Lencse Gábor

Hálózati alkalmazások

2. kiadás

Győr, 2017.

A 2008. évi azonos című jegyzet rövidített, frissített változata.

Tartalom

Bevezető a hálózati alkalmazásokról ... 4
Portszámok kiosztása ... 4
Domain Name System ... 6
A szimbolikus nevek ... 6
Subdomainek létrehozása ... 7
Domain és zóna közötti különbség ... 7
Root DNS szerverek ... 7
A névfeloldás menete ... 8
Reverse mapping ... 11
Caching ... 11
Domain név regisztráció ... 12
Domain nevek helyesírása ... 12
Eredeti állapot ... 12
Jelenleg érvényes szabályozás ... 12
Implementációk ... 13
További források ... 13
Telnet ... 14
Simple Mail Transfer Protocol ... 15
A levél átvitelének folyamata ... 15
Nem ASCII-ben kódolt információ átvitele ... 16
Az SMTP protokoll parancsai ... 17
Példa az SMTP protokoll működésére ... 18
Postafiók implementációk ... 19
Post Office Protocol Version 3 ... 20
Internet Message Access Protocol Version 4 ... 23
POP3S és IMAP4S ... 25
POP3S ... 25
IMAP4S ... 25
SSH: távoli elérés biztonságosan ... 26
Elméleti alapok ... 26
Titkos csatorna létrehozása a kép gép között ... 26
A felhasználó azonosítása ... 26
Titkosított kommunikáció ... 27
SSH a gyakorlatban ... 27
File Transfer Protocol ... 28

FTP kliens-szerver kommunikáció parancsai 28
FTP a felhasználó szemszögéből 30
HTTP 33
HyperText Markup Language 33
HyperText Transfer Protocol 35
HTTPS 41
Ajánlott irodalom 42

Bevezető a hálózati alkalmazásokról

Ez a jegyzet az azonos című jegyzet 2008. évi 1. kiadásának [1] a rövidített, átalakított, aktualizált változata. (Az eredeti jegyzet a [2] mű 13. fejezetének magyar fordításán alapul.) Elkészítéséhez felhasználtam a Számítógép-hálózatok tárgy fő jegyzetét is [3].

A TCP/IP felett működő hálózati alkalmazások magukban foglalják az OSI 5., 6., és 7. rétegének funkcionalitását. Nem minden alkalmazás igényli az 5. és 6. réteg szolgáltatásait. A hálózati alkalmazások foglalkoznak az alkalmazási réteg adatainak formázásával, küldésével és fogadásával. Azonban nem tartalmazzák a felhasználó felé nyújtott kezelői felületet.

Ez a segédlet a következő hálózati alkalmazásokat tárgyalja:

- Domain Name System (DNS)
- Levelező protokollok: SMTP, POP3, IMAP4, POP3S, IMAP4S
- Távoli elérési protokollok: TELNET, SSH, SCP
- Fájl átviteli protokoll: FTP
- Web hozzáférési protokollok: HTTP, HTTPS

A hálózati alkalmazások többsége (a fentiek közül az összes) kliens-szerver modell szerint működik, azaz egy szerver alkalmazás nyújt valamilyen szolgáltatást, amelyet a hálózaton keresztül hozzá kapcsolódó kliensek tudnak igénybe venni.

Hogyan találja meg egy kliens a szervert? Általában a kliensnek (illetve a kliens programot használó felhasználónak) tudnia kell, hogy az igénybe venni kívánt szerver program melyik számítógépen fut. A számítógépekre az IP-címük helyett a szimbolikus nevükkel hivatkozhatunk: ezt részletesen tárgyaljuk DNS-ről szóló részben. Egy gépen akár több szolgáltatás is futhat; a kliens program az általa kívánt szolgáltatást nyújtó szervert a *portszám* segítségével találja meg.

Portszámok kiosztása

A portszámok kiosztásáért az IANA (Internet Assigned Number Authority) felelős. Honlapján (http://www.iana.org) a portszámok kiosztásán kívül nagyon sok más információ is megtalálható, a téma iránt érdeklődőknek javasoljuk a tanulmányozását. A honlapról a portszámok kiosztásának módja és a kiosztás aktuális állása is elérhető: http://www.iana.org/assignments/port-numbers.

A portszámokat három tartományra bontották. Ezek megnevezése (az első kettő esetében) változott, az új név mellett zárójelben megadjuk a régit is, ugyanis a szakirodalomban azzal is találkozunk.

System Ports (Well Known Ports) rendszer portok („jól ismert” portok): 0-1023

Ezeket a portokat használják az alapvető, általánosan használt hálózati szolgáltatások. Ezekhez a portokhoz csak privilegizált (Unix alatt például root jogokkal elindított) szerver programok kapcsolódhatnak szolgáltatás nyújtása érdekében.

User Ports (Registered Ports) felhasználói portok (regisztrált portok): 1024-49151

Ezeken a portokon egyéb szolgáltatások érhetők el. Ha valaki kitalál egy szolgáltatást, igényelhet hozzá ilyen portszámot. Ezeken a portokon történő szolgáltatásnyújtáshoz nem szükséges privilegizált módban futtatni a szerver programot.

Dynamic/Private/Ephemeral Ports dinamikusan kiosztott portok: 49152-65535

Ebből a tartományból az operációs rendszer oszt ki portokat a kommunikációhoz a programoknak.

Megjegyezzük, hogy az operációs rendszerek nem minden esetben követik a szabványt (RFC 6335), előfordul, hogy például már 32768-tól kezdve osztanak ki dinamikus portokat. Továbbá NAT (Network Address Translation) használata esetén (a kellően nagyméretű tartomány érdekében) már 1024-től osztják ki a forrás portokat. A TCP kapcsolat felépítési iránya alapján azonban az így esetlegesen átlapolódó tartományok esetén is megkülönböztethető, hogy melyik a forrásport és melyik a célport. A szolgáltatást mindig a célport azonosítja.

A továbbiakban csak olyan hálózati alkalmazásokkal foglalkozunk, amelyek szerver programja valamelyik rendszer porton érhető el. Az alábbi táblázatban tájékoztatásul megadjuk néhány fontosabb hálózati alkalmazás portszámát, valamint az alkalmazások rövidített és teljes nevét.

port	alk. röv.	hálózati alkalmazás neve
20	FTP	File Transfer Protocol, data
21	FTP	File Transfer Protocol, control
22	SSH	Secure Shell
23	Telnet	Telnet
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name System
80	HTTP	HyperText Transfer Protocol
110	POP3	Post Office Protocol – Version 3
143	IMAP4	Internet Message Access Protocol – Version 4
443	HTTPS	HTTP over TLS/SSL
993	IMAPS	IMAP4 over TLS/SSL
995	POP3S	POP3 over TLS/SSL

Domain Name System

Mivel az IP-címek az emberek számára nehezen megjegyezhetők, ezért a számítógépeket az IP-cím helyett a sokkal könnyebben megjegyezhető *szimbolikus név*vel azonosítjuk.

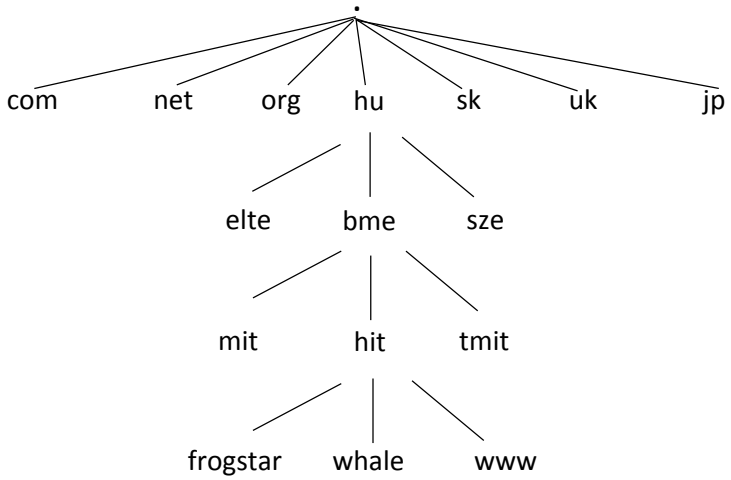
A szimbolikus nevek

Hogyan épülnek fel a szimbolikus nevek? Példaként tekintsük a következőt:

`whale.hit.bme.hu`

A szimbolikus név mezőit pontok választják el. Az első mező jelenti a *gép* (host) nevét (**`whale`**), a többi a *tartomány* (domain) neve. A tartományok felépítése hierarchikus, fa struktúrát követ. Az 1. ábrán látható, hogy a legfelső szinten egy pont („.”) van, amit a nevek írásakor általában elhagyunk. Alatta vannak a *legfelső szintű tartomány*ok (TLD: Top Level Domain). Ezek további tartományokra bomlanak, ami tetszőleges mélységig folytatható. Az ábrán értelemszerűen a „hu” Magyarországot, a „bme” a Budapesti Műszaki és Gazdaságtudományi Egyetemet, a „hit” pedig a Hálózati Rendszerek és Szolgáltatások Tanszéket jelenti. Egy számítógép *teljes nevét*[1] (FQDN: Fully Qualified Domain Name) tehát a gép nevétől a fa struktúrában felfele haladva olvassuk ki, az egyes mezők közé pontot teszünk, és a végét is ponttal zárjuk, például: „**`whale.hit.bme.hu.`**”. A végén álló pont teszi formálisan egyértelművé, hogy egy FQDN-nel állunk szemben. Az FQDN-t hívják még *abszolút név*nek (absolute name) is. Ezzel szemben egy Partially Qualified Domain Name nem tartalmaz minden labelt a záró pontig (gyakran csak az első label, azaz a gép hostneve szerepel, de szerepelhet benne több is), ezt más kifejezéssel *relatív név*nek (relative name) is hívják. A példában ilyen a **`whale`**, de hasonlóképpen a **`whale.hit`** is. Bizonyos alkalmazások esetén (a köznapi internet használtban nagyon gyakran) a záró pontot elhagyhatjuk. Vannak azonban olyan esetek, amikor a pont elhagyása a helyi tartománynak a névhez való automatikus hozzáírását eredményezi.

Megjegyzés: a záró pont különösen nem hagyható el névkiszolgálók adminisztrációjánál a zónafájlban, mert fontos, jelentést megkülönböztető szerepe van!



1. ábra. A DNS névhierarchia egy részlete

A TLD neveknek két fajtája van. A *generic TLD* (gTLD) nevek a szervezetek fajtája szerinti csoportosítást használják, erre bemutatunk néhány példát a következő táblázatban.

[1] Nehéz rá jó magyar kifejezést találni, lehetne például *teljesen kifejtett név*nek hívni, a legbiztosabb, ha FQDN-nek hívjuk.

gTLD	szervezet fajtája
com	üzleti vállalkozás
org	non-profit szervezet
net	hálózattal kapcsolatos
edu	oktatási intézmény
gov	USA közigazgatás

Mivel az Internet eredetileg az USA-ban indult, ezek eredetileg ottani kategóriákat jelentettek, aztán közülük bizonyosak az egész világon elterjedtté váltak (például: com, org, net), de van, amelyik ma is csak az USA-ban használatos (például: gov). A *country code TLD* (ccTLD) nevek az ITU (International Telecommunication Union) illetve az ISO 3166 szabvány szerinti kétbetűs országmegjelölést követik (kivétel: gb helyett inkább uk-t használnak), az 1. ábrán ilyenek: hu, sk, uk, jp.

Subdomainek létrehozása

Az FQDN középső része utal a szervezetre (és utalhat a szervezeten belüli egyéb szerkezetre, rendszerre is). Egy szervezet szabadon definiálhat *altartomány*okat (subdomain) a szervezeten belül, de ha megteszi, akkor fel kell töltenie a hozzá tartozó DNS kiszolgáló adatbázisát, hogy az végre tudja hajtani a névfeloldásokat. Példaként tekintsük a BME-t, melynek a domain neve a következő:

`bme.hu`

Ha ennek az egyetemnek különálló hálózatai vannak, például a *Hálózati Rendszerek és Szolgáltatások*, a *Méréstechnika és Információs Rendszerek*, valamint a *Távközlési és Médiainformatikai* tanszékeken, akkor definiálhat subdomaineket mint a **`hit`**, **`mit`**, és **`tmit`**. Ha ezt az információt megfelelő módon bejegyzi a BME DNS szerverének a **`bme.hu`** zónát leíró zónafájljába, hogy el tudják végezni a névfeloldást, akkor az egyes tanszékek szervereit következő domainek alatt lehet majd elérni (miután ezekbe a zónákba bejegyezték őket):

`hit.bme.hu`
`tmit.bme.hu`
`iit.bme.hu`

Nem szükséges minden domainhez egy DNS szervert alkalmazni, egy közös szerver elláthat több zónát is, de természetesen megtehető az is, hogy például a BME példájában a tanszékek külön névkiszolgálót üzemeltetnek.

Domain és zóna közötti különbség

A **`www.bme.hu`** és **`www.hit.bme.hu`** szimbolikus nevek mindegyike benne van a **`bme.hu`** domainben, mert a végződésük az, hogy „bme.hu”. Viszont amíg az első a „bme.hu” zónában van, addig a második a „hit.bme.hu” zónában, ugyanis az ezeket a zónákat leíró zónafájlban nyernek feloldást. Ha például a „hit.bme.hu” zónában létrehozunk egy „teszt.hit.bme.hu” zónát, akkor az abban bejegyzett, **`gep1.teszt.hit.bme.hu`** szimbolikus név egyaránt benne van a „hu”, a „bme.hu”, a „hit.bme.hu” és a „teszt.hit.bme.hu” domainekben, de csak a „teszt.hit.bme.hu” zónában van benne, a „hu”, a „bme.hu”, és a „hit.bme.hu”, zónákban nincs benne.

Root DNS szerverek

Számos névkiszolgáló kezeli a szimbolikus neveket a *root zónában*. Korábban logikailag 13 szerverre volt lehetőség, mert az összes névkiszolgáló IP-címét megadó válasznak bele kellett férnie egy minden host által kötelezően támogatott méretű UDP csomagba (a DNS üzenet számára 512 oktett

állt rendelkezésre), valójában azonban némelyik (ma már a legtöbb) szerver IP-címe *anycast* cím, így fizikailag ennél több szerver használta volt lehetséges. A csomagméret problémát az RFC 2671 megoldotta. Ez azért is fontos, mert 2008. február óta némelyik szerver már IPv6 címen is elérhető, ehhez mindenképpen szükség volt a nagyobb csomagméretre. A root domain névkiszolgálóit ma **a**-tól **m**-ig terjedő betűkkel jelöljük, és a nevük úgy néz ki, hogy: **a.root-servers.net**, **b.root-servers.net**, …, **m.root-servers.net**. Korábban az üzemeltetőjük névtartományából (domain) származó nevük volt. A következő táblázatban látható néhány a *root domain name server*ek közül.

Új név	Régi név	IP-cím
a.root-servers.net	**ns.internic.net**	198.41.0.4
b.root-servers.net	**ns1.isi.edu**	192.228.79.201
c.root-servers.net	**c.psi.net**	192.33.4.12
d.root-servers.net	**terp.umd.edu**	199.7.91.13
e.root-servers.net	**ns.nasa.gov**	192.203.230.10

A névfeloldás menete

A *névfeloldás* (name resolution) azt jelenti, hogy a szimbolikus név (domain name) alapján kiderítjük a hozzá tartozó IP-címet. Ne tévesszük össze a *címfeloldás*sal (address resolution), ami az IP-cím alapján határozza meg a MAC-címet!

Amennyiben egy hálózati alkalmazás találkozik egy szimbolikus névvel, akkor megkéri a *névfeloldó*t (name resolver), hogy adja meg a hozzá tartozó IP-címet. A névfeloldó egy könyvtári függvény, ami az alkalmazás része. Unix alatt C nyelvben korábban a **gethostbyname()** függvényt használták erre a célra, de az ma már hivatalosan *elavult* (obsolete), ezért helyette **getaddrinfo()** függvényt használjuk, ami IPv4-hez és IPv6-hoz egyaránt használható. (Bővebben majd a TCP/IP socket interface programozásáról szóló előadáson foglalkozunk vele.) Ezt a függvényt hívja meg az adott alkalmazás. A gépen lévő beállítások figyelembevételével megtörténik a névfeloldás. Legyen például Linux alatt a **/etc/nsswitch.conf** fájlban az alábbi bejegyzés:

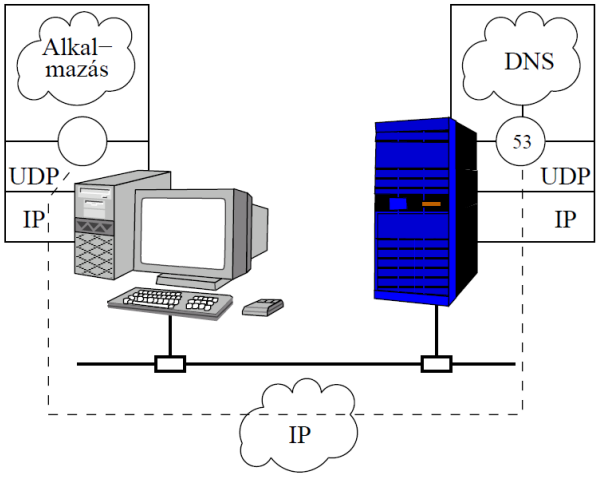
```
hosts:    files, dns
```

Ebben az esetben először a **/etc/hosts** fájlban keres, ahol össze vannak rendelve az IP-címek és szimbolikus nevek. (Tipikusan csak néhány darab, leginkább csak a gép saját interfészének a címei.) Először ebben keresi az adott névhez az IP-címet. Ha nem találja, akkor a **/etc/resolv.conf** fájlból kiolvassa a névkiszolgáló (name server) IP-címét.[2] A meghívott névfeloldó helyi függvény ilyenkor a névkiszolgálóhoz fordul, hogy adja meg például a kérdéses szimbolikus névhez tartozó IP-címet. Ez mutatja be a 2. ábra. Amint az ábrán látható, a DNS szerver programot az 53-as porton érjük el. Szállítási protokollként UDP-t használunk, hiszen általában egy rövid kérdésre egy rövid választ várunk (összesen 2 üzenet): ehhez gazdaságtalan lenne TCP kapcsolatot felépíteni, majd utána lebontani (összesen 3+2+4=9 üzenet). Amennyiben esetleg a kérdés vagy a válasz elveszne, akkor a kliens újra kérdez.

Megjegyzés: két névszerver közötti zónafájl áttöltés viszont TCP protokoll fölött történik (mivel az nem fér bele egy UDP csomagba). Sőt ma már lehetőség van arra is, hogy a kérdés-válasz alapú kommunikációra is TCP-t használjunk, ha tudjuk, hogy a válasz nem fér egy a DNS-nél szabványos 512

[2] Ilyen több is lehet. Ha az első (primary) valamiért nem válaszol, akkor a másodikhoz (secondary) fordul, és így tovább.

bájtos üzenetben, illetve ha az UDP fölötti kommunikáció során a szerver a válaszában azt jelezte, hogy csonkolnia kellett az üzentet.



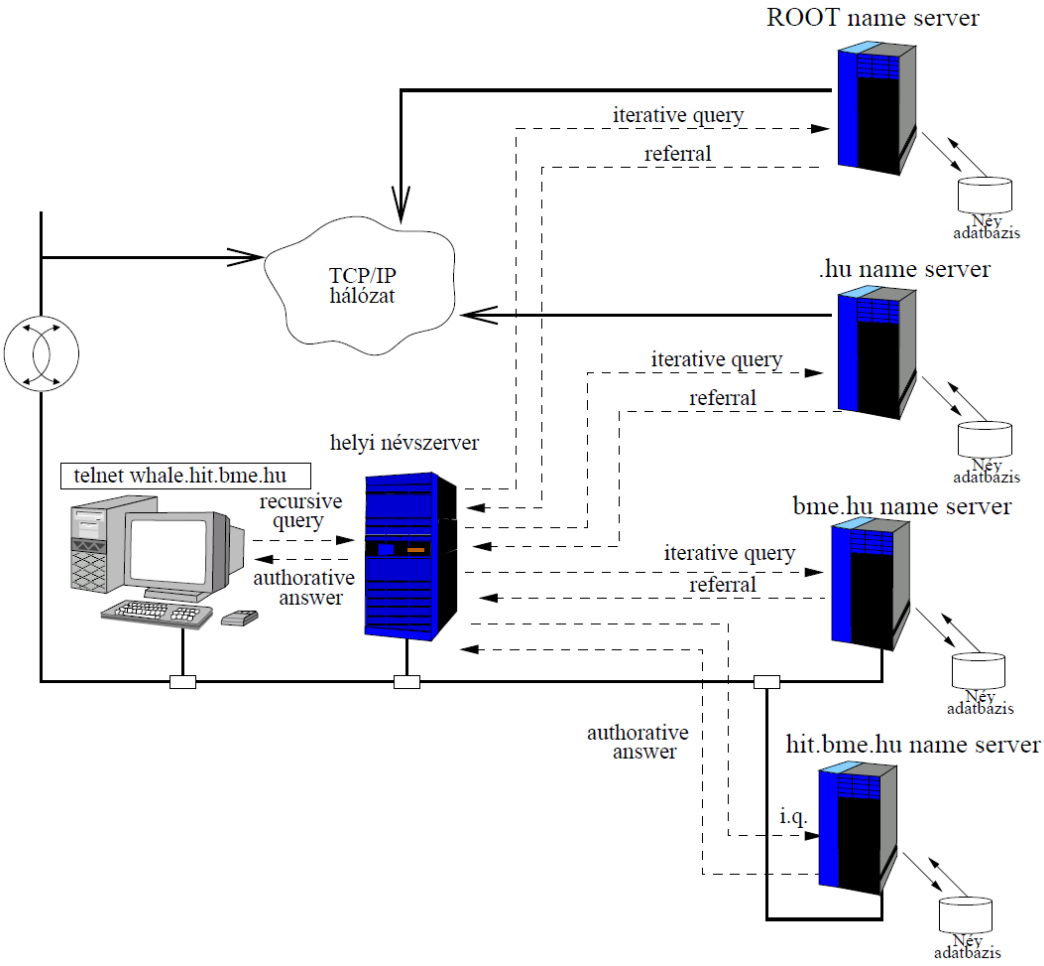
2. ábra. DNS kliens-szerver kommunikáció

A DNS protokoll üzenetei tehát UDP-be ágyazva haladnak, és speciális formátumuk van, amit most nem ismertetünk részletesen, csak néhány fontosabb jellemzőt említünk meg:

- A DNS üzenet egy fix méretű (6x16 bites) részből és további változó méretű rész(ek)ből épül fel.
- Az első 16 bites mezője a *Transaction ID*, ami egy véletlen szám, és azt a cél szolgálja, hogy a kliens több üzenet esetén is azonosítani tudja, hogy a DNS szervernek melyik válasza melyik kérdéshez tartozik.
- A következő 16 bites mező számos almezőre bomlik, többek között itt kódolják, hogy milyen típusú üzenetről van szó.
- Utána további 16 bites mezők következnek, amelyek az üzenetben található kérések, válaszok és egyéb erőforrás rekordok számát adják meg.
- A változó méretű részekben található például a feloldandó szimbolikus név, a hozzá tartozó IP-cím(ek), esetleg további rekordok.
- A szimbolikus nevet speciálisan kódolják, amelyek több, ismétlődő végződésű név esetén hatékony tömörítést tesz lehetővé.

A részleteket illetően további információ található a [4] cikk 2. fejezetében.

Fontos még, hogy az IPv4-címet „A” recordnak, az IPv6-címet „AAAA” recordnak nevezik (a DNS zónafájlban is így hívják őket). Egy DNS üzenetben elvileg több kérdés (query) is szerepelhetne, de a gyakorlatban csak egy kerül bele. (Ha IPv4 és IPv6 címre is szükség van, akkor két külön kérést küldenek.) Egy válasz viszont gyakran tartalmaz több erőforrás rekordot is. Ha például több IP-cím érkezik, akkor a kliens dönti el, hogy melyiket használja.



3. ábra. DNS névfeloldás mentére példa

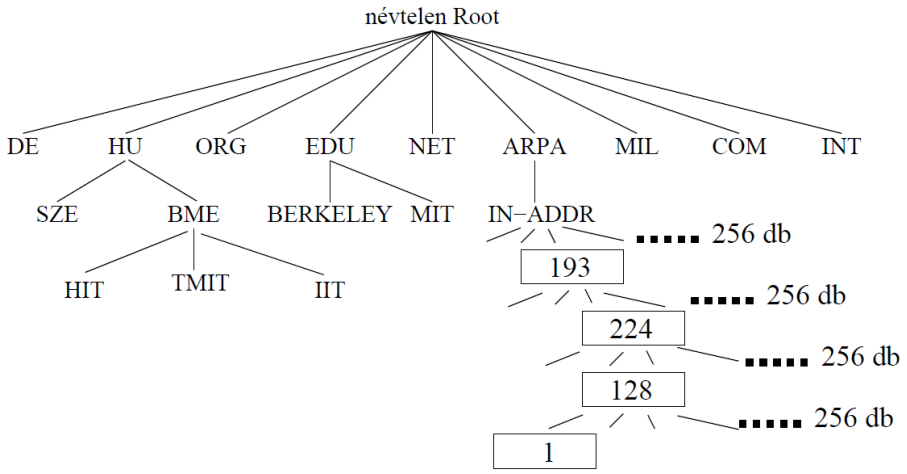
Kövessük nyomon a 3. ábra példáján, hogy mi is történik ezután! (A **`whale.hit.bme.hu`** gép IP-címét derítjük ki.) Azt a kérdést, amivel a name resolver megszólítja a helyi névkiszolgálót *recursive query*nek hívjuk. A recursive query azt jelenti, hogy a megkérdezett (helyi) névkiszolgáló köteles *végső válasz*t adni a kliens kérdésére.[3] Közben esetleg más névkiszolgálókat is meg kell szólítania. Ehhez a helyi névkiszolgálón elhelyeztek egy fájlt[4], amelyben root DNS szerver IP-címek találhatók, de nem feltétlenül mindegyik aktuális (up-to-date), hanem csak „tippek". A névkiszolgáló indulásakor ezek közül az első elérhető szervertől lekérte az aktuális root DNS server listát. Ebből a listából aztán kiderítette próbálkozással, hogy melyiknek a legkisebb a válaszideje (RTT – Round Trip Time: a teljes oda-vissza út ideje plusz a válasz előállításának együttes ideje). Ez a root névkiszolgáló van a „legközelebb" időben, ezért utána hozzá fog fordulni. A 3. ábrán látható, hogy a megkérdezett helyi névkiszolgáló *iterative query*vel fordul a kiválasztott root name serverhez, ami egy *referral*lal válaszol, hogy a példánkban a **`whale.hit.bme.hu`** névből a **`hu`** zónáért melyik szerver felelős. Ezután újabb kérdés most már ehhez, majd a válaszból kiderül, hogy a **`bme.hu`**-ért ki a felelős, és így tovább, míg végül a helyi névkiszolgáló a **`hit.bme.hu`** zónáért felelős szervertől megtudja egy *authoritative answer*rel a **`whale.hit.bme.hu`** IP-címét, és visszaküldi ezt az őt megkérdező kliensnek.

[3] Bizonyos esetben a recursive query visszautasítható, a lényeg azon van, hogy ha válaszol, akkor teljes választ kell adnia, szemben a később említendő *iterative query*re adandó *referral*lal.
[4] BIND esetén régebben **`root.hints`** volt a neve, újabban **`root.db`**.

Megjegyzés: *rekurzió*nak (resursion) nevezik azt a folyamatot, amelynek során egy névkiszolgáló a kapott recursive queryre való választ iterative queryk segítségével kideríti és visszaküldi a kliensnek. Az ilyen névkiszolgálókat pedig *recursor*nak nevezik. Létezik *forwarder* névkiszolgáló is, amely nem maga végzi el a rekurziót, hanem egy másik névkiszolgálót kér meg rá. (Ennek értelme lehet például az, hogy a hálózat biztonsági szabályzata nem engedi meg az adott kiszolgálónak a külső kommunikációt, a klienseknek pedig a recursorhoz való közvetlen hozzáférést.)

Reverse mapping

Míg a névfeloldás szimbolikus névhez ad meg IP-címet, addig a *reverse mapping* ennek a fordítottját végzi, IP-címhez deríti ki a hozzá tartozó szimbolikus nevet. Ehhez a leképzés tervezői akár létrehozhattak volna egy másik rendszert, de nyilvánvalóan célszerűbb volt a meglevő DNS-t felhasználni. Sőt, még új névfát sem kellett létrehozniuk! A leképzés alapelve a következő: a 4. ábrán látható fa egy pontját kinevezzük egy másik fa gyökerének. Ez a pont az **`in-addr.arpa.`** ág.[5] Az egyszerűség kedvéért induljon innen 256 ág, és azok mindegyikén szintén 256 ág. Ez történjen így összesen 4 szinten, mivel egy IP-cím négy oktettből áll. Tekintsük a 193.224.128.1 IP-címet. Az **`in-addr`** csomópontban válasszuk ki a 193-at, ott a 224-et, ott a 128-at, ott pedig az 1-et! Itt tároljuk el a hozzá tartozó szimbolikus nevet (**`rs1.sze.hu.`**). Vegyük észre, hogy amit így a névfában bejártunk, azt szimbolikus névként a levéltől a gyökér felé kiolvasva a következőt kapjuk: **`1.128.224.193.in-addr.arpa.`**, amiben az IP-cím oktettjei éppen fordított sorrendben leírva szerepelnek!



4. ábra. A DNS névhierarchia egy másik részlete

Persze a valóságban az alhálózatokra bontás nem 8 bites határokon történik, így a részfa nem lesz ennyire szabályos.

Fontos még, hogy a zónafájlban a bejegyzés neve „PTR” rekord, és így nevezik a DNS üzenetben is. Ezenkívül még egy rekord típust érdemes megjegyezni, ez a CNAME (Canonical Name), ami egy alias egy másik névre. A CNAME számunkra a virtuális webszerverek miatt fontos.

Caching

A névszerverek tárolják a már lekérdezett IP-címeket, és újabb kéréseknél felhasználják. Ez a közbenső eredményekre (adott zónáért felelős névkiszolgáló IP-címe) és a végeredményre (a kérdéses szimbolikus névhez tartozó IP-cím) egyaránt vonatkozik, sőt, a negatív eredményeket (adott

[5] IPv6 esetén pedig az `ip6.arpa.` ág.

szimbolikus névhez nem tartozik IP-cím) is tárolják. A tárolás legfeljebb az információ érvényességi idejének lejártáig (TTL: Time To Live) történhet, amit az információ gazdája (az adott zónáért felelős névkiszolgáló) az információval együtt szolgáltat.

A válasz üzenet fejrészének második 16-bites mezőjében az AA bit (Authoritative Answer) 1 értéke jelzi, ha a válasz a zónáért felelős névkiszolgáló „friss" információja, a 0 értéke pedig azt jelzi, hogy egy másik névszerver cache-éből származik.

Megjegyzés: az olyan névkiszolgálókat, amelyeknek nincs saját zónájuk, amelyért felelnének (a triviális helyi zónán kívül), hanem csak kliensektől fogadnak recursive queryket (és megválaszolják őket), *caching only name server*nek nevezik.

Domain név regisztráció

Ismerjünk meg néhány fontos fogalmat a domain nevek regisztrációjával kapcsolatban!

registry Egy TLD adminisztrálásának felelőse, delegálja az adott TLD subdomainjeit, de az ügyfelek közvetlenül nem fordulhatnak hozzá.

registrar Magyarul regisztrátor, általában egy profit orientált cég (de akár egyesület is lehet), akihez az ügyfelek fordulhatnak domain név regisztrációért.

registration A domain név regisztráció folyamata.

A **`hu`** domain adminisztrálásának felelőse az Internet Szolgáltatók Tanácsa (ISZT). Honlapján (http://www.domain.hu) megtalálható a hivatalos regisztrátorok listája, valamint a regisztrálással kapcsolatos szabályok. Az általuk közölt *Domainregisztrációs szabályzat* világos fogalom-magyarázattal kezdődik, megtalálhatók az igénylés, a regisztrálás, a vitás esetek kezelésének szabályai, valamint egy domain igénylő lap minta is, amin látható, hogy mit kell megadni domain igényléskor (például: kért domain név, névkiszolgáló, többféle kapcsolattartó, számlázási cím). A honlapon található egy kereső is, amivel ellenőrizhetjük, hogy egy név szabad-e még, illetve ha már nem, akkor megtudhatjuk, hogy ki regisztrálta.

Domain nevek helyesírása

Eredeti állapot

Az eredeti, RFC 1035 szerinti szabályok a következők. A szimbolikus nevekben (más kifejezéssel domain nevekben) nem különböztetjük meg a kis- illetve a nagybetűket. A pontok közötti szakaszokat angolul *label*nek hívják. Egy label hossza 1-63 karakter lehet, betűket („a"-„z"), számjegyeket („0"-„9") és kötőjelet („-") tartalmazhat, de az utóbbi csak a belsejében fordulhat elő, a határán nem.[6] Az FQDN teljes hossza nem haladhatja meg a 255 karaktert.

Jelenleg érvényes szabályozás

Már évek óta lehetőség van az ún. *nemzetköziesített domain nevek* (IDN: Internationalized Domain Name) használatára, ami a magyar ékezetes betűket is tartalmazza. Azonban a megoldás meglehetősen „fából vaskarika" jellegű: a unicode karaktereket tartalmazó stringet egy megfelelő algoritmussal (Punycode, RFC 3492) átkódolják ASCII karakterekből álló stringgé, majd elé teszik az ún. ACE (ASCII Compatible Encoding) prefixet („**`xn--`**") és a DNS rendszerben így tárolják. Természetesen az átkódolás után nyert ASCII stringre továbbra is érvényesek a korábbi szabályok (például egy label hossza legfeljebb 63 karakter lehet). A megoldás részleteivel számos RFC foglalkozik, amelyeket az ISZT összegyűjtött itt: http://www.domain.hu/domain/ekes/. Ezen az

[6] Egyes Microsoft szoftverek a fent felsorolt megengedett karaktereken kívül még az aláhúzás jelet („_") is használják.

oldalon számos további információ (beleértve a magyar domain nevekben használható karakterek felsorolását is), valamint egy konverter is található, amely megmutatja, hogy egy magyar ékezetes szóból mi lesz az elkódolása után. Például: „**`képesítés`**” helyett: „**`xn--kpests-bvae8a`**”.

Implementációk

A DNS legszélesebb körben használt megvalósítása a Berkeley Internet Domain Name Server (BIND), amely eredetileg a BSD Unix-ban volt elérhető. Most elérhető a legfontosabb Unix platformokon. A Unix rendszerekben a BIND-ot megvalósító program neve: *named* (name daemon). Elérhető az ISC (Internet Systems Consortium) honlapjáról: https://www.isc.org/downloads/bind/.

További fontos implementációk még:

- Unbound http://www.unbound.net/
- PowerDNS https://www.powerdns.com/.

További források

A DNS alapkoncepcióját az RFC 1034, implementációját az RFC 1035 írja le. Ezeket számos további RFC egészíti ki, amelyeket az RFC-k HTML formázású verziójába belinkeltek („Updated by…”). Végül ajánlunk egy szakkönyvet is a témához: [5].

Telnet

A telnet protokollt korábban távoli bejelentkezésre használták, de mivel mind a jelszavakat, mind a kommunikáció tartalmát titkosítás nélkül viszi át a hálózaton, ezért ma már ilyen célra nem használják. (Helyette az ssh használható, azt meg fogunk megismerni.) Más protokollok vizsgálatához azonban használni fogjuk, ezért most nagyon röviden megismerjük a számunkra érdekes jellemzőit.

Amennyiben a telnetet távoli bejelentkezésre használják, akkor a telnet kliens egy telnet szerverhez (Linux alatt **`telnetd`**, ejtsd: telnet-dí, telnet daemon) kapcsolódik, amely a 23-as porton várja a kliens csatlakozását. A felhasználói névvel és jelszóval történő azonosítás után, amit a felhasználó a telnet kliens előtt gépel, azt a kliens a szervernek továbbítja, amely átadja a távoli gép operációs rendszer parancsértelmezőjének (shell), mintha a felhasználó ott gépelte volna be. Az operációs parancsértelmezője a szokott módon működik, de a kimenet nem távoli gép a képernyőjére, hanem a telnet szerveren keresztül a telnet klienshez kerül, ami a felhasználónak karakteres képernyőn megjeleníti azt. Grafikus képernyő továbbítására a telnet nem alkalmas, ahhoz a *távoli asztal* (remote desktop) protokoll használható.

Amennyiben a telnetet különféle hálózati protokollok vizsgálatára használjuk, akkor Linux alatt a telnet kliensnek az elérni kívánt gép szimbolikus neve vagy IP címe után szóközzel elválasztva kell megadnunk a portszámot, ahol a vizsgált alkalmazás szerver programja elérhető. Például így:

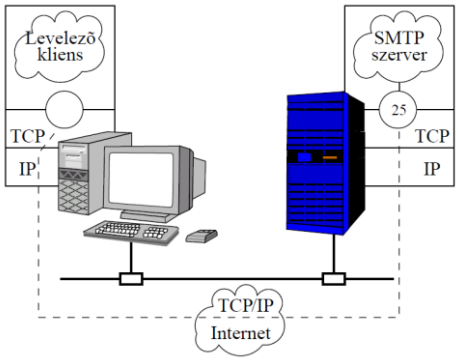
```
telnet localhost 25
```

A fenti paranccsal a helyi gép 25-ös portján figyelő SMTP szerverhez férhetünk hozzá tesztelési célból.

Simple Mail Transfer Protocol

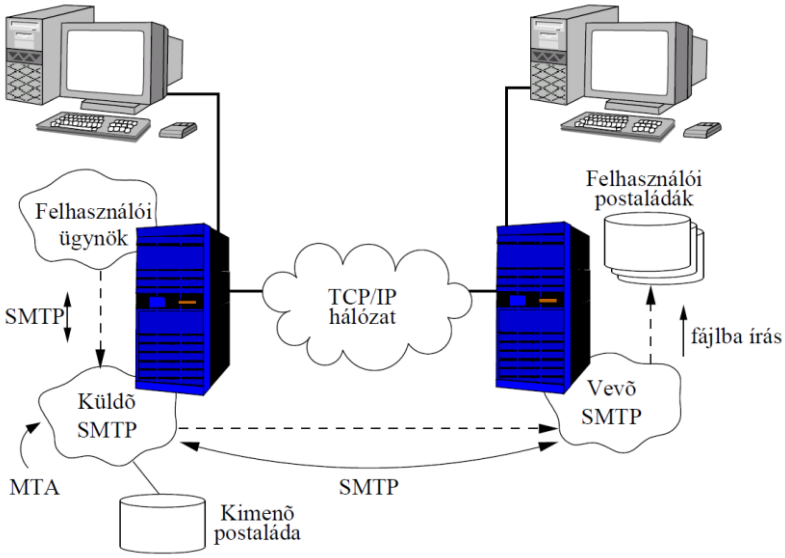
Az SMTP ASCII kódú szöveges üzenetek továbbítására képes TCP/IP protokollt használó hostok között, ha azok levelezésre is konfigurálva vannak.

A levél átvitelének folyamata



5. ábra. Az SMTP kliens-szerver architektúra

Amikor a felhasználó kezdeményezi egy levél küldését, a *felhasználói ügynök* (Mail User Agent, MUA vagy röviden csak UA, azaz a felhasználó által használt „levelező program", például Thunderbird) kapcsolatba lép a benne beállított *kimenő SMTP szerver*rel és átadja neki a levelet. Az 5. ábrán a kliens szerver kapcsolat látható. A TCP kapcsolat a szerver oldalon a 25-ös porton épül ki, míg kliens oldalon ez nem meghatározott előre (az operációs rendszer dinamikusan oszt ki egy szabad portot). A kliens az SMTP protokoll segítségével átadja a szervernek a levelet, és utána lebontják a TCP kapcsolatot.



6. ábra. Levélküldés SMTP-vel

Kövessük nyomon a 6. ábrán, hogy mi történik ezután! A kimenő SMTP szerver a levelet egy kimenő postafiókban tárolja. Amint rákerül a sor, az *üzenetátviteli ügynök* (Message Transfer Agent, MTA, Unix operációs rendszer esetén lehet például Postfix) megkísérli annak átvitelét, azaz az e-mail cím alapján a DNS által szolgáltatott információt felhasználva (MX record, lásd később) TCP kapcsolatot épít ki a címzett leveleit kezelő SMTP szerverrel és megtörténik az átvitel, melynek során szintén az SMTP protokollt használják. Ha ez nem sikerül, akkor a levél a kimenő postafiókban marad, és az MTA

bizonyos időközönként újra megkísérli az átvitelt.[7] A vevő oldalon található MTA fogadja a kapcsolódást, veszi az üzenetet, és elhelyezi a címzett bejövő postaládájába.

Vegyük észre a 6. ábrán, hogy az eredeti koncepció szerint ugyanazt az SMTP protokollt használják a felhasználói ügynök (MUA) és a kimenő SMTP szerver (küldő MTA), valamint a küldő és a fogadó MTA-k között. Ma viszont az elharapózó *spam* (kéretlen levélküldés) miatt a MUA és az küldő MTA között az úgynevezett *Message Submission Protocol* (RFC 6409) szerinti üzenetváltás zajlik, ami az SMTP-hez nagyon hasonlít, de a 25-ös helyett az 587-es porton kommunikál, és megköveteli a kliens azonosítását.

Az SMTP üzenetformátumát eredetileg az RFC 822-ben definiálták, ma az RFC 5322 az érvényes szabvány, de a levelek fejrészét ma is gyakran hívják 822-es fejrésznek. Egy példa a levélcímre:

`lencse@rs1.sze.hu`

A @ jel előtti szöveg megadja a postaláda nevét (**`lencse`**), a @ jel utáni szöveg pedig egyszerű esetben a host FQDN nevét (**`rs1.sze.hu`**). Az utóbbi azonban nem szükségszerű. Mutatunk egy másik példát, ahol nem így van:

`lencse@hit.bme.hu`

A @ jel utáni rész alapján az SMTP a DNS segítségével megkeresi az adott zónához tartozó levelező kiszolgálót.[8] Kérdezzük le mi is a **`host`** parancs segítségével:

```
lencse@dev:~$ host -t mx hit.bme.hu
hit.bme.hu mail is handled by 5 frogstar.hit.bme.hu.
```

A fenti címre érkező leveleket tehát a **`frogstar.hit.bme.hu`** gép fogja kezelni.

Nem ASCII-ben kódolt információ átvitele

Ha nem ASCII kódolású szöveges üzenetet akarunk küldeni SMTP-n keresztül, hanem bináris fájlt, hangot, képet, akkor megfelelő kódolási eljárást kell alkalmaznunk, amely az átviendő információt ASCII kódokká alakítja.

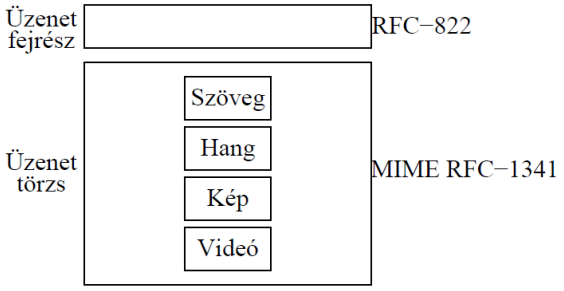
A klasszikus eljárás Unix alatt a **`uuencode`**. Az így kódolt üzenet visszaalakítására a **`uudecode`** parancs használható.

Egy korszerűbb megoldás (ilyenkor is szöveges üzenetet küldünk SMTP-n keresztül) a MIME (Multipurpose Internet Mail Extensions) protokoll. (Eredeti definíciója: RFC 1341, aktuális definíciója: RFC 2045-2049.) A MIME képes különböző típusú adatokat is kódolni, mint egyszerű szöveg, gazdagabban formázott szöveg (rich text), kép, hang, videó, HTML dokumentum és így tovább. A MIME megadja a tartalom típusát (pl. text/plain, application/pdf) és az átvitelnél alkalmazott kódolást (pl. quoted-printable, base64) is. A MIME üzenet törzse részekre osztott tartalommal rendelkezik, és a MIME felhasználói ügynök válogathat a megjelenítendő adatok közül. Például egy „dumb” (buta) terminál, amely nem képes sem hang lejátszásra, sem kép, sem videó megjelenítésre, az üzenetnek csak a szöveges részét írja ki a képernyőre. A MIME egy másik hasznos tulajdonsága, hogy használhat mutatót olyan adatra, amely valahol máshol került tárolásra. Például ez a mutató megadhat egy FTP helyet, ahonnan letölthető az adott dokumentum. Ha egy körlevélben ezt megoldást használjuk, akkor csak annak kell megvárnia a letöltést, akit érdekel az információ.

---

[7] További sikertelenség esetén bizonyos idő (néhány óra) után a feladónak figyelmeztető üzenetet (e-mail) küld, de még tovább próbálkozik. Ha néhány nap múlva végül feladni kényszerül, arról is e-mailt küld a feladónak.

[8] Konkrétan az *MX record* adja meg @ jel utáni rész, mint zóna ***M**ail e**X**changer*ét.



7. ábra. MIME üzenet

Az SMTP protokoll parancsai

Az alábbi táblázatban láthatók az SMTP parancsok, amelyekkel üzenetet küldhetünk.

Parancs	Jelentés
HELO *küldő*	A küldő gép, amelyen a MUA fut.
MAIL FROM: *feladó_címe*	Ez a parancs adja meg a feladó postaláda címét.
RCPT TO: *célcím*	Ez a parancs adja meg a cél postaláda nevét. Több címzett esetén többször kell alkalmazni a parancsot.
DATA	A parancs után lehet megadni a levél tartalmát. Az üzenet végét egy olyan sor jelzi, amelyben csak egy pont van.
QUIT	A parancs hatására a vevő OK választ küld és bezárja a kapcsolatot.
RSET	Ez a parancs törli az épp aktuális folyamatot, utána újra lehet kezdeni.
NOOP	Ez a parancs nem hajt végre műveletet. A vevő egy OK választ ad. A parancs akkor hasznos, ha meg akarunk győződni róla, hogy a kapcsolat rendben van-e, a szerver működik-e.

Minden SMTP parancs (vagy annak első tagja) négy karakter hosszú. Az SMTP szerver az SMTP parancsok fogadása után egy három számjegyes állapot kóddal és egy szöveges üzenettel válaszol a következő formátumban:

`nnn Szöveges üzenet`

Ahol az *nnn* a három számjegyből álló állapotkód, ami az SMTP kliens programnak szól. Az első számjegy az eredmény (hiba) jellegét adja meg, például: 2xx: pozitív eredmény, 4xx: tranziens hiba, 5xx: permanens hiba, stb. A számjegyeket követő szöveges üzenet az emberi értelmezést szolgálja. A lehetséges válaszok közül néhányat példaként bemutatunk az alábbi táblázatban (magyar fordításban).

Kód	Jelentés
250	Kért levél művelet OK, sikeresen befejeződött.
450	Kért levél művelet nem történt meg, a postaláda nem elérhető. Például a postaláda foglalt.
550	Kért levél művelet nem történt meg, a postaláda nem elérhető.
354	Kezdje a levelet. Befejezés: <CR><LF>.<CR><LF>. (Azaz egy sorban csak egy pont áll.)

A hagyományos postai levél küldésének példáján mutatunk be néhány fontos fogalmat. Amikor megírunk egy hivatalos levelet, a fejléces levélpapíron rajta van a saját céges címünk, illetve ráírjuk a címzett teljes címét is. Ugyanezeket a címeket ráírjuk a borítékra is. Az elektronikus leveleknél az ún. *envelope sender* és *envelope recipient* az, amit a „MAIL FROM:” és „RCPT TO:” parancsok segítségével

tudnunk megadni. A felhasználó levelező programjában megjelenő feladót és címzettet viszont a DATA parancs segítségével adjuk meg. Itt megadhatunk még további mezőket is (például a levél tárgya, dátuma, stb.), majd azoktól egy üres sorral elválasztva kezdődik a levél szövege.

Példa az SMTP protokoll működésére

Az alábbiakban bemutatunk egy példát arra, hogy SMTP parancsokkal hogyan lehet levelet küldeni. A példában az *envelope sender* és az *envelope recipient* valamint a levélben megjelenő feladó és címzett szándékosan eltérnek.

```
lencse@server:~$ telnet localhost 25   ← kapcsolódunk az MTA-hoz
Trying ::1...                          ← a telnet írja ki
Connected to localhost.                ← a telnet írja ki
Escape character is '^]'.              ← a telnet írja ki
220 server.test ESMTP Postfix (Debian/GNU)  ← a szerver köszönt bennünket
helo localhost                         ← a MUA a helyi gépen fut
250 server.test                        ← az MTA válasza, ez az FQDN
mail from: lencse@sze.hu               ← envelope sender megadása
250 2.1.0 Ok                           ← az MTA válasza
rcpt to: lencse@hit.bme.hu             ← envelope recipient megadása
250 2.1.5 Ok                           ← az MTA válasza
data                                   ← minden mást ezen belül adunk meg
354 End data with <CR><LF>.<CR><LF>    ← MTA: így kell befejezni a data részt
From: mikulas@meseorszag.hu            ← a levélben ez jelenik meg feladóként
To: ovodasok@pirosovi.hu               ← a levélben ez jelenik meg címzettként
Subject: Ajandek                       ← a levélben ez jelenik tárgyként
                                       ← üres sor a levél szövegének elválasztására
Kedves Gyerekek!                       ← a levél szövegének eleje
Hamarosan megyek, addig is rendesen viselkedjetek!
Sziaszok.                              ← a levél szövegének vége
.                                      ← ebben a sorban csak egy pont van
250 2.0.0 Ok: queued as 636EE60062     ← az MTA válasza
```

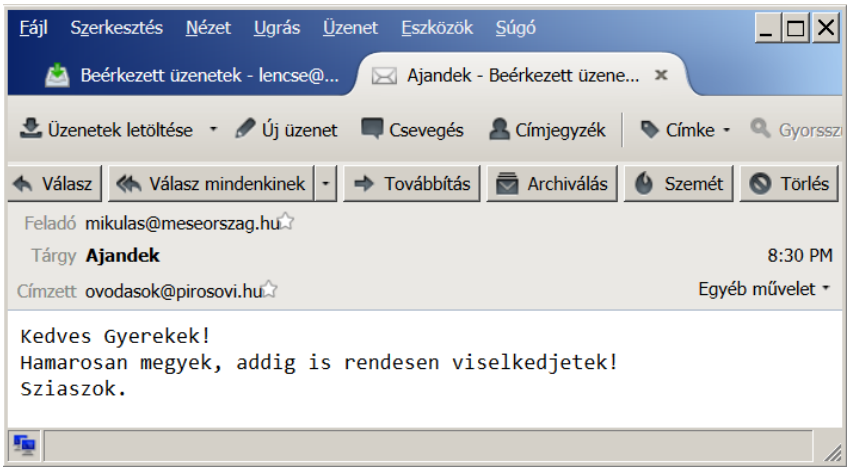
Ezt a levelet a **lencse@hit.bme.hu** postaládacímre a szerver kézbesítésre elfogadta, mivel a levél szerver válasza **Ok** volt. A kapcsolatot a szerver ekkor még nem zárja le, további leveleket lehet küldeni. Egy idő után aztán a szerver lezárja a kapcsolatot, ha nem adunk ki további parancsokat:

```
421 4.4.2 server.test Error: timeout exceeded  ← az MTA hibaüzenete
Connection closed by foreign host.             ← a telnet írja ki
lencse@server:~$                               ← újra a parancsértelmezőt (shell) látjuk.
```

Most nézzük meg a 8. ábrán, hogy a levelező kliens programban (MUA) hogyan jelenik meg az imént küldött levelünk. Természetesen a fentiek célja csak a demonstráció volt. A valóságban nem küldünk mások nevében levelet – még akkor sem, ha ezt technikailag meg tudjuk tenni!

Fontosnak tartjuk még megemlíteni, hogy egy MTA esetén alapvető fontosságú annak beállítása, hogy milyen domainekből fogadhat el levelet kézbesítésre (kiknek *relay*ezik). *Open relay*nek nevezik azokat, amelyek bárkitől elfogadnak levelet, és ezeket általában fekete listákra szokták tenni (más MTA-k nem fogadnak tőlük levelet), mert spam küldésre használhatók. Érdeklődőknek bővebben: https://en.wikipedia.org/wiki/Open_mail_relay.

Ezenkívül az MTA-kon (pl. Sendmail, Postfix, Exim) még számos korlátozás állítható be, például az üzenet méretére vonatkozólag, vagy már a levél átvételekor ellenőrizhető a feladó és a küldő címének érvényessége, stb. [7].



8. ábra. Így néz ki a küldött levél.

Postafiók implementációk

A levelek küldése és fogadása szempontjából a postafiók konkrét implementációja közömbös, de a mindennapi életben lehet jelentősége a konkrét implementációnak. Unix rendszerekben a klasszikus megoldás az ún. „mbox” formátum, amikor a postafiókot egyetlen egy fájlban tárolják, és az új leveleket a sor elején kezdődő öt karakteres „**`Mail `**” (az 5. karakter egy szóköz) kombinációról lehet felismerni. (Ha esetleg a levélben ilyen előfordul, akkor azt úgy tárolják, hogy: „**`>Mail `**”, és a megjelenítéskor kihagyják a „**>**” karaktert.) A megoldás de facto szabvány, és gyakran használják nem Unix rendszerekben is. Utólag, az RFC 4155-ben dokumentálták.

Az egyes felhasználók leveleit Linux rendszerekben alapértelmezés szerint sok esetben a **`/var/spool/mail/username`** fájlban tárolják, de ez nem szükségszerű, a fájl helyezhető akár a felhasználó home könyvtárába is.

A megoldás hátránya, hogy sok levél és jelentős méretű csatolt fájlok esetén a mailbox fájl mérete nagyon nagy lehet, a bennük való keresés, törlés jelentősen lassíthatja a levelezést egy sok felhasználóval rendelkező szerveren.

A megoldás alternatívája az ún. „maildir” formátum. Ebben az esetben minden levél egy külön fájlban található. (A korszerű fájlrendszerek a nagyszámú fájl problémáját például hasheléssel oldják meg.)

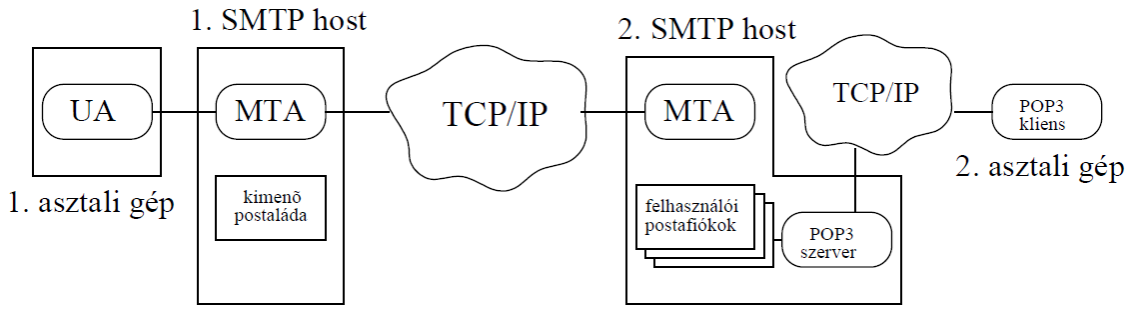
Elhelyezés: gyakori megoldás, hogy a felhasználó home könyvtárában egy **`Maildir`** nevű könyvtárat használnak, de természetesen lehet más nevet is választani. Ezen belül aztán még három könyvtár van: **`new`**, **`cur`**, **`tmp`**. Egy levelet érkezése közben az MTA mindig a **`tmp`** könyvtárba helyez egyedi névvel. Aztán amikor a teljes levél megérkezett, akkor áthelyezi a **`new`** könyvtárba. (Így a MUA sohasem találkozik félig megérkezett levéllel.) A MUA pedig a **`new`** könyvtárból a **`cur`** könyvtárba helyezi át a leveleket, mielőtt megnyitja őket.

Érdeklődőknek további információ: https://en.wikipedia.org/wiki/Maildir.

Léteznek még további postafiók implementációk is. Ezekről, valamint az egyes MTA és MUA implementációknak az egyes postafiók implementációkkal való kompatibilitásáról egy jó összefoglaló található itt: http://wiki2.dovecot.org/MailboxFormat.

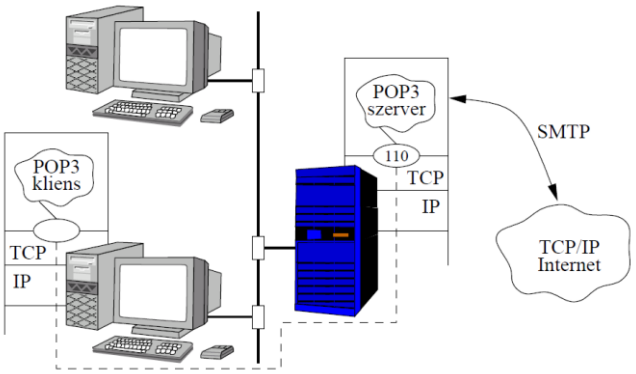
Post Office Protocol Version 3

Az SMTP protokoll elvárja, hogy a levelet fogadó mail szerver on-line (bekapcsolt és a hálózaton elérhető) legyen, különben a TCP kapcsolat nem hozható létre. Éppen ezért asztali számítógépeknél az SMTP önmagában nem alkalmas a levelezés feladatának teljes körű megoldására, mivel az asztali gépeket munka után ki szokták kapcsolni. Erre a problémára egy jó megoldás, ha az SMTP-vel küldött levelek fogadását egy olyan SMTP host végzi, amely mindig be van kapcsolva. Ez az SMTP host látja el a postafiók (mail-drop) szolgálatot. A munkaállomások kapcsolatba lépnek az SMTP hosttal, majd letöltik az üzeneteket egy kliens/szerver levelező protokoll segítségével, mint például a POP3 (Post Office Protocol version 3), melynek leírása megtalálható az RFC 1939-ben. Bár az üzenetek letöltéséhez a POP3-at használjuk, a munkaállomás felhasználója továbbra is az SMTP-t használja azok elküldéséhez. A 9. ábrán látható, hogy hol van az egyes protokollok helye az asztali gépen dolgozó felhasználók közti levelezésben. Természetesen a felhasználói levelező program mind a User Agent (UA), mint a POP3 kliens funkciót ellátja.



9. ábra. Asztali gépen dolgozó felhasználók közti levelezés megoldása SMTP és POP3 segítségével

A POP3 a TCP szállítási protokollt használja, és a POP3 szerver a szabványos 110-es TCP porton érhető el. A POP3 protokoll kliens-szerver architektúrája 10. ábrán látható.



10. ábra. POP3 kliens/szerver architektúra

Az SMTP-hez hasonlóan a POP3 kliens is négy ASCII karakterrel kódolt parancsokkal kommunikál a POP3 szerverrel. A válaszok itt némileg eltérnek, a szerver nem számmal és szöveges üzenettel válaszol, hanem +OK vagy -ERR üzenettel jelzi, hogy a parancs végrehajtása sikeres volt-e vagy sem. Majd a válasz érdemi része következik, amit parancsonként ismertetünk azok részletes leírásánál.

A POP3 parancsait két csoportra bontjuk, ezek a *kötelező* és az *opcionális* parancsok. A kötelező parancsoknak minden implementációban szerepelniük kell. A POP3 protokoll kötelező parancsait,

azok paraméterezését és jelentését a következő táblázatban mutatjuk be. (A szögletes zárójel a paraméter opcionális voltát jelenti.)

Parancs	Jelentés
STAT	Erre a parancsra olyan választ kapunk, amely egy +OK stringgel kezdődik, majd egy szóköz után az üzenetek száma és még egy szóköz után a levelek összmérete látható oktetekben megadva.
LIST [üzenet_sorszám]	Ha megadunk üzenet sorszámot, akkor a parancs hatására a szerver válaszképpen kiírja a kért üzenetazonosító mellé a méretét oktettekben. Ha nem adunk meg sorszámot, akkor egy többsoros válasz kapunk, melynek első sora egy +OK stringgel kezdődik, majd egy szóköz után az üzenetek száma és még egy szóköz után a levelek mérete látható (oktettekben kifejezve). A következő sorok elején az üzenet sorszáma és szóköz után hozzá tartozó üzenet mérete látható.
RETR üzenet_sorszám	Ez a parancs arra szolgál, hogy letöltsük a POP3 szerveren lévő üzenetünket. Válaszként ad egy +OK string-et, majd egy szóköz után a letöltött üzenet méretét oktettekben. Ezután következik maga az üzenet (az, amelyiknek a sorszámát megadtuk). Ha olyan azonosítót adunk meg, amely nem létezik, egy -ERR üzenetet küld a szerver.
DELE üzenet_sorszám	A parancs az adott üzenetet törölésre kijelöli.
NOOP	Ez a parancs nem hajt végre műveletet. A szerver egy +OK választ ad. A parancs akkor hasznos, ha meg akarunk győződni róla, hogy a kapcsolat rendben van-e, és a POP3 szerver működik-e.
RSET	Ez a parancs törli az összes törlésre való kijelölést. A szerver visszaad egy +OK stringet.
QUIT	A parancs hatására a szerver törli az összes törlésre kijelölt levelet, és az elvégzett művelettől függően +OK vagy -ERR stringet ad vissza, majd lezárja a kizárólagos elérést a postafiókon, és lebontja a TCP kapcsolatot.

A következőkben a POP3 opcionális parancsait mutatjuk be.

Parancs	Jelentés
USER név	Ezzel a paranccsal adható meg a kezelni kívánt postaláda.
PASS jelszó	Ez a parancs adja meg a kiválasztott postaládához tartozó jelszót.
TOP üzenet_sorszám n	A parancs hatására a szerver válasza egy +OK string, majd kiírja az üzenet fejrészét és egy üres sor után az üzenet törzsből n sort. Ha ez a szám nagyobb, mint ahány sor van az üzenetben, akkor a szerver elküldi az összes sort.
UIDL [üzenet_sorszám]	Minden üzenet kap egy egyedi azonosítót (UID), amely alapján bárhol azonosítható lesz a levél. Ez a parancs az UID azonosító alapján listázza ki az üzeneteket.
APOP név digest	A név azonosítja a felhasználót, a digest pedig egy MD5-ös (Message Digest version 5) digest string. Ezt a parancsot az egyszerű nyílt szöveg stringeket használó USER/PASS azonosítási metódus helyett szokták használni. A legfontosabb tulajdonsága, hogy a jelszót nem nyílt szövegként küldi el.

Bár a USER és a PASS opcionális parancsok az RFC 1939-ben, de általában támogatottak. A USER és PASS parancsok azért opcionálisak, mert van egy másik lehetőség: az APOP parancs, amely az MD5 (Message Digest version 5) hitelesítő eljárást használja.

A 11. ábrán bemutatunk egy példát arra, hogyan kommunikál egymással a POP3 kliens és szerver. (K jelöli a klienst, SZ pedig a szervert.)

SZ:	(kapcsolatra vár a 110-es TCP porton)	*Kapcsolódási állapot*
K:	(megnyitja a kapcsolatot)	
SZ:	+OK POP3 users.tilb.sze.hu v2000.70 server ready	
K:	USER <*felhasználó_ név*>	*Hitelesítési állapot*
SZ:	+OK User name accepted, password please	
K:	PASS <*jelszó*>	
SZ:	+OK Mailbox open, 8 messages	
K:	STAT	*Tranzakciós állapot*
SZ:	+OK 8 153681	
K:	LIST	
SZ:	+OK Mailbox scan listing follows	
SZ:	1 29486	
SZ:	2 512	
	...	
SZ:	8 65677	
SZ:	<CR><LF>.<CR><LF>	
K:	RETR 1	
SZ:	+OK 29486 octets	
	itt van a levél tartalma	
SZ:	<CR><LF>.<CR><LF>	
K:	QUIT	*Frissítési állapot*
SZ:	+OK Sayonara	
K:	(bezárja a kapcsolatot)	
SZ:	(vár a következő kapcsolatra)	
	K=Kliens SZ=Szerver	

11. ábra. POP3 kliens és POP3 szerver kommunikációja

A példában látható, hogy a POP3 kapcsolat kezdeti részében belépünk a *kapcsolódási állapotba.* A kapcsolódási állapotban létrehozzuk a TCP kapcsolatot a POP3 szerverrel. A következő lépésben a POP3 kapcsolat belép a *hitelesítési állapotba*. Ebben az állapotban a felhasználónak meg kell adnia a felhasználói nevét illetve a jelszavát, hogy hitelesíthesse a szerver. A korábbi POP3 implementációkban a felhasználói név és jelszó információkat nyílt szövegként továbbították, ami azt jelenti, hogy ha valaki megszerezte a csomagokat, kiolvashatta belőlük a felhasználói név + jelszó kombinációkat. Biztonságosabb alternatívát nyújt az RFC 1939-ben leírt MD5-ös hitelesítési folyamat. Miután a felhasználót hitelesítette a szerver, a POP3 kapcsolat átlép a *tranzakciós állapotba*. Ebben az állapotban számos parancs kiadására van lehetőségünk, mint például a STAT, LIST, RETR, DELE, RSET és így tovább. A példában a POP3 kliens kiad egy STAT parancsot, amire válaszul megkapja, hogy 8 üzenete érkezett, és azok együttes mérete 153681 oktett. A POP3 kliens ezután a LIST parancsot használja, hogy lekérdezze az üzenetek listáját. A szerver válaszképpen megadja az üzenet azonosító száma mellé az adott üzenet méretét is. Ezután a kliens a RETR parancsot használja az üzenet azonosítóval, hogy letöltse a levelét. A POP3 kliens beállításaitól függően a kliens letörölheti a letöltött üzenetet a szerverről. Ha a POP3 kliens kiadja a QUIT parancsot, a POP3 kapcsolat átlép a *frissítési állapot*ba. Ebben az állapotban mind a POP3 kliens, mind a POP3 szerver frissíti a belső állapotait, hogy rögzítse, mennyi üzenet van a postaládáikban. Végül a TCP kapcsolat bezárul.

Internet Message Access Protocol Version 4

A POP3 egy jó kliens/szerver protokoll a leveleink letöltéséhez, azonban néhány esetben ez kevés lehet. Például POP3-on keresztül nem vizsgálhatjuk meg a leveleinket, mielőtt letöltöttük volna azokat, és a POP3 nem teszi lehetővé a levelekkel (azok részeivel) való közvetlen műveletvégzést a szerveren. Tehát, ha ilyen feladatokat szeretnénk megoldani, akkor az IMAP4-et célszerű használnunk a POP3 helyett.

Az Internet Message Access Protocol Version 4 (revision 1) egy olyan kliens/szerver protokoll, mellyel elérhetjük és szerkeszthetjük elektronikus leveleinket a szerveren. A protokoll lehetővé teszi, hogy a távoli postafiókon elvégezhessük ugyanazokat a műveleteket, amelyeket a helyi postaládánkon el tudunk végezni. Az IMAP4 továbbá támogatja a kapcsolat nélküli munkát, és lehetőséget biztosít a szerverrel való újraszinkronizálásra kapcsolatfelvétel esetén.

Egy IMAP4 kliens szolgáltatásai lehetővé teszik a következőket:

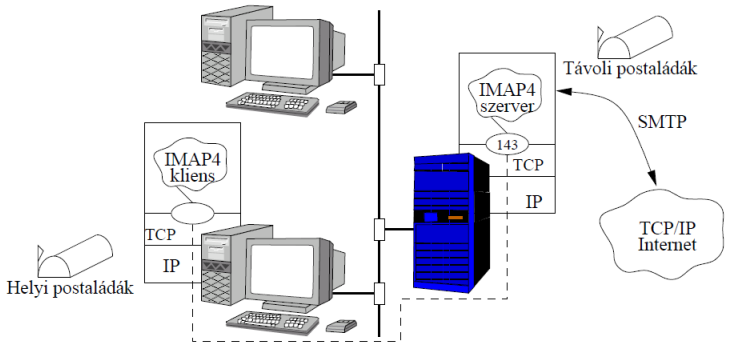
- levelek elérése és szerkesztési lehetősége a szerveren anélkül, hogy letöltenénk őket
- levelek és csatolt fájlok áttekintése anélkül, hogy letöltenénk őket
- levelek letöltése kapcsolat nélküli munkához
- szinkronizálás a helyi és a szerveren lévő postaládák között.

Az IMAP4 műveletei között szerepelnek:

- postaládák létrehozása, átnevezése, törlése
- új levelek érkezésének ellenőrzése
- levelek eltávolítása a postaládából
- levelek állapotjelzőinek beállítása, és törlése
- RFC-822 fejrész felismerése és MIME kódolású levelek elemzése („érti” a MIME kódolást)
- keresés és szelektív letöltés az üzenet tulajdonságaiból, szövegéből, illetve annak részeiből.

Az üzenetek eléréshez az IMAP4-ben számokat használunk. Ezek a számok vagy az üzenetek sorszámai vagy az egyedi azonosítói.

Ugyanúgy, mint a POP3, az IMAP4 sem képes a levelek feladására. Ezt a funkciót általában az SMTP segítségével oldják meg. A 12. ábra egy IMAP4-es kliens szerver kapcsolatot mutat be.

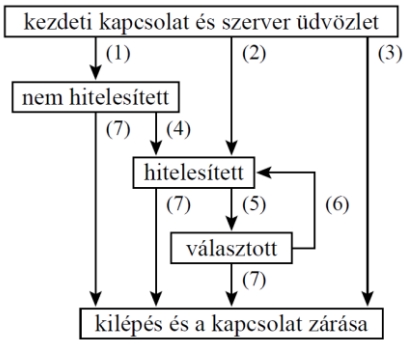


12. ábra. IMAP4 kliens/szerver architektúra

Az IMAP4 viselkedését írja le a 13. ábra egy állapotátmenet diagram segítségével. Az ábrán számozással jelölt állapotátmenetek a következők:

1. kapcsolat előzetes azonosítás nélkül (OK üdvözlés)

2. előzetesen azonosított kapcsolat (PREAUTH üdvözlés)
3. visszautasított kapcsolat (BYE üdvözlés)
4. sikeres LOGIN vagy AUTHENTICATE (azonosítás) parancs
5. sikeres SELECT (választás) vagy EXAMINE (vizsgálat) parancs
6. CLOSE (bezárás) parancs, vagy sikertelen SELECT, illetve EXAMINE parancs
7. LOGOUT parancs, a szerverről való kilépés és a kapcsolat bontása.



13. ábra. IMAP4 állapotátmenet diagram

Az IMAP4 a 13. ábrán látható állapotok valamelyikében található. A legtöbb IMAP4 parancs csak bizonyos állapotban adható ki. Ha a kliens nem megfelelő állapotban adja ki a parancsot, akkor protokoll hiba generálódik.

A következő IMAP4 állapotokat definiáljuk ebben a részben:

1. azonosítás előtti állapot
2. azonosított állapot
3. választott állapot
4. kijelentkezett állapot.

Az *azonosítás előtti állapot*ban az IMAP4 kliens azonosítási „okmányokat” nyújt be. A legtöbb parancs nem használható, míg a felhasználó nem azonosította magát. A kapcsolat kezdetén ebbe az állapotba kerülünk, hacsak nem történt előzetes azonosítás (preauthentication).

Az *azonosított állapot*ban a felhasználó már hitelesített, de mielőtt kiadná a parancsokat, ki kell választania egy postaládát. Ebbe az állapotba akkor lépünk, ha az előzetesen azonosított kapcsolat kezdődik, vagy ha a kliens elfogadható azonosítási okmányokat nyújtott be. Amennyiben hiba történik a postaláda kiválasztásánál, újra bekerülünk ebbe az állapotba, hogy egy másik postaládát választhassunk ki.

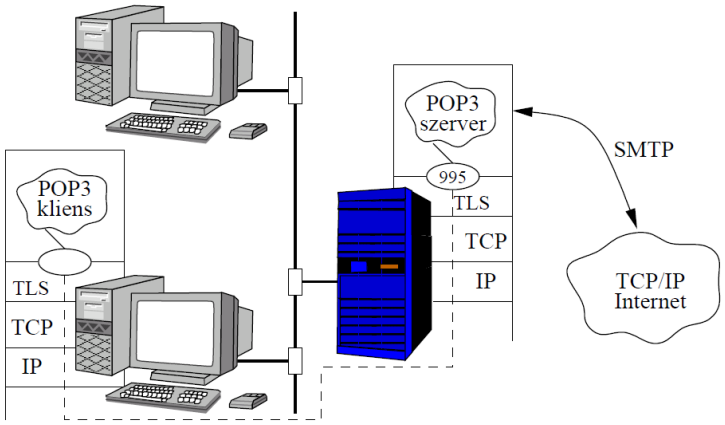
A *választott állapot*ban vagyunk, ha sikeresen kiválasztottuk a kezelni kívánt postaládát.

*Kijelentkezett állapot*ba a kliens kérése, vagy a szerver döntése nyomán kerülhetünk. Ekkor az IMAP4 szerver bezárja a kapcsolatot.

POP3S és IMAP4S

POP3S

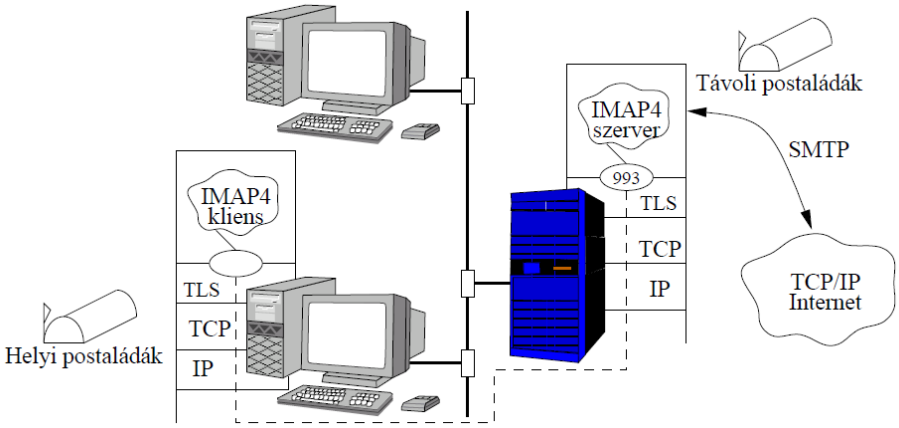
A POP3 protokoll biztonságos verziója. Gyakorlatilag kiegészül egy újabb réteggel: SSL 3.0 (Secure Socket Layer 3-as verzió), vagy TLS 1.0 (Transport Layer Security 1.0). Lásd 14. ábra. A szerver a 995-ös porton kommunikál. A protokoll parancsai megegyeznek a POP3 parancsaival, csak az a különbség, hogy itt a TLS réteg miatt a kliens-szerver kommunikáció egy titkosított csatornán folyik, így nem lehallgatható.



14. ábra. POP3S kliens/szerver architektúra

IMAP4S

Az IMAP4 protokoll biztonságos verziója. Gyakorlatilag kiegészül egy újabb réteggel: SSL 3.0 (Secure Socket Layer 3-as verzió) vagy TLS 1.0 (Transport Layer Security 1.0). Lásd 15. ábra. A szerver a 993-as porton kommunikál. A protokoll parancsai megegyeznek az IMAP4 parancsaival, csak az a különbség, hogy itt a TLS réteg miatt a kliens-szerver kommunikáció egy titkosított csatornán folyik, így nem lehallgatható.



15. ábra. IMAP4S kliens/szerver architektúra

SSH: távoli elérés biztonságosan

Elméleti alapok

Az SSH csomag arra szolgál, hogy számítógép-hálózaton keresztül egyik gépről egy másik gépre biztonságosan tudjunk információt továbbítani.

Az SSH is kliens-szerver architektúrában működik, az ssh szerver (**`sshd`**) a 22-es TCP porton várja az ssh kliens csatlakozását. Egy ssh kapcsolat létrejöttéhez azonban szükség van néhány előzetes feltételre, konkrétan bizonyos kulcsok rendelkezésre állására, amelyek a következők:

- gépenként nyilvános és titkos kulcs
- felhasználónként nyilvános és titkos kulcs (ha kulcs alapú authentikációt szeretnénk használni)

Egy ssh kapcsolat létrejöttének és működésének a menete a következő

1. titkos csatorna létrehozása a kép gép között
2. a felhasználó azonosítása
3. titkosított kommunikáció

Ezeket most egy kicsit részletesebben is megnézzük.

Titkos csatorna létrehozása a kép gép között

Előfeltétel: A kliens programot futtató gépnek ismernie kell a szerver programot futtató gép nyilvános kulcsát. (Egyébként lásd: sebezhetőség.)

A lényeg: kapcsolatkulcs létrehozása, amit a kapcsolat lezárásáig a két fél minden további kommunikációja során titkos kulcsú titkosítás kulcsaként használ. Ezzel a kulccsal valamilyen titkos kulcsú algoritmussal (3DES, blowfish, CAST128, Arcfour) titkosítják az adatfolyamot.

Érdeklődőknek: Diffie-Hellman kulcscsere protokoll [6]. (Valójában nem kulcsok cseréjéről, hanem kulcsmegegyezésről van szó.)

Sebezhetőség: Ha a kliens gépen nincs meg a szerver gép nyilvános kulcsa, és a felhasználó úgy dönt, hogy elfogadja a – remélhetőleg a szerver által felajánlott – kulcsot, akkor azt kockáztatja, hogy amennyiben a kulcs a támadótól származik, akkor nem a szerverrel, hanem a támadóval hozott létre közös kapcsolatkulcsot.

A felhasználó azonosítása

Az authentikáció a következő három, erejében egyre gyengülő megoldás valamelyikével történik:

1. Erős azonosítás nyilvános kulcsú módszerrel
2. Jelszavas azonosítás
3. Berkeley r*[9] (szerű megoldás) használata

Ezek közül a nyilvános kulcsú módszert a gyakorlatban is megismerjük.

A jelszavas azonosítás azért lehetséges, mert a jelszó is az első lépésben létrehozott titkos csatornán megy át. Éppen ebből származik a sebezhetősége is, ha a felhasználó a támadó által felajánlott nyilvános kulcsot fogadott el!

[9] Bizalomra épülő transzparens távoli elérési megoldás, bővebben lásd: [1].

Megjegyzés: A Berkeley r* további fájlokkal bővül (a **/etc/hosts.equiv** és a **~/.rhosts** fájlokon kívül): **/etc/ssh/shosts.equiv** és **~/.shosts**. De ezzel a megoldással bővebben nem foglalkozunk, a gyakorlatban le szokták tiltani.

Titkosított kommunikáció

Amennyiben az előző két lépés hibátlan volt, akkor az említett hagyományos titkosítók valamelyikével titkosítják az ssh kliens és szerver közötti adatfolyamot.

SSH a gyakorlatban

Az egyes gépeken **~/.ssh/known-hosts** fájlban lehet tárolni bizonyos távoli gépeknek a nyilvános kulcsát. Ha az SSH kliens kapcsolatot vesz fel a távoli géppel és a nyilvános kulcsa itt megtalálható, akkor létrejöhet a biztonságos kommunikáció. Ha egy távoli gép kulcsát elfogadtuk, de valójában nem azzal a géppel kommunikálunk, amellyel szándékoztunk, akkor jelszavas azonosítás esetén a támadó kezébe kerül a jelszavunk.

Ha távoli gépre való belépéskor erős azonosítást szeretnék használni, akkor előbb létre kell hoznunk az **ssh-keygen** programmal egy kulcspárt. Ekkor létrejön két fájl a **~/.ssh/** könyvtárunkban: a DSA algoritmus használata esetén az **id-dsa** tartalmazza a titkos és az **id-dsa.pub** tartalmazza a publikus kulcsunkat; RSA esetén a fájlok: **id-rsa** és **id-rsa.pub**. A publikus kulcsunkat el kell helyeznünk azon a gépen, ahova erős azonosítással szeretnék belépni a megfelelő fájlban (ha az SSH protokoll 2-es verzióját használjuk, akkor ez a **~/.ssh/authorized_keys2**). Ezt akár „kézzel” is megtehetjük, de célszerű a következő parancs használata:

```
ssh-copy-id -i ~/.ssh/id_dsa.pub felhasznalo@gepnev
```

(Természetesen RSA esetén az **id_rsa.pub** fájlt adjuk meg.)

Amennyiben valaki a jelenlegitől eltérő felhasználói névvel szeretne belépni egy gépre, akkor a felhasználói nevet megadhatja így: **ssh felhasznalo@gepnev** vagy **ssh -l felhasznalo gepnev**.

Az SSH-t nem csupán távoli gépre történő karakteres terminállal való bejelentkezésre lehet használni, hanem port forwardolásra is, amivel más adatfolyamokat is titkosan vihetünk át. A gyakorlatban például X elérésre is szokták használni.

Az SSH csomag fontos eleme az **scp** parancs is, ami helyi és távoli gép közötti másolást tesz lehetővé a szokásos Unix szintaxissal[10]. Működését két példával mutatjuk be. Tegyük fel, hogy a **pc6** gép előtt ülünk, és a **pc2** gépen **joska** a felhasználói nevünk. Másoljuk át a helyi gépen az aktuális könyvtárban található **cica.jpg** fájlt a **pc2** gépre a **/tmp** könyvtárba **macska.jpg** névre:

```
[jozsi@pc6 ~]$ scp cica.jpg joska@pc2:/tmp/macska.jpg
```

Amennyiben a **pc2** gépen a home könyvtárunkban van egy **kutya.jpg** nevű fáj, amit a helyi gép aktuális könyvtárába szeretnénk másolni, akkor azt megtehetjük például így is:

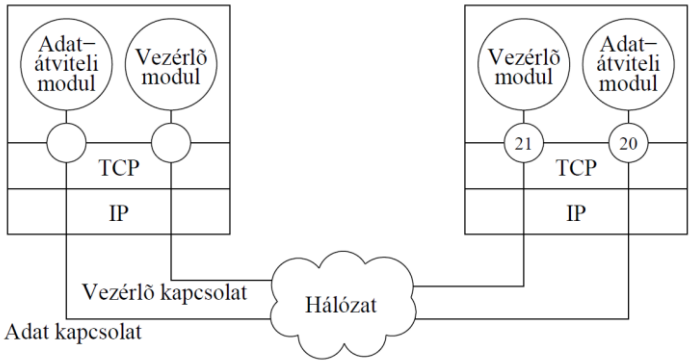
```
[jozsi@pc6 ~]$ scp –l joska pc2:kutya.jpg .
```

Figyeljük meg a felhasználói név megadásának alternatív módját, a home könyvtár jelölésének (**/~joska/** vagy **/~/**) elhagyhatóságát, valamint azt, hogy a helyi aktuális könyvtárat (**.**) viszont kötelező jelölni!

[10] A segédlet utolsó fejezetében adunk egy rövid bevezetőt a Linux rendszerekről.

File Transfer Protocol

A korábbiakhoz hasonlóan az FTP (RFC 959) is kliens-szerver architektúrában működik, amint azt a 16. ábra mutatja. Az FTP kliens segítségével a felhasználó fájlokat tud a szerverről letölteni, illetve – ha a szerveren engedélyezve van –, akkor oda feltölteni. Az FTP az IP fölött TCP-t használ a fájlok átvitelére. Az FTP kliens a felhasználó gépén fut, és kapcsolatot kezdeményez a szerverrel. Az FTP szerver a szabványos 21-es TCP porton várja a kapcsolat felépítését. Amikor a kapcsolat kérés megérkezett, lejátszódik a háromutas kézfogás a TCP kapcsolat felépítéséhez. Ezt a TCP összeköttetést *vezérlő kapcsolatnak* hívják, és az FTP parancsok és válaszok küldésére használják. Amikor az FTP kliens kiad egy parancsot az adatátvitel megkezdésére, akkor egy másik TCP kapcsolat épül ki. Ennek iránya kétféle lehet. Az ún. *"aktív mód"* esetén a szerver kezdeményezi az *adat kapcsolat* felépítését (SYN) a kliens felé. Ehhez a kliens a vezérlőkapcsolaton keresztül elküldi az IP-cím + portszám párost, ahol a kapcsolat felépítését fogadja. Mikor az FTP szerver megkapja ezeket, akkor a szabványos 20-as TCP portról elindítja a TCP kapcsolat felépítését megadott IP-cím és port szám alapján a kliens felé. Az *adat kapcsolat* kizárólag a kért fájl (vagy könyvtárlista) adatainak átvitelére szolgál, utána lebontják. A tűzfalak miatt ma nagyon gyakran úgynevezett *"passzív módú"* FTP-t használunk, ami azt jelenti, hogy a szerver küldi el az IP-cím + portszám párost a kliensnek, és a kliens kezdeményezi az adatkapcsolatot a szerver gép megadott portja felé.



16. ábra. Az FTP működési modellje

Összegzésül, a következő kétféle összeköttetés jön létre FTP használatakor:

- Vezérlő kapcsolat (TCP, KLIENS_IP, KLIENS_PORT1, SZERVER_IP, 21)
- Adat kapcsolat (TCP, KLIENS_IP, KLIENS_PORT2, SZERVER_IP, 20[11])

Az adat kapcsolat csak a fájl (vagy könyvtárlista) átvitelének idejére jön létre és bezáródik, amint az átvitel befejeződött. Új kapcsolat jön létre minden alkalommal, amikor egy fájlt átviszünk. A vezérlő kapcsolat fennmarad, amíg azt a kliens vagy a szerver be nem zárja. Általában a kliens szakítja meg a kapcsolatot, azonban túlterhelés vagy más probléma esetén a szerver is bezárhatja. Például ha adott ideig (timeout) az FTP kliens nem fordul hozzá, akkor be szokta zárni.

FTP kliens-szerver kommunikáció parancsai

A kliens a vezérlő kapcsolaton keresztül küldött parancsokkal kommunikál a szerverrel. Az FTP parancsok maximum négy karakterből állnak, opcionálisan paraméterekkel kiegészítve. A parancsokat és jelentésüket a következő táblázatban adjuk meg. Az opcionális parancsok nevét dőlt betűvel írtuk. (A szögletes zárójel a szokott módon opcionális argumentumot jelöl.)

[11] Passzív mód esetén más portszám a szerver dinamikus portszám tartományából.

Vezérlő parancs	Jelentés
USER <felhasználói_név>	Megadja az FTP kapcsolatot kezdő felhasználó nevét.
PASS <jelszó>	Megadja a USER parancsban megadott felhasználóhoz tartozó jelszót.
ACCT <account>	Megadja a PASS parancs után a további account információkat.
CWD <könyvtár>	Megváltoztatja az aktuális könyvtárat a szerveren.
CDUP	Átvált az aktuális könyvtárról annak szülő könyvtárára.
SMNT <elérési_út>	Mountol egy külső fájlrendszer adatstruktúrájából anélkül, hogy megváltoztatná a felhasználói nevet vagy jelszót.
REIN	Visszaállítja a kezdeti értékeket a felhasználó azonosítása előtti állapotba. A USER parancsnak kell követnie.
QUIT	Bezárja az FTP kapcsolatot.
PORT h1,h2,h3,h4,p1,p2	Megadja a TCP végpont információkat. Segítségével megadhatjuk az FTP kliens portszámát az FTP szervernek. A h1-h4-ig az IP-cím, p1 és a p2 pedig a portszám oktettjei decimálisan, vesszőkkel elválasztva. Értelmezésük a hálózati bájtsorrend (MSB) szerinti.
PASV	Megadja, hogy az FTP szerver várakozzon az adatkapcsolatra, amit a kliens fog létrehozni. Erre a parancsra adott válasz tartalmazza a TCP végpontot, ahol az FTP szerver várja a kapcsolódást.
TYPE <kód>	Megadja az adatmegjelenítés típusát.
STRU <kód>	Megadja a fájl struktúráját. (Például: fájl, rekord, oldal.)
MODE <kód>	Megadja az átvitel módját. (Például: byte folyam (stream), blokk (block), tömörített (compressed).)
RETR <elérési_út>	Megadott fájl letöltése.
STOR <elérési_út>	Megadott fájl tárolása a szerveren (feltöltés).
STOU <elérési_út>	Hasonló, mint a STOR, azonban az eredményfájlnak, melyet létrehoz, egyedi neve kell, hogy legyen a könyvtárban.
APPE <elérési_út>	Hasonló, mint a STOR, azonban ha a megadott fájlnév már létezik, az új adatot folyamatosan hozzáírja a régihez.
ALLO <argumentumok>	Helyet foglal a szerveren a fájlnak.
REST <argumentumok>	Meghatároz a szerveren egy pontot, ahonnan a fájl átvitele folytatódhat.
RNFR <elérési_út>	Megadja a régi elérési utat ahhoz a fájlhoz, amely át lesz nevezve. Az RNTO parancsnak kell követnie.
RNTO <elérési_út>	Megadja az új elérést a fájlhoz. Az RNFR parancs után kell használni.
ABOR	A szerver megszakítja az aktuális parancs végrehajtását.
DELE <elérési_út>	Megadott fájl törlése.
RMD <könyvtár>	Megadott könyvtár törlése.
MKD <könyvtár>	Megadott könyvtár létrehozása.
PWD	Kiírja az aktuális könyvtár elérési útját a szerveren.
LIST [könyvtár]	Kiírja a megadott könyvtárban lévő fájlok neveit és attribútumait. Ha nincs megadva könyvtár, akkor az aktuális könyvtárat listázza.
NLST [könyvtár]	Kiírja a megadott könyvtárban lévő fájlok neveit attribútumok nélkül. (Könyvtár neve nélkül az aktuálisat. Az mget parancs használja.)
SITE <paraméterek>	Megadja a szerver specifikus parancsait.
SYST	Megkapható a paranccsal, hogy milyen operációs rendszer fut a távoli gépen.
HELP [parancs]	Segítséget nyújt a parancs használatához. Ha nem adunk meg parancsot kiírja a szerveren használható összes parancsot.
NOOP	Nincs művelet, a szerver küld egy visszaigazolást.

Az FTP szerver válasza egy három számjegyű kódból, illetve szóköz után opcionálisan szöveges üzenetből áll:

`nnn Szöveges üzenet`

Az *nnn* a három számjegyű kódot jelenti. Például az FTP szerver válaszolhatja a következőt:

`200 Command okay.`

Az alábbi táblázatban példaként megadtuk az FTP szerver néhány lehetséges válaszát és azok jelentését (magyar fordításban).

Kód	Jelentés
200	Parancs rendben
500	Szintaktikus hiba, nem értelmezett parancs
125	Adat kapcsolat már nyitva, átvitel kezdődik
425	Adat kapcsolatot nem tudja megnyitni
226	Adat kapcsolat bezárása
426	Kapcsolat bezárva, átvitel megszakadt
227	Passzív módba lépés (h1,h2,h3,h4,p1,p2)
331	Felhasználó neve rendben, jelszóra vár
250	Kért fájl művelet rendben lezajlott
350	Kért fájl művelet további információra vár
450	Kért fájl művelet nem történt meg

FTP a felhasználó szemszögéből

Az FTP lehetővé teszi a felhasználó számára a távoli gépen történő interaktív fájl- és könyvtárhozzáférést és a következő műveletek végrehajtását:

- A helyi vagy távoli gépen lévő könyvtárak listázása
- Fájlok átnevezése és törlése (jogosultság szükséges)
- Fájlok átvitele a távoli gépről a helyi gépre (letöltés)
- Fájlok átvitele a helyi gépről a távoli gépre (feltöltés, általában csak adott könyvtárba engedélyezik)

Ahhoz, hogy FTP-t használjunk, szükséges egy kliens alkalmazás. Ilyen kliens alkalmazások sok platformra elérhetők. Bizonyos platformok rendelkeznek grafikus felhasználói interfésszel (Graphical User Interface, GUI), míg mások parancssort használnak. Az oktatás szempontjából érdemesebb a parancssoros interfészt tárgyalni, mert könnyebb az egyes parancsok azonosítása.

Ahhoz, hogy megnyissunk egy FTP kapcsolatot, a következő parancsot kell kiadnunk:

`ftp [host_név]`

A *host_név* annak az FTP szervernek a szimbolikus neve vagy IP címe, amelyre be szeretnénk jelentkezni. Ha a host nevét nem adjuk meg, akkor csak néhány FTP parancs adható ki.

Ha be szeretnénk jelentkezni egy FTP szerverre, akkor az **`ftp`** parancs kiadása után a következő lépés az azonosítás, melyben meg kell adni a felhasználói nevet és a jelszót. Az FTP szerver az FTP host hitelesítő mechanizmusát használja. Tehát ha van egy **`joska`** felhasználónév bejegyzésünk a távoli hoston, akkor bejelentkezhetünk az FTP szerverre **`joska`** néven, és használhatjuk a hoston megadott jelszavunkat.

Számos FTP host támogatja az **anonymous** (névtelen) bejelentkezést. Amennyiben **anonymous**t adunk meg felhasználói névként, régebben jelszóként a rendszerben az e-mail címünket szokták kérni (amit 20 évvel ezelőtt még meg is adtunk, és általában nem éltek vissza vele) ma azonban ez már nem szokás, általában ki is írják, hogy bármit meg lehet adni jelszóként. (Ha mégis e-mail címet kérnének, akkor jó tudni, hogy néhány esetben csak a „@" karakter meglétét figyelik.) Ha sikerült belépnünk, akkor a helyi adminisztrátor által korlátozott jogokkal rendelkezünk. Tipikusan a **/pub** könyvtárból lehet valamit letölteni, és esetleg az **incoming** könyvtárba feltölteni.

A következő példa egy rövid FTP kapcsolatot kísér végig.

1. Kiadjuk az FTP parancsot és mellé a host nevét.

```
C:\Users\student>ftp ftp.hu.debian.org
```

2. Ha a host elérhető, a következőhöz hasonló üzenetet kapunk. A bejelentkezés részletei eltérhetnek.

```
Connected to ftp.freepark.org.
220 ftp.freepark.org FTP server (Version 6.00LS) ready.
User (ftp.freepark.org:(none)):
```

3. Megadjuk a felhasználónevet és a jelszót.

```
User (ftp.freepark.org:(none)): anonymous
331 Guest login ok, send your email address as password.
Password:
230 Guest login ok, access restrictions apply.
ftp>
```

4. A bejelentkezés után használhatjuk a **?** vagy a **help** parancsot:

```
ftp> help
Commands may be abbreviated.  Commands are:

!               delete          literal         prompt          send
?               debug           ls              put             status
append          dir             mdelete         pwd             trace
ascii           disconnect      mdir            quit            type
bell            get             mget            quote           user
binary          glob            mkdir           recv            verbose
bye             hash            mls             remotehelp
cd              help            mput            rename
close           lcd             open            rmdir
ftp>
```

5. A **pwd** paranccsal megkaphatjuk, hogy a távoli hoston mi az aktuális könyvtár.

```
ftp> pwd
257 "/" is current directory.
ftp>
```

6. Könyvtár váltásra a **cd** parancsot használjuk

```
ftp> cd /pub/linux/distributions/debian/dists/wheezy/
250 CWD command successful.
ftp>
```

7. Az aktuális könyvtár fájljainak listáját az **ls** vagy **dir** parancsokkal kaphatjuk meg.

```
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for 'file list'.
Contents-kfreebsd-i386.gz
Contents-kfreebsd-amd64.gz
Contents-mips.gz
Contents-ia64.gz
Contents-i386.gz
Release
Contents-mipsel.gz
Contents-amd64.gz
Release.gpg
Contents-s390x.gz
contrib
main
Contents-armhf.gz
non-free
Contents-armel.gz
ChangeLog
226 Transfer complete.
ftp: 263 bytes received in 0,00Seconds 263000,00Kbytes/sec.
ftp>
```

8. Ha tudjuk, hogy egy Unix rendszer FTP szerverére csatlakoztunk, akkor a következő hasznos parancsot használhatjuk. Ha **ls** parancsot használunk, a Unix **ls** parancsa fog lefutni. Kiegészíthető az **ls** parancs bármely Unix opcióval, mint például a **-lR** opció, amely egy rekurzív hosszú formátumú listát ad a fájlokról és alkönyvtárakról. A **-lR** opció nem szabványos FTP parancs, de alkalmazható Unix szervereken, illetve ott, ahol emulálják azt. A parancs segítségével egy gyors áttekintést kaphatunk az FTP hoston elérhető fájlokról.

9. Ahhoz, hogy egy fájlt másoljunk a távoli gépről a helyi hostra, a következő FTP parancsot kell kiadnunk:

```
get <távoli_fájl> [helyi_fájl]
```

Ha a **helyi_fájl**-t nem adjuk meg akkor ugyanaz lesz a fájl neve a helyi gépen, mint a távolin. Szövegfájloknál az FTP támogatja a sorvégjel konverziót a különböző operációs rendszereknél, ha az átviteli mód ASCII. A bináris fájlok továbbításához használjuk a **bin** parancsot, hogy kikapcsoljuk ezt a konverziót.

Unix alatt a sorok végét a 10-es kódú karakter (<LF>) jelzi. Windows alatt a sorok végét a 13-as és 10-es kódú karakterek (<CR><LF>) együttesen jelzik. ASCII módú átvitel esetén Windows -> UNIX irányban minden <CR><LF> karakterpárt az <LF> karakter helyettesít. Unix -> Windows átvitel esetén minden <LF> karakter helyett <CR><LF> karakterpár kerül a szövegbe.

10. Az adott szerverrel való kapcsolatot bezárhatjuk a **close** paranccsal, úgy, hogy az **ftp** programból még nem lépünk ki. A **quit** a parancs bezárja az esetleg élő kapcsolatot, és az **ftp** programból is kilép:

```
ftp> quit
221 Goodbye.

C:\Users\student>
```

HTTP

A *HTML* (HyperText Markup Language) nyelvű oldalakat a *HTTP* (HyperText Transfer Protocol) segítségével tölthetjük le. A HTML nyelvű oldalak összességét *World Wide Web*nek, gyakran csak *web*nek (magyarul pókháló) hívják. A web sok *web szerver*ből áll. A web szerverek dokumentumokat tárolnak, amelyek tartalmazhatnak például szöveget, képet, hangot és videót, melyeket *weblapokba* (web pages) szerveztek. Minden weblap tartalmazhat *link*eket (hyperlink), amelyek mutatók web dokumentumokra. A hyperlinkek mutathatnak azonos, illetve különböző web szerveren lévő dokumentumokra egyaránt. A linkekkel keresztül-kasul behálózott dokumentumok szövevénye átnyúlik ország- és kontinenshatárokon, ezt fejezi ki a World Wide Web (világméretű pókháló) egyik kedves korai (de sajnos el nem terjedt) magyarítása: világszőttes.

A HTML dokumentumokat a web kliensek, azaz web böngészők segítségével érheti el a felhasználó, melyet saját gépén futtat. Miután a böngésző letöltötte a HTML dokumentumot, megjeleníti azt a képernyőn grafikusan, esetleg szöveges formátumban. A HTML leíró nyelv alkalmas a HTML dokumentum különböző elemeinek leírására, melyeket felhasználva a böngésző grafikusan megjelenítheti a weblapot. A linkek általában aláhúzott szövegként jelennek meg. (Természetesen nemcsak szöveg, hanem kép is lehet link.) A linkekre kattintva letölti azt a dokumentumot, amire a link mutat.

A következőkben aprócska ízelítőt adunk a HTML nyelvből, a HTTP protokollt majd utána mutatjuk be.

HyperText Markup Language

A HTML a Standard Generalized Markup Language (SGML) egy implementációja. Az SGML szabvány megad egy általános módszert, hogy miképpen készítsünk olyan dokumentumokat, amelyek hyperlinkeket tartalmaznak. A *hyperlink* egy kiemelve látható kifejezés a szövegben, amelyet kiválasztva egy újabb dokumentumot kapunk. Egy hyperlink kiválasztása többféle cselekményt is kiválthat, mint például egy kép megjelenítése, levél küldése, felhasználótól adatok kérése *form*on keresztül, távoli bejelentkezés vagy fájl átvitel kezdeményezése, adatbázis lekérdezése, program futtatása és így tovább. Az olyan dokumentumot, amely hyperlinket tartalmaz, *hyperdokumentum*-nak is hívjuk.

Minden HTML dokumentum *tag*eket tartalmaz (magyarul leginkább *címkének* lehetne nevezni), a következő struktúrában:

```
<tag>
</tag>
```

A *tag* egy hivatkozás egy speciális kulcsszóra, amelyet a HTML dokumentum különböző komponenseinek megadására használunk. A lezáró tag **</tag>** megadja a komponens végét. Majdnem minden HTML tagnek megvan a hozzá tartozó lezáró tagje. Például minden HTML dokumentum eleje és vége követi a következő megadást:

```
<HTML>
A HTML különböző elemeit e két tag közé kell elhelyezni.
</HTML>
```

A **<HTML>** és **</HTML>** tagek között meg kell adni a HTML dokumentum fejrészét és törzsét. A fejrészt a **<HEAD>** és **</HEAD>** tagek között, és a törzset pedig a **<BODY>** és **</BODY>** tagek között kell megadni:

```
<HTML>
<HEAD> Ez itt a fejrész helye. </HEAD>
<BODY> Ez itt a törzs helye. </BODY>
</HTML>
```

Az üres sorokat és a sortörés karaktereket a tagek között nem vesszük figyelembe. Például a következő HTML kód jelentésben teljesen megegyezik az előzővel:

```
<HTML><HEAD> Ez itt a fejrész helye. </HEAD>

<BODY> Ez itt a törzs helye. </BODY></HTML>
```

Az üres sorok és a sortörések általában javítják a HTML dokumentumok forrásának olvashatóságát, ezért ajánlatos alkalmazni őket. A tagek nem érzékenyek a kis- illetve nagybetűkre és az egyéb szövegformázásra. Ha azt szeretnénk, hogy a böngésző által megjelenített weblapon sortörést, újsort, vagy egyéb formázást hajtsunk vége, akkor speciális HTML tageket kell a forrásban elhelyeznünk. A weblap címét a **<TITLE>** és **</TITLE>** tagek között kell megadni. A következő szöveg a böngésző címrészében (az ablak címsorában) fog megjeleni:

```
<HTML>
<HEAD>
<TITLE>Ez az első egyszerű weblapom!</TITLE>
</HEAD>
<BODY> Ez itt a törzs helye. </BODY>
</HTML>
```

Hyperlink megadásához használjuk a következő taget:

```
<A HREF="URL" egyéb_paramtérek > hyperlink szöveg </A>
```

A tagnek számos paramétert lehet megadni az *egyéb paraméterek* résznél, melyek módosítják a tag viselkedését. Általában a HREF paramétert használjuk az *URL* cím megadására. A *Uniform Resource Locator* feladata, hogy meghatározza egy erőforrás helyét az Interneten. Az URL általános szintakszisa:

```
protokoll://gépnév/elérési_út
```

A *protokoll* többféle lehet az Interneten használt protokollok közül, mint például: **http**, **ftp**, **telnet**, **gopher**, **file**. Ha megadunk egy HTML dokumentumot egy másik gépen, akkor a **http** (HyperText Transfer Protocol) protokollt szoktuk használni a letöltésére. Ha egy dokumentumot FTP-vel szeretnénk letölteni, az **ftp** protokollt kell megadnunk, és így tovább. Ha egy helyi fájlt szeretnénk megnyitni a saját gépünkről a web böngészőben, akkor pedig a **file** protokollt kell megadnunk.

A *gépnév* egy IP cím vagy egy DNS szimbolikus név, amely az erőforrást birtokló hostot adja meg. Az *elérési_út* a könyvtárat és a fájl nevét adja meg, ahol az erőforrás elérhető. Az *elérési_út* érzékeny lehet a kis- és nagybetűkre, ha a web szerver Unixot használó hoston fut. Az *elérési_út* opcionális. Ha nem adjuk meg, akkor a HTML dokumentum alapértelmezett neve Unix alapú szervereknél általában **index.html**. Itt látható néhány példa az URL címekre:

```
http://www.hit.bme.hu/~lencse
ftp://ftp.bme.hu
telnet://users.tilb.sze.hu
file://~lencse/public_html/index.html
```

mailto:lencse@hit.bme.hu

Megjegyzés: az URL-nél általánosabb az URI, ami a Uniform Resource Identifier rövidítése. Érdeklődőknek ajánljuk: http://en.wikipedia.org/wiki/Uniform_Resource_Identifier.

Nézzük meg a következő HTML forrást, amelyben bemutatunk néhány hasznos tag-et:

```
<HTML><HEAD><TITLE>Hasznos tag-ek</TITLE></HEAD>
<BODY> <!-- Ez itt egy megjegyzés.>
<H1>Bemutatunk néhány tag-et</H1>
<P>Ezt a bekezdést egy bekezdés taggel kezdtük. A bekezdés tag
automatikusan formázza a szöveget megjelenítéskor.</P>
<BR><!-- Az előző tag sortörést idéz elő a szövegben.>
<HR><!-- Az előző tag egy vízszintes vonalat húz.>
</BODY></HTML>
```

A HTML lehetőséget ad számozott és számozatlan listák létrehozására, melynek tagjai külön-külön sorban lesznek. A számozatlan lista minden tagja elé egy jelet tesz. A lista szintaktikája a következő:

```
<OL> <!-- Számozott lista>
<LI> Lista elem 1
<LI> Lista elem 2

<LI> Lista elem N
</OL>

<UL> <!-- Számozatlan lista>
<LI> Lista elem 1
<LI> Lista elem 2

<LI> Lista elem N
</UL>
```

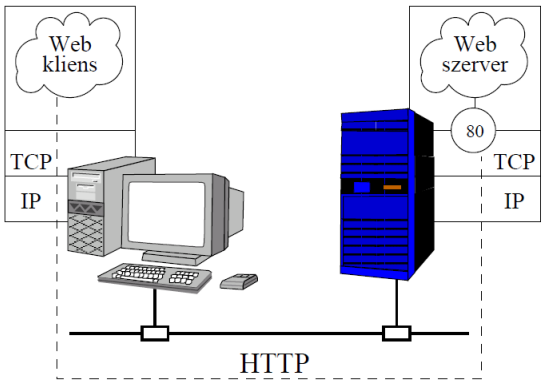
Érdemes még megjegyezni, hogy a **<B>** és **</B>** tag-ek közé írt szöveg félkövér, míg a **<I>** és **</I>** tagek közé írt szöveg dőltbetűs lesz. Továbbá használhatunk fejrészeket, amelyek **<H1>**-től **<H6>**-ig különböző megjelenítésűek. A weboldalainknak az oldalon (tehát nem a böngésző címsorában) megjelenő címét **<H1>**-gyel szoktuk megadni.

HyperText Transfer Protocol

A web kliens (böngésző) a HTTP protokoll segítségével kommunikál a web kiszolgálóval. A web kiszolgálót hívják web szervernek, HTTP szervernek is. Az egyik legelterjedtebb megvalósítása az Apache **httpd**.

Érdekesség: a segédlet készítésekor ez volt az egyes web szerverek piaci részesedésének aránya: http://news.netcraft.com/archives/2015/08/13/august-2015-web-server-survey.html.

Mielőtt letöltünk egy HTML dokumentumot, létrejön egy kapcsolat a böngésző és a web szerver között. A TCP kapcsolat alapértelmezetten a 80-as porton jön létre (17. ábra). Megadhatunk más portot is, (például egy második web szervert tipikusan a 8080-as portra szoktak tenni), de alapértelmezetten a 80-as portot foglalták le a HTTP számára.



17. ábra. HTTP kliens-szerver kapcsolat

Ha azt szeretnénk, hogy a web böngészőnk ne az alapértelmezett portot használja, akkor az URL címben meg kell adnunk a kívánt portot. Az alábbi példában egy URL-t láthatunk, amellyel a 8080-as portra kapcsolódunk:

```
http://www.szerverem.hu:8080
```

A kapcsolat létrejötte után a web böngésző küld egy kérést a HTTP protokollt használva. A kérés tartalmazza a letölteni kívánt objektum nevét és az általunk használt HTTP protokoll verziószámát. A HTTP protokoll szöveg stringeket használ, így az emberek számára is könnyen érhetőek a kiadott parancsok. Például a következő HTTP kérést küldi el a web böngésző a web szervernek egy dokumentum letöltéséhez:

```
GET /index.html HTTP/1.0
```

Itt a **`GET`** HTTP parancsot használjuk az **`/index.html`** letöltéséhez. A **`HTTP/1.0`** argumentummal pedig megadtuk, hogy milyen verziószámú HTTP protokollt szeretnénk használni. Vegyük észre, hogy a kérésből teljesen hiányzik a web kiszolgáló címe! Ez alapesetben azért nem okoz(ott) gondot, mert a TCP kapcsolat már úgyis a megfelelő kiszolgálóval jött létre. Virtuális webszerverek esetén azonban egyetlen kiszolgáló több név alatt is működik, és szüksége van a host információra, ezért az 1.1-es HTTP verzióban el is küldik.[12] Például:

```
GET /index.html HTTP/1.1
HOST: ipv6.tilb.sze.hu
```

A szerver a HTML dokumentum letöltésére irányuló kérés megérkezése után küld egy HTTP választ. A HTTP válasz három részből áll:

- státusz (response status)
- fejrész (response header)
- adat (response data)

A *válasz státusz* egyetlen sor, amely tartalmazza a szerver HTTP verziójának számát; a státusz kód megadja a kérés eredményét (ezt könnyű a web böngészőnek értelmeznie), majd az utána következő szöveg emberek számára értelmezi a státusz kód jelentését. A következő példában látható egy válasz státusz:

[12] A HTTP 1.0 verzióját az RFC 1945-ben, az 1.1-es verzióját eredetileg az RFC 2068-ban definiálták, amit felváltott az RFC 2616, de már az is obsolete, és helyette egy sereg RFC definiálja a különféle részeit: RFC 7230-7235. (És ez még mindig csak az 1.1-es verzió!)

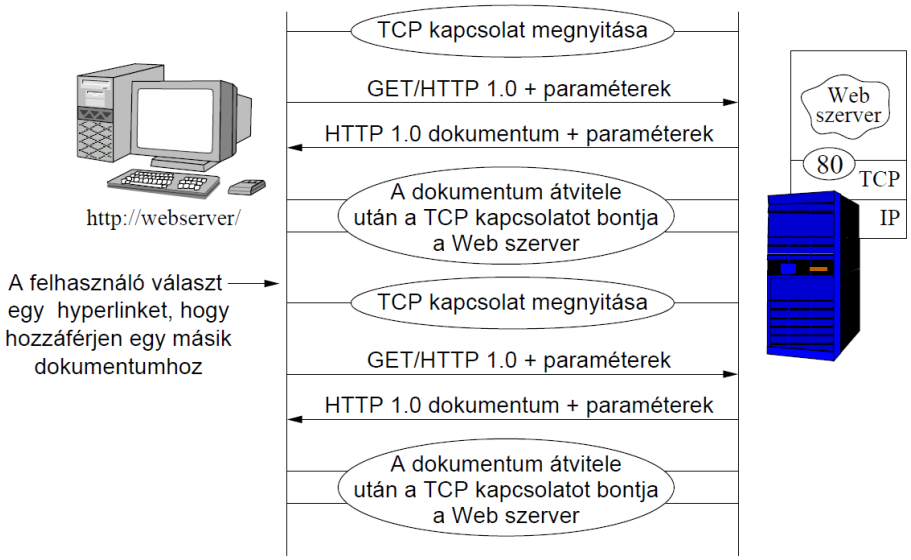
```
HTTP/1.1 200 OK
```

A válasz státusz után következik a *válasz fejrész*. A válasz fejrész tartalmazza a következő információkat: a szerver típusa, MIME verziószám (ha használatos), a tartalom típusa[13] (megadja, hogy az adat részben milyen típusú tartalom van), és végül egy üres sor. A fejrészt és az adatot mindig egy üres sor választja el. Az alábbiakban látható egy példa a válasz fejrészre:

```
Date: Tue, 01 Sep 2015 11:59:34 GMT
Server: Apache/2.2.16 (Debian)
Last-Modified: Mon, 05 Mar 2012 07:10:59 GMT
ETag: "19a09c-d9-4ba79a0d586c0"
Accept-Ranges: bytes
Content-Length: 217
Vary: Accept-Encoding
Content-Type: text/html
```

Ha a dokumentum, amit lekérdezünk egy HTML dokumentum, akkor a válaszadat egyszerű szöveg lesz, amely a HTML dokumentumot tartalmazza.

1.0 verzió esetén, miután a HTTP kérés/válasz lebonyolódott, a TCP kapcsolat bezáródik, és külön TCP kapcsolat jön létre minden egyes HTTP kérés/válasz folyamathoz. A 18 ábrán látható egy tipikus HTTP/1.0 kapcsolat. Az első lépés a TCP kapcsolat megnyitása a 80-as porton. A következő lépésben a kliens küld egy GET HTTP/1.0 parancsot. A dokumentum átvitele után a web szerver bezárja a kapcsolatot. Egy másik dokumentum letöltéséhez egy új kapcsolatot kell létrehozni.



18. ábra. Egy tipikus HTTP/1.0 kommunikáció

1.1-es verzió esetén a kapcsolat bontása vagy fenntartása beállítás kérdése, tipikusan több objektumot is le szoktak tölteni egy kapcsolaton keresztül.

Ismerjük meg jobban a HTTP működését! Egy telnet kliens segítségével jelentkezzünk be egy web szerver 80-as portjára a következő módon:

```
telnet vip.tilb.sze.hu 80
```

[13] A levelezésnél már láttunk példát a MIME által támogatott tartalomtípusokra és kódolásokra.

A kapcsolat létrehozása közben a következőhöz hasonló üzeneteket láthatunk:

```
Trying 2001:738:2c01:8000:193:224:130:171...← A telnet írja ki
Trying 193.224.130.171...                   ← ezeket a sorokat:
Connected to vip.tilb.sze.hu.               ← látjuk, hogy IPv4-en
Escape character is '^]'.                   ← sikerül csatlakoznunk.
```

A következő lépésben úgy teszünk, mintha mi lennénk a web böngésző (kliens) és kiadjuk a GET parancsot a következő módon:

```
GET / HTTP/1.1
Host: ipv6.tilb.sze.hu
```

Megjegyzés: példánkban egyben a virtuális webszerverek használatát is bemutatjuk. A DNS rendszerben az **ipv6.tilb.sze.hu** szimbolikus név egy CNAME (alias) a **vip.tilb.sze.hu** szimbolikus névre. A névfeloldás után az azonos IP-cím miatt a kiszolgáló program csak a Host mező tartalma alapján képes eldönteni, hogy mely oldalt kell küldenie.

Miután lenyomtuk az Enter billentyűt, nem történik semmi, mivel a HTTP kérést két <CR><LF> karakternek kell követnie. Miután másodszor is lenyomjuk az enter billentyűt, megjelenik a HTTP válasz:

```
HTTP/1.1 200 OK                             ← válasz státusz
Date: Tue, 01 Sep 2015 12:12:17 GMT         ← válasz fejrész eleje
Server: Apache/2.2.9 (Debian)
Last-Modified: Tue, 01 Sep 2015 12:01:15 GMT
ETag: "dc48a-117-51eae4e93b8c0"
Accept-Ranges: bytes
Content-Length: 279
Vary: Accept-Encoding
Content-Type: text/html
X-Pad: avoid browser bug                    ← válasz fejrész vége
                                            ← üres sor (elválasztás)
<HTML>                                      ← válasz adatrész eleje
<HEAD>
   <META charset="utf-8">
   <TITLE>ipv6.tilb.sze.hu</TITLE>
</HEAD>
<BODY>
<H1>IPv6 kutatócsoport</H1>
<P>Az oldal szerkesztés alatt áll. Kérjük, addig tekintse meg
publikációinkat <A
HREF="http://www.hit.bme.hu/~lencse/publications/">itt</A>.</P>
</BODY>
</HTML>                                     ← válasz adatrész vége
Connection closed by foreign host.          ← ezt a telnet írta ki
```

A HTTP válaszban megtalálható a státusz, a fejrész, illetve egy üres sor után az adat rész is. A HTML dokumentum átvitele után a kapcsolat nem rögtön, hanem egy idő után bomlott le.

Az előbb látott példában sikeresen letöltöttünk egy HTML dokumentumot. Próbáljuk ki, hogy mi történik, ha egy olyan dokumentumot szeretnénk letölteni, ami nem létezik. Hozzunk létre egy újabb telnet kapcsolatot a web szerverrel és adjuk ki például a következő HTTP parancsot:

```
GET /kutya.html HTTP/1.0
```

Nyomjuk le kétszer az enter-t és a következő választ kapjuk:

```
HTTP/1.1 404 Not Found
Date: Tue, 01 Sep 2015 12:33:01 GMT
Server: Apache/2.2.9 (Debian)
Vary: Accept-Encoding
Content-Length: 288
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL /kutya.html was not found on this server.</p>
<hr>
<address>Apache/2.2.9 (Debian) Server at vip.tilb.sze.hu Port
80</address>
</body></html>
Connection closed by foreign host.
```

Várakozásunknak megfelelően a szerver nem találta a kért dokumentumot. Ezt jelzi a visszaadott a 404-es hibakód és a mellette levő szöveges üzenet is.

A HTTP válasz ilyenkor is tartalmaz egy HTML dokumentumot, amelynek tartalma elárulja, hogy a kért dokumentumot nem találta. Az a HTML dokumentum, amelyet a web szerver visszaadott, egy *virtuális dokumentum*. Ilyen dokumentumok nem találhatók meg a web szerveren, ezeket dinamikusan generálja a szerver hasonló természetű hibák esetén.

Most próbáljuk ki, milyen válasz küld a web szerver egy olyan hibásan kiadott HTTP parancsra, mint például a következő:

```
GET ezt elrontottuk

HTTP/1.1 400 Bad Request
Date: Tue, 01 Sep 2015 12:35:29 GMT
Server: Apache/2.2.9 (Debian)
Vary: Accept-Encoding
Content-Length: 306
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not
understand.<br />
</p>
<hr>
<address>Apache/2.2.9 (Debian) Server at vip.tilb.sze.hu Port
80</address>
</body></html>
Connection closed by foreign host.
```

Megemlítjük, hogy a GET-en kívül más eljárások is vannak, például nagyobb mennyiségű adat felküldésére használható a POST, ahol a kérés törzsében (body) van a feltöltendő adat.[14]

A fejlődés nem állt meg, jelenleg már szabványnak számít a HTTP/2 (RFC 7540), aminek a céljai közé tartozik, hogy tovább csökkentse a felhasználó által érzékelt késleltetést a fejrész mező tömörítésének és még több objektum párhuzamos letöltésének a segítségével. Lehetővé teszi, hogy a szerver a kliens kérése nélkül küldjön neki olyan adatot, amiről valószínűsíti, hogy majd szüksége lesz rá (push). Viszont a HTTP/2 *nem helyettesíti* az 1.1-et (does not obsolete), hanem alternatívája annak.

[14] GET-nél csak az URL-ben lehet, ami kis mennyiségű adatnál nem okoz gondot.

HTTPS

A HTTPS a HTTP biztonságos változata, TLS (Transport Layer Security) vagy SSL (Secure Socket Layer) fölött működő HTTP. Jelentősége igen nagy, hiszen a mindennapi élet során egyre több tranzakciót hajtunk végre webes felületen keresztül. (Például: banki műveletek, rendszerek távoli adminisztrációja.) Biztonsági kockázatot jelent, hogy az SSL 2.0-s verziója biztonsági rést tartalmaz. Ezért kizárólag SSL v3.0-t vagy TLS 1.0-t használjunk!

Az ssh/scp-hez hasonlóan ez is nyilvános kulcsú kriptográfián alapul, itt egy (potenciálisan hamis) tanúsítvány elfogadása jelenti a legnagyobb veszélyt! További veszélyforrás, ha a böngészőnk nem ellenőrzi a CRL-eket (Certificate Revocation List - a visszavont tanúsítványok listája).

A biztonsági megfontolások ismertetése sajnos meghaladja e segédlet kereteit, de szeretnénk felhívni mindenkinek a figyelmét, hogy a mai felhasználói szokások (tanúsítványok automatikus elfogadása) mellett egy sikeres *DNS elleni támadás*[15] esetén a felhasználók gyakorlatilag védtelenek. A hálózati támadásokról az ellenük való védekezésről szól a következő jegyzet [8], amely jelenleg bárki számára elérhető.

[15] A támadó eléri, hogy a kliens programok (például web böngészők) az általa hamisan megadott IP című gépre csatlakozzanak.

Ajánlott irodalom

Az ajánlott irodalom célja az, hogy az egyes témakörök iránt mélyebben érdeklődő hallgatóknak legyen hol tájékozódniuk.

[1] Lencse Gábor: *Hálózati alkalmazások*, (elektronikus jegyzet) Széchenyi István Egyetem, Távközlési Tanszék, Győr, 2008. https://www.tilb.sze.hu/tilb/targyak/NGB_TA027_1/p12.pdf

[2] K. S. Siyan, *Inside TCP/IP*, Third Edition, New Riders Publishing, Indiana, 1997.

[3] Lencse Gábor: *Számítógép-hálózatok*, Universitas-Győr Nonprofit Kft, Győr, 2008.

[4] G. Lencse and A. G. Soós, "Design of a Tiny Multi-Threaded DNS64 Server", *38th International Conference on Telecommunications and Signal Processing (TSP 2015)*, Prague, Czech Republic, July 9-11, 2015, Brno University of Technology, pp. 27-32. http://www.hit.bme.hu/~lencse/publications/TSP-2015-MTD64-for-review.pdf

[5] P. Albitz and C. Liu, *DNS and BIND*, 4th ed., O'Reilly, Sebastopol, CA, 2001.

[6] Wikipedia, "Diffie-Hellman_key_exchange", http://en.wikipedia.org/wiki/Diffie-Hellman_key_exchange

[7] Kallai Péter, Kovács Ákos, Lencse Gábor, Molnár Zoltán Vilmos, Sinkó Gergely: Hálózati operációs rendszerek: Linux, (elektronikus jegyzet) Széchenyi István Egyetem, Távközlési Tanszék, Győr, 2011. http://www.tilb.sze.hu/tilb/targyak/NGB_TA047_1/unix-2011-11-20.pdf

[8] Lencse Gábor: *Hálózatok biztonsága*, (elektronikus jegyzet) Széchenyi István Egyetem, Távközlési Tanszék, Győr, 2005. http://www.tilb.sze.hu/tilb/targyak/NGB_TA0028_1/Halozatok_biztonsaga_0.401.pdf