File Transfer Protocol - FTP

TCP/IP class

outline

- intro
    - kinds of remote file access mechanisms
- ftp architecture/protocol
- traditional BSD ftp client
- ftp protocol command interface
- ftp trace (high-level)
- higher-level services on ftp + summary

intro

- divide world into **file transfer utilities**
  - ftp - put/get single files, multiple files with mget but not file tree (except via 3rd party archive utility like tar), password required
  - tftp - trivial file xfer, no password
  - rcp - BSD utility like UNIX cp, can do recursive tree copy (-r) , weak security (.rhost)
- and **distributed file systems**
  - NFS - UDP based, “local” file systems
  - Unix V.3 RFS, Andrew, non-TCP/IP Novell

FTP/telnet vs BSD apps

	IETF	**BSD**
file xfer	ftp/tftp	rcp/rsh/rshd
virtual terminal	telnet	rlogin

intro

- RFC 959, more in RFC 1123
- ftp well-established file xfer mechanism
- ftp/telnet classic IETF apps
- ftp servers offer up files with a certain amount of ad hoc per-site organization, basically used for file xfer when you already know what it is you are after, not browsing so much

intro - what ftp can do

- you can xfer single files, ASCII/binary
- you can’t xfer a file tree
- you can do multiple files in the same directory at once though (mget/mput)
- xfer file directory workaround as follows:

intro - directory xfer workaround

- to compress a file tree:
  - tar -cvf foo.tar foo.dir
  - compress foo.tar -> foo.tar.Z
- to fetch and unwrap
  - (ftp) get foo.tar
  - cd <desired location>
  - uncompress foo.tar.Z
  - tar -xvf foo.tar

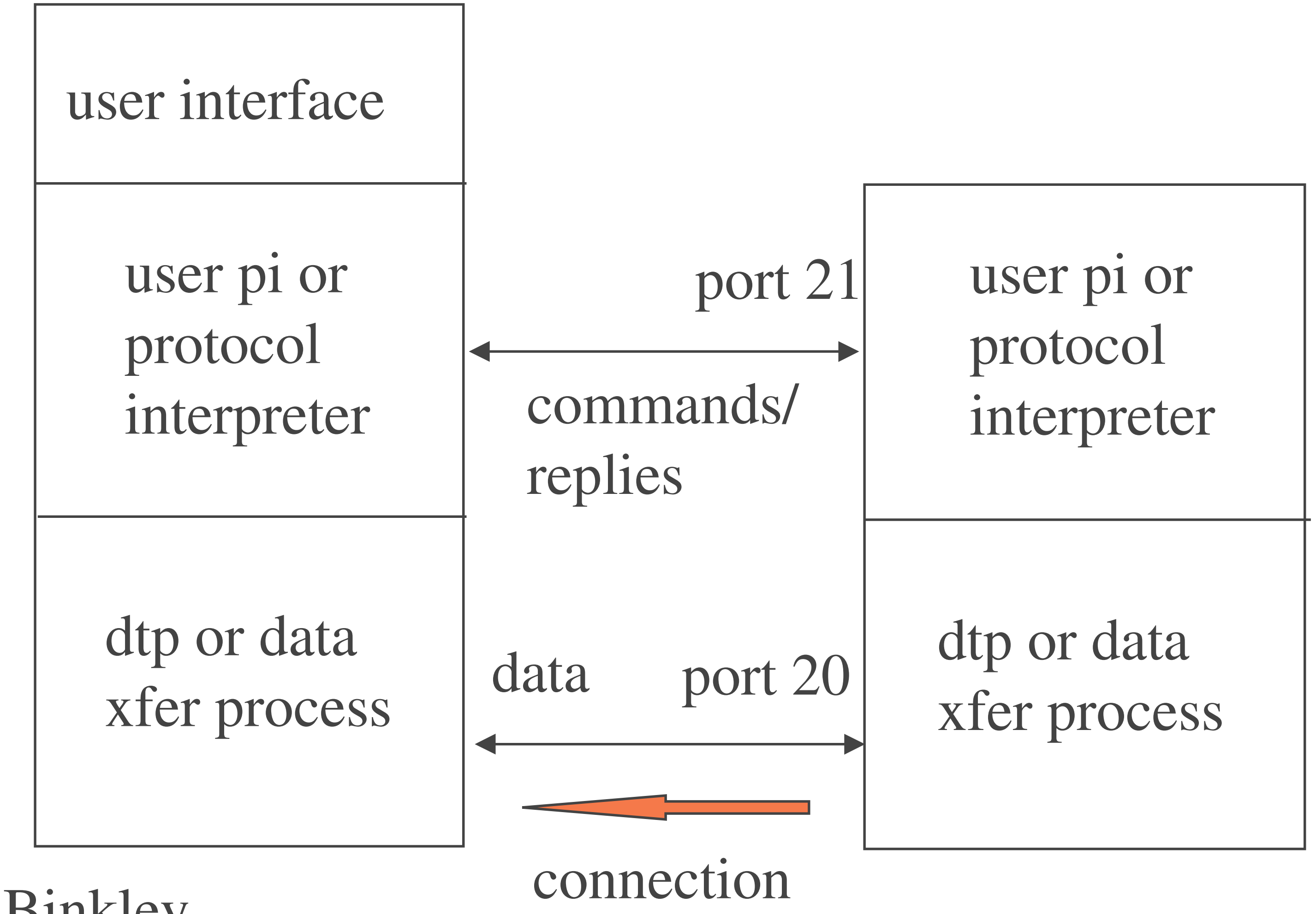
intro - anonymous ftp

- anonymous ftp servers offers up files on server with no need to for password for user convenience, server security isn't impaired
- unix anon-ftp server runs via **chroot(2)** call to /usr/ftp (or wherever), appears to client (and server) as root of file system
- anon login:
  user: ftp (or anonymous)
  password: username@dns.site

ftp architecture/protocol

- ftp uses ASCII commands for ftp protocol on TCP port 21
- protocol commands are simple verb object <cr> <nl>
  RETR *filename* <cr> <nl> (get a file)
- ASCII success/error status comes back from server; e.g.,
  226 transfer complete
- separate socket channel used for data xfer
- after client RETR, **server connects** from port 20 to client port sent via client PORT command OR
- PASV command can be used to tell server to wait for client connection

ftp architecture picture



user interface

user pi or
protocol
interpreter

port 21

commands/
replies

user pi or
protocol
interpreter

dtp or data
xfer process

data

port 20

dtp or data
xfer process

connection

ftp protocol versus ftp client

- understand that an ftp client uses the ftp protocol to talk to a server
- the client “get file” command is translated somehow into the ftp protocol command; e.g.
- BSD ftp client has command: **get** *file*
- ftp protocol uses **RETR** *file* to implement “get”

some ftp lingo

- ASCII - ftp uses ASCII char. set for commands borrowed from TELNET definitions
- control connection - to server port 21
- data connection - from server port 20
- EOL - cr/lf. telnet eof.
- mode: data has modes, **stream**/block/compressed
- NVT - network virtual terminal, telnet abstraction
- reply - ftp command ack, number followed by human readable message

more ftp lingo

- data structure: **file**, record, page, file means byte stream
- type - data is typed, ASCII, EBCDIC, IMAGE. image means binary
- 3-party xfer. possible for one client to talk to two ftp servers and tell them to xfer a file

file typing

- boo, hiss
- BSD client defaults to ASCII
- (ftp) *binary*
- if ASCII mode, both client server must translate lines into telnet eol (cr/nl)
- and translate back acc. to native host ASCII

file typing

- con: xfer binary file from UNIX to DOS in (default?) ASCII mode
  - file maybe (likely) corrupted
  - assume original has <nl>, unix adds <cr><nl>, dos leaves it alone, file now larger...
  - slower anyway because we have to scan per char
- pro: xfer ASCII file from DOS to UNIX
  - 1. DOS converts from <cr,nl> to <cr,nl>
  - 2. UNIX converts from <cr,nl> to <nl>
    you don't need to do anything else...

“classic” BSD client

- syntax:

  % ftp *host* | *ip address*

  % ftp
- connection
  - (ftp) open *host* | *ip*
  - (ftp) user *name [password]*
  - (ftp) password (prompted for after user)
  - (ftp) close
- help
  - (ftp) help [command]

ftp commands, cont.

- file xfer
  - (ftp) ascii - set ASCII xfer type
  - (ftp) binary - set IMAGE xfer type
  - (ftp) hash - print hash mark during file xfer
  - (ftp) get remote-file [local-file]
  - (ftp) put local-file [remote-file]
  - (ftp) mget remote-files
    - (ftp) prompt - toggle, on by default, you want it off
  - (ftp) mput local-file

file commands, cont

- directory ops
  - (ftp) cwd <pathname> - change on server
  - (ftp) lcwd <pathname> - change on client
  - (ftp) pwd - print cwd on server
  - (ftp) !pwd - print on client
  - (ftp) dir - same as
  - (ftp) ls
    - can do ls -lR as unix server-side hack

ftp commands, cont

- misc.
  - (ftp) !sh - escape to UNIX command shell
  - (ftp) get file
  - (ftp) !vi file
  - (ftp) delete file
  - (ftp) mdelete files
  - (ftp) rmdir dir
  - (ftp) mkdir dir

unix ftp replacement - ncftp

- many features over stock BSD ftp
- default is to do anon. login, you don't have to do it
- (ncftp) get foo*bar - works with wildcards
- IMAGE type is default
- shows you how much of file is xferred
- mget works automatically - no prompting

ftp protocol commands

- sent to server on well-known port 21
- typically gets FTP response, success/error
  - 200 “fine by me...”
- connection start/shutdown
  - USER <sp> <username> <crlf>
  - PASS <sp> <password> <crlf>
  - QUIT <crlf>
  - SYST <crlf> - find out server os type

ftp protocol commands

- file xfer
  - RETR <sp> <pathname> <crlf> (get file)
  - STOR <sp> <pathname> <crlf>
  - PORT <sp> <host,port> <crlf>
    - » 6 bytes in decimal, h1,h2,h3,h4,p1,p2
  - PASV <crlf> (tells server to go to passive mode for data xfer)
  - ABOR <crlf> - abort file xfer

ftp protocol commands

- directory ops
  - CWD <sp> <pathname> <crlf> (on server)
  - PWD <crlf>
  - LIST [<sp> <pathname>] <crlf>
  - NLST [<sp> <pathname>] <crlf>
  - RMD <sp> <pathname> <crlf> (rmdir)
  - MKD <sp> <pathname> <crlf> (mkdir)

LIST or NLST for dir list?

- according to RFC, LIST is for humans, NLST for machines (e.g., for mget)
- LIST may have non-interoperable file list (e..g, UNIX ls -l)
- NLST should just be the filenames with nothing else
- use NLST for mget, where you want to get list of filenames to start with

LIST/NLIST cont.

- BSD mget works as follows
  (ftp) mget *.foo
  PORT command sent
  NLST *.foo
  then will get a.foo, b.foo... z.foo
- SunOs ftp and ncftp
  - (ftp) ls -> NLST (like ls in format)
  - (ftp) dir -> LIST (which is ls -l in format)
- 4.4 BSD ftp has ls and dir as LIST

replies (and errors)

- every command must have at least 1 reply
- reply can be multi-line (e.g., SYST, STAT, etc. are most common)
- format: 3 digit number<sp>text
- multiline: replace <sp> with -, last line has space
- theory: 3 digit number for “machine”, text for people
- digit 1: good, bad, incomplete
- digit 2: function groupings (syntax/info/connection/authentication/file system)
- digit 3: particular meaning

reply codes - 1st digit

- 1yz - positive preliminary reply
- 2yz - positive completion reply
- 3yz - positive intermediate reply
- 4yz - transient negative completion reply (try again later)
- 5yz - permanent negative completion reply

ftp replies- examples

- 200 okay
- 226 Transfer complete
- 550 foobar: No such file or directory
- 150 Opening ASCII mode data connection for /bin/ls
- 331 Guest login ok, send “guest” as password
- 220-howdy, howdy
- 220 howdy...

ftp protocol trace

- use ftp client and turn **debug** switch on
- result is that ftp commands are shown
- note PORT gives 6 decimal bytes, ip address in dotted decimal + client port # h1,h2,h3,h4,p1,p2
- e.g., 127,0,0,1,4,7, ip addr == 127.0.0.1
- client port == (4*256)+7 = 1031

protocol trace 1 of 3

```
% ftp nic.ddn.mil
220-****Welcome to the Network Information Center
...
220 and more
Name: anonymous
331 Guest login ok, send “guest” as password
Password:
230 Guest login ok, access restrictions apply
(ftp) debug
(ftp) ls
---> PORT 131,252,20,183,8,107
200 Port command successful
---> NLST
```

protocol trace 2 of 3

```
150 Opening ASCII mode data connection for file list
lost+found
netinfo
bin
ietf
...
rfc
...
226 Transfer Complete
170 bytes received in 0.047 seconds (3.5 Kbytes/s)
(ftp) cd rfc
---> CWD rfc
250 CWD command successful
```

protocol trace 3 of 3

```
(ftp) binary
---> TYPE I
(ftp) get rfc959.txt
---> PORT 131,252,20,183,8,109
200 PORT command successful.
---> RETR rfc959.txt
150 Opening BINARY mode data connection
        for rfc959.txt (157316 bytes)
226 Transfer complete.
local: rfc959.txt remote: rfc959.txt
147316 bytes received in 27 seconds (5.3 Kbytes/s)
(ftp) quit
---> QUIT  (followed by server's 221 Goodbyte)
```

services built on ftp

- archie - find src code by filename
  - ftp servers register to be walked
  - archie uses “ls -lR” listing to index filenames
  - email/web access
- http - single shot ftp access
  - ftp is connected though and you can have problems with anon server limits
- “alex” - distributed anonymous ftp files

  /alex/edu/pdx/cs/ftp/pub/blackadder

ftp summary

- great majority of use is anonymous ftp
- simple ASCII commands, similar mechanism used by http/smtp/nntp too. avoids byte-swapping problems
- ftp is still workhouse of Inet for mass file xfer
- available in web browsers, con is per connection file xfer or dir listing
  - web browser not good way to xfer a billion files
- lack of support for recursive dir. xfer is con
- another con: passwords in the clear