

# FTP

## FILE TRANSFER PROTOCOL

INTRODUCTION TO FTP, THE INTERNET'S  
STANDARD FILE TRANSFER PROTOCOL

Peter R. Egli  
[peteregli.net](http://peteregli.net)

## Contents

1. FTP versus TFTP
2. FTP principle of operation
3. FTP trace analysis
4. FTP File Transfer Protocol RFC959
5. FTP Active mode versus passive mode
6. FXP File Exchange Protocol
7. FTP clients


## 1. FTP versus HTTP

Before the advent of HTTP, FTP (RFC959) was the prime protocol for file transfer in the Internet. FTP has still some advantages over HTTP.


### • FTP

  FTP session (stateful).

 FTP is comparatively simple.


 FTP is better suited (faster, more efficient) for large files.


 FTP has a control and a data connection and communicates TCP port numbers for data connection in control connection (so-called ,non-well-behaving' protocol).


 With FTP the user ,sees' the directory structure on the server.


### • HTTP

  No session (stateless).

 Web clients and servers became very complex since they need to support many protocols, scripting languages, file types etc. Complexity is also a security problem.

 HTTP is better suited for the transfer of many small files (from HTTP 1.1 on TCP connections are used for many objects).

 HTTP uses a single TCP connection for control and data (better for passing through firewalls).

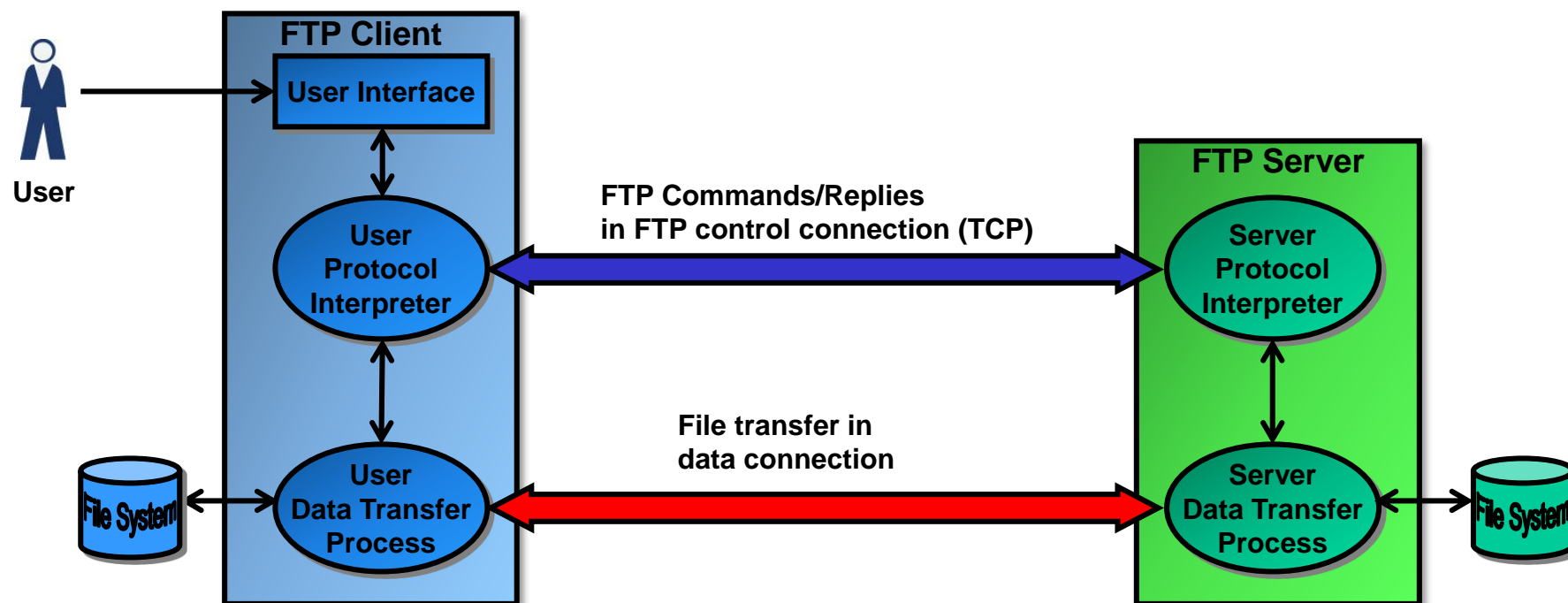
 Through use of Hyperlinks the directory structure on the server may be hidden to the user.

## 2. FTP principle of operation (1/2)

FTP uses a TCP **control** and **data connection**.

The TCP control connection is used for opening / closing an FTP session and for transferring commands from client to server.

The data connection is used for transferring individual files between client and server. Every file transfer uses a separate data connection.



## 2. FTP principle of operation (2/2)

### Steps of a file transfer session:

#### *1. Open control connection*

The Client opens a control connection to the server (TCP).

This control connection (server port 21, the client uses an ephemeral port) is used for FTP commands (C→S) and FTP replies (S→C).

#### *2. File transfer command*

The client initiates a file transfer (C→S: STOR, S→C: RETR).

#### *3. Data connection for transfer*

A new data connection (TCP, usually server port 20, the client uses an ephemeral port) is created for the transfer of the file. In active mode (see below), this connection is opened by the server (for RETR and STOR operations).

#### *4. End of file transfer*

The sender (server for RETR operation, client for STOR operation) closes the data TCP connection when the file is completely transferred.

#### *5. Close control connection*

At the end of the FTP session, the client closes the control connection.

## 3. FTP trace analysis (1/8)

Trace of a typical session using a simple command line front-end (1/3):

*Blue: Control connection*

*Red: Data connection*

Wireshark trace of FTP session:

```
1 0.000000 192.168.1.15 -> 193.5.54.110 TCP 2349 > ftp [SYN] Seq=1573931081 Len=0 MSS=1460
2 0.024683 193.5.54.110 -> 192.168.1.15 TCP ftp > 2349 [SYN, ACK] Seq=434812839 Ack=1573931082 Win=50400 Len=0 MSS=1440
3 0.024823 192.168.1.15 -> 193.5.54.110 TCP 2349 > ftp [ACK] Seq=1573931082 Ack=434812840 Win=65535 Len=0
4 0.205645 193.5.54.110 -> 192.168.1.15 FTP Response: 220 marge FTP server ready.
5 0.391427 192.168.1.15 -> 193.5.54.110 TCP 2349 > ftp [ACK] Seq=1573931082 Ack=434812869 Win=65506 Len=0
6 2.236433 192.168.1.15 -> 193.5.54.110 FTP Request: USER pegli
7 2.262494 193.5.54.110 -> 192.168.1.15 TCP ftp > 2349 [ACK] Seq=434812869 Ack=1573931094 Win=50400 Len=0
8 2.262794 193.5.54.110 -> 192.168.1.15 FTP Response: 331 Password required for pegli.
9 2.469465 192.168.1.15 -> 193.5.54.110 TCP 2349 > ftp [ACK] Seq=1573931094 Ack=434812903 Win=65472 Len=0
10 3.909289 192.168.1.15 -> 193.5.54.110 FTP Request: PASS *****
11 3.934343 193.5.54.110 -> 192.168.1.15 TCP ftp > 2349 [ACK] Seq=434812903 Ack=1573931107 Win=50400 Len=0
12 5.222065 193.5.54.110 -> 192.168.1.15 FTP Response: 230 User pegli logged in.
13 5.422846 192.168.1.15 -> 193.5.54.110 TCP 2349 > ftp [ACK] Seq=1573931107 Ack=434812930 Win=65445 Len=0
14 19.210650 192.168.1.15 -> 193.5.54.110 FTP Request: PORT 192,168,1,15,9,47
15 19.233713 193.5.54.110 -> 192.168.1.15 FTP Response: 200 PORT command successful.
16 19.234858 192.168.1.15 -> 193.5.54.110 FTP Request: NLST
17 19.260158 193.5.54.110 -> 192.168.1.15 TCP ftp-data > 2351 [SYN] Seq=440929149 Len=0 MSS=1440 WS=0
18 19.260380 192.168.1.15 -> 193.5.54.110 TCP 2351 > ftp-data [SYN, ACK] Seq=689664515 Ack=440929150 Win=65535 Len=0 WS=0
19 19.284812 193.5.54.110 -> 192.168.1.15 TCP ftp-data > 2351 [ACK] Seq=440929150 Ack=689664516 Win=50400 Len=0
20 19.285344 193.5.54.110 -> 192.168.1.15 FTP Response: 150 Opening ASCII mode data connection for file list.
21 19.287364 193.5.54.110 -> 192.168.1.15 FTP-DATA FTP Data: 190 bytes
22 19.287384 193.5.54.110 -> 192.168.1.15 TCP ftp-data > 2351 [FIN, ACK] Seq=440929340 Ack=689664516 Win=50400 Len=0
23 19.287504 192.168.1.15 -> 193.5.54.110 TCP 2351 > ftp-data [ACK] Seq=689664516 Ack=440929341 Win=65345 Len=0
24 19.288407 192.168.1.15 -> 193.5.54.110 TCP 2351 > ftp-data [FIN, ACK] Seq=689664516 Ack=440929341 Win=65345 Len=0
25 19.312277 193.5.54.110 -> 192.168.1.15 TCP ftp-data > 2351 [ACK] Seq=440929341 Ack=689664517 Win=50400 Len=0
26 19.422456 192.168.1.15 -> 193.5.54.110 TCP 2349 > ftp [ACK] Seq=1573931137 Ack=434813015 Win=65360 Len=0
27 19.447621 193.5.54.110 -> 192.168.1.15 FTP Response: 226 Transfer complete.
28 19.641203 192.168.1.15 -> 193.5.54.110 TCP 2349 > ftp [ACK] Seq=1573931137 Ack=434813039 Win=65336 Len=0
29 25.987480 192.168.1.15 -> 193.5.54.110 FTP Request: CWD temp
30 26.015339 193.5.54.110 -> 192.168.1.15 FTP Response: 250 CWD command successful.
31 26.203675 192.168.1.15 -> 193.5.54.110 TCP 2349 > ftp [ACK] Seq=1573931147 Ack=434813068 Win=65307 Len=0
```

## 3. FTP trace analysis (2/8)

Trace of a typical session using a simple command line front-end (2/3):

Wireshark trace of FTP session:

```
32 33.740317 192.168.1.15 -> 193.5.54.110 FTP Request: PORT 192,168,1,15,9,48
33 33.764851 193.5.54.110 -> 192.168.1.15 FTP Response: 200 PORT command successful.
34 33.765840 192.168.1.15 -> 193.5.54.110 FTP Request: RETR textfile1.txt
35 33.789123 193.5.54.110 -> 192.168.1.15 TCP ftp-data > 2352 [SYN] Seq=445135172 Len=0 MSS=1440 WS=0
36 33.789770 192.168.1.15 -> 193.5.54.110 TCP 2352 > ftp-data [SYN, ACK] Seq=3866753703 Ack=445135173 Win=65535 Len=0 WS=0
37 33.812075 193.5.54.110 -> 192.168.1.15 TCP ftp-data > 2352 [ACK] Seq=445135173 Ack=3866753704 Win=50400 Len=0
38 33.813205 193.5.54.110 -> 192.168.1.15 FTP Response: 150 Opening ASCII mode data conn. for textfile1.txt (333 bytes).
39 33.815523 193.5.54.110 -> 192.168.1.15 FTP-DATA FTP Data: 340 bytes
40 33.815618 193.5.54.110 -> 192.168.1.15 TCP ftp-data > 2352 [FIN, ACK] Seq=445135513 Ack=3866753704 Win=50400 Len=0
41 33.815735 192.168.1.15 -> 193.5.54.110 TCP 2352 > ftp-data [ACK] Seq=3866753704 Ack=445135514 Win=65195 Len=0
42 33.816254 192.168.1.15 -> 193.5.54.110 TCP 2352 > ftp-data [FIN, ACK] Seq=3866753704 Ack=445135514 Win=65195 Len=0
43 33.969243 192.168.1.15 -> 193.5.54.110 TCP 2349 > ftp [ACK] Seq=1573931191 Ack=434813169 Win=65206 Len=0
44 33.993594 193.5.54.110 -> 192.168.1.15 TCP ftp-data > 2352 [ACK] Seq=445135514 Ack=3866753705 Win=50400 Len=0
45 33.997429 193.5.54.110 -> 192.168.1.15 FTP Response: 226 Transfer complete.
46 34.187974 192.168.1.15 -> 193.5.54.110 TCP 2349 > ftp [ACK] Seq=1573931191 Ack=434813193 Win=65182 Len=0
47 52.491156 192.168.1.15 -> 193.5.54.110 FTP Request: PORT 192,168,1,15,9,49
48 52.515619 193.5.54.110 -> 192.168.1.15 FTP Response: 200 PORT command successful.
49 52.517063 192.168.1.15 -> 193.5.54.110 FTP Request: STOR textfile2.txt
50 52.552801 193.5.54.110 -> 192.168.1.15 TCP ftp-data > 2353 [SYN] Seq=450659857 Len=0 MSS=1440 WS=0
51 52.553014 192.168.1.15 -> 193.5.54.110 TCP 2353 > ftp-data [SYN, ACK] Seq=3891456526 Ack=450659858 Win=65535 Len=0 WS=0
52 52.579850 193.5.54.110 -> 192.168.1.15 TCP ftp-data > 2353 [ACK] Seq=450659858 Ack=3891456527 Win=50400 Len=0
53 52.581755 193.5.54.110 -> 192.168.1.15 FTP Response: 150 Opening ASCII mode data connection for textfile2.txt.
54 52.619464 192.168.1.15 -> 193.5.54.110 FTP-DATA FTP Data: 340 bytes
55 52.619635 192.168.1.15 -> 193.5.54.110 TCP 2353 > ftp-data [FIN, ACK] Seq=3891456867 Ack=450659858 Win=65535 Len=0
56 52.643223 193.5.54.110 -> 192.168.1.15 TCP ftp-data > 2353 [ACK] Seq=450659858 Ack=3891456867 Win=50400 Len=0
57 52.656957 193.5.54.110 -> 192.168.1.15 TCP ftp-data > 2353 [ACK] Seq=450659858 Ack=3891456868 Win=50400 Len=0
58 52.657494 193.5.54.110 -> 192.168.1.15 TCP ftp-data > 2353 [FIN, ACK] Seq=450659858 Ack=3891456868 Win=50400 Len=0
59 52.657638 192.168.1.15 -> 193.5.54.110 TCP 2353 > ftp-data [ACK] Seq=3891456868 Ack=450659859 Win=65535 Len=0
60 52.781676 192.168.1.15 -> 193.5.54.110 TCP 2349 > ftp [ACK] Seq=1573931235 Ack=434813282 Win=65093 Len=0
61 52.801452 193.5.54.110 -> 192.168.1.15 FTP Response: 226 Transfer complete.
62 53.000363 192.168.1.15 -> 193.5.54.110 TCP 2349 > ftp [ACK] Seq=1573931235 Ack=434813306 Win=65069 Len=0
```

## 3. FTP trace analysis (3/8)

### Trace of a typical session using a simple command line front-end (3/3):

Wireshark trace of FTP session:

```
63 60.670981 192.168.1.15 -> 193.5.54.110 FTP Request: TYPE I
64 60.691198 193.5.54.110 -> 192.168.1.15 FTP Response: 200 Type set to I.
65 60.875378 192.168.1.15 -> 193.5.54.110 TCP 2349 > ftp [ACK] Seq=1573931243 Ack=434813326 Win=65049 Len=0
66 71.614301 192.168.1.15 -> 193.5.54.110 FTP Request: PORT 192,168,1,15,9,50
67 71.642352 193.5.54.110 -> 192.168.1.15 FTP Response: 200 PORT command successful.
68 71.643428 192.168.1.15 -> 193.5.54.110 FTP Request: STOR tux.gif
69 71.686138 193.5.54.110 -> 192.168.1.15 TCP ftp-data > 2354 [SYN] Seq=456418333 Len=0 MSS=1440 WS=0
70 71.686773 192.168.1.15 -> 193.5.54.110 TCP 2354 > ftp-data [SYN, ACK] Seq=527148705 Ack=456418334 Win=65535 Len=0 WS=0
71 71.715006 193.5.54.110 -> 192.168.1.15 TCP ftp-data > 2354 [ACK] Seq=456418334 Ack=527148706 Win=50400 Len=0
72 71.716245 193.5.54.110 -> 192.168.1.15 FTP Response: 150 Opening BINARY mode data connection for tux.gif.
73 71.718485 192.168.1.15 -> 193.5.54.110 FTP-DATA FTP Data: 1440 bytes
74 71.718581 192.168.1.15 -> 193.5.54.110 FTP-DATA FTP Data: 118 bytes
75 71.718744 192.168.1.15 -> 193.5.54.110 TCP 2354 > ftp-data [FIN, ACK] Seq=527150264 Ack=456418334 Win=65535 Len=0
76 71.752674 193.5.54.110 -> 192.168.1.15 TCP ftp-data > 2354 [ACK] Seq=456418334 Ack=527150146 Win=48960 Len=0
77 71.836515 193.5.54.110 -> 192.168.1.15 TCP ftp-data > 2354 [ACK] Seq=456418334 Ack=527150264 Win=50400 Len=0
78 71.839919 193.5.54.110 -> 192.168.1.15 TCP ftp-data > 2354 [ACK] Seq=456418334 Ack=527150265 Win=50400 Len=0
79 71.840134 193.5.54.110 -> 192.168.1.15 TCP ftp-data > 2354 [FIN, ACK] Seq=456418334 Ack=527150265 Win=50400 Len=0
80 71.840260 192.168.1.15 -> 193.5.54.110 TCP 2354 > ftp-data [ACK] Seq=527150265 Ack=456418335 Win=65535 Len=0
81 71.922082 192.168.1.15 -> 193.5.54.110 TCP 2349 > ftp [ACK] Seq=1573931281 Ack=434813410 Win=64965 Len=0
82 71.944708 193.5.54.110 -> 192.168.1.15 FTP Response: 226 Transfer complete.
83 72.140912 192.168.1.15 -> 193.5.54.110 TCP 2349 > ftp [ACK] Seq=1573931281 Ack=434813434 Win=64941 Len=0
84 75.049330 192.168.1.15 -> 193.5.54.110 FTP Request: QUIT
85 75.070911 193.5.54.110 -> 192.168.1.15 FTP Response: 221-You have transferred 2238 bytes in 3 files.
86 75.076639 193.5.54.110 -> 192.168.1.15 FTP Response: 221-Total traffic for this session was 3244 bytes in 4 transfers.
87 75.076787 192.168.1.15 -> 193.5.54.110 TCP 2349 > ftp [ACK] Seq=1573931287 Ack=434813616 Win=64760 Len=0
88 75.080600 192.168.1.15 -> 193.5.54.110 TCP 2349 > ftp [FIN, ACK] Seq=1573931287 Ack=434813616 Win=64760 Len=0
89 75.132379 193.5.54.110 -> 192.168.1.15 TCP ftp > 2349 [ACK] Seq=434813616 Ack=1573931288 Win=50400 Len=0
```

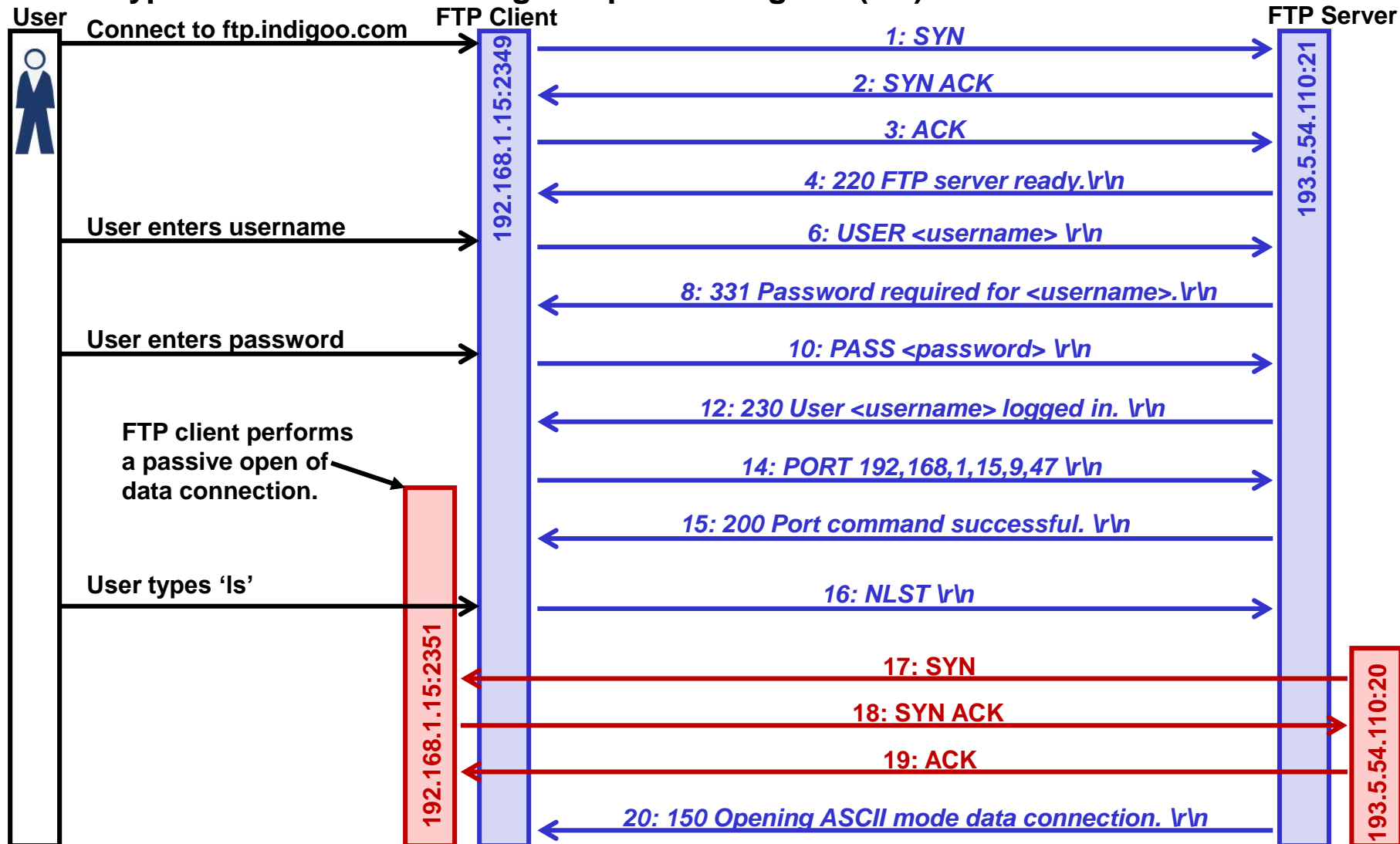


## 3. FTP trace analysis (4/8)

Trace of a typical session as message sequence diagram (1/5):

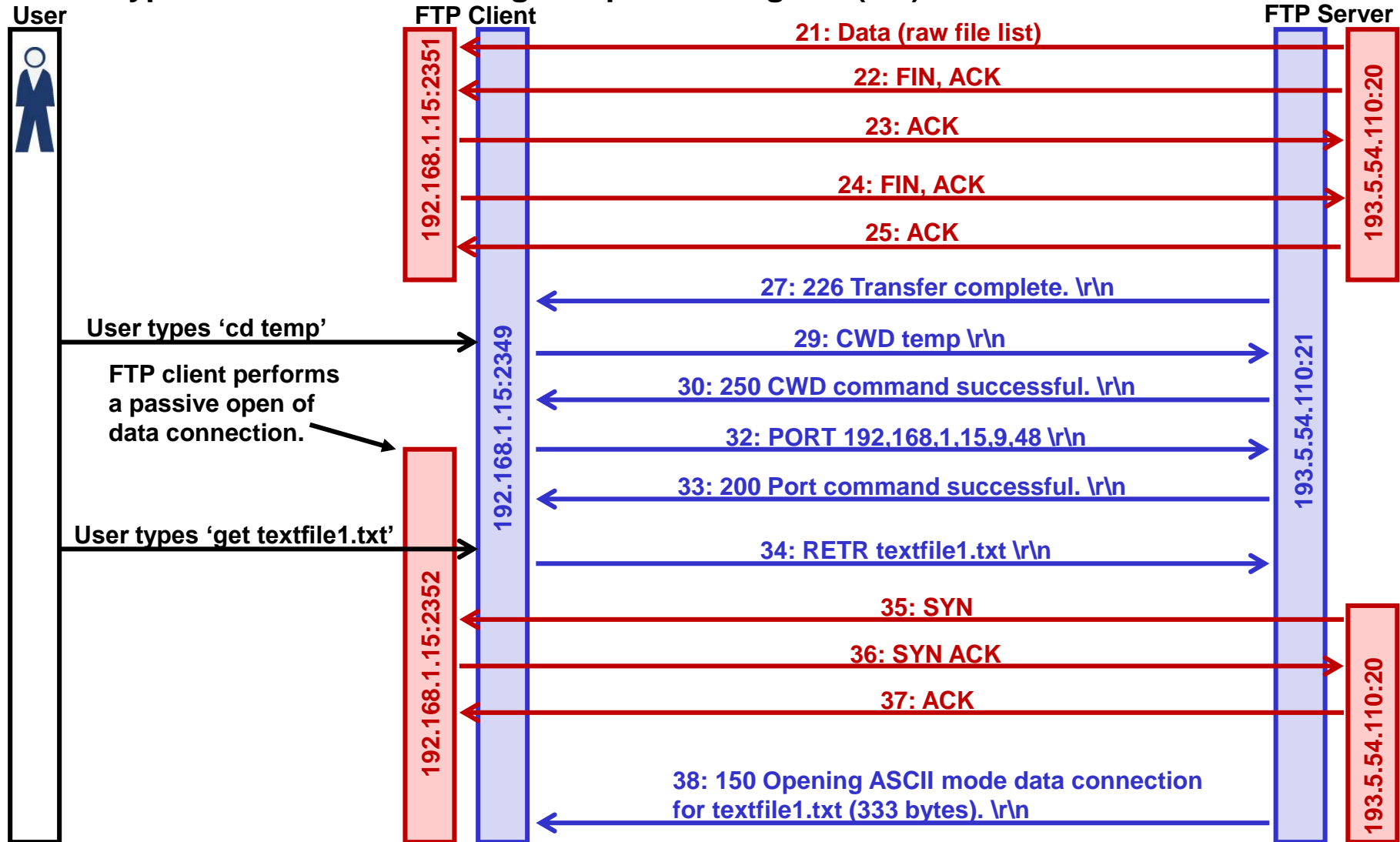
Blue: Control connection

Red: Data connection



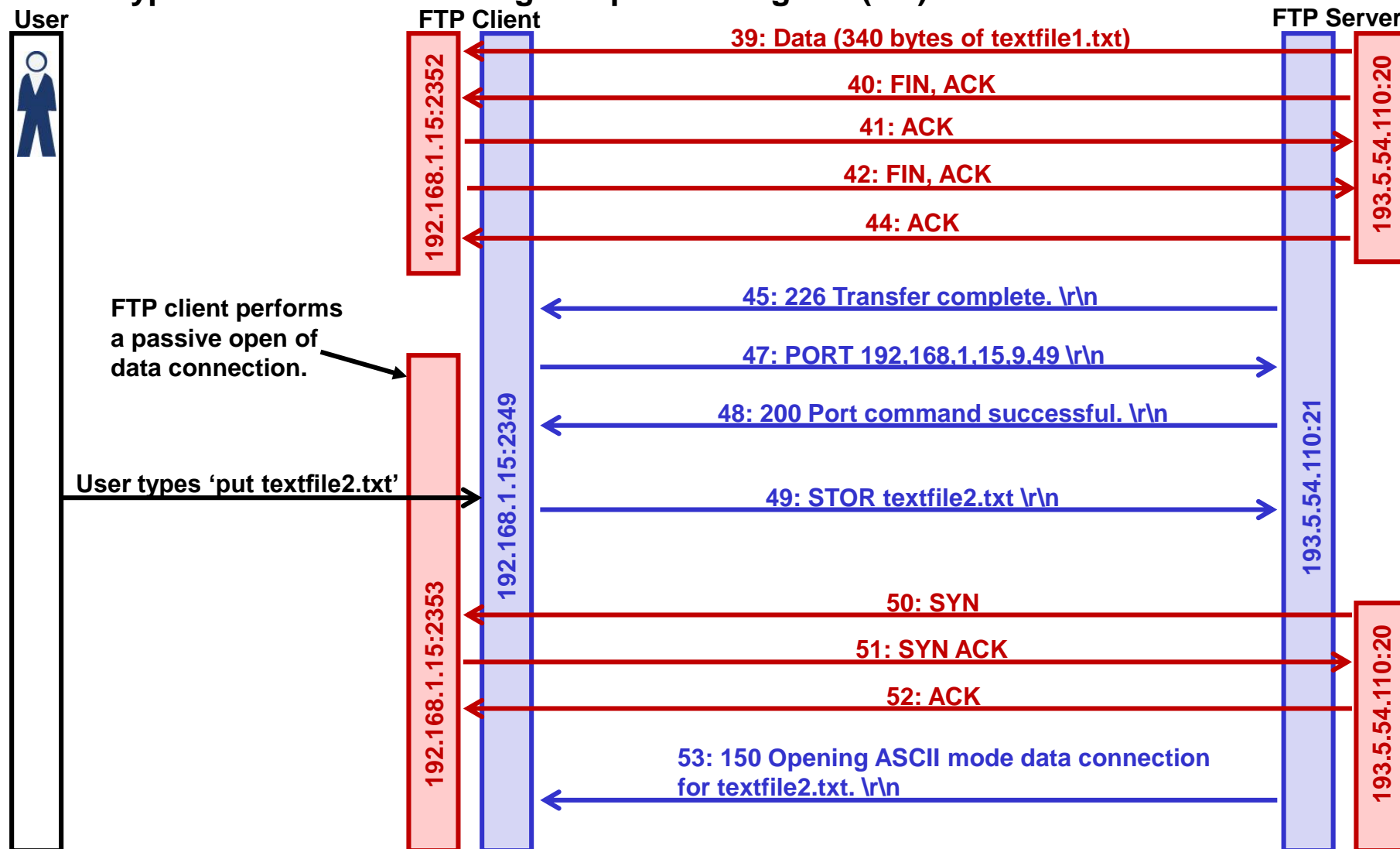
## 3. FTP trace analysis (5/8)

Trace of a typical session as message sequence diagram (2/5):



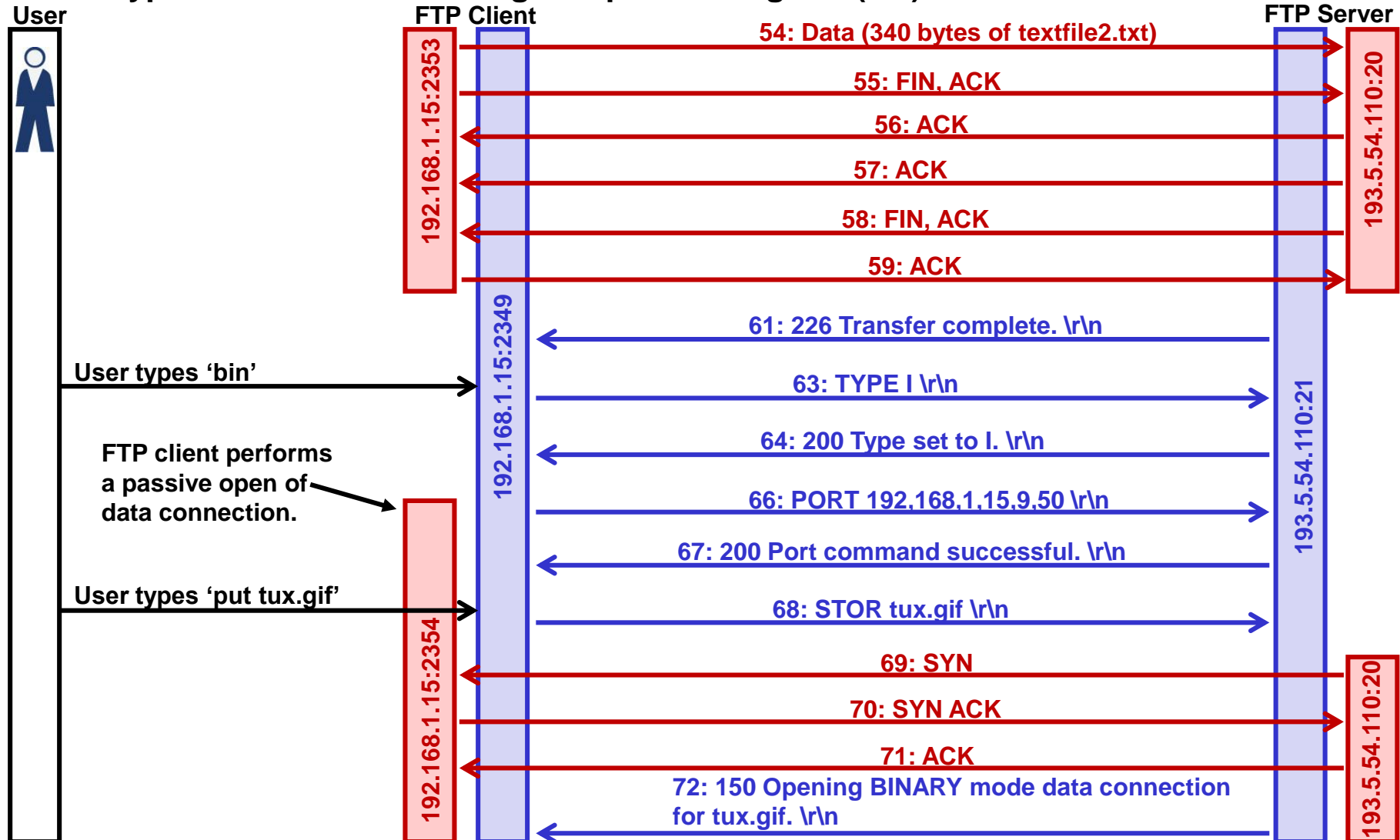
## 3. FTP trace analysis (6/8)

Trace of a typical session as message sequence diagram (3/5):



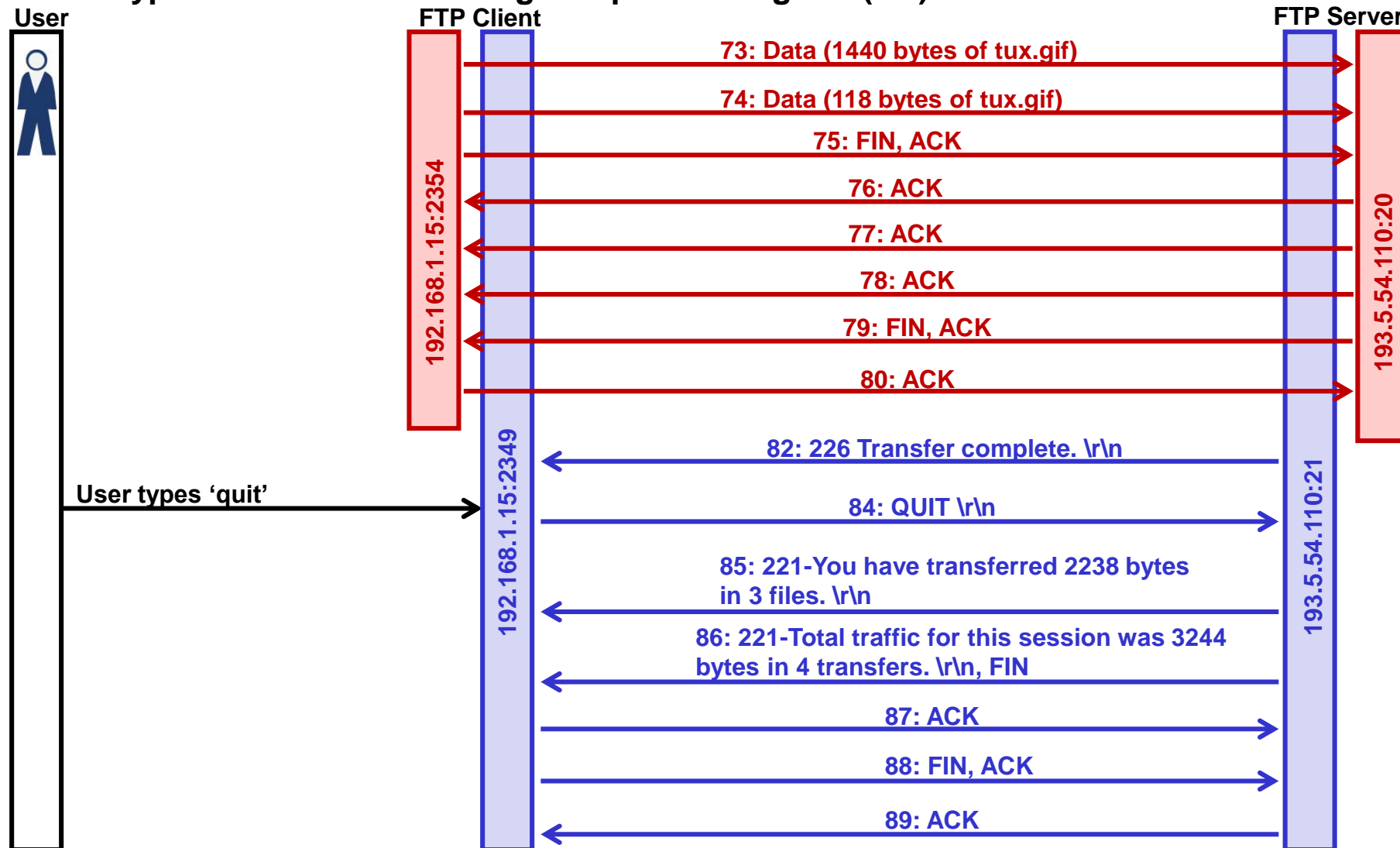
## 3. FTP trace analysis (7/8)

Trace of a typical session as message sequence diagram (4/5):



## 3. FTP trace analysis (8/8)

Trace of a typical session as message sequence diagram (5/5):



## 4. FTP File Transfer Protocol RFC959 (1/3)

### The protocol in a nutshell:

1. FTP is a simple request / reply protocol.

Requests are 4 characters followed by an (optional) argument (e.g. „STOR“, „RETR“).

Replies are 3 character codes followed by an (optional) human readable text (e.g. „200 ...“):

**Client Request:**       “PORT 192,168,1,15,5,91 \r\n”

**Server Reply:**         “200 Port command successful. \r\n”

2. FTP uses the NVT (Network Virtual Terminal) protocol, i.e. ASCII requests and replies terminated by a CRLF (Carriage Return Line Feed) combination (= „\r\n“).

3. The PORT command tells the server the IP address and port number of the client's TCP connection for the data transfer.

With this command, the client tells the server that it is listening for the data connection on the given port. The server opens the data connection both for STOR and RETR operations.

**PORT n1,n2,n3,n4,n5,n6 \r\n**

**n1,n2,n3,n4 is the client's IP address for the TCP data connection (comma decimal notation).**

**n5 is ASCII decimal of the 8 most significant bits of the port number of the TCP data connection.**

**n6 is ASCII decimal of the 8 least significant bits of the port number of the TCP data connection.**

**Example Request:**     “PORT 192,168,1,15,5,91 \r\n”

**Listening IP address = 192.168.1.15, listening TCP port  $5 \cdot 256 + 91 (=1371)$ .**

## 4. FTP File Transfer Protocol RFC959 (2/3)

### File types (ASCII, binary):

In the early days of data networks, bandwidth was precious thus a character set with only 7 bits was defined (ASCII).

Today bandwidth is usually high enough so saving 1/8 of bandwidth through using only 7 bits is not justified anymore.

FTP supports different file types, the most important being ASCII (text) and binary. Some clients and servers make a difference between ASCII and binary, some don't. E.g. UNIX clients strip everything past non-ASCII characters. The windows client transmits all characters anyway.

It's always save to switch to binary mode since then files are transmitted (and stored) ,as is' (unchanged).

## 4. FTP File Transfer Protocol RFC959 (3/3)

### Control connection commands and responses:

The control connection is used to send ASCII commands and response codes between FTP client and server.

### Important control commands:

*USER <username> <CRLF>  
Open FTP session for user <username> (PASS command to follow as next command).*

*PASS <password> <CRLF>  
Enter password (USER <username> entered previously).*

*NLST <CRLF>  
Raw list files or directories (no additional information).*

*LIST <CRLF>  
List of files and directories with additional human readable information.*

*PORT <n1,n2,n3,n4,n5,n6> <CRLF>  
Client IP address (n1.n2.n3.n4) and port (n5\*256+n6) for data connection.*

*RETR <filename> <CRLF>  
Retrieve (get) a file.*

*STOR <filename> <CRLF>  
Store (put) a file.*

*QUIT <CRLF>  
Quit current FTP session.*

### The response codes are grouped into ranges (like SMTP, HTTP, SIP etc.):

100 Series: The requested action is being initiated, expect another reply before proceeding with a new command.

200 Series: The requested action has been successfully completed.

E.g. "200 Command okay".

300 Series: The command has been accepted, but the requested action is dormant, pending receipt of further information.

400 Series: The command was not accepted and the requested action did not take place, but the error condition is temporary and the action may be requested again.

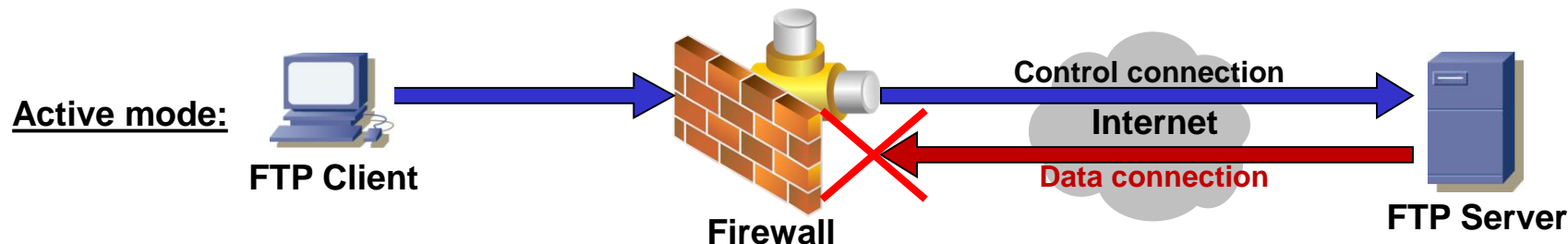
500 Series: The command was not accepted and the requested action did not take place.

E.g. 500 Syntax error, command unrecognized. This may include errors such as command line too long.

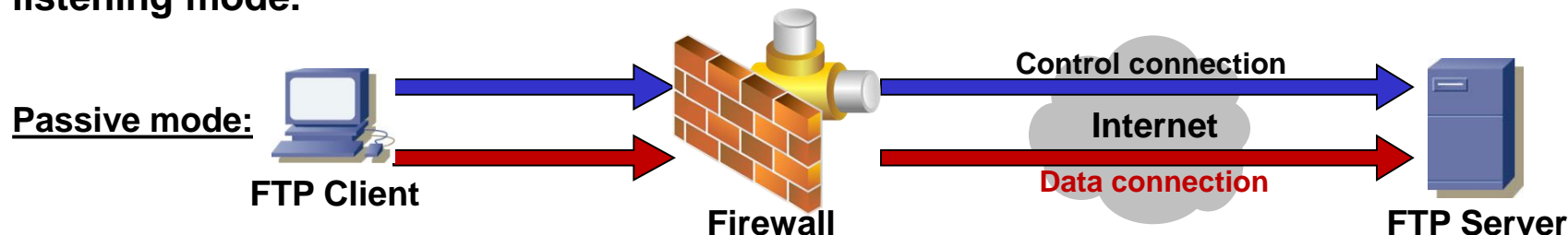


## 5. FTP Active mode versus passive mode (1/3)

**Problem:** Many clients are behind firewalls which do not allow incoming TCP connections (incoming SYN). This blocks data transfer since with active FTP the server opens the TCP data connection (incoming TCP connection for the client for both STOR and RETR commands).



**Solution:** FTP passive mode. The client opens the TCP data connection, thus there are only outgoing TCP connections (control and data). The server is passive, i.e. opens a TCP socket in listening mode.



Passive mode is initiated by the client with the PASV command. The server responds with its IP address and TCP port number for the TCP data connection.

Client Request:  
Server Reply:

“PASV \r\n”  
“227 Entering Passive Mode (n1,n2,n3,n4,n5,n6) \r\n”


## 5. FTP Active mode versus passive mode (2/3)

Sample FTP session with Telnet, active mode (1/2):

FTP uses NVT characters for the control connection. TELNET uses NVT too so a simple TELNET connection can be used to mimick an FTP control connection.

The data connection cannot be established with TELNET since it opens an outgoing connection. For the incoming data connection on the client, an additional tool is needed (e.g. netcat) that allows starting (listening) incoming connections on the client:

```
cmd>telnet ftp.indigoo.com 21
S: 220 marge FTP server ready.
C: USER pegli
S: 331 Password required for pegli.
C: PASS *****
S: 230 User pegli logged in.
C: LIST
S: 425 Can't build data connection: Connection refused.c
C: PORT 193,5,54,26,26,133
S: 200 PORT command successful.
C: CWD temp
S: 250 command successful.
C: RETR textfile1.txt
S: 226 Transfer complete.
C: QUIT
S: 221-You have transferred 0 bytes in 0 files.
S: 221-Total traffic for this session was 681 bytes in 1 transfers.
S: 221-Thank you for using the FTP service on marge.
S: 221 Goodbye.
```



Client data connection TCP port is  $26*256+133=6789$ . Start netcat (listening on TCP port 6789) and redirect received data to ,outputfile`:  
nc -l -p 6789 > outputfile

## 5. FTP Active mode versus passive mode (3/3)

Sample FTP session with Telnet, passive mode (2/2):

In passive mode the client opens the data connection to the server (server is listening). Thus a second TELNET connection can be used for the data transfer from client to server:

```
cmd>telnet ftp.indigoo.com 21
S: 220 marge FTP server ready.
C: USER pegli
S: 331 Password required for pegli.
C: PASS *****
S: 230 User pegli logged in.
C: PASV
S: 227 Entering Passive Mode (193,5,54,110,32,193)
C: CWD temp
S: 250 CWD command successful.
C: RETR textfile1.txt
S: 150 Opening ASCII mode data connection for textfile1.txt (1772 bytes).
S: 226 Transfer complete.
```

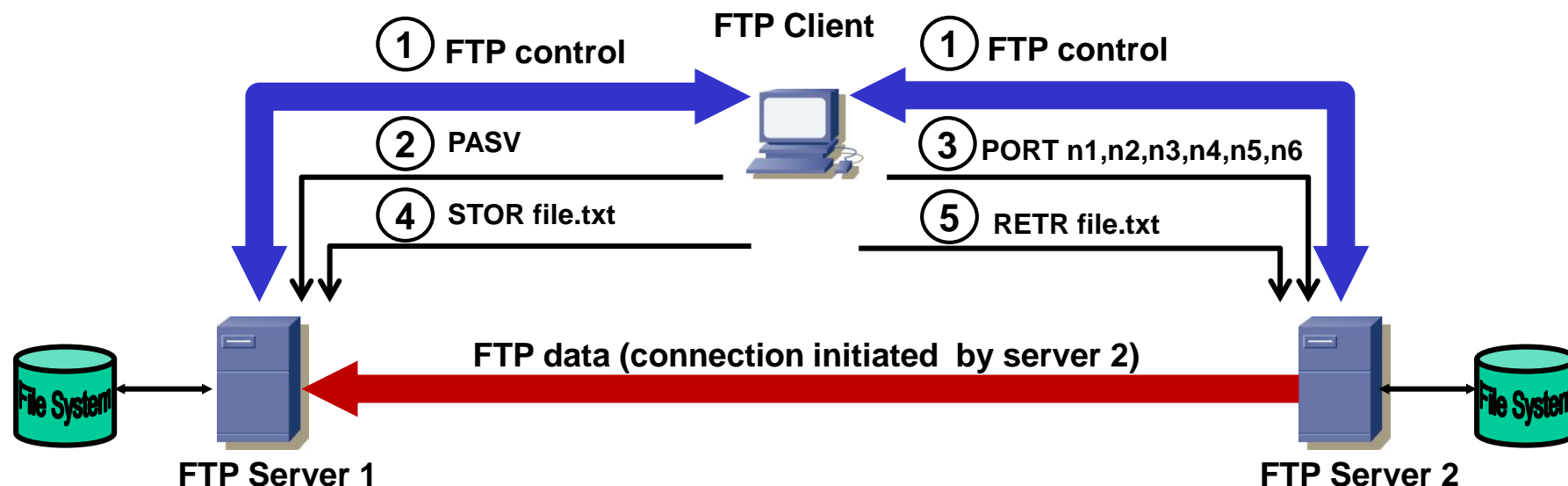
Start second TELNET session  
to ftp.indigoo.com port  
 $32*256+193=8385$

**N.B.:** The second TELNET session will be closed after the transfer of the file is complete.

## 6. FXP File Exchange Protocol (1/2)

FXP uses FTP transfers to directly transfer files between 2 FTP servers. The transfer is controlled by an FTP client.

**Advantage:** The transfer speed depends solely on the connection between the servers which is typically higher than between client and server (a low speed connection is sufficient for the client).



## 6. FXP File Exchange Protocol (2/2)

1. The client opens the control connections to FTP server 1 and 2.
2. The client sends the PASV command to server 1. Server 1 responds with its IP address and listening data connection port number (e.g. 172,16,64,15,4,137).  
Server 1 opens a passive (listening) data connection on the specified IP address and port number.
3. The client sends the PORT command to server 2 with IP address and port number returned by server 1.
4. The client sends the STOR <filename> (upload „client→server“) command to server 1.
5. The client sends the RETR <filename> (download „server→client“) command to server 2. This triggers server 2 to open the data connection to the IP/port specified by the previous PORT command (server 1).  
Since server 1 is in passive mode, it accepts the connection request, receives the file transferred through the data connection and stores it locally.

**N.B.:** This will not work on all servers. Often the servers perform a check if the data connection terminates on the same host as the client connection (security).

## 7. FTP clients

Web browser can be used as FTP front-end (Web browser runs ftp scheme / protocol):

FTP URL: ftp://user:password@host:port/path

Most web browsers are full-fledged FTP clients that allow to get and put files from/to the server with the FTP protocol.

---

FTP download managers:

More sophisticated FTP clients (GUI) are able to „resume“ an FTP transfer in case of transfer failure. This is particularly helpful for large files (Murphy's law dictates that the transfer fails at >95%). These download managers make use of the FTP Restart (REST) command.