat switch som er del af samme VLAN som den anden. Flytter frames over flere seperate switches.



VLAN frame: 2 ekstra felter tag control information og tag protocol information. Control indeholder 12-bit VLAN ID field og et 3 bit priority field)

**Network security:**

Fortrolighed: Kun afsender og modtager der forstår beskeden.

Authentication: Afsender og modtager vil gerne være sikre på hvem hinanden er.

Beskedens integritet: Afsender og modtager vil gerne være sikre på at beskeden ikke er blevet ændret undervejs.

Hvem vil gerne kommunikere sikkert? Webbrowser/server ved electroniske transaktioner.

On-line bank-services.

DNS servere.

Routere der udvikler routing table updateringer.

De onde kan: Lytte med uden at man ved de er der.

Selv indsætte beskeder i "samtalen".

De kan lade som om de er en af parterne, og agere på deres vejne.

Kan bytte modtager eller afsender ud med sig selv.

DoS.

Ciphertext: Krypteret text, som er blevet krypteret med en nøgle.

m plaintext message

KA(m) ciphertext, encrypted with key KA

m = KB(KA(m))

Eksempel på simple encryption: Sustitution Cipher. Byt en ting ud med en anden. Det var være at alle bogstaver bliver byttet ud med et 3 bogstaver længere fremme i alfabetet.

Breaking encryption: Kræver en af to scenarier. Enten har den onde et stykke ciphertext, eller også har han et noget plaintext med tilsvarende ciphertext.

Kan løses ved at analysere, eller prøve alle muligheder. (Brute Force) Eller nogle gange kan ting genkendes.

Polyalphabetic encryption: Monoalphabetic M1, M2, M3, M4 osv.

M = en monoalphabetisk enkryption.

Cycling pattern: n = 4: M1, M3, M4, M2, M1, M3...Altså man bruge flere forskellige monoalphabetic enkryptioner i en enkryption.

På den måde dukker det samme ord ikke så nemt op på samme måder flere steder.

Symmetric key crypto: Både afsender og modtager har en ens nøgle, f.eks. at de kender mønsteret som det er blevet krypteret i. De aftaler nøglen igennem en sikker kilde, f.eks. telefonen, personligt møde, osv.

To typer symmetrisk ciphers: Stream ciphers - Krypterer en bit af gangen.

Block ciphers - Deler plaintext op i mindre blokke af samme størrelse. Krypter hver blok.

Bits bliver så krypteret ud fra et mønster af blokke tilsvarende størrelsen.

Hvis man kører denne algoritme flere gange, så er der større chance for at man ikke gætter det, fordi så er krypteringen krypteret i en kryptering.

Public key: En anden løsning er at skjule krypteringen med en public kryptering. Den krypteres med en public nøgle, som alle har, men kræver så en privat nøgle af modtageren, før beskeden kan læses. Den dekrypteres altså ført med den offentlige, og derefter den private nøgle.

Det smart er at det er nøgler der er genereret af modtageren, så modtageren har den private nøgle. På den måde er det kun ham der kan dekryptere beskeden.

Det er bare vigtigt at man ikke kan udregne privatnøglen ud fra den offentlige nøgle.

Message integrity: Tillader modtageren af være sikker på at beskeden ikke har været ændret siden den blev sendt, samt at modtageren er den rigtige, beskeden ikke har været vist før nu.

Message digests: En Hash funktion der tager en besked af vilkårlig længde, og laver en fixed- length besked "signatur".

Skal være nem at udregne, men umulig at tilbageregne. Skal være umuligt at kunne reproducere. Skal virker vilkårlig.

Internet checksum: Internet checksum har nogle Hash funktion properties. Den har fixed length output (16-bit sum), og laver et output ud fra mange inputs. Bare ikke særligt stærk, da man kan reproducere de samme signaturer med andre inputs.

Hash funktioner: Meget brugt hash funktion er MD5. Laver en 128-bit message i en 4-skrid proces.

En anden er SHA-1, en US standard som laver 160-bit message digest.

Message Authentication Code (MAC): Bruges til at udregne beskedens integritet. Afsender og modtager har en fælles nøgle, som kun de ved. Afsender sætter så nøglen på beskeden, hasher den, og klistrer hashværdien fast på beskeden og sender de 2 ting. Modtageren klistrer så hemmeligheden på igen, hasher det igen og sammenligner med den påklistrede hashværdi. Hvis de er ens, så er beskeden intakt.

HMAC: En populær standard, som retter nogle enkelte småfejl.

Sammenkæder hemmeligheden foran beskeden, hasher det hele, sammenkæder hemmeligheden med den nye hash værdi, og hasher det hele igen.

Digitale signaturer: Afsender hasher en besked. Så krypterer han hash værdien med sin personlige nøgle, derefter sender han både plaintext beskeden og hans krypterede besked.

Modtager Hasher så beskeden og gemmer hash værdien, decrypterer den hashværdi der var krypteret med afsenders personlige nøgle, med afsenders offentlige nøgle og sammenligner de to hash værdier. Hvis ens, så er afsenderen den rigtige, og beskeden er intakt.

Problemet er, at hvis en person udgiver sig for at være en anden, og sender en public key som personen siger er fra ham selv (altså den han udgiver sig for at være), og de så passer, så tror modtager at afsenderen er en forkert. Derfor nødvendigt at signere den offentlige nøgle.

Certification Authorities: (CA) er en tredjepart som bruges til dette problem. En person registrerer en public nøgle med CA, CA laver så et certicate der binder den offentlige nøgle, så når modtager får den, så kan de se at CA siger god for den nøgle.

Når modtageren så vil have afsenders public key, så bruger de CA's offentlige nøgle på certificate og så får de afsenderens public nøgle.

Certificate: Indeholder navn på udgiver, personoplysninger (navn, adresse, domænenavn osv), personens public key, digital signatur som er signeret med personens private nøgle.

End-point authentication: En måde at sikrer på at det er personen der lavede beskeden, der sendte den.

Vi ved at det er afsender der lavede den.

Playback-attack: Den onde optager data der bliver sendt fra afsender til modtager, og afspiller derefter samme besked igen. Kunne være et online-køb, bank overførsel, besked.

Hvordan beskytter man mod playback-attack?

En Nonce. Nonce er et nummer som en protocol kun bruger en gang i dens livstid. Så når den er brugt en gang, så bliver den aldrig brugt igen. Modtager giver afsende en nonce, som afsender så hasher sammen med beskeden og deres nøgle, og sender det. Modtager kan så kigge og sammenligne selv om hashværdierne passer. Da noncen kun kan bruges en gang, er det ikke muligt at sende beskeden igen, da hashen så ville være anderledes.

Secure email: Afsender vil sende en hemmelig email til en.

Afsender krypterer mailen med en tilfældig symmetrisk nøgle (samme nøgle til kryptering og dekryptering), og krypterer derefter den symmetriske nøgle med modtagers offentlige nøgle og sender disse ting.

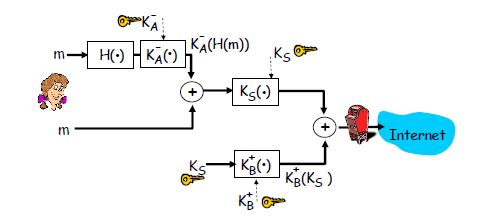
Når modtager sår får disse, så dekrypterer han den symmetriske nøgle først med sin private nøgle, og dekrypterer derefter beskeden med den symmetriske nøgle.

Hvis afsender gerne i stedet vil bevise at det faktisk er afsender, så kan afsender hashe beskeden, kryptere hashværdien med sin private nøgle, sende beskeden og hashen, hvor modtageren så dekrypterer hashen med afsenders offentlige nøgle, og sammenligner med beskedens hash.

Hvis mailen nu BÅDE skal være hemmelig OG afsenderen skal være bevist, så bliver det kompliceret. Så skal der 3 nøgler til. Afsenders private nøgle (A), modtagers offentlige nøgle (M) og en tilfældig symmetrisk nøgle (S), samt hashværdien af beskeden (H) og beskeden selv (B).

H krypteres med A, klistres sammen med B, som krypteres med S, hvor efter S bliver krypteret med M og disse sendes.

For at åbne det igen, så skal modtageren først dekryptere symmestrisk nøgle som så dekrypterer den samlede pakke, der nu bliver dekrypteret bagefter med afsenders offentlige nøgle, hvorefter man har beskeden og en hash, som nu kan sammenlignes.



Secure -email: PGP: Pretty Good Privacy.

Netværks sikkerhed på applikationsniveau. Er tilgængeligt gratis på mange platforme. Blev ikke udviklet af regeringen eller store organisationer. Brugere skulle ikke bekymre sig om certificate authority. Er bygget op af kendte algoritmer.

Hash: SHA-1, 160 bits.

Public key cryptography: RSA (Lav besked-bits om til integer, som bliver krypteret)

PGP key rings: Hver node har to key rings; public/private key par, som er ejet af noden, og en public key til andre brugere. Hver bruger har så en public key, som ligger på en server, og en private key, og hver gang brugeren skal bruge sin private key, så skal der bruges et password. Brugeren kan så kryptere sin data, og serveren kan så læse det. Når modtageren så får beskeden så kan de blot finde key'en på serveren.

Web of trust: En måde at autorisere public keys som er anderledes. En person der allerede er i web of trust kan godkende en anden brugers key-par (offentlig og private), samt username, og sige god for dem ved at signere med sin egen private nøgle. Den nye autoriserede person kan nu gøre det samme til andre, som kan gøre det samme til andre, osv.

Senere kan det være at andre personer ender med at autorisere den sammen person, og sikrer på den måde endnu mere at den person er ægte.

PGP generelt: PGP giver sikkerhed på applikationsniveau til en enkelt applikation. Den giver: Authentication, integrity og confidentiality.

SSL: (lille udgave) Secure Sockets Layer.

Udbredt sikkerheds protokol. Bliver brugt at næsten alle browsers og web-browsers. Designet af Netscape i 1993. Giver Authentication, integrity og confidentiality.

Er en tjeneste der gør TCP sikker, og kan derfor bruges af alle applikationer der kører over TCP. Det fungerer som et ekstra lag imellem application og transport laget. Der er i basis 4 faser af en SSL forbindelse:

Handshake: Afsender og modtager bruger deres certificates og private nøgler til at autorisere hinanden og uddele en fælles nøgle.

Key Derivation: Afsender og modtager bruger den delte nøgle til at lave et sæt nøgler.

􀂁Kc = encryption key for data sent from client to server

􀂁Mc = MAC key for data sent from client to server

􀂁Ks = encryption key for data sent from server to cl

􀂁Ms = MAC key for data sent from server to client

Data transfer: Da TCP er en byte-stream protokol, så kan man ikke kryptere den on-the-fly, og man kan ikke vente til efter alt data er sendt før man autoriserer og tjekker integritet osv. Derfor bliver data brudt op i samlede stykker som hedder "Records", hvor man kan smide en MAC på. Afsender og modtager har de samme fire nøgler. For at lave MAC'en, bruges record + Mc nøglen. Record+MAC bliver så krypteret med Kc.

Når modtager får det hele bruger modtager Ks og Ms til at pakke det hele ud igen.

SSL records: En record har 5 fields;

Type: som fortæller om det er en handshake eller en application data message.

Version

Length: Længden på data fæltet.

Data: Dataen der skal krypteres.

MAC: Til integrity check.

Problemer og løsninger ved SSL:

Den onde kan opfange pakker. Løsningen kan være at putte sequence numre inde i MAC'en, så ville det bliver opdaget.

Den onde kan lave Playback-attack. Løsning kan være at bruge Nonce.

Control information: Truncation attack: Den onde forfalsker TCP connection close segment, hvilket får en eller begge sider til at tro at der er mindre data end der faktisk er.

Det er derfor der er et Type-field i SSL record'en. Der specificeres det hvilken type besked det er.

Hvordan SSL faktisk gør:

Handhake: Klienten sender en liste over algoritmer den understøtter, sammen med en nonce.

Server vælger en algoritme fra listen, sender tilbage: Valg af algoritme, certificate, nonce

Klienten verificerer certificate, udvinder serverens public key af certificate, genererer pre\_master\_secret, krypterer den med serverens public key, sender den tilbage.

Klienten og serveren udregner hver især de 2 nøgler (MAC og Encryption) nøglerne ud fra pre\_master\_secret og nonce.

Klienten sender en MAC af alle handshake beskederne, det samme gør server. (Krypteret)

Det sidste er for at beskytte mod at den onde måske har slettet de stærke algoritmer fra listen over tilgængelige algoritmer. Men ved at lave MAC af alle beskederne, så kan de se om de har fået de samme beskeder som der blev sendt.

Key derivation fuldt: Klient og server udregner Master secret ud fra pre\_master\_secret, klient nonce og server nonce som puttes ind i en number generator. Master secret hashes så i en anden number generator der laver en "key block" som indeholder: Klient og server MAC nøgle (2 nøgler), klient og server encryption nøgle (2 nøgler), klient og server Initialiserings vector.

**Virtual Private Networks (VPNs):**

Det er en måde at skabe et krypteret "lokalt" netværk over internettet, f.eks. til fjernadgang til netværket. Hjemmearbejdspladser f.eks.

VPN sikrer: Data integrity, Oritin authentication, Replay Attack prevention, Confidentiality.

2 måeder at have IP Sec: Transport mode, hvor den enkelte computer har IP sec, hvor alle datagrammer der kommer ud af den, er krypterede.

Den anden er Tunneling mode, hvor det er en router der har IP sec, hvor computerne ikke behøver. (Bedre hvis der er mange computere der alle skal bruge IP sec.) Alle datagrammer der kommer ind i routeren fra det lokale netværk er ikke krypterede, men routeren krypterer dem så, før de sendes ud.

Eller en kombination af de to måder.

To protokoller: AH (Authentication Header): Sikrer afsenders identitet og integritet af data, men giver ikke confidentiality.

ESP (Encapsulation Security Protocol): Giver det hele, og bliver mere brugt end AH.

Security associations (SAs): Før der bliver sendt data, så bliver der oprettet en virtuel forbindelse imellem afsender og modtager, som hedder en "Security association", da IPsec er connection- baseret.

Det der gemmes i en SA: En 32-bit identifier for den unikke SA.

SA Afsenderens interface (IP)

SA Modtageres interface (IP)

Type af kryptering der bliver brugt.

Krypterings nøgle.

Hvilken type af integritets kontrol der bliver brugt, (f.eks. HMAC med MD5)

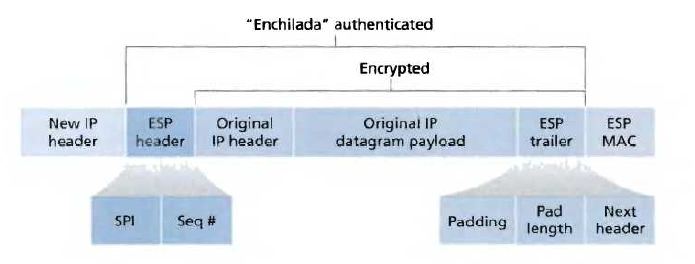
Autorisations nøgle.

SAD (Security Association Database): De to slutpunkter indeholder state af SA i en database. Der er 2 + 2n mændge SA i afsenders SAD.

Når der bliver sendt et datagram, så kigger routeren eller hosten i dens SAD, og finder ud af hvordan den skal kryptere osv. ud fra hvad der er aftalt med SA modtager.

Når modtageren så modtager IPsec datagrammet, så kigger den på SPI (32-bit identifier) og indekserer så SPI'en i SAD'en og åbner så datagrammet som aftalt.

Hvordan sker det: Afsender routeren laver datagrammet om til et IPsec datagram, ved at sætte en trailer på efter det originale datagram, med padding, padding length og hvilken type af header den næste er. Den krypterer derefter dette, og sætter sætter en ESP header på den krypterede data. Det hedder nu en enchilada. Den laver herefter en MAC af hele enchiladaen ved at bruge den algoritme og nøgle den har fået fra SA. Kæder MAC'en back på enchiladaen, og til sidst sættes en helt ny IP header som er magen til en normal IPv4 header, som sider forrest.



For en helt ny SA, der er sekvensnummeret 0. for hver IPsec pakke der bliver sendt over SA, der bliver seq # incremented, og værdi smides i seq# fielded i headeren.

Dette er for at undgå at nogen sniffer og playback'er pakkerne.

SPD: Security Policy Database:

Udover SAD så bibeholder SA også en anden data struktur, nemlig en SPD. SPD dikterer hvilke typer af datagrammer der skal IPsec processes, så kun de pakker der er en del af VLAN, bliver krypteret. SPD dikterer altså hvad der skal gøre med pakkerne, hvor SAD dikterer hvordan.

**Firewalls:**

En firewall isolerer en organisation eller host's lokale net fra internettet, og tillader kun nogle pakker at passere igennem.

Hvorfor bruge firewall?

Forebygger DOS angreb. (SYN flooding)

Forebygger ændringer i systemet bag ved firewallen.

Tillader kun autoriserede brugere på indersiden af firewallen.

3 typer firewalls: Stateless packet filters, Stateful packet filters, application gateways.

Stateless packet filtering:

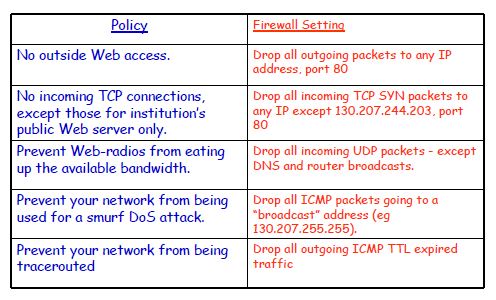
Internt netværk som er tilsluttet internettet via firewall router. Routeren filtrer alle pakker der kommer ind og ud og bedømmer om de skal tillades adgang eller droppes ud fra:

Source og destination IP

TCP/UDP source og destination port

ICMP message type

TCP SYN og ACK bits.



ACL: Access Control Lists:

En tabel med regler som bliver pålagt fra toppen og ned på alle indkommende pakker.

Stateful packet filtering: Holder styr på status af all TCP connections.

Den hølder øje med den enkelte forbindelses status, i forhold til om der er en forbindelse på den givne port, for at se om den indkommende pakke giver mening. (Hvis der kommer en pakke på et tidspunkt hvor der ingen TCP forbindelse er, så giver den ikke mening)

ACL indeholder også et fælt i denne packet filtering, om der skal kontrolleres connection state når der kommer pakke på specifikke porte.

Application gateways: Filtrerer pakke på application data, men også på IP/TCP/UDP fields.

Application gateways er en måde at lave specifikke tilladelser på. Det kunne f.eks. at give tilladelse til noget, men kun igennem gateway'en. Gateway er jo bare en slags proxy.

Fejl og mangler på firewall:

IP spoofing kan finde sted, da routeren ikke kan vide om pakker kommer fra den rigtige afsender.

Filtrer bruger også all or nothing policy på UDP, på grund af UDP's natur.

Tradeoff: Ved at have mere kommunikation med omverdenen, sænker man også sikkerheden, og omvendt.

Mange højt beskyttede hjemmesider oplever stadig angreb på trods af sikkerheden.

Intrusion detection: Packet filtering kigger normalt kun på TCP/IP headeren.

IDS (Intrustion Detection System): Kigger på pakke-indholdet og sammenligner f.eks. character strings med en liste af kendte virusser, attack strings osv.

Kigger på sammenhængen mellem flere pakker modtaget, om det f.eks. er et DoS angreb, port scanning eller network mapping.