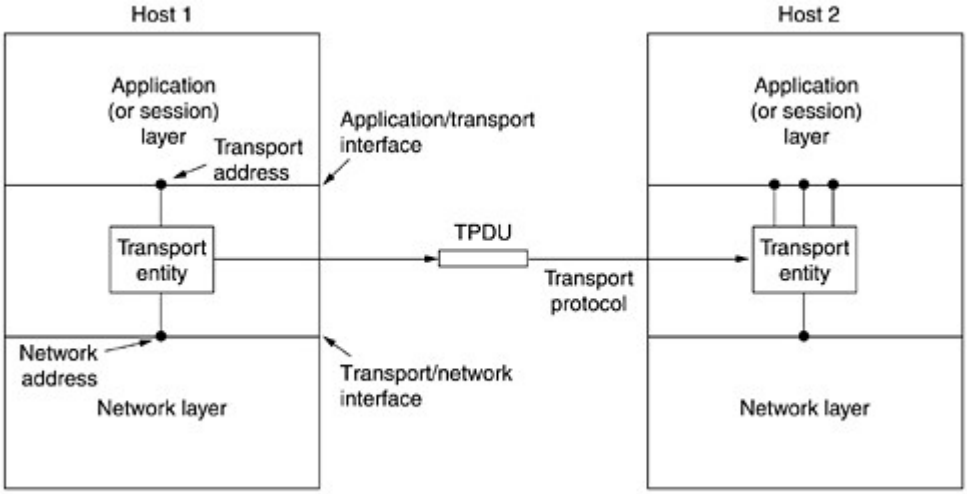
Prenosna plast

Vloga plasti je zagotavljati zanesljiv (nadzor napak), časovno ugoden prenos podatkov iz točke v točko neodvisno od omrežja, nadzor pretoka , fragmentacija.Je osrednja plast protokolarnega sklada, brez katere vsa struktura omrežja (razdeljena na sloje) ne bil imela velikega pomena. Hkrati je podobna povezavni plasti. Implementiramo pa jo lahko programsko ali tudi strojno. Implementaciji pravimo tudi osebek (transport entity). Osebek je lahko del jedra operacijskega sistema, lahko je samostojen proces,lahko je v obliki (dinamične) knjižnice...

Transportna plast nudi logično komunikacijo (strani ki komunicirata nista nujno neposredno povezani) med aplikacijskimi procesi:
• ponuja (zanesljivo) povezavno storitev (preko TCP protokola)
• ponuja (nezanesljivo) nepovezavno storitev (preko UDP protokola)
• podaja parametre, ki določajo kvaliteto storitev (QoS = Quality of Service)
• podobno naslavljanje in krmiljenje pretoka



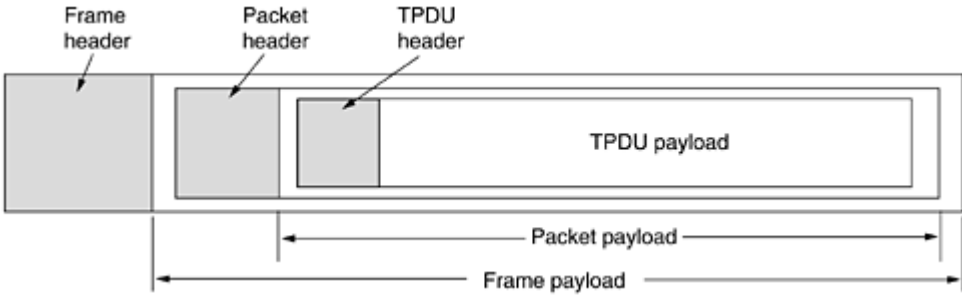
Obstajata 2 vrsti transportne storitve, povezavna in nepovezavna, kakor tudi na mrežni plasti, in sta tudi zelo podobni.
V mrežni plasti uporabniki nimajo nobenega nadzora nad dogajanjem. Zato ne morejo rešiti morebitnih napak, ki bi nastale pri prenosu (ali npr. pomanjkanje določenih primitivov). Tako moramo nad mrežno plastjo postaviti še prenosno plast, ki bo za to skrbela.

Za razvoj aplikacij, ki bi bile neodvisne od storitev, ki jih ponuja vršilec storitve, raznolikosti podmrežnih vmesnikov in nezaneslijvega prenosa, so na voljo primitivi, s katerimi lahko programerji pišejo svoje aplikacije.

Primitiv	poslani TPDU	Pomen
Listen	none	Blokira dokler ni poskusa za povezavo.
Connect	Connection request	Poskus vzpostavitve povezave.
Send	Data	Pošiljanje podatkov.
Receive	none	Blokira dokler podatkovni TPDU ne pride.
Disconnect	Disconnection req.	Stran hoče razpustiti povezavo.

Slika: primer primitivov za enostavno transportno storitev.

Osebka med sabo komunicirata s pomočjo PETS (Transportne Enote Transportnega Sloja, angl. **TPDU**)



Slika: Gnezdenje več PETS v paket in gnezdenje paketa v okvir.

Sloj nudi podporo kakovosti storitev (angl. QoS) prek parametrov:
• zakasnitev vzpostavitve povezave (angl. connection establishement delay);
• napovedovanje verjetnosti nevzpostavitve pravočasne povezave (angl. connection establishement failure probability);
• pretok (angl. throughput), ki meri število prenesenih B uporabnika v čas. enoti;
– zakasnitev (angl. delay), ki meri čas do vročitve paketa prejemniku.

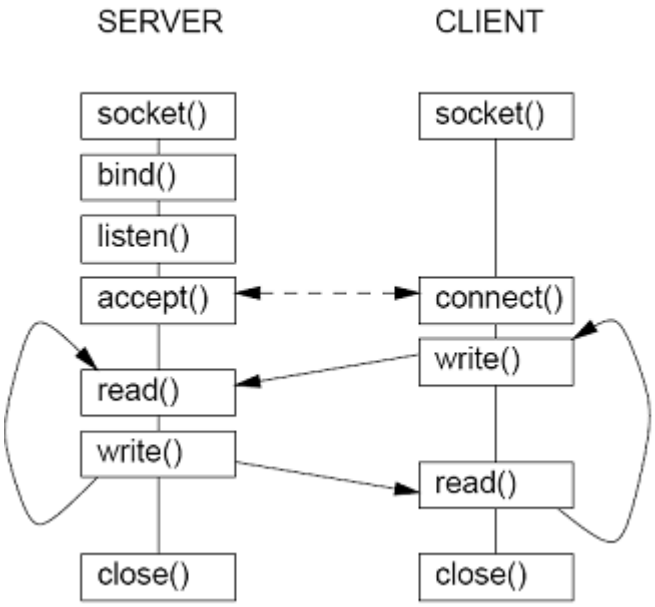
Kakovost storitev
• Parametre določi uporabnik ob vzpostavitvi povezave.
• V kolikor so parametri neizpolnljivi, sloj povezave niti ne poskuša vzpostaviti.
• Če so vrednosti parametrov blizu minimuma, se osebka pogajata.
• V primeru uspešne pogoditve se vzpostavi povezava z dogovorjenimi parametri, ki veljajo za časa trajanja povezave.
• Kakovost storitev proti »best effort«.

Vtičnice - Berkeley Sockets

Primitiv	Pomen
SOCKET	Tvori vtičnico. Parametri določajo format naslavljanja, tip storitve (npr. zaneljiv tok podatkov) in protokol. Uspešen klic SOCKET vrne navaden opis datoteke za nadaljnjo uporabo, podobno kot klic OPEN.
BIND	Da lokalni naslov novo tvorjeni vtičnici.
LISTEN	Zasede prostor za čakalno vrsto v primeru, ko bi se več uporabnikov istočasno hotelo povezati.
ACCEPT	Blokira klicatelja do poskusa vzpostavitve povezave. Dokler TPDU z zahtevkom za povezavo ne pride, je server blokiran.
CONNECT	Blokira klicatelja in aktivno sproži proces klicanja, dokler ne dobi od strežnika ustrezne PETS,nakar se odjemalec deblokira in povezava se vzpostavi.
SEND	Pošlje podatke - *full-duplex*
RECEIVE	Sprejme podatke - *full-duplex*
CLOSE	Omogoča razpustitev povezave (razpustitev pri vtičnicah je simertrična, obe

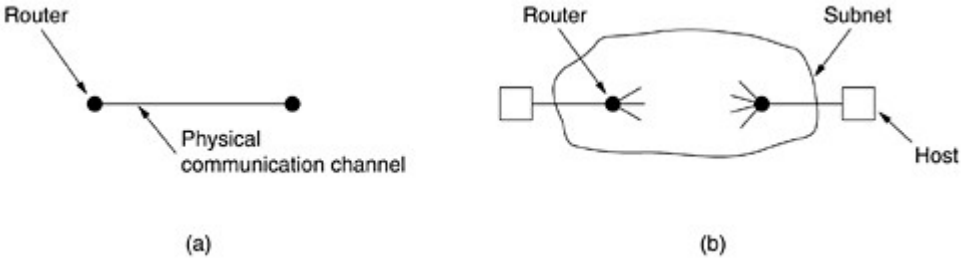
	strani izvršita CLOSE).

Ločimo 2 vrsti razpustitve povezave,
• simetrično - ena stran pošlje zahtevo za prekinitev in čaka drugo stran za potrditev.
• asimetrično - ena stran razpusti povezavo, a zaradi tega lahko pride do izgube podatkov.



Slika: primer pri povezavni comunikaciji. Na strežnikovi strani se socket, bind, listen in accept izvršijo zaporedoma.
"Puščice" delujejo na mrežni plasti. Na transportni plasti jih moramo implementirati tako, da bo delovalo. Če primerjamo z nepovezavno komunikacijo, vidimo da je v tem primeru še "listen" , "connect" in "accept", ki pri nepovezavni niso potrebni.

Razlike in podobnosti



Slika: podobnost med povezavno in prenosno plastjo. Pri prenosni plasti strani ni nujno da se vidita.

V neki meri je prenosna plast zelo podobna povezavni plasti. Obe pazita na morebitne napake, "flow control", "buffering" ipd. vendar ne nujno na enak način.

Razlike:
• V transportnem sloju pa je potrebno eksplicitno naslavljanje naslovnika.
• Vzpostavitev, držanje in razpustitev povezave
•V podmrežje je lahko možnost hranjenja podatkov, se lahko shranijo za nekaj časa in ne nujno takoj usmerjajo naprej (zakasnitve). Med večkratnimi zahtevami za povezavo in razpustitev povezave bi lahko prišlo do izgube podatkov. Zato potrebujemo posebne protokole.

• Lahko pride do različnih povezav, kjer vsak potrebuje svojo rešitev (npr. paketi lahko gredo po različnih poteh in tudi zato pridejo z zakasnitvijo)

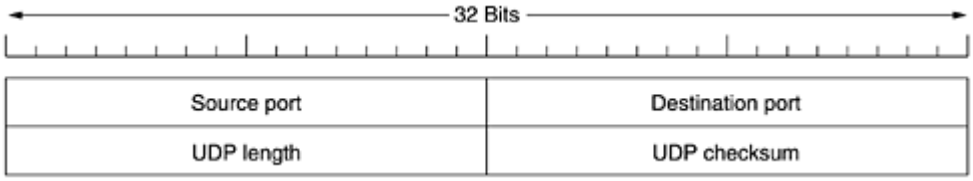
TCP in UDP

TCP (Transmission Control Protocol) in UDP (User Datagram Protocol) sta protokola transportne plasti. Spodaj so naštete njune osnovne lastnosti:

TCP:
- zanesljiv
- povazava točka v točko
- dvosmeren promet
- RFC 793 in 1323

UDP:
- nezanesljiv
- brezpovezaven
- hiter
- RFC 768

UDP
- UDP se uporablja predvsem za storitve tipa "pošljemo vprašanje, pričakujemo odgovor". V takšnih storitvah je primeren zaradi tega, ker sam po sebi ne zagotavlja nikakršne kvalitete storitev (ni potrjevanja, ni kontrole pretoka, ...)
- Glava UDP paketa:

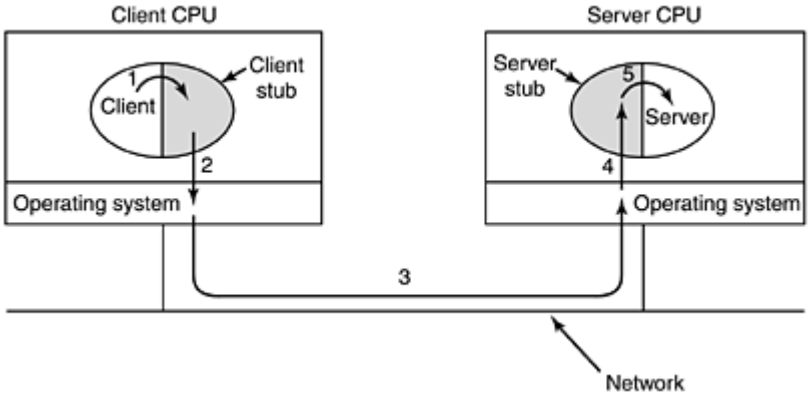


- Source port in Destination port sta izvorna in ponorna vrata
- UDP length je dolžina paketa
- UDP checksum je kontrolna vsota
- Kontrolna vsota se računa na podlagi psevdo UDP glave, ki vsebuje nekatera polja iz IP glave. To pomeni, da mrežna in transportna plast nista popolnoma neodvisni (v teoriji bi morale biti vse plasti neodvisne). Razlog je zgodovinski, saj sta bili mrežna in transportna plast sprva načrtovani kot ena sama plast.

Uporaba UDP protokola:

Oddaljeno klicanje funkcij:
- RPC (Remote Procedure Call) v C-ju
- RMI (Remote Method Invocation) v Javi

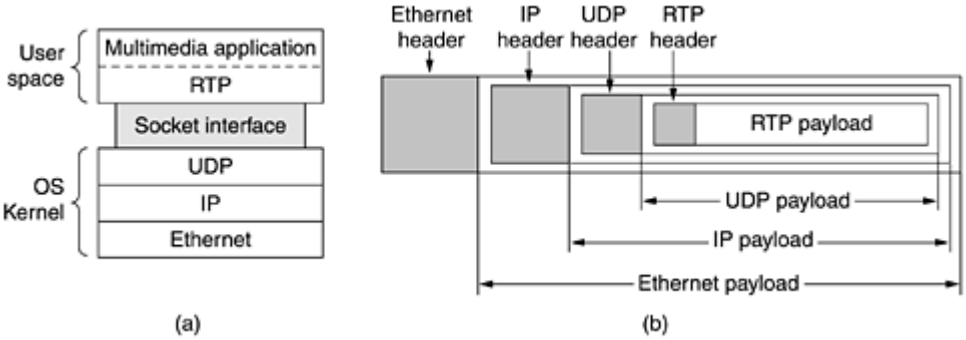
- Grafični prikaz potrebnih korakov:



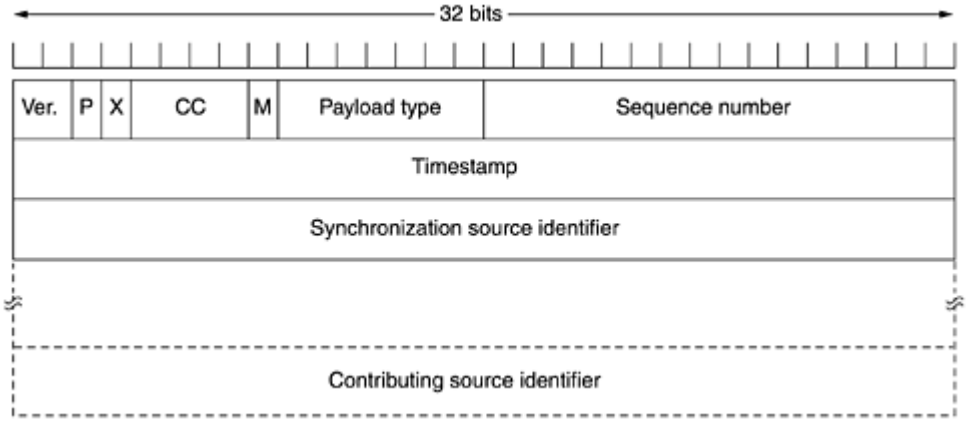
- Client Stub in Server Stub skrbita za pravilno obliko podatkov (marshalling)
    - primer: klient uporablja pravilo debelega konca za predstavitev podatkov, strežnik pa pravilo tankega konca. Oba stub-a (krclja) morata poskrbeti, da oddaljena procedura vrne pravilne rezultate in da jih klient pravilno razume.

Realno časovne aplikacije:

- RTP (Real-Time Transport Protocol)
- aplikacijska plast
- umestitev RTP v sklad protokolov:



- RTP glava:



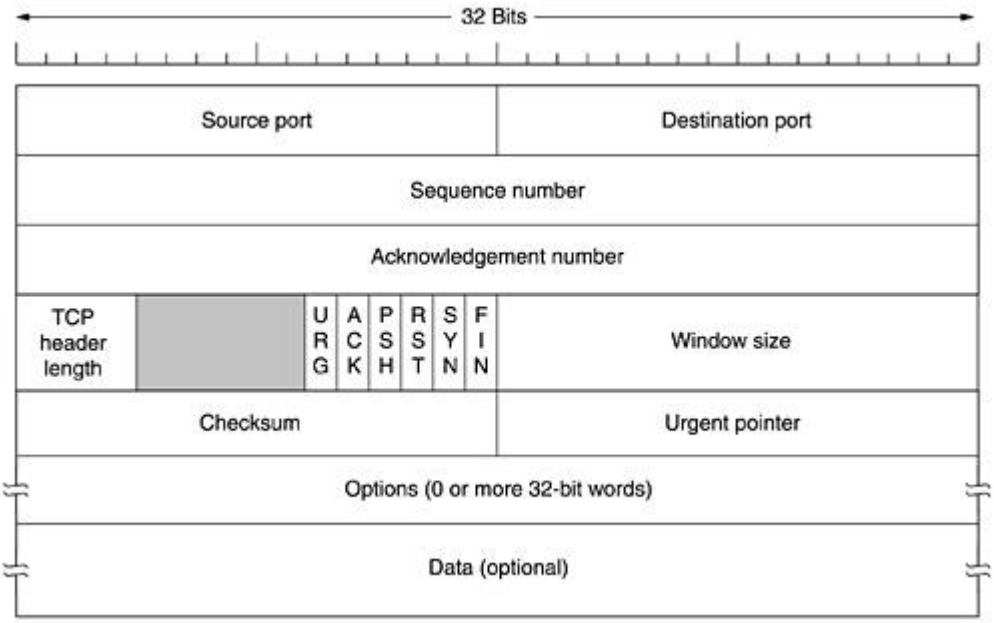
- v glavi je pomembno polje timestamp (ker sam UDP ne zagotavlja prejemanja paketov v pravilnem vstnem redu)
- trenutna verzija je 2
- RTP se uporablja za multipleksiranje večih tokov podatkov v en UDP tok
    - primer: video na zahtevo
        - en tok za samo "sliko"
        - več tokov za zvok

TCP

- Povezave so dvosmerne in dvotočkovne (ni razpršenega oddajanja)
- Vsak računalnik ima osebek TCP, ki je lahko samostojen proces ali del jedra operacijskega sistema
- Osebek TCP sprejema podatke v nizih, ki ne presegajo 64 KB
- Nize fragmentira v datagrame IP, prenese v pravilnem zaporedju, združi in preda dalje
- Prenos podatkov je v obliki toka oktetov
- 2 izjemi (prenosne plasti)

1. URGENT ... nujno pošiljanje
2. PUSH ... podatki se takoj pošljejo na drugo stran (npr. pri tipkanju)

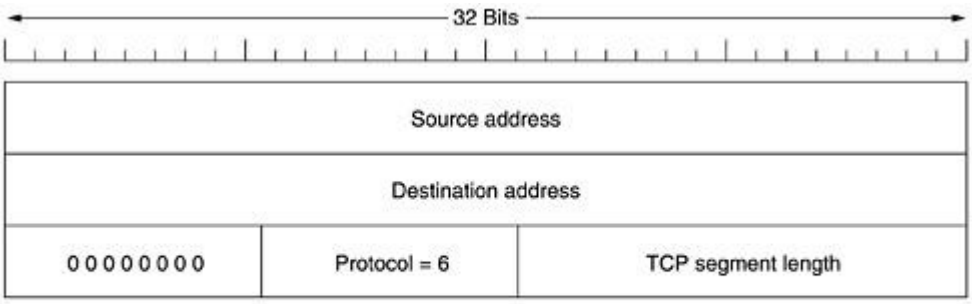
Glava segmenta TCP



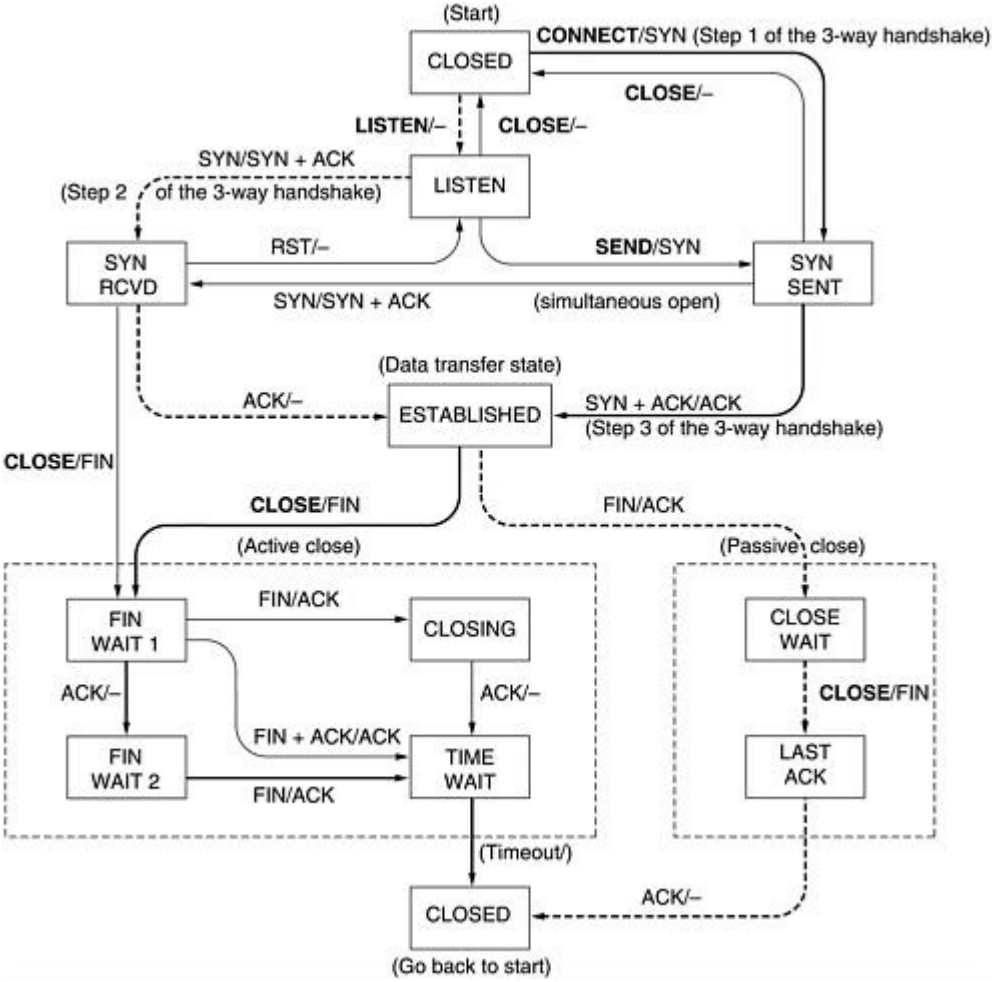
Source port ... številka izvornih vrat
Destination port ... številka ponornih vrat
Sequence number/Acknowledgement ... zaporedna številka zadnjega zloga in zaporedna številka naslednjega pričakovanega zloga. Številki morata biti naključni, zaporedni, povezava ne sme biti "živa".
Window size ... velikost drsečega okna (koliko podatkov lahko sprejemnik sprejme)
Checksum ... kontrolna vsota (seštevanje eniškega komplementa vseh besed glave in podatkov, nato pa še izračun eniškega komplementa vsote)
Options ... dodatna polja (npr. če so paketi večji od 2^{32})
Data ... podatki

Zastavice:
URG ... če je ena pomeni, da polje Urgent pointer vsebuje odmik
ACK ... potrditev
PSH ... prejemnik naj "potisne" podatke aplikaciji brez hranjenja v medpomnilniku
RST ... če je ena, se povezava postavi na začetek (reset)
SYN ... zahteva za vzpostavitev povezave
FIN ... prekinitev povezave

Psevdo-glava TCP



Končni avtomat TCP

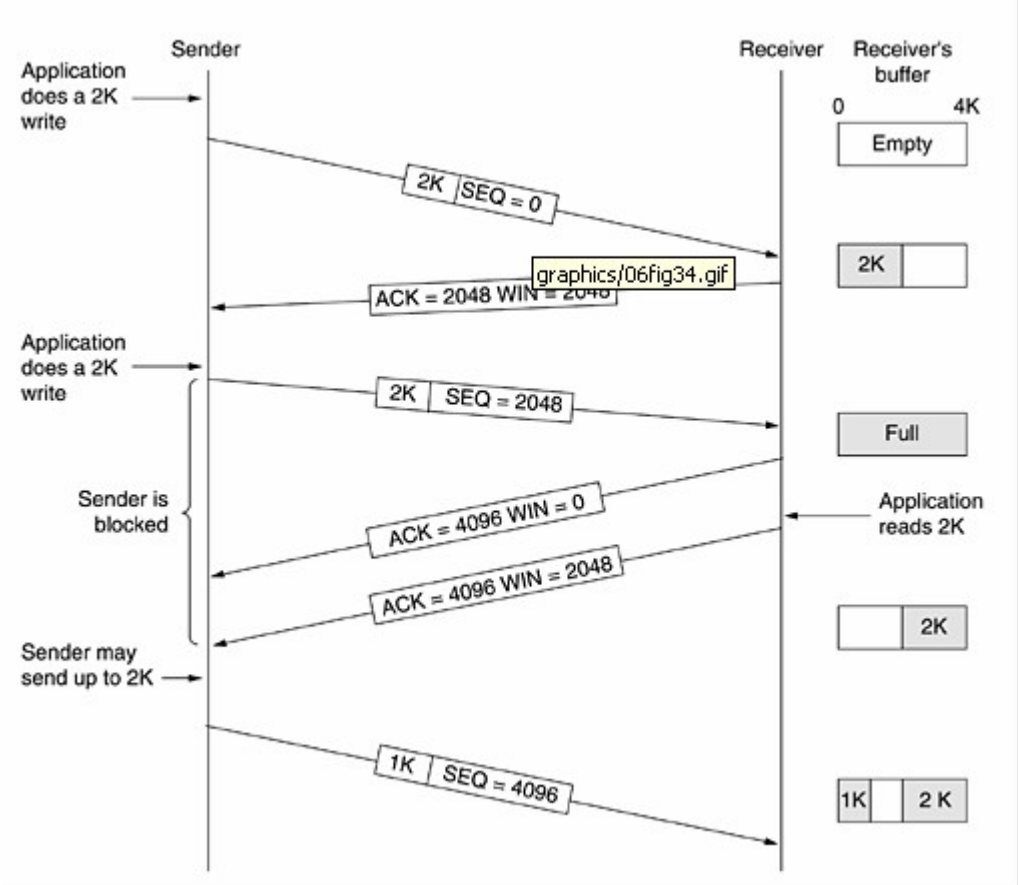


Stanje	Opis
CLOSED	ni aktivnih povezav ali pričakovanja povezav
LISTEN	čakaj na prihajajoč klic
SYN RCVD	zahteva za povezavo prispela, čakaj na ACK

SYN SENT	aplikacija je začela odpirati povezavo
ESTABLISHED	stanje normalnega prenosa podatkov
FIN WAIT 1	aplikacija pravi, da je končala
FIN WAIT 2	druga stran se strinja z razpustom
TIMED WAIT	počakaj, da paketi odmrejo
CLOSING	sočasna obojestranska zaključitev povezave
CLOSE WAIT	sogovornik je iniciral zaključek povezave
LAST ACK	počakaj, da paketi odmrejo

Upravljanje oken pri protokolu TCP

Pri TCP upravljanje oken ni neposredno povezano na potrditve (ang. ACKs). Prejemnik in pošiljatelj oba posedujeta medpomnilnik, kjer hranita segmente sporočila. Potek pošiljanja z uporabo medpomnilnika prikazuje naslednja slika.



- Prejemnik ima medpomnilnik velikosti 4KB
  - Pošiljatelj pošlje segment dolžine 2KB.
  - Prejemnik ga pravilno prejme in pošlje potrditev, kjer oznani okno 2K.

- Pošiljatelj pošlje nov segment dolžine 2K, ki ga prejemnik pravilno prejme, a ima prvi segment še vedno v medpomnilniku, zato pošlje potrditev (ACK), oznani pa okno velikosti 0.

- V zgornjem primeru nastopi pojem okna (ang. Window), ki predstavlja informacijo o tem koliko podatkov je prejemnik še sposoben prejeti.

- Ko je velikost okna 0 mora pošiljatelj prenehati z oddajanjenjem, edini izjemi sta:

    - nujni podatki, da uporabnik lahko ustavi oz. Prisilno zaključi (angl. *kill*) oddaljeni proces;

    - pošiljatelj lahko pošlje 1 B segment, da prejemnik ponovno pošlje št. naslednjega pričakovanega zloga in velikost okna.

- Opazimo, da je za vsak poslani segment potrebna potrditev. Tako potrditve večkrat po nepotrebnem prispevajo k povečanem prometu med pošiljateljem in prejmenikom.

- PRIMER:

    - Prek TELNET-a poganjamo urejevalnik, kjer imamo reakcijo na vsako tipko.

    - To sproži pri pošiljateljevem TCP procesu kreiranje 21 B segmenta, ki ga preda sloju IP, ki kreira 41 B datagram.

    - Po vročitvi proces TCP takoj kreira potrditev (20B glava TCP z ACK in 20 B glava IP).

    - Ko editor prebere znak, TCP oznani novo velikost okna, kar zahteva 40 B.

    - Ko pa editor sprocesira znak, vrne 41 B echo → **torej 162 B prometa za en bajt interakcije z urejevalnikom!**

Trudimo se optimizirati protokol, tako, da bo promet čim manjši, ne da bi povečali entropijo. Ena strategija za optimizacijo je zakasnjevanje potrditev. Na tej ideji temelji tudi **Nagle-ov algoritem.**

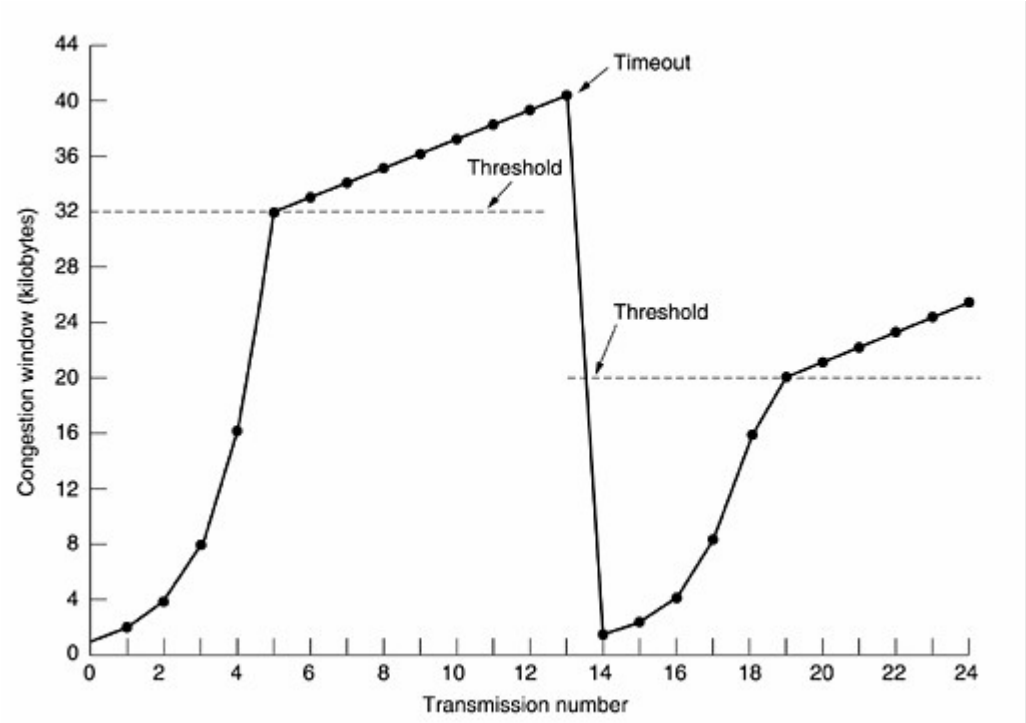
- • Ko pošiljatelj prejema podatke kot posamezne oktete, mora kar takoj poslati prvi oktet (zlog).
- • Pošiljatelj nato nalaga v medpomnilnik prihajajoče oktete, dokler ne prejme potrditve prvega zloga.

- • Po prejemu posredne potrditve pošlje znake iz medpomnilnika v enem segmentu TCP in nalaga v medpomnilnik nove do naslednje potrditve.

Nadzor zamašitve pri TCP

Včasih, ko so bili prenosni mediji manj zanesljivi in je verjetnost šuma bila veliko večja kot danes, se je težko ugotovilo, kaj je povzročilo izgubljene pakete. Danes lahko z veliko verjetnostjo napovemo, da je za izgubljene pakete odgovorna zamašitev, ki se lahko zgodi zaradi prejemnika bodisi zaradi prepustnosti prenosnega medija. Zato moramo pri velikosti okna upoštevati oboje. Torej je velikost okna enaka velikosti manjšega od dejavnikov (kapacitete povezave in velikosti prejemnikove mize). Pošiljatelj pa seveda pred pošiljanjem ne ve kakšna je priporočljiva velikost okna. En način, kako to ugotoviti je algoritem **slow start.** Pošiljatelj exponentno povečuje velikost okna tako dolgo dokler ne preteče časovna omejitev odgovora ali dokler ni dosežena velikost prejemnikovega okna. Na primer, če pošiljanje z oknom velikosti 1024,2048,4096 B uspe, pri pošiljanju z velikostjo 8192 B pa se časovnik izteče, potem je boljše izbrati za velikost okna 4096 B.

Še en algoritem je **Internet congestion control** algoritem. Ta uporablja dodaten parameter imenov prag ali zgornja meja (**angl. threshold**).



Na sliki vidimo, da je prag okna 32 KB. Od te meje naprej se okno povečuje le linearno. Če pa pride do neke velikosti, kjer časovna omejitev odgovora poteče, dobi prag vrednost polovice velikosti pri kateri je prišlo do poteka časovnika. V našem primeru je časovnik potekel pri velikosti okna 40 KB, torej je nov prag 20 KB.

Če ne pride do nobenih potekov časovnika bo velikost okna naraščala tako dolgo dokler nebo enaka velikosti okna prejemnika.

Upravljanja časovnikov pri TCP

TCP uporablja več časovnikov od katerih je najbolj pomemben časovnik za ponovno pošiljanje (**retransmission timer**). Časovnik za ponovno pošiljanje se vklopi vsakič ko pošljemo nov segment. Če se časovnik izteče preden do pošiljatelja dospe potrditev, se poslan

segment pri katerem je prišlo do izteka pošlje ponovno. Kako določiti časovno mejo za ponovno pošiljanje pa predstavlja pereč problem.

Jacobson je predlagal dinamičen algoritem, ki bazira na nenehnem merjenju zmogljivosti omrežja. Za vsako povezavo TCP hrani spremenljivko RTT (angl. round trip time), ki napove najboljši približek krožnega časa, to je čas, ki preteče od takrat, ko pošiljatelj pošlje segment do takrat, ko dobi potrditev.

$$RTT = \alpha RTT + (1 - \alpha)M$$

Če potrditev prispe pred iztekom časovnika se krožni čas zapiše v spremenljivko M. Vrednost α v enačbi pove koliko pozornosti posvetimo oziroma koliko pomembnosti pripišemo stari vrednosti RTT in koliko izmerjeni vrednosti M. Ponavadi vzamemo α =7/8.

Torej RTT imamo, potrebno je še določiti čas izteka časovnika. Jacobson je zato predlagal še eno spremenljivko. Poimenoval jo je D in bi naj spominjala na standardno deviacijo.

$$D = \alpha D + (1 - \alpha)\ |RTT - M|$$

Princip računanja je enak kot pri izračunu RTT. Na podlagi vrednosti alfa določimo pomembnost starega D in pomembnost na novo izračunane vrednosti | RTT – M |.

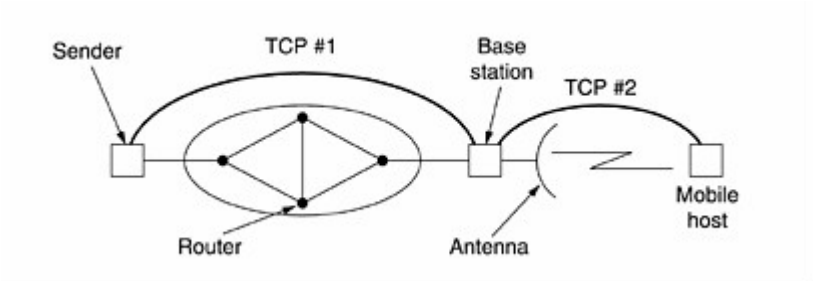
Tako pridemo do enačbe za iztek časovnika.

$$\text{Timeout} = RTT + 4 \times D$$

Brezžično omrežje, TCP in UDP

V teoriji transportni protokoli naj nebi bili odvisni od tehnologije in medija po katerem se prenašajo podatki. V praksi pa vemo, da je TCP bil optimiziran za trdo ožičeno omrežje. V takem omrežju predpostavljamo, da so v glavnem zamašitve odgovorne za izgubljene pakete. Brezžično omrežje ima druge lastnosti, saj je verjetnost šuma pri prenosu podatkov veliko večja kot pri ožičenem omrežju. Implementacija TCP v brezžičnem omrežju nebi vodila v nepravilno delovanje, vendar bi veliko izgubili na zmogljivosti. Če predpostavimo, da brezžično omrežje izgubi v povprečju 20 % paketov bi pri prenosu 100 paketov/s do prejemnika prispelo 80 paketov/s. Pri uporabi opisanih algoritmov za nadzorovanje zamašitev, bi potem pošiljatelj zmanjšal pošiljanje na 50 paketov/s. In, če še vedno izgubimo 20 % paketov, bi na nasprotno stran prišlo le 40 paketov/s. Torej si želimo, da namesto, da pošiljatelj upočasni pošiljanje, se raje potrudi še bolj.

Problem nastopi, ker velikokrat med pošiljateljem in prejemnikom ni homogenega omrežja. Rešitev, ki sta jo predlagala Bakne in Badrinath je **indirect TCP.** TCP povezavo razdelimo na dve ločeni povezavi kot prikazuje spodnja slika. Bazna postaja pa samo kopira pakete in pošilja naprej.

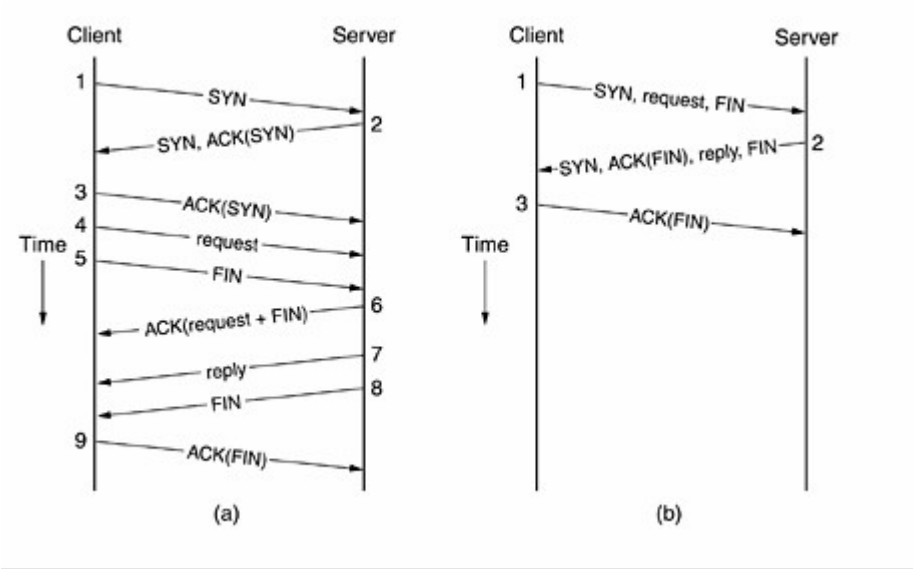


Pozorni pa moramo biti na semantiko TCP. Če pošiljatelj dobi potrditev še to ne pomeni, da je paket zares prišel do prejemnika. Bazna postaja mora biti pozorna na potrditve, ki pripotujejo po brezžičnem omrežju. Če za določeni paket ne dobi potrditve, ga pošlje ponovno, pošiljatelj pa tega sploh ne zazna. Nezanesljivo brezžično omrežje bi zato lahko povzročilo iztek časovnika pošiljatelja in ta bi začel z algoritmom za nadzorovanje zamašitev.

Tudi UDP protokol ima svoje težave z brezžičnim omrežjem. Programi, ki uporabljajo UDP smatrajo, da gre za zanesljivo skoraj popolno povezavo, ki pa je daleč od tega. To predvidevanje večkrat vodi v obupno zmogljivost komunikacije

Transakcijski TCP

Spodnja slika (a) prikazuje potek RPC (Remote procedure call) pri TCP protokolu. Zanesljivost TCP protokola je tudi njegova poguba, kar je dobro razvidno iz števila paketov, ki jih je potrebno prenesti pri RPC.



Slika b prikazuje izboljšan TCP protokol, T/TCP (Transactional TCP). Vidimo, da se je število prenesenih paketov zmanjšalo za več kot polovico. T/TCP protokol malo spominja na UDP, vendar ohranja značilnosti TCP.

Zmogljivosti omrežja

Da bi čim bolj optimizirali omrežja so potrebne nenehne meritve. Tako nam meritve nudijo rezultate na podlagi katerih si lahko razložimo dogajanje v omrežjih.

Eden izmed pokazateljev, ki nam pomaga oceniti obremenjenost posamezne povezave, je produkt pasovne širine in zakasnitve (rezultat je v bitih). Vzemimo za primer 1Gbps povezavo iz San Diega do Bostona. Da pošiljatelj odda vse namenjene pakete porabi 500 mikro sekund časa. Da iz Bostona dobi potrditev preteče 40 mili sekund. Torej je izkoriščenost povezave le 1.25 %. Prej omenjen produkt pa znaša 40 milijonov bitov, kar pomeni, da če pošiljatelj hoče popolnoma koristiti dano povezavo mora v Boston poslati 40 milijonov bitov.

Kakorkoli so meritve pomembne pri izboljšanju omrežij, pa nam nič ne pomagajo, če je omrežje bilo slabo načrtovano.

Varnost v Internetu

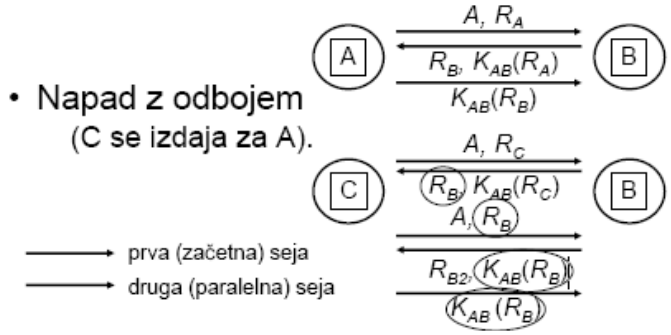
Secure Sockets Layer (SSL) To je protokolni sloj, ki nudi varnostne storitve in je skupen vsem aplikacijam. **Cilj protokola SSL je zagotavljanje zasebnosti in varnosti med dvema aplikacijama, ki komunicirata preko nezavarovanega komunikacijskega kanala** SSL vzpostavi varen kanal z overjanjem in izmenjavo **sejnih ključev** s pomočjo sledečega protokola:

- odjemalec pošlje strežniku sporočilo “Hello”
- strežnik vrne svoj lastni certifikat – za identifikacijo strežnika
- odjemalec kreira sejni ključ in ga pošlje strežniku šifriranega z javnim ključem strežnika (lahko preveri, ali je ključ veljaven)
- strežnik potrdi prejem
- tako strežnik in odjemalec varno izmenjata ključe

Kriptografski protokoli za overjanje Overjanje s pomočjo skupnega ključa.

- *A* pošlje *B* sporočilo s svojim imenom
- *B* vrne *A*-ju izziv R_B (random number- A in B se s tem prepričata, da imata opravka s svežo, novo sejo)
- *A* kriptira (z Bj) izziv R_B in ga pošlje *B*-ju
- *A* pošlje *B*-ju izziv R_A.
- *B* kriptira izziv R_A in ga vrne *A*-ju

Optimizacija predhodnega protokola.



Kriptografski protokoli za vzpostavitev sejnega ključa – **Diffie-Hellman protokol**

• Temelji na kompleksnosti izračuna diskretnega logaritma.

• Izberemo dve veliki praštevili *n* in *g*, ki izpolnjujeta določene pogoje.

• *A* si izbere veliko skrivno število *x*, *B* pa *y*.

• Potek: • *A* pošlje *B* g_x mod *n*, *B* pa *A* g_y mod *n*. (potenciranje in računanje po modulu sta počasni operaciji) • *A* in *B* izračunata g_{xy} mod *n* in ta vrednost predstavlja **sejni ključ** (da se zgornji izračuni ne ponavljajo vsakič, ko se pogovarjata).

Protokol se izkaže za neučinkovitega v primeru posrednika: »Man in the middle attack«

Problem je izmenjava neoverjenih paketov
A verjame, da je C = B problematično, ko se A in B pogovarjata o sejnem ključu Sorodni problemi (ponovitve paketov).

IPsec – IP Security

IPsec se uporablja za varnejši, kriptiran promet po omrežju. Ob vzpostavitvi povezave se uporabita javna in zasebna ključa, nato pa se generira krajši, sejni ključ. Le-ta je krajši, kar je pomembno, saj se tako lahko kodiranje/dekodiranje izvaja hitreje in manj ovira pretok podatkov.Kljub temu so omenjene operacije procesorsko zahtevne, zato se pogosto uporablja strojna implementacija.
IPsec paketi:

Kriptiranje poteka precej nizko – že na IP nivoju. V IP glavi je polje, ki lahko omogoči *extension*, kar se uporablja pri IPsec-u. Za IP glavo se tako vstavi ESP (Encapsulated Security Payload) glava, ki vsebuje 64-biten *index* (zap. številka) in avtentikacijske podatke, s katerimi lahko potrdimo veljavnost kriptiranih podatkov v paketu. Le-ti so podpisani po prej omenjenih postopkih/metodah. Sledi dejanska vsebina paketa, torej sami podatki, ki so enkriptirani. Ker smo omenili, da poteka enkripcija že na IP nivoju, se tu nahajajo tudi recimo TCP/UDP glave in podobne zadeve, ki jih najdemo na višjih nivojih. Nazadnje je v paketu podpis za preverjanje, ki bi se sicer lahko nahajal pred podatki, kar bi bilo organizacijsko morda bolj logično, razlog za njegovo lokacijo pa je predvsem bistveno lažja strojna izvedba preverjanja podatkov. Le-ti se lahko spustijo čez določen čip, ki postopoma na njih izvaja hash in ga nazadnje primerja s podpisom. Če bi se slednji nahajal na začetku, bi ga bilo treba začasno shraniti v nekakšen medpomnilnik, dokler ga ne bi rabili po prejemu ostalih podatkov.
ESP in AH se pri IPv4 prometu sicer lahko uporabljata tudi skupaj, kar pa je precej redek pojav, ponavadi se enega od njiju izbere glede na potrebe zaščite. Drugače je pri IPv6, tam sta tako ESP kot AH že standardna komponenta glave in

se vedno uporabljata.

VPN – Virtual Private Network

Virtualna zasebna omrežja se uporabljajo, ko želimo preko interneta varno povezati manjša, zasebna omrežja. Znotraj the omrežij je promet varen in lahko poteka nezaščiteno, medtem ko moramo pri prometu, ki ga pošiljamo čez internet, poskrbeti za varnost. Poleg povezave med dvema omrežjema se VPN pogosto uporablja tudi za povezovanje samo enega sistema, ki se nahaja nekje v internetu, z oddaljenim omrežjem. V tem primeru se ponavadi uporablja programska enkripcije, saj nakup strojne opreme ni smotrn. Pri tem nastanejo določeni problemi – med zunanjimi in notranjimi omrežji se ponavadi nahaja NAT, ki skrbi za spremembo IP naslova iz zunanjega v notranjega in obratno. V tem procesu se paketom seveda spremenijo IP glave, kjer pa nastane težava. Znotraj podatkov v IP paketu, ki so v tem primeru zaradi uporabe IPsec enkriptirani, se recimo v TCP ali UDP paketih v psevdo-glavi nahaja checksum, ki je izračunan na podlagi določenih podatkov, med drugim tudi izvornega in ciljnega IP naslova. Ker se ta naslova, ko prečkata NAT, paketu spremenita, checksum ni več pravilen, popravi pa se ga ne, ker je že enkriptiran znotraj podatkovnega dela IP paketa. Ta problem se rešuje s pomočjo procesa, ki se imenuje NAT Traversal, prav tako pa mora usmerjevalnik, ki je točka z zunanjim svetom, omogočati t.i. *NAT passthrough*, ki vključuje posredovanje prometa z določenih vrat do končnega prejemnika.

Varnost v internetu

Asimetrični sistem + overjanje
Pri asimetričnih sistemih je ključ prirejen osebku, pri simetričnih pa povezavi.
• V vsakodnevni uporabi kombinacije obeh vrst šifriranj:
– asimetrična za izvajanje overjanja
– simetrična za šifriranje velikih množin podatkov.
• Temeljni problem upravljanja ključev: Kako zagotoviti, da nek (javni) ključ res pripada tistemu osebku, ki naj bi mu pripadal?
Upravljanje s ključi
Temeljne razlike

- rast števila ključev: simetrični sist. O(n2) varuje povezavo med osebkoma⇓
  Asimetrični sist. O (n) varuje osebek⇓
- razlika v »ceni« igra ključno vlogo pri izbiri kriptiranja

Pri kriptiranju imamo vedno opraviti s povezavnimi storitvami.
Poznamo javne in zasebne ključe s katerimi zakriptiramo sporočilo.
Kreiranje ključev:
Naključen niz
-generator psevdonaključnih števil
-metanje kovanca
Hranjenje ključev
-pomnenje (človek)

-pametne kartice s PINi
Postavitev strukture overiteljev (angl. certification authoritites ali CAs) in uvedba overjenih dig. potrdil (angl. certificates).
Pri agenciji za certificiranje, lahko zaprosimo za osebno digitalno potrdilo.

Asimetrični postopki – RSA

Zahteva modul n, ki je produkt dveh velikih praštevil p in q (n = p* q).
B naj bo osnovni tekst, T skritopis, e enkripcijski ključ in d dekripcijski ključ:

T=Be mod n
B=Td mod n.
Za e mora veljati, da je tuj (p-1) * (q-1), za d pa e*d º 1 (mod (p-1)*(q-1)).
Enkripcijski ključ (e,n)
Dekripcijski ključ (d,n)

Primer:

Naj bo p = 11 in q = 13, njun produkt je torej n= 143.
(p-1)(q-1) = 10*12 = 120.
– izberemo e tako, da je tuj številu 120 - naj bo to 11.
e*d mod 120 = 1. > Ugotovimo, da je d = 11, ker e*d mod 120 = 1 (enkripcijski in dekripcijski ključ sta v našem primeru enaka).
• Enkripcija in dekripcija.

Naj bo naš čistopis B = 8.
– Tajnopis izračunamo z 8^11 mod 143 = 96, torej T = 96.
– Kontrola – izračun čistopisa iz tajnopisa:
96^11 mod 143 = 8 = B.
– Izračun je torej pravilen, dobili smo osnovni tekst, to je čistopis.

APLIKACIJSKI SLOJ

Uvod:

Skrbi za **interakcijo z uporabniki**. Upravlja s prenosom datotek ter protokoli oddaljenega dostopa. **Aplikacije**, ki so na vrhu strukture plasti **dostopajo do spodnjih slojev** preko **vtičnic,** ki predstavljajo samo vmesnik (vse druge plasti so nekako skrite).

Ker navadne vtičnice o varnosti ne vedo nič, je za varnost lahko poskrbljeno na mrežni plasti(IPsec, podpira UDP in TCP), ali pa z **varnimi vtičnicami** SSL(Secure Socket Layer), ki pa podpirajo le TCP.

Aplikacijski sloj sestavljajo:

- • Sistem domenskih strežnikov (angl. *Domain Name System* - DNS
- • E-poštni sistem (angl. *e-mail*).

- • Novičarski sistem (NEWS).
- • Svetovni splet (protokol HTTP in jezik HTML).
- • Večpredstavnost (angl. multimedia).
- • Protokol za upravljanje omrežij (*Simple Network Management Protocol* – SNMP).

Sistem DNS (Domain Name System)

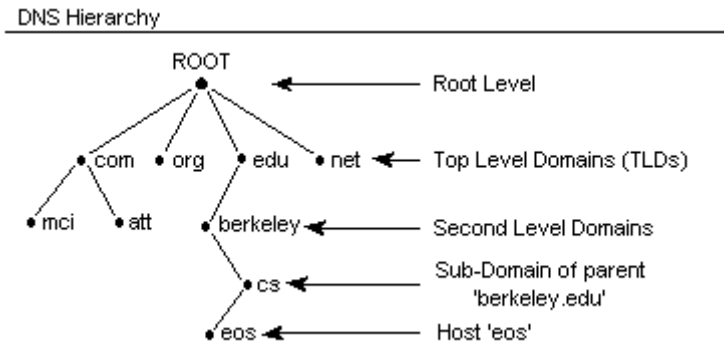
- • Računalniki komunicirajo z 32 bitnimi (IPv4) oz. 128 bitnimi (IPv6) naslovi IP.
- • Deluje kot RPC (Remote Procedure Call), torej z UDP paketi

Dva razloga za DNS:

1. **Predstavitev v bitni obliki je pretežka** za navadne uporabnike, zato se poslužujemo poimenovanja računalnikov z "običajnimi" (človeku lažje dojemljivimi) imeni. Tudi ime domene je tržni material.

2. Pojavi se **problem preslikav med množico („običajnih") imen in naslovi IP**. (potrebujemo sistem, ki pove celemu svetu, da se je IP za nek naslov spremenil)

- • **Spočetka** (v Arpanet) za preslikave služi datoteka ***hosts.txt***, ki si jo izmenjujejo računalniki, vendar sta se pojavila naslednji težavi:
  - problem decentraliziranega upravljanja omrežja
  - problem fizične rasti omrežja
- • **Rešitev je porazdeljen in hierarhično strukturiran sistem**
- • Celotni imenski prostor je razdeljen na **domene,** te naprej na **poddomene,** itd.
- • Imamo **drevesno strukturo**:



  - **koren je neimenovan** (to je začetna domena, ki jo označujemo s "**.**")
  - v vozliščih so domene nižjih nivojev (poddomene)

- v listih so domene najnižjega ranga, ki nimajo več poddomen

- • **ICANN** je organizacija, ki določa zgornje krovne domene. Organizacija se odloča, če se vpelje nova krovna domena. (Za vpeljavo mora obstajati dober razlog).

- • **Vrhovne domene** (počasi se dopolnjujejo):

1. **generične**, npr. *int, com, edu, net, mil, gov, org* 2. **državne**, npr. *si, us, de, au, ca, uk, at, it* (standard ISO 3166)

- • Poddomene (vozlišča) so lahko **dolžine** do **največ 63 znakov**, **niz poddomen** (pot od poddomene do korena) pa **ne sme biti daljši od 255 znakov**.
- • Celotno ime je niz (pod)domen, ki so ločena s pikami.

- • **Za kreiranje poddomene potrebujemo dovoljenje nadrejene (pod)domene**:

  - upravljavec nadrejene (pod)domene dodeli enolično ime znotraj svojega imenskega prostora

  - vzpostavi ustrezne kazalce na strežnike, ki so odgovorni za novo domeno

  - princip se lahko ponavlja na enak način na naslednjem nižjem nivoju.

Primer:

fri.uni-lj.si -> Upravljalec (pod)domene **fri** potrebuje soglasje (pod)domene **uni-lj**, ta potrebuje soglasje upravljalca (pod)domene **si**(pri nas je to **ARNES**), ta pa potrebuje soglasje **ICANN(.)** za kreiranje te domene..

- **Kako registrirati domeno?**
- Domeno registriramo preko ponudnika domenskih storitev in ne direktno preko upravitelja domene. Upravitelj domene samo določi pravila, ki veljajo pri registraciji.

- • S pomočjo storitve **WHOIS** lahko pogledamo, **kdo** je **lastnik** neke domene.

- • Imena ↔ organizacijska shema / naslovi ↔ struktura omrežja.

Kako deluje DNS sistem?

Uporabniški program (npr. Internet Explorer) vpraša za storitev (prevod nekega imenskega naslova v IP naslov) svojega tolmača (resolver). Tolmač na različne načine poskuša ta naslov spremeniti v IP:

- z lokalno datoteko
- vpraša oddaljeni DNS strežnik (za to se uporablja DNS protokol)
- z lokalnim DNS strežnikom (tudi ta komunicira z oddaljenim strežnikom le da vsebuje še svoje master datoteke - primer lokalna mreža, kjer ima vsak računalnik svoje ime)

Za vsako krovno domeno obstaja en glavni strežnik (primarni). Za bolj varno delovanje se ponavadi uporablja tudi sekundarni strežnik, ki prevzema njegove funkcije ob primer ne delovanja primarnega.

V glavni (master) datoteki so navedeni vsi podatki za to domeno v obliki zapisov:

Type	Meaning	Value
SOA	Start of Authority	Parameters for this zone
A	IP address of a host	32-Bit integer
MX	Mail exchange	Domain willing to accept e-mail
NS	Name Server	Name of a server for this domain
CNAME	Canonical name	Domain name
PTR	Pointer	Alias for IP address
HINFO	Host description	CPU and OS in ASCII
TXT	Text	Uninterpreted ASCII text

Struktura posameznega zapisa je naslednja:

<name> <ttl>? <class>? <type> <data>

- name - ime
- ttl - time to last (koliko časa naj računalnik potem to domeno drži v predpomnilniku)
- class - razred
- type - tip
- data - podatki

Imesnki naslov se preslika iz desne proti levi.

Sistem DNS

Primer slovnice za tip (type) SOA.

- <name> <ttl> <class> SOA <origin> <person>
- (<serial><refresh><retry><expire><minimum>)

- *SOA označuje začetek domene, <name> je ime domene, <origin> pa ime primarnega računalnika* za to domeno;

- <person> - naslov skrbnika (“.” namesto “@”);

- <serial> je različica datoteke, <refresh> označ č uje as, v katerem mora sekundarni računalnik osvežiti podatke,

- <retry> označuje, kako dolgo mora sekundarni strežnik poskušati dobiti sveže podatke;

- <expire> podaja zgornjo mejo veljavnosti podatkov na sekundarnem strežniku,

- <minimum> pa je minimalna vrednost za TTL za zapise

- DLE. – se uporablja namesto @

Primer preslikovanja naslova cs.yale.edu:

- naš zunanji DNS server flits.cs.vu.nl nima prave datoteke in zato vpraša svojo naddomeno cs.vu.nl za domeno ki jo hoče dobiti(v tem primeru .edu).
- edu-server.net ugotovi kdo je domenski strežnik za to domeno, ker ne ve gre vprašat poddomeno yale.edu.
- .edu nas napoti na cs.yale.edu, ki ima prave podatke

Zanimivo:

Obstaja le 10 posebnih strežnikov(naslovov), ki vejo kje so krovni strežniki za vsako od domen. So zelo statični, večina jih je v ZDA, nekaj pa v Evropi in na Japonskem. Za lokacije ve le ICANN.

Master datoteka lahko vsebuje več zapisov za isto ime. Najbolj pogosta uporaba tega je, da se uporabniku vrne tisti naslov, ki je njemu najbližje.

E-POŠTA

- druga storitev aplikacijske plasti
- RFC=request for comments; tj. internetni standard.
- RFC821 in RFC822 sta definirala, kako deluje poštni sistem
- RFC821 opisuje kako se 2 računalnika "pogovarjata", ko si izmenjujeta pošto (protokol SMTP)
- RFC822 opisuje kako naj izgleda samo e-poštno sporočilo (format sporočila)

RFC821:

- SMTP(Simple Mail Transfer Protocol): skrbi za učinkovit in zanesljiv prenos sporočila iz enega na drug računalnik. Ta protokol se je uveljavil leta 1969 in ga uporabljamo še danes. SMTPju konkurenčna funkcija je bila X.400, ki je bila namenjena pošti, ampak ta sistem je že izumrl. Danes je še vedno ohranjen sistem X.500, ki ga uporabljamo pri varnosti.

GLAVNE KOMPONENTE POŠTNEGA SISTEMA:

-User Agents so uporabniški programi, preko katerih dostopamo do e-pošte. Osnovne funkcije so sestavljanje, pošiljanje, prikazovanje in urejanje. Primeri user agentov: Outlook, Thunderbird.
-Mail Servers (tudi transfer agents) so serverji, ki posredujejo pošto dalje
-Protokoli kot so SMTP, POP, IMAP

IZGLED POŠTE(RFC822):

- 2 dela: glava in telo, med njima pa je ena prazna vrstica, ki ju ločuje.
- glava: sestoji iz posameznih polj(začne se s črkovnim nizom kateremu sledi dvopičje in vsebina). **Obvezna polja** so: *To* (za koga je sporočilo, naslov), *From* (pošiljatelj sporočila), *Date* (datum); **opcijska polja** pa so *Subject* (zadeva), *Reply-to (odgovori)*, *CC* (carbon copy; v vednost), *BCC*(blind carbon copy; ostali prejemniki ne vedo za prejemnike navedene v tem polju).
- telo: niz znakov, ki dopušča 1000 znakov v vrstici. Vsebuje čisti tekst(7-bitni, printable ASCII tekst; oglati oklepaji in ostali "nenavadni" znaki niso dovoljeni).

POMEN POLJ:

- Sender: naslov aktualnega kreatorja sporočila(tajnica)
- From: naslov pošiljatelja(šef). To polje se da zlahka ponarediti in mu zato ne smemo preveč zaupati (rešitev: pošiljatelj uporabi digitalni podpis)
- Recieved: 1.polje je naslov zadnjega prejemnika pošte, 2.polje je naslov predzadnjega prejemnika...
- Reply-to: naslov tistega,kamor naj pošta potuje, če odgovoriš na mail (poljuben naslov, ki je neodvisen od

naslova pošiljatelja).
- Message-Id: to polje določa vsebino poslane pošte(identifikacija sporočila).
- Return path: po kateri poti naj pošta potuje nazaj-k pošiljatelju (danes se skoraj ne uporablja)
- In-reply-to: določa na kateri message id odgovajamo - tako imamo lahko nit sporočila in lahko imamo pošto urejeno po pogovoru (iste pogovore imamo združene)
- Poleg znanih polj pa si lahko uporabniki (ali ponudniki poštnih predalov ali User Agent) izmislijo svoja polja za privatno uporabo, ki pa se morajo začeti z X- (npr. polje X-Vnesite-tekst-vasega-polja).
- Seznam vseh standardnih polj glave si lahko ogledate na naslovu http://www.iana.org/assignments/message-headers/perm-headers.html

POŠILJANJE POŠTE:

- uporabniški vmesnik komunicira s SMTP sistemom, ki skrbi za prenos sporočil(RFC821).
- komunikacija med uporabniškim sistemom in poštnim sistemom:
- programa se dogovorita, kje bosta shranjevala podatke(izhodni "spooler"); - vhodni "spooler", tudi uporabniški nabiralnik;pošta je shranjena nekje na disku
- na Unix sistemih imamo 2 načina pri vhodnem "spooler"; mailbox format-vsa pošta v eni sami veliki datoteki (starejša zadeva - razlog za to je v tem da je zasedeno manj prostora na disku (vsaka datoteka zasede vsaj eno gručo!)); (nek drug format) poštnemu predalu se dodeli imenik na disku, kjer je vsako sporočilo shranjeno v svoji datoteki (ta način je novejši)

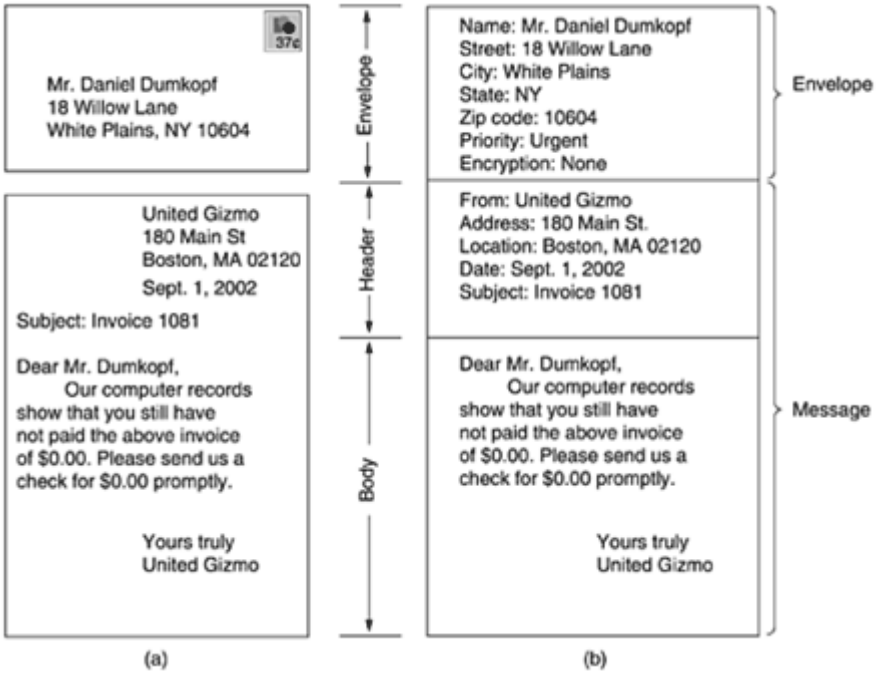
ZASEBNOST:

- elektronska pošta običajno ni zakodirana;
- sporočila morajo iti na svoji poti do cilja skozi vmesne računalnike, kar pomeni, da je drugim osebam relativno enostavno prestreči in prebrati sporočila;
- mnogi ISP-ji si shranijo kopije sporočil na svojih poštnih strežnikih, preden se sporočila dostavijo. Te kopije lahko ostanejo shranjene več mesecev, četudi jih vi izbrišete iz vaših nabiralnikov;
- polja Recieved in preostale informacije v sporočilu običajno določijo pošiljatelja in tako preprečujejo anonimno komunikacijo.

E-POŠTA

E-pošta simulira navadno pošto, njen format se ni spremenil že od okoli leta 1970 naprej. Sestoji iz *ovojnice* in *sporočila*, sporočilo pa je sestavljeno iz dveh delov: *glave* (opis

sporočila) in *telesa* (vsebina sporočila):



Standard RFC821 govori o tem, kako izgleda komunikacija med dvema računalnikoma, ki bi si rada izmenjala e-pošto.

Komunikacija poteka po TCP/IP - definiramo naslov na katerega bi radi prenesli pošto.Ciljni naslov dobimo preko name serverja kamor pošljemo zahtevek za MX zapis: Preko DNS sistema ugotovimo IP število.

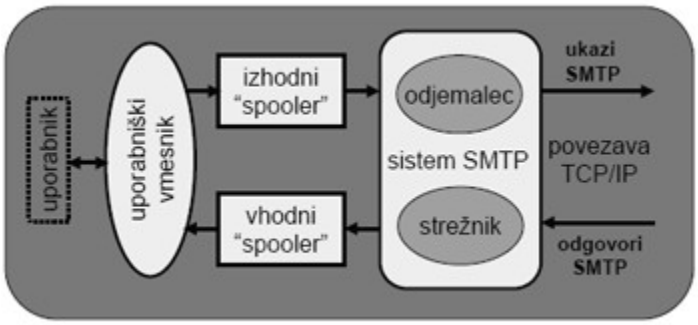
postni_predal@domena

@ = at , slov. grdi prevod: afna ali bolše 'pri' (če hočte bit kul)

<u>MX zapis:</u>
-mail exchange
-zapis poštnega strežnika, kamor se priklopimo

Kako poizvedujemo v DNS sistem? Povpraševanje po zapisu poteka prek ukazov (s katerimi si ogledamo posamezne zapise v bazi) :
-dig
-nslookup

Komunikacija se vzpostavi preko TCP/IP protokola. IP smo določili zgoraj, za vzpostavitev komunikacije potrebujemo še vrata (port).



Pri pošiljanju imamo opravka s SMTP (Simple Mail Transfer Protocol) protokolom , ki operatuje na vratih s številko 25.
(vrata definira organizacija IANA)
Ko pošto oddamo v "poštni sistem", nimamo nobenega zagotovila, da bo prejemnik dejansko prejel pošto. Zanesljivosti prenosa v tem pomenu praktično ni.

Tisti, ki pravijo da je to staromoden način komuniciranja se motijo.

Primer dialoga - potek protokola :

S: <strežnik čaka na vratih 25 na odjemalca>
S: 220 www.primer.com ESMTP Postfix
C: HELO mojadomena.com
S: 250 Hello mojadomena.com
C: MAIL FROM: <posiljatelj@mojadomena.com>
S: 250 Ok
C: RCPT TO: <prijatelj@novadomena.com>
S: 250 Ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>
C: Subject: test message
C:
C: Pozdravljen,
C: To je samo test.
C: Lep pozdrav.
C: .
S: 250 Ok: queued as 12345
C: QUIT
S: 221 Bye

Razlaga protokola:

Uvodni del:
- odjemalec pošlje zahtevo za povezavo
- strežnik vzpostavi povezavo s sporočilom "220 ...."
- odjemalec odgovori s HELO, strežnik pa z nizom "250" kar pomeni "OK"
- vsak niz je zaključen z ASCII znakom EOL (End Of Line)

Osrednji del:
- odjemalec predaja sporočila z ukazi "MAIL From:" in "RCPT To:", ki jih strežnik potrdi

- sledijo ukazi "DATA" in predaja elektronskih sporočil, zaključek vsakega je podan z vrstico, ki vsebuje le piko "."
- strežnik ločeno potrdi prejem vsakega sporočila
- odjemalec zaključi prenos z ukazom "QUIT", ki ga strežnik potrdi z odgovorom "221".

TELNET
Je TCP/IP povezava do poljubnih vrat na strežniku (pošilja 1 byte naenkrat - tako kot tipkamo). Z njim lahko komuniciramo z vrsto protokolov (npr. prej omenjeni SMTP - seveda če poznamo ukaze).
Protokol POP3 je TCP/IP protokol, ki omogoča zelo malo. POP3 (Post Office Protocol) prenese e-pošto s strežnika na lokalni disk.
Protokol IMAP (Interactive Mail Access Protocol) je v bistvu nadgrajen protokol POP3 ki omogoča centralizirano administracijo.

Protokol DMSP (Distributed Mail System Protocol) podpira več strežnikov. Kar pride v poštev pri večjem številih uporabnikov.

Protokol MAPI (Messaging Application Programming Interface) je zasebni protokol MS za sodelovanje z MS Exchange strežnikom.

Ker pošta prihaja iz naslovov in z vsebinami, ki jih nočemo imeti se dodajajo "antispam" filtri, lahko pa dodajamo tudi svoje filtre, ki jih kreiramo z orodji za delo z pošto (fetchmail, procmail, postfix, qmail).

E-pošta – MIME

- • MIME (*Multipurpose Internet Mail Extensions*):
- RFCs 1341, 1342, 1521, 1522, 1590, 2045, 2049

- Problemi zaradi, katerih se je pojavil MIME:

    1. jeziki s poudarki (Francoščina, Nemščina…)
    2. ne-latinske abecede (cirilica,hebrejščina…)
    3. jeziki, ki nimajo abecede (Kitjaščina, Japonščina)
    4. sporočila, ki ne vsebujejo texta (vsebujejo video, audio…)

MIME definira 5 novih glav v sporočilu:

Header	Pomen
MIME-Version	Različica MIME
Content-description	Opis vsebine v običajnem zapisu (navadno v Stringu)

Content-ID	Enolični identifikator
Content-Transfer-Encoding	Kako je vsebina kodirana za prenos
Content-Type	Vrsta podatkov v sporočilu

Content-Transfer-Encoding:

Najbolj primerno kodiranje binarnega sporočila je "*base 64*"

Vsebina	R								K								O											
ASCI	82								75								79											
Bit pattern	0	1	0	1	0	0	1	0	0	1	0	0	1	0	1	1	0	1	0	0	1	1	1	1				
Index	20								36								45								15			
Base 64	U								k								t								P			

Kako kodiramo?

Vzamemo ASCII kodo vsakega znaka in jo zapišemo binarno. Nato pa gledamo samo po 6 bitov namesto 8.

Iz tistih 6 bitov tvorimo decimalno število, ki jo pretvorimo v znak po naslednjem ključu (velja angleška abeceda):

A=0 a=26 0=52

B=1 b=27 1=51

….. ….. ……

Z=25 z=51

Kmalu so se potrebe povečale in nastali so novi tipi in podtipi sporočil:

Kratki opisi tipov:

- Text: za prenos tekstovnih sporočil (enriched/plain), text je lahko navaden ali pa formatiran (uporaba »Markup« oznak; npr <bold> odebeljeno </bold)
- Image: za prenos slik. Najpogostejša sta jpg in gif, vendar so na voljo tudi mnogi drugi formati, ki so definirani v RFC

- Audio: za prenos audio sporočil

- Video: za prenos video sporočil. Video sporočila se nanašajo zgolj na vizualno informacijo in ne vključuje audio informacije. Če hočemo tako prenesti recimo film, moramo posebej prenesti audio in video.

- *application* - program s podtipoma:

- *Octet-stream* (neinterpretiran niz bitov) in

- *Postscript*

- *message* – omogoča enkapsulacijo in fragmentacijo sporočil, podtipi so *Rfc822*, p*artial* (fragmentirano sporočilo) in *External-body* (sporočilo vsebuje kazalec na obsežno vsebino, ki se prenese prek npr. FTP)

- *multipart* - omogoča več tipov v pošti, podtipi so *Mixed*, *Alternative*, *Parallel* in *Digest*

<u>Protokol FTP</u>:

- File Transfer Protocol - protokol za prenos datotek
- sam protokol je že zelo star, zato tudi nekaj pomankljivosti
- zagotavlja zanesljiv prenos med datotečnim strežnikom in lastnim računalnikom:
    - pravilno prenešena datoteka in podatki o datoteki
    - pravilno odpiranje/zapiranje
    - upošteva pravilo debelega/tankega konca
- protokol ne vključuje nobene varnosti
- ločuje med binarnim (za prenos je pomembno samo pravilo debelega/tankega konca) in črkovnim formatom (zna transformirat datoteko glede na platformo, npr.: različno označen konec vrstice; unix: ctrl-j , win:ctrl-m-j)
- je neodvisen od okolja (platforme)

Prenos med strežnikom in odjemalcem vključuje dve TCP povezavi (podatkovno in kontrolno povezavo):

- odjemalec strežniku sporoči svoj IP naslov in vrata na katerih pričakuje podatke
- strežnik vzpostavi podatkovno povezavo z odjemalcem (za nadzorni protokol strežnik uporablja vrata 21, za podatkovno povezavo pa vrata 20)

Problemi protokola:

- Odjemalec skrit za NATom:

    - da lahko pravilno preusmeri pakete nazaj k pravemu odjemalcu, mora NAT razumeti FTP in spremeniti vsebino FTP paketov (deep package inspection):
    - NAT v FTP zahtevi menja izvorni naslov in vrata s svojim naslovom in vrati (strežnik vzpostavi podatkovno povezavo z NATom)
    - na poti nazaj do odjemalca mora NAT ponovno menjat naslov in vrata
- FTPS (FTP Secure):
    - nad TCP nivo je dodan SSL
    - prenašajo se zakriptirani paketi
    - Nat ne more delati s kriptiranimi paketi, ker ne more pregledati vsebine paketa --> rešitev: pasivni strežnik - namesto da 2. povezavo odpre strežnik, jo odpre client (strežnik mu sporoči na kateri port naj se poveže)

URI (Uniform Resource Identifier):

```
URI / URL

Osnovna tehnološka inovacija svetovnega spleta je Universal Resource
Identifier (Locator). URI/URL je nadgradnja poimenovanja, ki ga je
uvedel že DNS. Kjer DNS naslavlja računalnike URL naslavlja datoteke.

URL si lahko predstavljamo kot funkcijo, ki smo jo srečali pri Remote
Procedure Call.
Format URL (nekateri elementi se lahko izpustijo):

shema://uporabnik:geslo@streznik.net:1234/pot/datoteka.abc?ime=abc#fragment
```

Ima tridelno strukturo:

protokol za dostop	ime računalnika (domena)	pot do objekta na računalniku
http://	en.wikipedia.org	/wiki/URI

HTTP zahteva:

Uporabnik vtipka url v brskalniki. Ta od DNS sistema dobi IP od strežnika. Vzpostavi se povezava preko vrat 80. Brskalnik pošlje strežniku metodo GET v kateri je pot do objekta na računalniku. Strežnik pošlje objekt odjemalcu. Odjemalec oblikuje element in ga prikaže.

Odjemalec:

Odjemalec skrbi za povezavo s strežniki in prikazovanje vsebine. Vsebina so danes lahko text, slike, zvok, video, interaktivne aplikacije (javascript, flash), oblikovne datoteke(css). Če Odjemalec ne ve kaj narediti z določeno zahtevo ima več možnosti:

1. ne naredi ničesar
2. požene drugo aplikacijo

3. uporabi t.i. plug-in

Protokol HTTP

HTTP ali **HyperText**? Transfer Protocol je metoda za prenos informacij po spletu. Gre za komunikacijski protokol med strankami (npr. brskalniki) in strežniki, ki poteka preko TCP povezave na izbranem gostitelju. Po poslanem zahtevku stranke na strežnik ta odgovori z MIME sporočilom (podobno kot pri elektronski pošti), nato pa sledijo podatki. Po prejetem zahtevku strežnik najprej pošlje vnaprej določen odgovor in v primeru, da pri zahtevku ni bilo zaznanih napak posreduje podatke.

Tipi HTTP zahtevkov:

- **GET** Metoda za pridobivanje vsebine datoteke preko spletnega naslova.
- **POST** Podobno kot GET, vendar tu telo zahtevka vsebuje podatke, ki jih želimo posredovati strežniku.
- **HEAD** Kot GET, le da strežnik odgovori samo z glavo datoteke.
- **PUT** Pošlje datoteko na strežnik.
- **DELETE** Zbriše datoteko iz strežnika (le redko implementiran)
- **TRACE** Vrne zahtevek.
- **OPTIONS** Vrne seznam podprtih HTTP zahtevkov.
- **CONNECT** Uporaba samo pri vzpostavitvi SSL tunela.

Tipi odgovorov na zahtevke:

- **1xx** Informacijske narave; poslan zahtevek je bil sprejet in je v obdelavi.
- **2xx** Uspeh; zahtevek je bil uspešno sprejet.
- **3xx** Preusmeritev; na odjemalca se pošlje odgovor, ki zahteva dodatno interakcijo z odjemalcem.
- **4xx** Napaka; prišlo je do napake zato zahteveka ni mogoče obdelati.
- **5xx** Napaka na strežniku; strežnik ne more izpolniti načeloma pravinega zahtevka.

Primer izmenjave sporočil med stranko in strežnikom:

Zahtevek:
```
GET www.google.si
Host: www.google.si
```

Odziv:
```
200 OK
```

Response Headers
```
Cache-Control: private, max-age=0
Date: Tue, 26 May 2009 00:30:30 GMT
Expires: -1
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
Server: gws
Content-Length: 3024
```

Request Headers

```
Host: www.google.si
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; sl;
rv:1.9.0.10) Gecko/2009042316 Firefox/3.0.10 (.NET CLR 3
.5.30729)
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: sl,en-gb;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-2,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie:
PREF=ID=e745f14466152b82:U=c87ce58b0eba1fc6:TM=1242254149:LM=1
242288258:S=zjyoHLbUXb1PI_n2; NID=23=n
mLS-kFsFr1emseWDj_lDQKqweJXkZwb-
PrH2w_kdCO3x2UQle9WjB4D4mrRkfVmJHa190Yb_Nxjrhawc8VBSPqTjXhAxFU
Ie7S2u3NOUk1fRuSZohXYbh95Xkygq?-q9
;
SID=DQAAAHoAAAAvgPTFYAt_hYXCZ9bq7kwQERgydOEdcmj82oTUWY2H6ZnBEE
21Xk8_MTZJVOmFRYbEuqX6izGTNPrTGFi?-2r
H_oRjvvjUCgybvuaqTQW20GMqhKJQQ6Wm_l_dN3tLtaXnoEQ0SGlgbTIiI_0Ad
OlOCDVck4g0P8DrzgFGMxaO5BQ?
Cache-Control: max-age=0
```

Večino današnjega prometa preko protokola HTTP predstavljajo HTML dokumenti, ki so osnova spletnih strani. Vsak HTML dokument sestavljajo značke (angl. tags), ki tekstu dodajo specifične lastnosti. Tako lahko tekst oblikujemo v odstavke, v tekst dodajamo nove vrstice, ustvarjamo oštevilčene in neoštevilčene sezname, v tekst vstavljamo slike, tabele, obrazce itd. Vsaka izmed značk ima že vnaprej določene lastnosti s katerimi jih lahko do neke mere preoblikujemo tako funukcionalno kakor tudi estetsko. Za grafično oblikovanje tako elementom znotraj dokumenta lahko določimo stile (class, id), prav tako pa lahko preko oznake id spremenimo tudi funkcionalnost elementa v kombinaciji z **JavaScriptom**? in Java appleti.

Nekaj osnovnih HTML oznak:

```
<p>...</p> - odstavek
<strong>...</strong> - odebeljen tekst
<a>...</a> - povezava
<form ...>...</form> - začetek obrazca
<br /> - nova vrstica
<img ... /> - slika
<input ... /> - vnosno polje
<!-- ... --> - komentar
```

Zgradba dokumenta HTML:

```
<html>
<head>
```

```
<title>Primer</title>
<!-- V glavi dokumenta se nahajajo meta informacije, ki jih
največkrat uporabljajo brskalniki ter povezave do zunanjih
datotek, ki vsebujejo stile, skripte, ikone ... -->
</head>
<body>
Primer
<!-- Tu se nahaja telo dokumenta, ki bo prikazano v našem
brskalniku -->
</body>
</html>
```

Z značko <a href> v HTMLju ustvarimo povezavo na neko drugo stran. Če uporabnik v brskalniku klikne na to povezavo se ustvari nova TCP seja, izvrši GET ukaz, nato se seja zapre.

Prav tako se nova seja vzpostavi in nato zapre ob nekaterih drugih značkah v kodi HTMLja - npr. pri znački <img src> se vzpostavi nova TCP seja za prenos slike, slika se prenese, nato se seja zapre. Podobno tudi za druge multimedijske vsebine. Več o standardu HTML je na voljo tu.

Spletne storitve (web services)

V splošnem so polja v HTTP glavi lahko karkoli - morda navodila za izvajanje programa - to so potem spletne storitve. Gre za podoben princip kot pri RPCju. Najpogosteje so uporablja standard SOAP, za katerega je HTTP prenosni protokol.

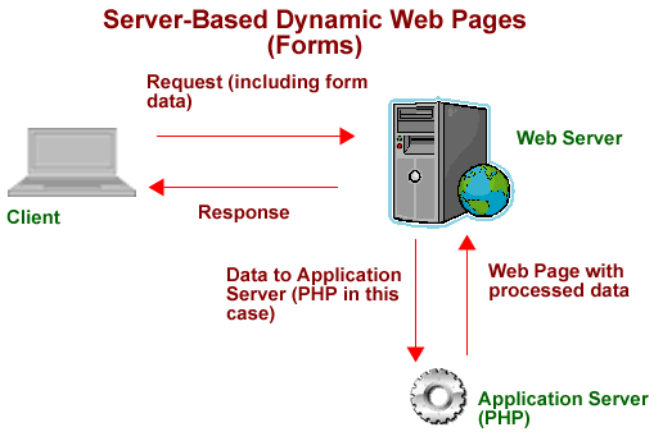
XML in XSL

XML je okrajšava za eXtensible Markup Language oziroma razširljiv označevalni jezik. Je nekakšen univerzalni jezik namenjen strukturiranju, shranjevanju in pošiljanju podatkov. Hrani lahko poljubne podatke (npr. v XML dokumentu imamo lahko bazo podatkov, ...). XSL je kratica za eXtensible Stylesheet Language. Namenjen je oblikovanju XML dokumentov za prikaz. XSL oblikuje XML za izhodne medije (npr. brskalnik).

Dinamični spletni dokumenti

Pri navadnem HTMLju uporabnik pošlje zahtevo za prenos HTML dokumenta, spletni strežnik pa mu zahtevani dokument samo pošlje nazaj.

Pri dinamičnih spletnih dokumentih pa spletni strežnik po zahtevi za prenos nekega dokumenta ta dokument najprej požene v okolju v katerem je bil napisan (npr. PHP, Python, ASP.NET, CGI) in nato vrednost izhoda, ki mu ga je vrnil prevajalnik oziroma interpreter pošlje nazaj.



Dinamični dokumenti imajo navadno v svoji vsebini posebne značke, ki označujejo kateri del njihove vsebine se mora interpretirati dinamično (npr. pri PHPju so to značke <? oziroma <?php in ?> ter php?>).

Dinamični dokumenti na strani odjemalca

Dinamične dokumente lahko generiramo tudi na strani odjemalca z jezikom **JavaScript**?. V tem primeru je brskalnik interpreter **JavaScript**? kode. Kodo **JavaScript**? znotraj HTML dokumenta označimo z značko <script>.

Caching (predpomnenje)

V primeru da imamo povezanih več računalnikov v LAN omrežje lahko postavimo proxy strežnik (npr. Squid), ki nam omogoča da si strani zapomnimo na proxy strežniku. Ko uporabnik zahteva stran, ima proxy to stran že shranjeno in je (ker je povezava do lokalnega računalnika precej hitrejša) dostop hitrejši.

Večpredstavnost

Zvok

Zvok je analogen signal, zato ga pretvorimo v digitalen signal z vzorčenjem.

Pri vzorčenju moramo upoštevati Nyquistov teorem (frekvenca vzorčenja mora biti vsaj 2x večja od največje frekvence fiz. fenomena). Ker človek sliši frekvence med približno 20Hz in **20kHz** je v našem primeru frekvenca vzorčenja 44.1kHz, možne so tudi druge frkvence 44.1, 48, 88.2 in 96 KHz. Veliko **HiFi** naprav tudi ponuja frekvenčno vzorčenje pri 176.4 and 192 KHz.

Za prenos zvoka je najbolj primeren UDP protokol, ker izguba posameznih vzorcev nima kritičnega vpliva na zvočni signal, ki ga reproducira predvajalnik. Zato je uporaba TCP protokola dokaj nesmiselna, saj po nepotrebnem obremenjujemo omrežje.

Če pri pošiljanju izgubimo posamezen vzorec, kompenziramo z interpolacijo.

Tu se pojavita dva problema:

- Pri pošiljanju ne želimo izgubljati zaporednih paketov (to bi onemogočilo dobro interpolacijo)
- Ne smemo imeti preveč podatkov saj bi prenos predolgo trajal

Problem izgube večih zaporednih paketov rešujemo na mrežnem nivoju. Usmerjevalniki morajo zavreči pakete, ki niso zaporedni. To dosežemo z RED protokolom (random early detection), kar pomeni, da zavržemo naključne pakete. S tem dosežemo minimalno verjetnost izgube dveh zaporednih paketov.

Problem prevelike količine podatkov pa rešujemo s kompresijo. Tu si lahko pomagamo z dejstvom, da povprečen človek določenih kombinacij frekvenc ne sliši. Tako uporabimo Fourierjevo transformacijo in zavržemo tiste vzorce.

Tipičen tak primer kompresije zvoka je MP3, ki na podlagi hevrističnih spoznanj o človeškem sluhu, ovrže in prilagodi vzorce glede na sposobnost sluha.

Pretočni zvok (streaming)

Vendar je 44000 paketov na sekundo še vedno preveč, zato zvok kompresiramo - z analizo frekvenc(nekatere se prekrivajo, nekatere so neslišne,...). Poleg tega, pa tudi pakete pošiljamo po 220 vzorcev naenkrat. Ker pa ne želimo izgubiti 220 zaporednih vzorcev, pošljemo enrat sode vzorce in nato lihe vzorce.

Pri poslušanju daljših zvokovnih posnetkov se pojavi nov problem. Gre namreč za preveliko količino podatkov, da bi si celoten posnetek najprej naložili na disk, nato pa predvajali (traja predolgo).

Tako se je razvilo tokovno predvajanje (streaming).

Za učinkovit prenos je potrebno, da je podatkovni del paketa, ki ga pošiljamo dosti večji od režije (glave paketov). Zato bi bilo precej potratno pošiljati pakete, ki bi vsebovali po en vzorec. Presežka se torej znebimo tako, da združimo čim več vzorcev (220) v en paket.

Tu se pa pojavi nov problem. Izguba takega paketa bi lahko imela precej hude posledice, če bi zapakirali zaporedne vzorce. Zato v pakete zaporedno vključimo ločeno sode in lihe vzorce. Tako pri izgubi enega paketa izgubimo samo vsak drugi vzorec. Tako se zmanjša resolucija signala, ne pride pa do lukenj v signalu.

Predvajalnik zakasnele pakete zavrže.

Predvajalnik strežniku pošilja zahteve po protokolu RTSP, s katerimi nadzoruje predvajanje toka:

- DESCRIBE
- SETUP
- PLAY
- PAUSE
- RECORD
- TEARDOWN

Voice over IP

VoIP omogoča prenos komunikacijskih podatkov (govor in faks) preko internetnega omrežja z uporabo internetnega protokola (IP). Prenos temelji na podatkovnem prenosu preko omrežja, kjer se analogni govorni signal pretvori v stisnjen digitalni format in prenese signal v IP paket za prenos preko omrežja.

Za delovanje potrebujemo 4 gradnike - 2krat mikrofon in 2krat terminal. ATM protokol skrbi za automatično povezovanje in rezervira pasovno širino.

Vzpostavitev protokola H.323, ki opisuje kako se vzpostavi povezava in kako gradniki skupaj delujejo. Spoznan za zelo kompliciran protokol, zato je nadomeščen z boljšim in bolj prijaznim SIP, ki je bil zasnovan s strani IETF. V ospredju je misel, da se telefon vključi v internet. Posledica tega je boljša učinkovitost in lažja integracija.

VSTAVI H.323 sliko

Med telefonskim in paketnim omrežjem je "Media Gateway", ki povezuje obe omrežji in zagotavlja, da ni nedovoljene interakcije med obema omrežjema. Podobno delovanje kot Proxy. Gateway poskrbi za to, da ne zaznamo prehoda iz analognega sveta v digitalen.

Za kodiranje VoIP signala skrbi voice codec. Naloga codec-a je, da zmanjša velikost podatkov za prenost našega govora prek omrežja. Avdio signal zakodira podobno kot se kodira mp3 zapis, na podlagi znanja iz psihoakustike odstranime dele informacije, ki niso bistvenega pomena. To so frekvence, ki so nam neslišne, ali pa frekvence, ki jih preglasije druge, saj imajo večjo amplitudo. Pri telefoniranju je bistvenega pomena, da nimamo zaksnitev, da se torej lahko pogovarjamo s sogovornikom v realnem času. To pa gre seveda na račun kakovosti zvoka. S tem ko zmanjšamo kakovost, zmanjšamo velikost podatkov, ki jih moramo poslati in s tem zmanjšamo tudi možne zakasnitve.

THE JPEG STANDARD

JPEG ali JPG je rastrski slikovni format. Za razliko od GIF formata, ki uporablja brezizgubno kompresijo, JPEG format uporablja izgubno. To pomeni, da se določeni deli informacije izgubijo. Tu se zopet uporablja znanje, ki ugotavlja kako naše oko zaznava, kakšne detajle in različne odtenke še zaznamo in jih lahko razločimo.

JPEG kompresija poteka tako, da barvni format najprej iz RGB zamenjamo v **YCbCr**. Y tu predstavlja svetlost, Cb in Cr pa barvni komponenti. Ker oko ne zazna oziroma je razlika zelo neopazna, lahko zmanjšamo število različnih odtenkov, saj svetlost igra pomembnejšo vlogo. Nato sliko razdelimo na 8x8 enakih blokov, ter vsak blok obdelamo s Diskretno Kosinusno Transformacijo (DCT), ki je podobna Fourierjevi transformaciji. Sliko nato še obdelamo s postopkom kvantizacije (quantization). Zadnji postopek je še, da podatke kodiramo s Huffmanovim kodom.

Vse to seveda zmanjšuje kakovost slike, zmanjšuje pa se tudi količina prostora, ki jo zasede taka slika na disku. To je tudi naš namen, saj so GIF slike neprimerno večje kot JPEG slike, razlika v kakovosti pa ni tako velika. Seveda jo ostro oko opazi, povprečen uporabnik pa je povsem zadovoljen s kakovostjo.

THE MPEG STANDARD

Glavna naloga je, da združi in sihronizira avdio s sliko.

VIDEO ON DEMAND

Omogoča, da so v omrežje t.i. spooling serverji, ki hranijo trenutne podatke. Ko naročnik naroči, se vsebina z glavnega strežnika prenese na najbljižji podstrežnik. S tem se razbremeni glavno omrežje.

kaj vas čaka tam zunaj

- Kaj vas čaka "tam zunaj"
- **virtualizacija in konsolidacija**
- **varnostni izzivi**
- uvajanje MPLS (Multiprotocol Label Switching)/VPN (Virtual Private Network) omrežij in prenosa podatkov čez Internet
    - namesto množice dragih rešitev za komunikacijo med podjetji, se promet seli na Internet
    - VPN, da so podatki varni
- združene komunikacije (unified communications)
    - poenostaviti in združiti vse oblike komunikacij
    - npr. centralizacija telefona, spletne pošte, instantnega sporočanja, glasovne pošte, … kjer bi na enem mestu lahko sprejemali in pošiljali na vse tipe medijev
- brezžična omrežja (wireless)
    - npr. skeniranje vsebine skladišča, brezžični POS terminali
- uporaba javnih storitev (Gmail, Google Docs)
    - sicer poceni, ampak ni varno, ameriška firma ima na vpogledu vse datoteke
    - lokalen email ne deluje, če ne dela internet (podjetje ni neodvisno), isto telefon v primeru uporabe Centrex-a
    - zahteva pozavo do interneta

- dokumenti so dostopni od kjerkoli na svetu
- cloud computing
  - kupimo prostor na strežnikih in procesorske cikle ter izvajamo procesorsko zahtevne aplikacije
  - dobro, če ne rabimo takoj (lahko traja, da pridemo na vrsto) in pri "burst" računanju
  - procesorsko zahtevne aplikacije so redke, zato je najemanje procesorke moči smiselno

VIRTUALIZACIJA

- ena fizična in več logičnih infrastruktur
- začetki v telekomunikacijah (na eni žici več navideznih povezav)
- navidezni računalniki so ločeni od strojne opreme na kateri tečejo - med delovanjem jih lahko po potrebi preselimo na drugo strojno opremo
- virtualna omrežja
- virtualna diskovna omrežja: logical volume
- diski tako veliki, kot jih rabimo

KONSOLIDACIJA

- premik strežnikov na eno centralno lokacijo, namesto da so fizično blizu uporabnikom
  - večja varnost, centraliziran backup in administracija
- dostop uporabnikov do strežnika preko Interneta

Cilj: čim več funkcij stlačiti v čim manj škatel

ZAKAJ ZDAJ

- hitre, poceni in zanesljive komunikacije
- hitri in neizkoriščeni računalniki
- veliki diski, virtualni so bolj izkoriščeni
- VARČEVANJE
  - z virtualizacijo prihranimo tako na stroških strojne opreme kot tudi na elektriki, manj računalnikov pa se tudi manj greje, torej prihranimo še na elektriki za hlajanje

- manj napajalnikov, manj izgub

VIRTUALIZACIJA JE TUDI OMREŽNA STORITEV

- kako zagotoviti komunikacijo med virtualnimi strežniki v istem fizičnem strežniku
- omrežje se seli v same strežnike - virtualna stikala, s katerimi upravljajo administratorji strežnikov, ne omrežij

- rešitev: integracija, virtualno stikalo postane del omrežja in vsebuje nekatere funkcionalnosti omrežnih stikal (npr Cisco programska oprema za virtualno stikalo)

VIRTUALIZACIJA ODJEMALCEV

- spletne aplikacije, ki tečejo znotraj brskalnika, ni potrebe po razvoju za različne platforme
- virtualizacija namizja
    - uporabniki rabijo samo poceni "thin client" da se povežejo na strežnik, vse kar počnejo se dogaja na strežniku na navideznih namizjih
    - večja varnost, lažja administracija
    - hitro dodajanje navideznih računalnikov
    - uporabnik lahko dostopa do svojega službenega računalnika tudi od doma
    - zahteva po visoki dostopnosti: kaj se zgodi, če mreža ne dela, kolikšne so takrat izgube v primerjavi s prihranki
    - zaenkrat ni primerna za poganjanje aplikacij z visokimi I/O in grafičnimi zahtevami

VIRTUALNE STRUKTURE

- PREDNOSTI
- učinkovitost in cenovna optimalnost

- hitrejše vpeljevanje novih storitev in prototipov

- preprostost posameznih storitev

- enostavna rast, upravljanje in optimizacija zmogljivosti

- nižja kompleksnost zaradi poenotene infrastrukture
- povečana predvidljivost in višja varnost
- manjša poraba energije
- SLABOSTI
- višja začetna kompleksnost
- organizacijske težave
- zmanjšana predvidljivost/bolj zahtevno načrtovanje zmogljivosti