

Camada Protocolar de Aplicação (Application Layer)

Redes de Comunicações 1

Licenciatura em Engenharia de Computadores e
Informática

DETI-UA, 2021/2022

TFTP - TRIVIAL FILE TRANSFER PROTOCOL

2

Trivial File Transfer Protocol (TFTP)

- Transfer file service with very simple client/server interactions.
- A TFTP client, to receive a file, needs to know the name and directory where the file is stored.
- This service can be used to configure network elements.
- Most *Routers* and *Switches* allow its configuration to be performed through a TFTP server to receive the configuration file.
 - They need the IP address of the server and the name of the file.
 - Very small files.

3

O TFTP é um protocolo de suporte a um serviço de transferência de ficheiros em que as necessidades de interacção entre cliente e servidor são simples. Um cliente TFTP, para receber um ficheiro do servidor, precisa de saber o seu nome e em que directório do servidor ele está armazenado. Este serviço é muito usado na configuração de elementos de rede. A maior parte dos *routers* e *switches* existentes no mercado permitem a sua configuração através de TFTP. No arranque estes equipamentos ligam-se a um servidor de TFTP e recebem o seu ficheiro de configuração. Para isso, basta configurar-lhes o endereço IP do servidor e o nome do ficheiro a pedir.

Este protocolo é implementado sobre o UDP e o porto normalizado do servidor é o 69. Dado que o UDP não garante fiabilidade, o TFTP implementa um mecanismo de controlo de fluxos simples (bem menos complexo que o do TCP) de modo a garantir a integridade da informação transferida (como veremos ilustrado no próximo slide).

O TFTP baseia-se apenas em 5 tipos de mensagens:

RRQ: permite o cliente pedir para ler um ficheiro do servidor

WRQ: permite o cliente pedir para escrever um ficheiro para o servidor

DATA: permite enviar um conjunto de bytes do ficheiro

ACK: permite validar a recepção de uma das primitivas anteriores

ERR: permite sinalizar a impossibilidade de uma das primitivas anteriores

Trivial File Transfer Protocol (TFTP)

- Basic file transfer service (IETF RFC 1350)
 - It does not allow to list server files
 - It does not support authentication
- TFTP runs over UDP
 - The initial client message is sent to server port number 69
 - TFTP is the one that has to address packet retransmission
 - TFTP server answers from another locally selected port number
 - The following messages are exchanged with the selected port number
- TFTP uses Stop and Wait ARQ mechanism
- TFTP has 5 messages:
 - Read Request (RRQ)
 - Write Request (WRQ)
 - Data
 - Acknowledgement (ACK)
 - Error (ERR)

4

O TFTP é um protocolo de suporte a um serviço de transferência de ficheiros em que as necessidades de interacção entre cliente e servidor são simples. Um cliente TFTP, para receber um ficheiro do servidor, precisa de saber o seu nome e em que directório do servidor ele está armazenado. Este serviço é muito usado na configuração de elementos de rede. A maior parte dos *routers* e *switches* existentes no mercado permitem a sua configuração através de TFTP. No arranque estes equipamentos ligam-se a um servidor de TFTP e recebem o seu ficheiro de configuração. Para isso, basta configurar-lhes o endereço IP do servidor e o nome do ficheiro a pedir.

Este protocolo é implementado sobre o UDP e o porto normalizado do servidor é o 69. Dado que o UDP não garante fiabilidade, o TFTP implementa um mecanismo de controlo de fluxos simples (bem menos complexo que o do TCP) de modo a garantir a integridade da informação transferida (como veremos ilustrado no próximo slide).

O TFTP baseia-se apenas em 5 tipos de mensagens:

RRQ: permite o cliente pedir para ler um ficheiro do servidor

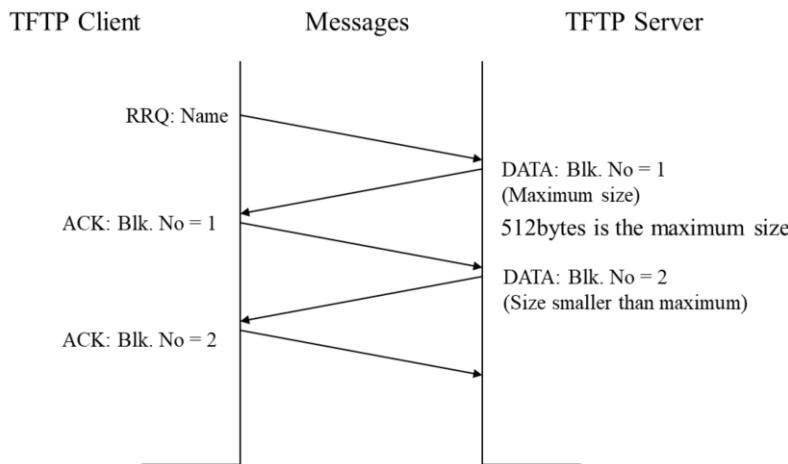
WRQ: permite o cliente pedir para escrever um ficheiro para o servidor

DATA: permite enviar um conjunto de bytes do ficheiro

ACK: permite validar a recepção de uma das primitivas anteriores

ERR: permite sinalizar a impossibilidade de uma das primitivas anteriores

Read Request session



Client detects the last block of DATA when the data block size is lower than the maximum size. If the size of the file is **multiple** of the maximum size, an additional data block is sent with **0** bytes of data. RRQ always with odd number of packets (RRQ+pairs of data and ACK).

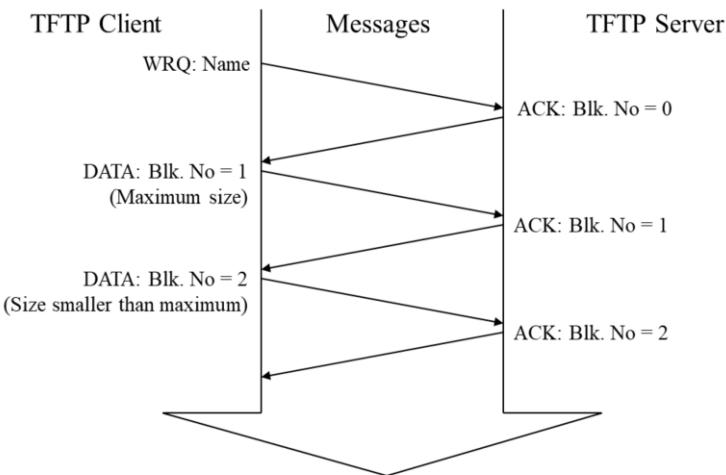
Uma sessão TFTP é sempre iniciada pelo cliente ou com um WRQ (para enviar um ficheiro) ou com um RRQ (para receber um ficheiro).

Numa sessão *Write Request*, após o envio do WRQ, o servidor responde com um ACK para confirmar que está preparado para receber o ficheiro. Depois, o cliente começa a enviar o ficheiro com mensagens DATA seccionando a informação em blocos do tamanho máximo definido por configuração. A cada mensagem DATA, o servidor responde com um ACK confirmando que recebeu o respectivo bloco. O cliente só manda um DATA após receber o ACK do DATA anterior. Se uma qualquer mensagem DATA não chega ao destino (ou o respectivo ACK), o cliente tem um mecanismo de timeout (também configurável na aplicação) em que ao fim do tempo máximo de espera se não recebeu o respectivo ACK, volta a enviar o mesmo bloco de dados.

O servidor detecta a última mensagem DATA quando o tamanho do bloco de dados é menor que o tamanho máximo. Neste caso, a mensagem ACK serviu para o servidor validar a recepção das mensagens WRQ e DATA enviadas pelo cliente.

Cada um dos lados implementa timeout e retransmissão. Se o lado que está a transmitir dados faz timeout, retransmite o último bloco de dados; se o lado responsável por transmitir ACKs faz timeout, retransmite o último ACK.

Write Request session



Server detects the last block of DATA when the data block size is lower than the maximum size. If the size of the file is **multiple** of the maximum size, an additional data block is sent with **0** bytes of data. WRQ always with even number of packets (WRQ+ACK0+pairs of data and ACK).

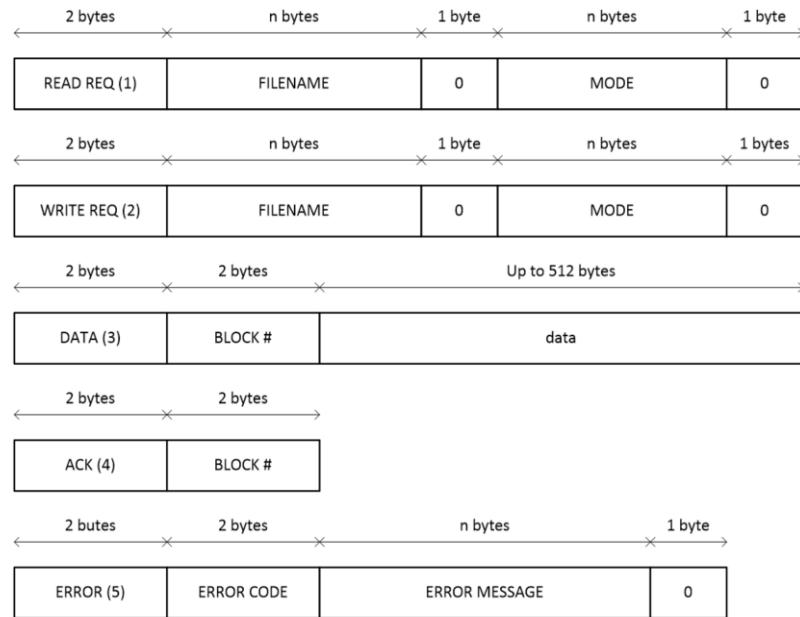
Uma sessão TFTP é sempre iniciada pelo cliente ou com um WRQ (para enviar um ficheiro) ou com um RRQ (para receber um ficheiro).

Numa sessão *Write Request*, após o envio do WRQ, o servidor responde com um ACK para confirmar que está preparado para receber o ficheiro. Depois, o cliente começa a enviar o ficheiro com mensagens DATA seccionando a informação em blocos do tamanho máximo definido por configuração. A cada mensagem DATA, o servidor responde com um ACK confirmado que recebeu o respectivo bloco. O cliente só manda um DATA após receber o ACK do DATA anterior. Se uma qualquer mensagem DATA não chega ao destino (ou o respectivo ACK), o cliente tem um mecanismo de timeout (também configurável na aplicação) em que ao fim do tempo máximo de espera se não recebeu o respectivo ACK, volta a enviar o mesmo bloco de dados.

O servidor detecta a última mensagem DATA quando o tamanho do bloco de dados é menor que o tamanho máximo. Neste caso, a mensagem ACK serviu para o servidor validar a recepção das mensagens WRQ e DATA enviadas pelo cliente.

Cada um dos lados implementa timeout e retransmissão. Se o lado que está a transmitir dados faz timeout, retransmite o último bloco de dados; se o lado responsável por transmitir ACKs faz timeout, retransmite o último ACK.

Messages format



7

FILENAME – string de caracteres ASCII que especifica o nome do ficheiro a ler ou a escrever.

MODE – string de caracteres ASCII que especifica o modo da mensagem; octet ou netascii; octet – 8 bits de raw data, usado para transferir ficheiros; netascii – ASCII de 7 bits standard, usado para enviar mensagens (strings de caracteres).

BLOCK # no ACK é igual ao número de bloco da mensagem recebida. O servidor utiliza o ACK para confirmar a recepção dos blocos de dados e o cliente usa os blocos de dados para confirmar a recepção dos ACK, excepto no caso de ACKs duplicados e de um ACK que termina uma ligação. A recepção de um ACK duplicado pode acontecer apenas quando o primeiro ACK se atrasa, provocando o envio de um bloco de dados (duplicado). Para quebrar o ciclo de transmissões duplicadas (Sorcerer's Apprentice Bug), o bloco de dados corrente nunca é retransmitido como resultado da recepção de um ACK duplicado.

ERROR actua como um NACK; pode causar retransmissão da mensagem ou quebra da ligação.

ERROR MESSAGE – string ASCII que explica o tipo de erro.

ERROR CODE:

00 – Not defined

01 – File not found

02 – Access violation

03 – Disk full

04 – Invalid operation code

05 – Unknown port number

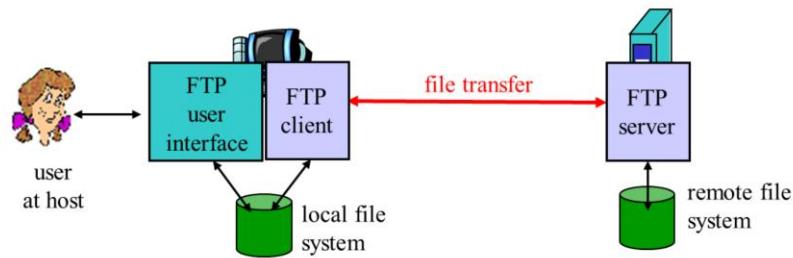
06 – File already exists

07 – No such user

FTP – FILE TRANSFER PROTOCOL

8

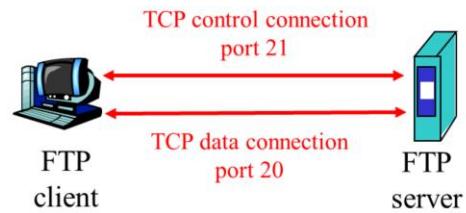
FTP: the file transfer protocol



- transfer file to/from remote host (IETF RFC 959)
- client/server model
 - ❖ *client*: side that initiates file transfer (either to/from remote host)
 - ❖ *server*: remote host
- ftp: runs over TCP (TCP takes care of packet retransmission)
- ftp server ports: 21 and 20
 - ❖ 21 for the control connection
 - ❖ 20 for each data connection

FTP: separate control, data connections

- FTP client contacts FTP server at port 21, TCP is transport protocol
- client authorized over control connection
- client browses remote directory by sending commands over control connection
- when server receives file transfer command, server opens 2nd TCP connection (for file transfer) to client
- after transferring one file, server closes data connection.



- server opens another TCP data connection to transfer another file.
- control connection: "out of band"
- FTP server maintains the "state" of client interactions: current directory, earlier authentication, etc...

FTP commands, responses

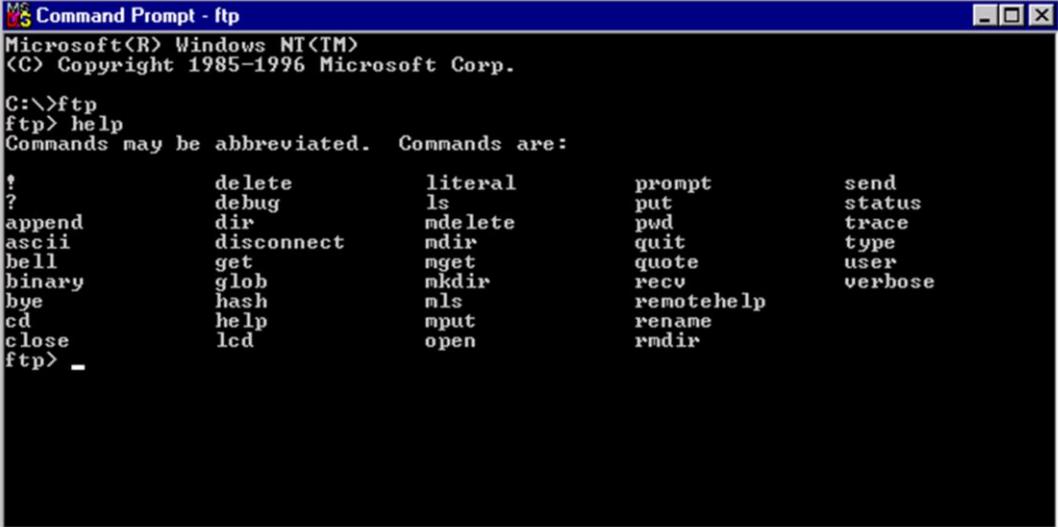
Sample commands:

- sent as ASCII text over control channel:
 - USER *username***
 - PASS *password***
 - LIST** (return list of files in current directory)
 - RETR *filename*** (retrieves file from server)
 - STOR *filename*** (stores file onto server)

Sample return codes

- status code and phrase (as in HTTP):
 - 331 Username OK, password required**
 - 125 Data connection already open; transfer starting**
 - 425 Can't open data connection**
 - 452 Error writing file**

FTP client: user commands

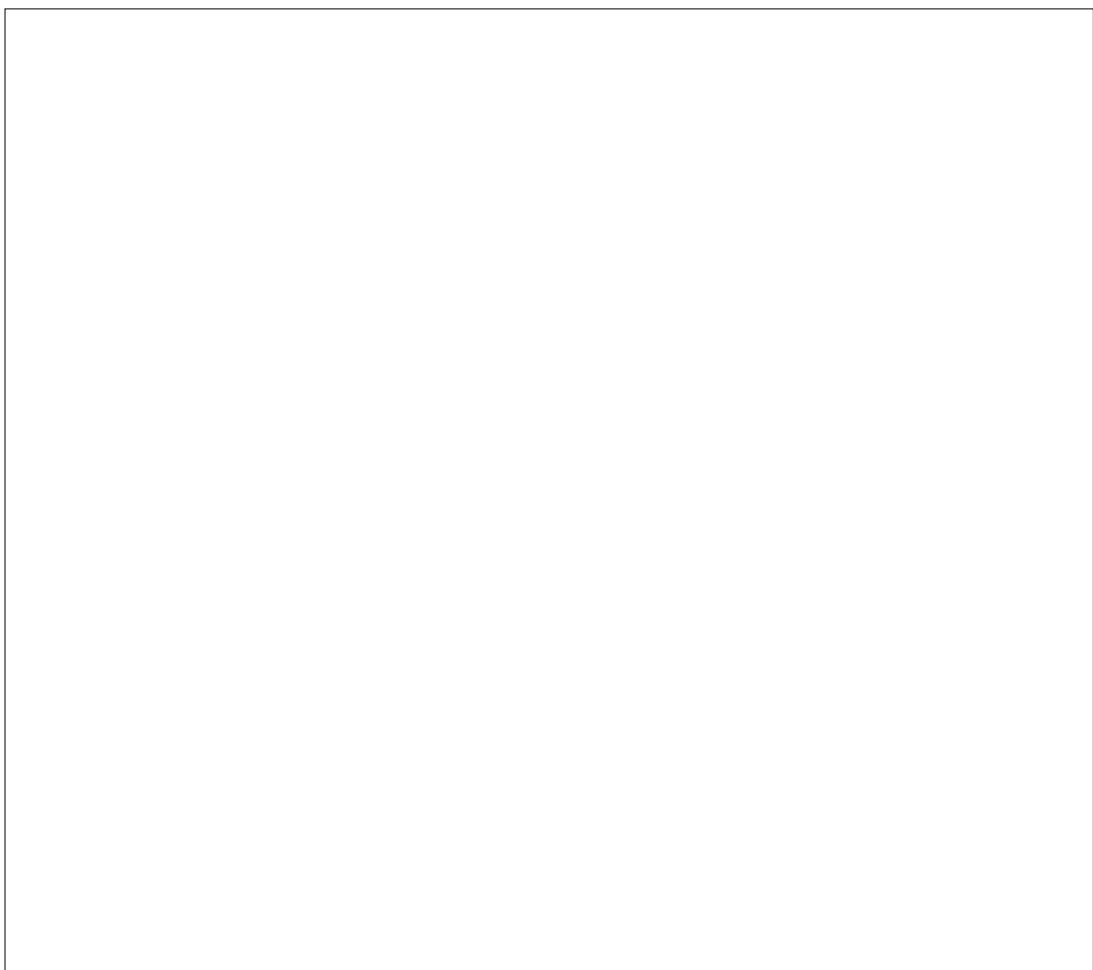


A screenshot of a Windows Command Prompt window titled "Command Prompt - ftp". The window shows the following text:

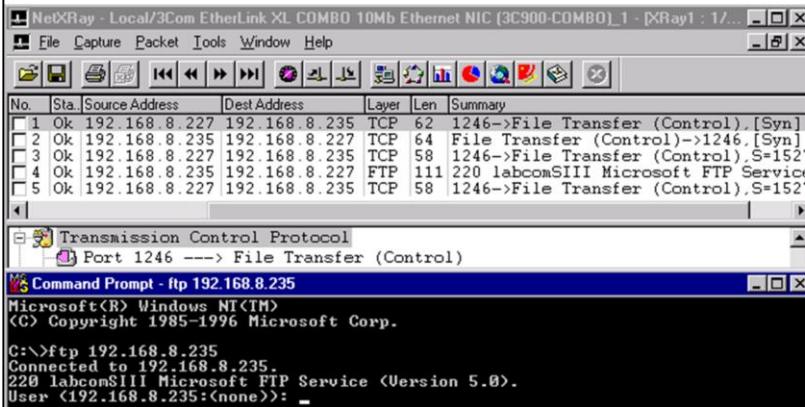
```
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\>ftp
ftp> help
Commands may be abbreviated. Commands are:
!
?           delete      literal      prompt      send
append      debug       ls           put         status
ascii       dir         mdelete    pwd         trace
bell        disconnect  mdir        quit        type
binary      get         mget        quote      user
bye         glob        mkdir      recv        verbose
cd          hash        mls         remotehelp
close      lcd         mput        rename
ftp> -
```

12



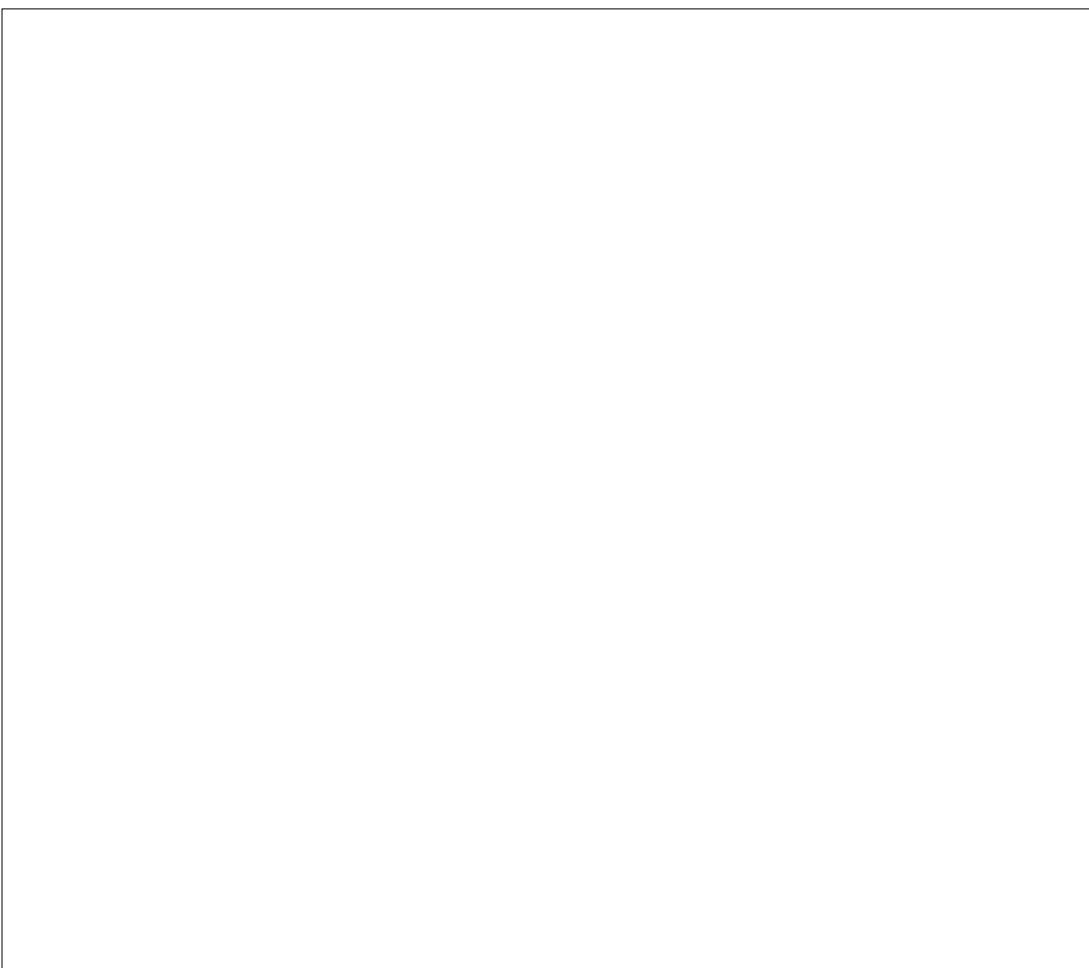
Initial connection to FTP server



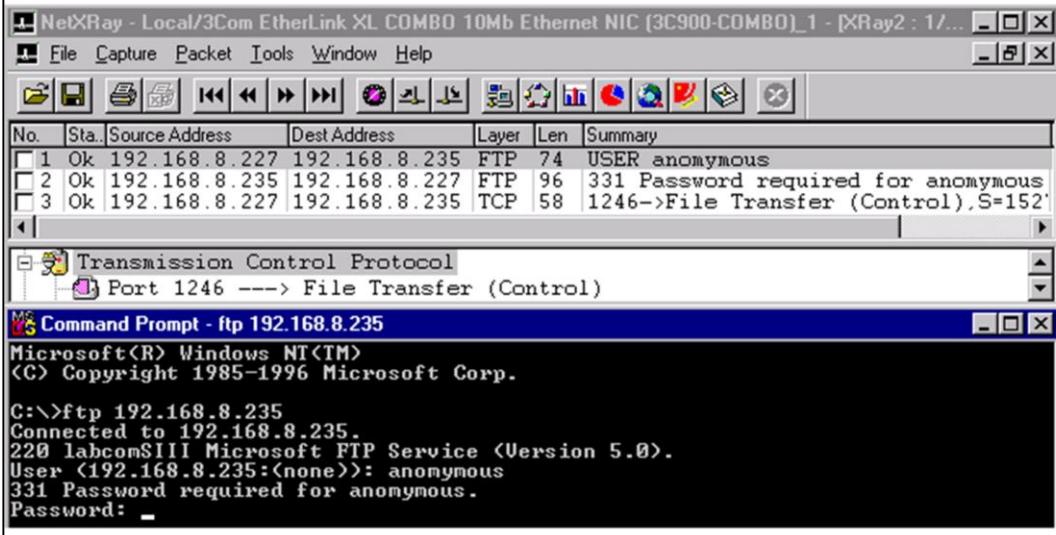
TCP 3-way handshake
to initiate the control
connection in port 21

For each packet sent, a
TCP ACK is needed to
confirm the previous
packet

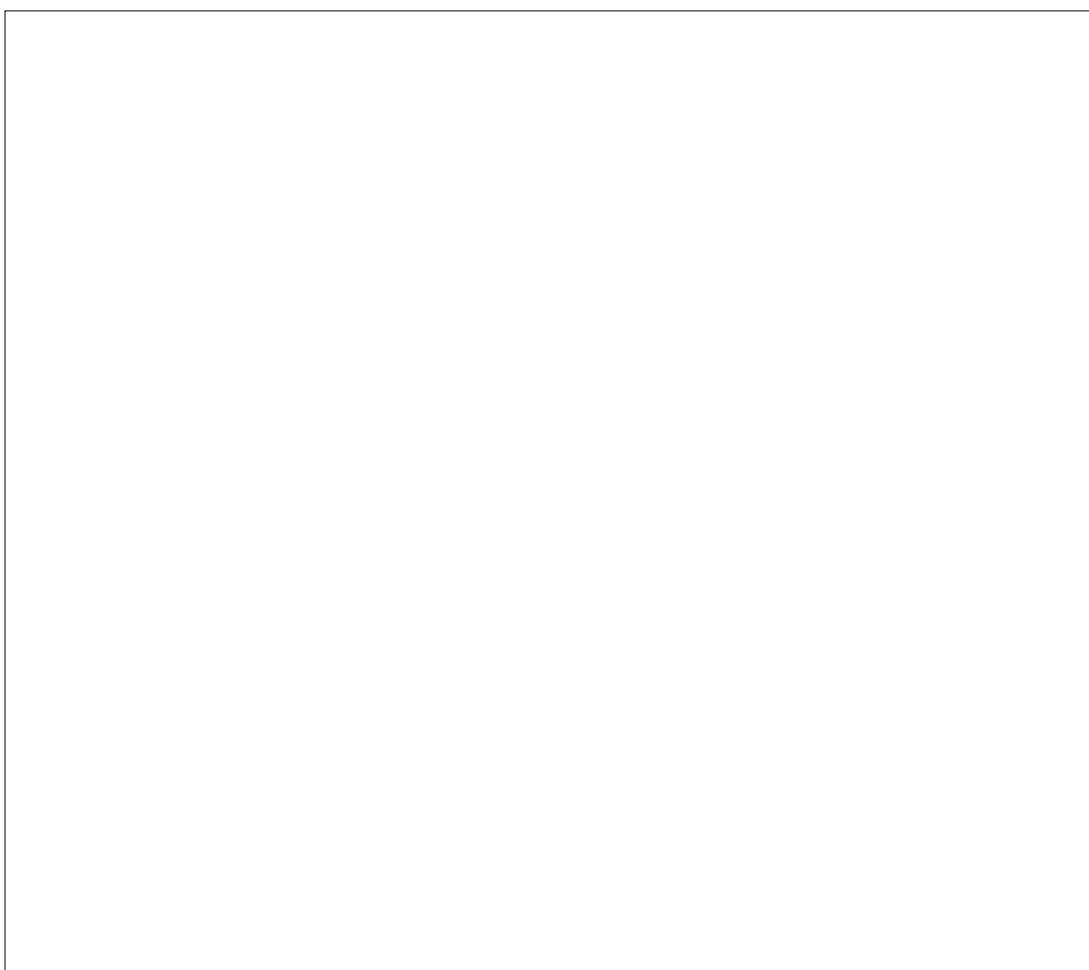
13



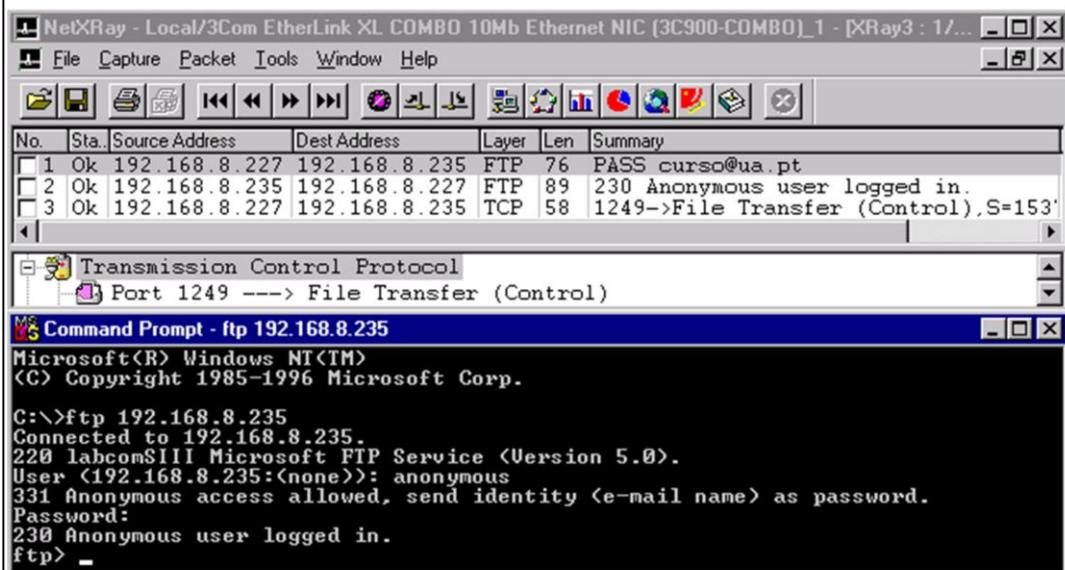
Introduction of username



14

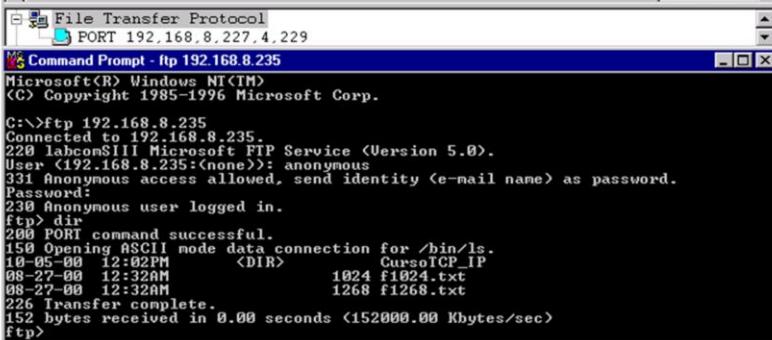


Introduction of password



Introduction of DIR command

| No. | Sta. | Source Address | Dest Address | Layer | Len | Summary |
|-----|------|----------------|---------------|-------|-----|---|
| 1 | Ok | 192.168.8.227 | 192.168.8.235 | FTP | 84 | PORT 192.168.8.227.4.229 |
| 2 | Ok | 192.168.8.235 | 192.168.8.227 | FTP | 88 | 200 PORT command successful. |
| 3 | Ok | 192.168.8.227 | 192.168.8.235 | FTP | 64 | LIST |
| 4 | Ok | 192.168.8.235 | 192.168.8.227 | FTP | 111 | 150 Opening ASCII mode data connection for /bin/ls. |
| 5 | Ok | 192.168.8.235 | 192.168.8.227 | TCP | 66 | File Transfer (Default Data)->1253. |
| 6 | Ok | 192.168.8.227 | 192.168.8.235 | TCP | 62 | 1253->File Transfer (Default Data). |
| 7 | Ok | 192.168.8.235 | 192.168.8.227 | TCP | 64 | File Transfer (Default Data)->1253. |
| 8 | Ok | 192.168.8.235 | 192.168.8.227 | FTP | 210 | Data (total 152 bytes). (More data) |
| 9 | Ok | 192.168.8.235 | 192.168.8.227 | TCP | 64 | File Transfer (Default Data)->1253. |
| 10 | Ok | 192.168.8.227 | 192.168.8.235 | TCP | 58 | 1253->File Transfer (Default Data). |
| 11 | Ok | 192.168.8.227 | 192.168.8.235 | TCP | 58 | 1253->File Transfer (Default Data). |
| 12 | Ok | 192.168.8.235 | 192.168.8.227 | TCP | 64 | File Transfer (Default Data)->1253. |
| 13 | Ok | 192.168.8.227 | 192.168.8.235 | TCP | 58 | 1252->File Transfer (Control), S=154. |
| 14 | Ok | 192.168.8.235 | 192.168.8.227 | FTP | 82 | 226 Transfer complete. |
| 15 | Ok | 192.168.8.227 | 192.168.8.235 | TCP | 58 | 1252->File Transfer (Control), S=154. |



```
File Transfer Protocol
PORT 192.168.8.227.4.229
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\>ftp 192.168.8.235
Connected to 192.168.8.235.
220 labcomSIII Microsoft FTP Service <Version 5.0>.
User <192.168.8.235:<none>>: anonymous
331 Anonymous access allowed, send identity <e-mail name> as password.
Password:
230 Anonymous user logged in.
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
10-05-00 12:02PM <DIR> CursoTCP_IP
08-27-00 12:32AM 1024 f1024.txt
08-27-00 12:32AM 1268 f1268.txt
226 Transfer complete.
152 bytes received in 0.00 seconds (152000.00 Kbytes/sec)
ftp>
```

Information to start the data connection

FTP command for DIR

TCP 3-way handshake to initiate the data connection in port 20

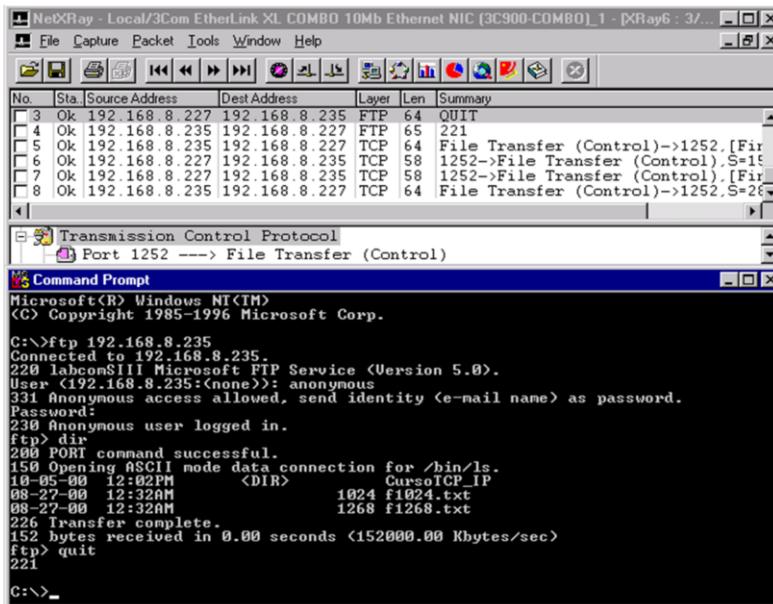
Data is sent

FIN and ACK packets (4 in total) to finish the data connection in port 20

16

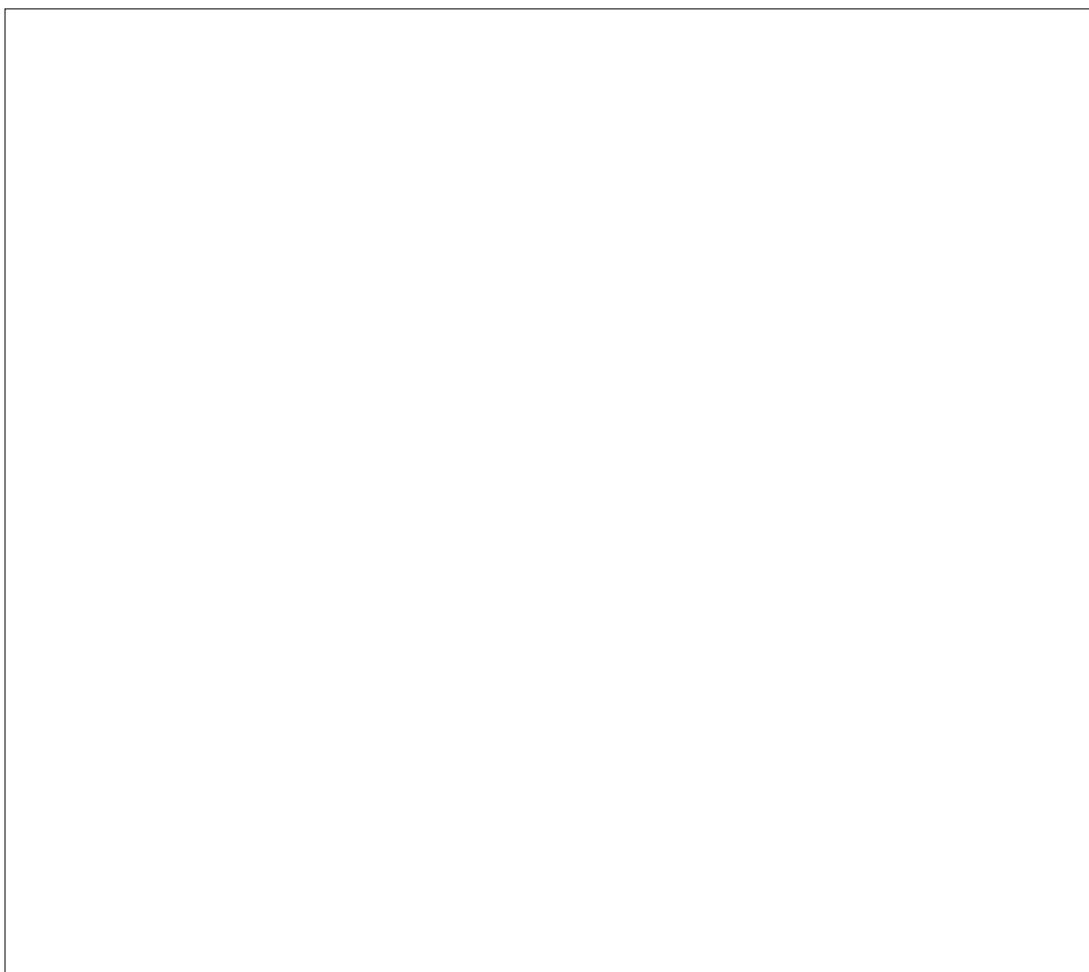
O comando PORT indica o porto usado para a ligação de dados. Cliente (227) indica porto para a ligação de dados. Servidor (235) inicia o estabelecimento da ligação. Depois do comando Data os 4 segmentos TCP fecham a ligação.

Termination of connection (quit)



FIN and ACK packets (4 in total) to finish the control connection in port 21

17



17

DNS – DOMAIN NAME SYSTEM

18

DNS: Domain Name System

People: many identifiers:

- ❖ SSN, name, passport #

Internet hosts, routers:

- ❖ IP address (32 bit) - used for addressing datagrams
- ❖ "name" (for example, www.yahoo.com) - used by humans

DNS: provides a mapping between IP addresses and names

Works in UDP port 53

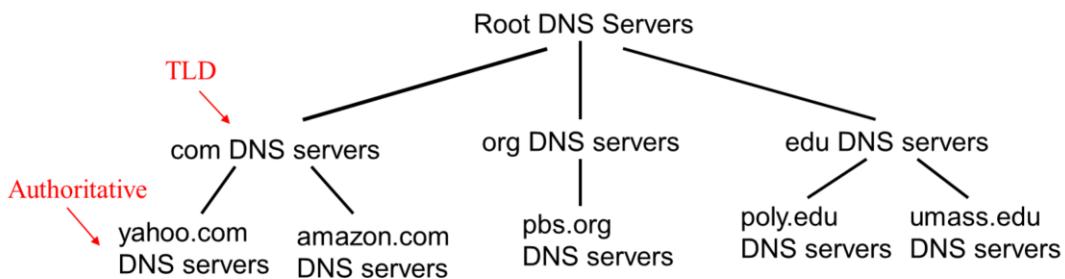
Domain Name System:

- ***distributed database***
implemented in a hierarchy of many **Name Servers**
- ***application-layer protocol***
hosts communicate with Name Servers to **resolve** names (name - IP address translation)
 - ❖ note: core Internet function, implemented as application-layer protocol
 - ❖ complexity at network's "edge"

2: Application Layer 19

Dynamic DNS is a system which allows the domain name data held in a name server to be updated in real time. The most common use for this is in allowing an Internet domain name to be assigned to a computer with a varying (dynamic) IP address. This makes it possible for other sites on the Internet to establish connections to the machine without needing to track the IP address themselves. A common use is for running server software on a computer that has a dynamic IP address, as is the case with many consumer Internet service providers.

Distributed, Hierarchical Database



Client wants IP for www.amazon.com; 1st approximation:

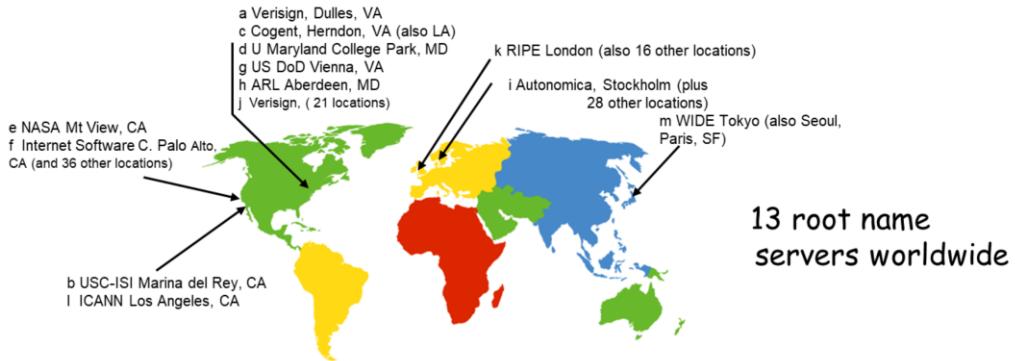
- client queries a root server to find 'com' DNS server
- client queries 'com' DNS server to get 'amazon.com' DNS server
- client queries 'amazon.com' DNS server to get IP address for www.amazon.com

DNS: Root Name Servers

| Server | Operator | Cities | IP Addr | Home AS | Answers ICMP? |
|--------|---|--|---|---------|---------------|
| A | VeriSign Global Registry Services | Herndon VA, US | 198.41.0.4 | 19836 | yes |
| B | Information Sciences Institute | Marina Del Rey CA, US | 128.9.0.107 | tba | yes |
| C | Cogent Communications | Herndon VA, US | 192.33.4.12 | 2149 | yes |
| D | University of Maryland | College Park MD, US | 128.8.10.90 | 27 | yes |
| E | NASA Ames Research Center | Mountain View CA, US | 192.203.230.10 | 297 | yes |
| F | Internet Software Consortium | Palo Alto CA, US; San Francisco CA, US; Madrid, ES; San Jose, CA, US; New York, NY, US; Hong Kong, HK | IPv4: 192.5.5.241 IPv6: 2001:500::1035 | 3557 | yes |
| G | U.S. DOD Network Information Center | Vienna VA, US | 192.112.36.4 | 568 | no |
| H | U.S. Army Research Lab | Aberdeen MD, US | IPv4: 128.63.2.53 IPv6: 2001:500:1::803f:235 | 13 | yes |
| I | Autonomica | Stockholm, SE | 192.36.148.17 | 8674 | yes |
| J | VeriSign Global Registry Services | Herndon VA, US | 192.58.128.30 | 26415 | yes |
| K | Reseaux IP Europeens - Network Coordination Centre | London, UK | 193.0.14.129 | 5459 | yes |
| L | Internet Corporation for Assigned Names and Numbers | Los Angeles CA, US | 198.32.64.12 | 20144 | no |
| M | WIDE Project | Tokyo, JP | 202.12.27.33 | 7500 | yes |

DNS: Root Name Servers

- contacted by local Name Server that cannot resolve name



2: Application Layer 22

Top Level Domains (TLD)

Organizational domains:

- ❖ com - commercial organizations
- ❖ edu - educational institutions
- ❖ gov - govern institutions
- ❖ mil - military institutions
- ❖ net - network operators
- ❖ int - international organizations
- ❖ org - other organizations

Country domains:

- ❖ pt - Portugal
- ❖ es - Spain

Local Name Server

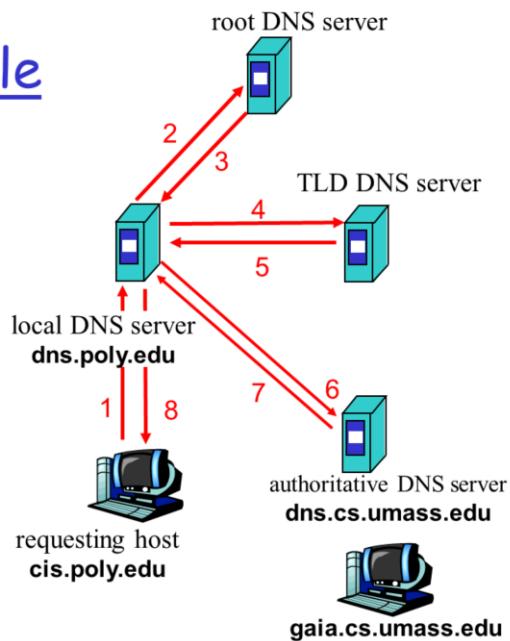
- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one.
 - ❖ also called "default Name Server"
- when host makes DNS query, query is sent to its local DNS server
 - ❖ acts as proxy, forwards query into hierarchy

DNS name resolution example

- Host at cis.poly.edu wants IP address for gaia.cs.umass.edu

iterated query:

- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"

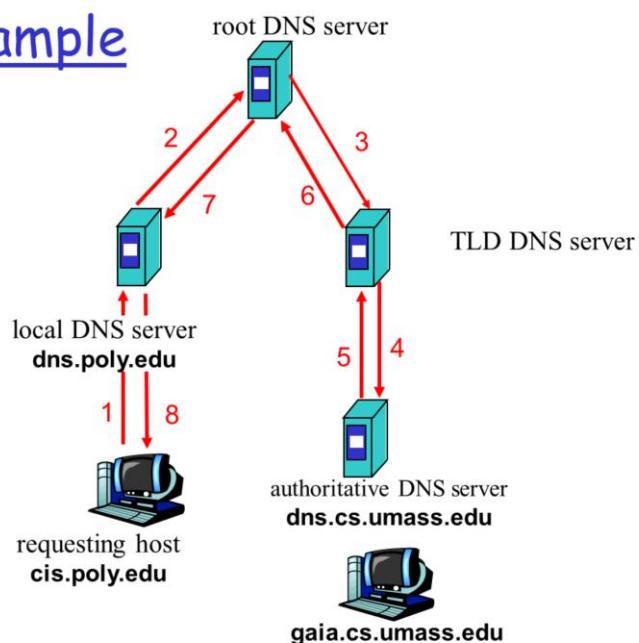


2: Application Layer 25

DNS name resolution example

recursive query:

- ❑ puts burden of name resolution on contacted name server



Recursive vs. iterated resolution

Recursive resolution:

- ❖ More efficient: minimizes the time between the query and the answer
- ❖ Requires more processing power in DNS servers: each server has more simultaneous ongoing requests, on average

Iterated resolution:

- ❖ Less efficient: the time between the query and the answer is larger, on average
- ❖ Minimizes the processing power required on DNS servers: each server replies immediately to each received query

DNS: caching and updating records

- once (any) Name Server learns mapping, it *caches* mapping
 - ❖ cache entries timeout (disappear) after some time
 - ❖ TLD server addresses are typically cached in local Name Servers
 - Thus, Root Name Servers are not often visited

2: Application Layer 28

DNS records

DNS: distributed database storing Resource Records (**RR**)

RR format: `(name, value, type, ttl)`

Type=A

- ❖ **name** is hostname
- ❖ **value** is IP address
- ❖ e.g. (relay1.bar.foo.com, 145.37.93.126, A)

Type=NS

- ❖ **name** is domain (e.g. foo.com)
- ❖ **value** is hostname of authoritative name server for this domain
- ❖ e.g. (foo.com, dns.foo.com, NS)

Type=CNAME

- ❖ **name** is alias name for some "canonical" (the real) name
- ❖ **value** is canonical name
- ❖ e.g. (foo.com, relay1.bar.foo.com, CNAME)

Type=MX

- ❖ **value** is name of mailserver associated with **name**
- ❖ e.g. (foo.com, mail.bar.foo.com, MX)

Type=AAAA

- ❖ **name** is hostname
- ❖ **value** is IPv6 address

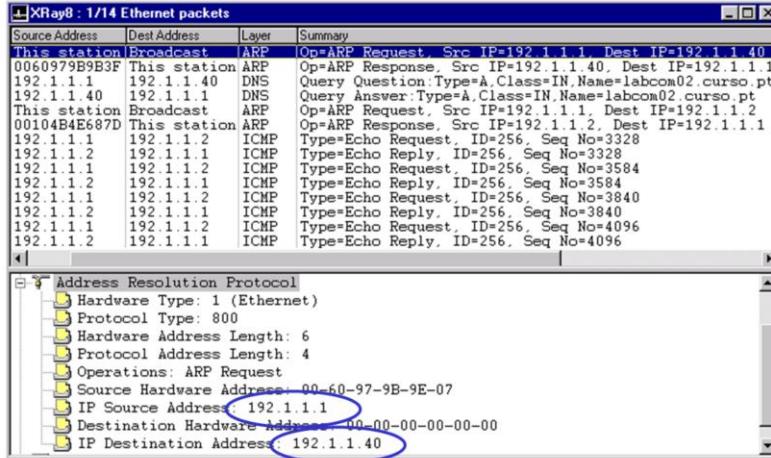
2. Application Layer 29

DNS messages

host: 192.1.1.1

C:\>ping labcom02.curso.pt

DNS: 192.1.1.40



2: Application Layer 30

A captura apresenta uma captura de um pedido bem sucedido de resolução do nome *labcom02.curso.pt* feito pelo terminal com endereço 192.1.1.1 configurado com o endereço de DNS 192.1.1.40. Este pedido de resolução de endereço foi originado pelo comando *ping labcom02.curso.pt*.

Como inicialmente o terminal não sabe o endereço MAC do DNS, começa por emitir um *ARP Request* para o endereço 192.1.1.40 (pacote 1) e recebe de seguida do servidor de DNS o *ARP Response* respectivo.

DNS messages

The screenshot shows a Wireshark capture titled "XRay8 : 3/14 Ethernet packets". The table lists several network frames, including ARP requests and responses, and DNS queries. A specific DNS query from source port 1052 to destination port 53 is highlighted. The detailed view pane shows the DNS message structure:

- User Datagram Protocol:
 - Port 1052 →> Domain Name Server
 - Protocol length: 43 (Octets)
 - Checksum: 0x5446
- Domain Name Service:
 - HEADER SECTION:
 - Identifier: 1
 - Flags: Req.Query.Non-Auth.Recv.Desr.Recv.Not Ava.RCode=No Error
 - Section Entries: QDCount=1 ANCount=0 NSCount=0 ARCount=0
 - QUESTION SECTION[1]: Type=A, Class=IN labcom02.curso.pt

Below the structure, the raw hex and ASCII data of the DNS packet is shown, with the question section circled in blue.

Source port: 1052

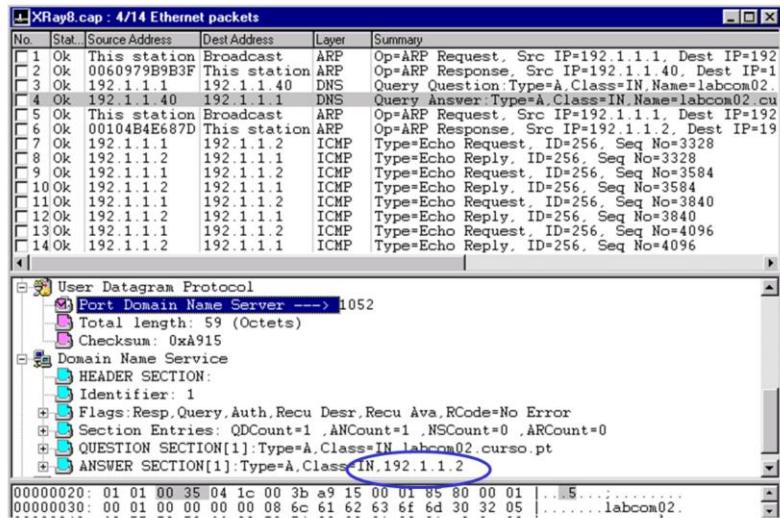
Destination port: 53

2: Application Layer 31

Após a identificação do endereço MAC do servidor DNS, o terminal envia um comando DNS encapsulado num pacote UDP com um número de porto origem escolhido localmente (no exemplo, 1052) e o número de porto destino típico do serviço DNS (porto 53).

Este comando DNS contém apenas um *QUESTION SECTION* em que é enviado o nome que se pretende resolver (conforme assinalado na figura).

DNS messages



2: Application Layer 32

O comando DNS enviado pelo terminal é recebido pelo servidor que responde com outro comando DNS. Este comando é encapsulado num pacote UDP em que os números de porto usados são os mesmos do comando recebido.

A resposta a um pedido de resolução DNS é constituída por dois *Section Entries*: o *QUESTION SECTION* que foi recebido e o *ANSWER SECTION* com o endereço IP associado ao nome recebido.

Após a recepção do endereço IP pretendido, os pacotes seguintes têm um comportamento semelhante ao que se observa se o comando *ping* tivesse sido executado com o endereço IP. Neste caso, é primeiro executado o protocolo ARP para descobrir o endereço MAC do destino e depois são gerados os pacotes ICMP típicos.

HTTP

33

Web and HTTP

First some basic concepts

- Web page consists of objects
- Object can be HTML file, JPEG image, Java applet, audio file,...
- Web page consists of base HTML-file which includes several referenced objects
- Each object is addressable by a URL
- Example URL:

www.someschool.edu/someDept/pic.gif

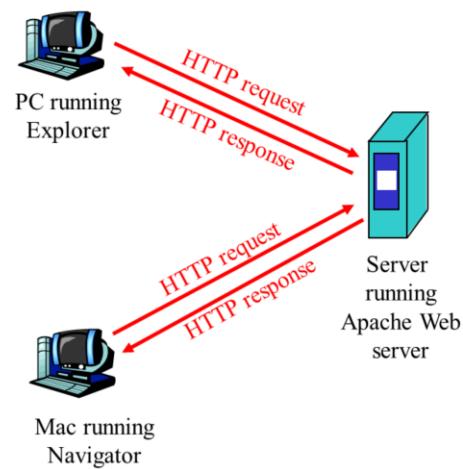
host name

path name

HTTP overview

HTTP: HyperText Transfer Protocol

- Web's application layer protocol
- client/server model
 - ❖ *client*: browser that requests, receives, "displays" Web objects
 - ❖ *server*: Web server sends objects in response to requests
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068



HTTP overview (continued)

Uses TCP:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

HTTP is "stateless"

- server maintains no information about past client requests

Protocols that maintain “state” are complex!

- past history (state) must be maintained
- if server/client crashes, their views of “state” may be inconsistent, must be reconciled

HTTP connections

Nonpersistent HTTP

- At most one object is sent over a TCP connection
- HTTP/1.0 uses nonpersistent HTTP

Persistent HTTP

- Multiple objects can be sent over single TCP connection between client and server
- HTTP/1.1 uses persistent connections in default mode

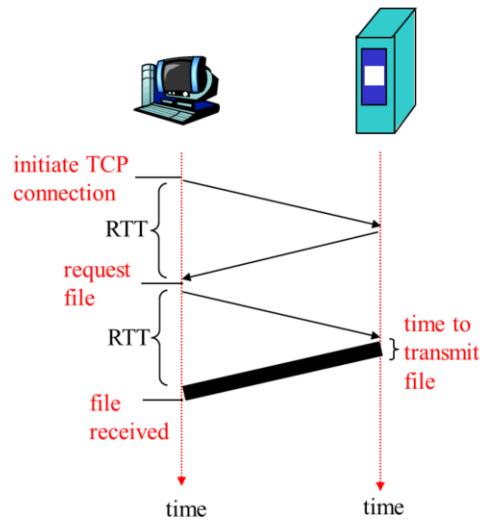
Nonpersistent HTTP: Response time

Round Trip Time (RTT):

time to send a small packet to travel from client to server and back.

Response time:

- one RTT to initiate TCP connection
 - one RTT for HTTP request and first few bytes of HTTP response to return
 - file transmission time
- total = 2RTT+transmit time**



Persistent HTTP

Nonpersistent HTTP issues:

- requires 2 RTTs per object
- OS overhead for each TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

Persistent HTTP

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection

Persistent without pipelining:

- client issues new request only when previous response has been received
- one RTT for each referenced object

Persistent with pipelining:

- default in HTTP/1.1
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

HTTP request message

- two types of HTTP messages: *request, response*

- **HTTP request message:**

- ❖ ASCII (human-readable format)

request line
(GET, POST,
HEAD commands)

header lines

Carriage return,
line feed
indicates end
of message

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close      Close: nonpersistent connection – 1 TCP
Accept-language: fr    connection per object to send
(extra carriage return, line feed)
```

2: Application Layer 40

HTTP response message

status line
(protocol
status code
status phrase)

header
lines

data, e.g.,
requested
HTML file

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 .....
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

2: Application Layer 41

HTTP Request - starts with TCP connection

| No. | Time | Source | Destination | Protocol | Length | Info | |
|-----|----------|----------------|----------------|----------|--------|---|---|
| 2 | 0.000374 | 192.168.50.1 | 192.168.50.100 | TCP | 62 | 58323 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 S | TCP 3-way handshake to initiate the HTTP connect in port 80 |
| 4 | 0.001036 | 192.168.50.100 | 192.168.50.1 | TCP | 62 | 80 → 58323 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 | GET: request of HTTP version 1.1 |
| 5 | 0.001209 | 192.168.50.1 | 192.168.50.100 | TCP | 54 | 58323 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0 | 200 OK: answer with data transmission |
| 6 | 0.001851 | 192.168.50.1 | 192.168.50.100 | HTTP | 615 | GET / HTTP/1.1 | |
| 7 | 0.002524 | 192.168.50.100 | 192.168.50.1 | TCP | 60 | 80 → 58323 [ACK] Seq=1 Ack=562 Win=63954 Len=0 | |
| 8 | 0.003805 | 192.168.50.100 | 192.168.50.1 | HTTP | 473 | HTTP/1.1 200 OK (text/html) | |
| 9 | 0.044556 | 192.168.50.1 | 192.168.50.100 | TCP | 54 | 58323 → 80 [ACK] Seq=562 Ack=420 Win=63821 Len=0 | |

▼ Hypertext Transfer Protocol
 > GET / HTTP/1.1\r\n
 Host: www.arqedes.pt\r\n
 Connection: keep-alive\r\n
 Cache-Control: max-age=0\r\n
 Upgrade-Insecure-Requests: 1\r\n
 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.71 Safari/53
 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed
 Sec-GPC: 1\r\n
 Accept-Encoding: gzip, deflate\r\n
 Accept-Language: en-US,en;q=0.9\r\n
 If-None-Match: "4f-5d5ddd3551e77-gzip"\r\n
 If-Modified-Since: Tue, 18 Jan 2022 16:33:13 GMT\r\n

2: Application Layer 42

HTTP Request - answers with data, ACKed with TCP

| No. | Time | Source | Destination | Protocol | Length | Info | |
|-----|------------|----------------|----------------|----------|--------|---|--|
| - | 2 0.000374 | 192.168.50.1 | 192.168.50.100 | TCP | 62 | 58323 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1468 S | |
| 4 | 0.001036 | 192.168.50.100 | 192.168.50.1 | TCP | 62 | 80 → 58323 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 | |
| 5 | 0.001209 | 192.168.50.1 | 192.168.50.100 | TCP | 54 | 58323 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0 | |
| + | 6 0.001851 | 192.168.50.1 | 192.168.50.100 | HTTP | 615 | GET / HTTP/1.1 | |
| 7 | 0.002524 | 192.168.50.100 | 192.168.50.1 | TCP | 60 | 80 → 58323 [ACK] Seq=1 Ack=562 Win=63954 Len=0 | |
| 8 | 0.003805 | 192.168.50.100 | 192.168.50.1 | HTTP | 473 | HTTP/1.1 200 OK (text/html) | |
| 9 | 0.044556 | 192.168.50.1 | 192.168.50.100 | TCP | 54 | 58323 → 80 [ACK] Seq=562 Ack=420 Win=63821 Len=0 | |

> HTTP/1.1 200 OK\r\nDate: Tue, 18 Jan 2022 17:33:07 GMT\r\nServer: Apache/2.4.38 (Debian)\r\nLast-Modified: Tue, 18 Jan 2022 16:33:13 GMT\r\nETag: "4f-5d5dd3551e77-gzip"\r\nAccept-Ranges: bytes\r\nVary: Accept-Encoding\r\nContent-Encoding: gzip\r\nContent-Length: 84\r\nKeep-Alive: timeout=5, max=100\r\nConnection: Keep-Alive\r\nContent-Type: text/html\r\n

Keep-alive: persistent connection, all objects are sent in the same TCP connection

TCP 3-way handshake to initiate the HTTP connect in port 80
GET: request of HTTP version 1.1
200 OK: answer sending the data

2: Application Layer 43

HTTP: The content of the page

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|----------|----------------|----------------|----------|--------|---|
| 2 | 0.000374 | 192.168.50.1 | 192.168.50.100 | TCP | 62 | 58323 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 S |
| 4 | 0.001036 | 192.168.50.100 | 192.168.50.1 | TCP | 62 | 80 → 58323 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 |
| 5 | 0.001209 | 192.168.50.1 | 192.168.50.100 | TCP | 54 | 58323 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0 |
| 6 | 0.001851 | 192.168.50.1 | 192.168.50.100 | HTTP | 615 | GET / HTTP/1.1 |
| 7 | 0.002524 | 192.168.50.100 | 192.168.50.1 | TCP | 60 | 80 → 58323 [ACK] Seq=1 Ack=562 Win=63954 Len=0 |
| 8 | 0.003805 | 192.168.50.100 | 192.168.50.1 | HTTP | 473 | HTTP/1.1 200 OK (text/html) |
| 9 | 0.044556 | 192.168.50.1 | 192.168.50.100 | TCP | 54 | 58323 → 80 [ACK] Seq=562 Ack=420 Win=63821 Len=0 |

> Frame 8: 473 bytes on wire (3784 bits), 473 bytes captured (3784 bits) on interface \Device\NPF_{0C3DD97D-3D58-4976-978E-B16B8

> Ethernet II, Src: PcsCompu_B8:31:e1 (08:00:27:8b:31:e1), Dst: 0a:00:27:00:00:03 (0a:00:27:00:00:03)

> Internet Protocol Version 4, Src: 192.168.50.100, Dst: 192.168.50.1

> Transmission Control Protocol, Src Port: 80, Dst Port: 58323, Seq: 1, Ack: 562, Len: 419

> Hypertext Transfer Protocol

Line-based text data: text/html (6 lines)

```
arqredes.pt
Porto 80
```

2: Application Layer 44