Virtual Terminal Apps telnet/rlogin

TCP/IP class

Jim Binkley

outline

- intro
- telnet
  - architecture
  - nvt
  - option negotiation
- rlogin
- what else?

intro - network terminals

- **telnet** - TCP/IP classic application, RFC 854, 1983
- **rlogin** - BSD UNIX derived application, similar to telnet, RFC 1282, 1991. RFC done after rlogin
- **X11** too - servers run on hw terminal, clients may be here or on server, use TCP to communicate over net
- telnet/rlogin - ASCII based 80 column by 24 rows, run UNIX command shell OR what?
- both telnet/rlogin use TCP for client/server, point to point virtual circuit connection

intro - what are they for?

- % **telnet** *ip-addr* | *dns-name [port]*
- you can aim it at an arbitrary port and
  - 1. talk to a telnetd (default port 23) and get a UNIX login shell or who knows what at other ports
  - 2. talk to a mud server
  - 3. talk to a database system (library)
  - 4. do debug-oriented things with sendmail, news, http servers, that speak ASCII
- you can’t aim rlogin at a port, only a host running rlogind, won’t take ip address either

intro - summary

- telnet is a network debug tool - more versatile than rlogin
- rlogin may be “convenient”, especially if .rhost “weak authentication” used
  - assumption was unix to unix
  - but that’s dubious in and of itself
  - my opinion is turn it off... replace with ssh/slogin/sshd with RSA authentication

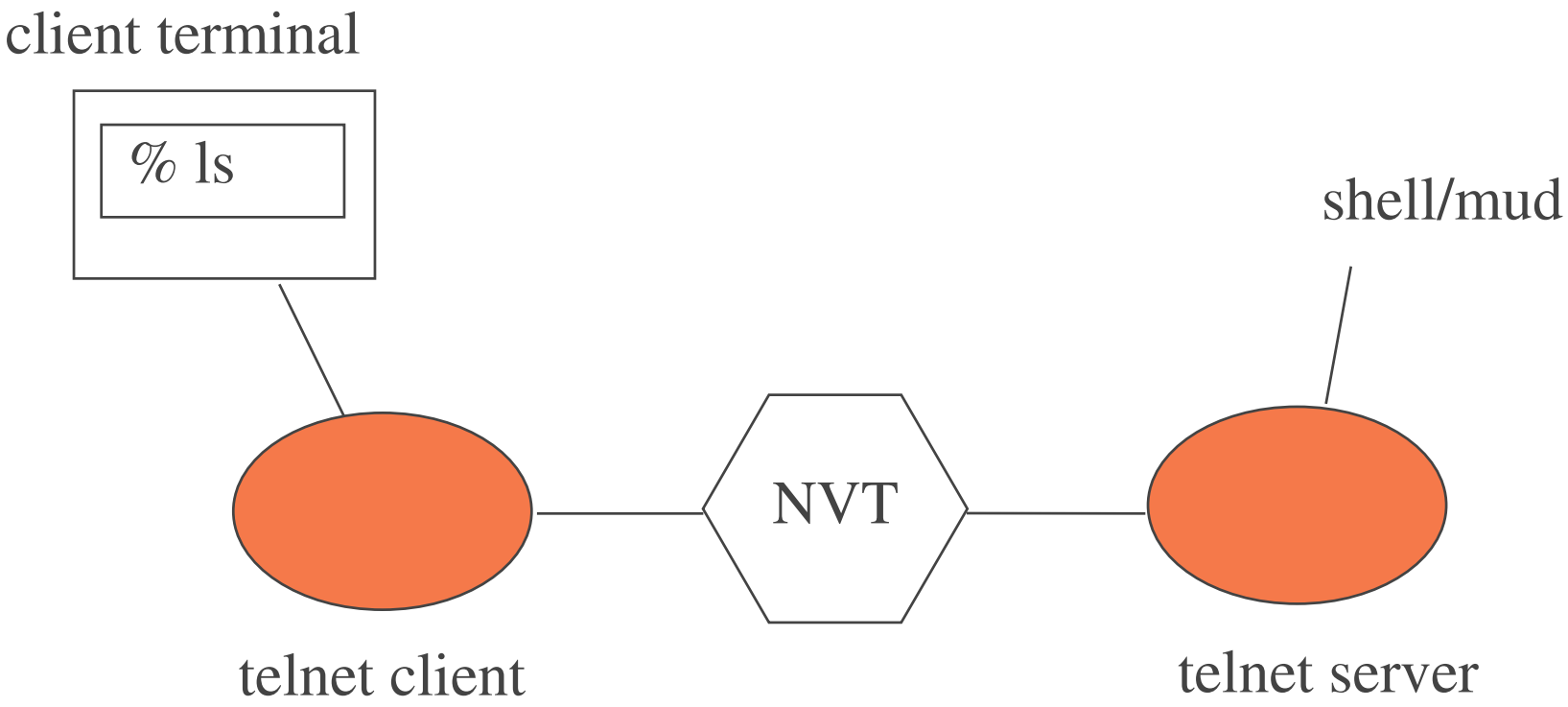
telnet

- telnet can be given a DNS name or ip address
  - if DNS address, use gethostbyname(3)
  - % telnet 199.0.1.2
- one tcp connection, client obtains dynamic port, server uses port 23
- telnet commands encoded and sent on single channel, data + commands both (unlike ftp)
- defines lowest common denominator imaginary terminal called **NVT - network virtual terminal**

telnet

- client must map real terminal onto NVT
- server must map NVT onto backend software (UNIX - “shell console”, DOS box under windows)
- data from client sent to server as input, output from server displayed at client

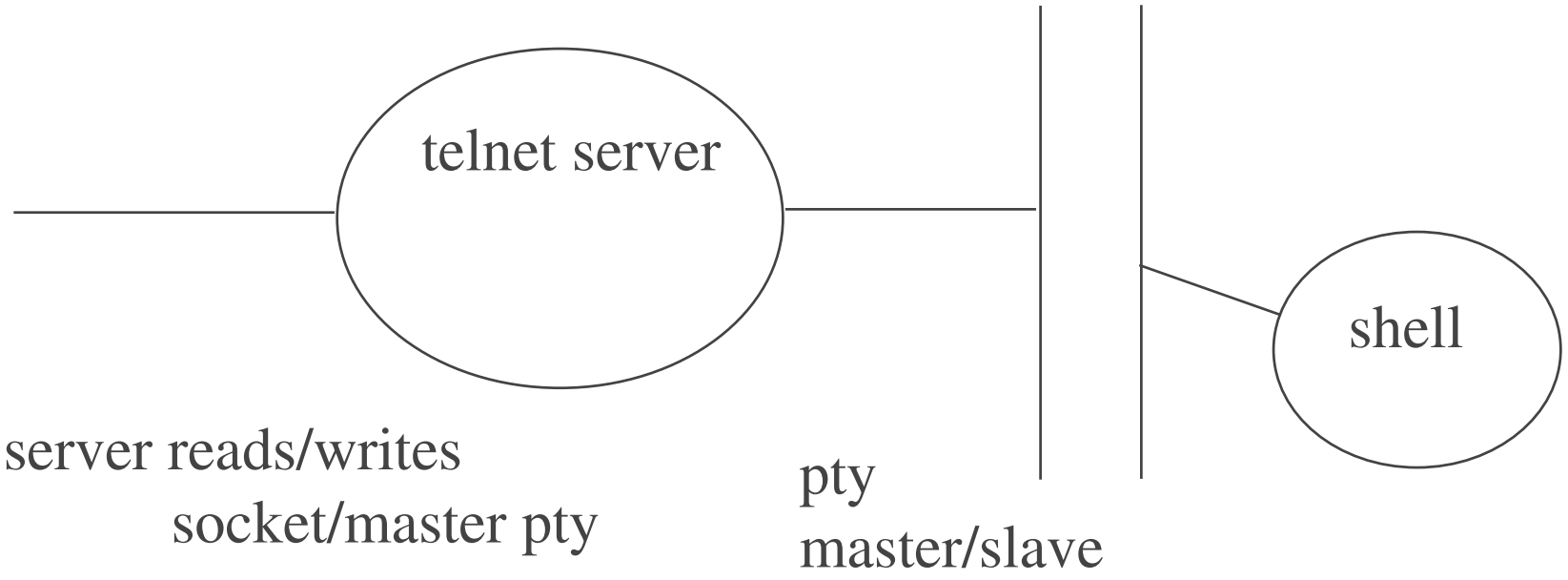
telnet - network architecture



client-side UNIX architecture

select(2) from network socket OR terminal device
if terminal device ready
    read from terminal
    process acc. to telnet NVT
    write to socket
if network device ready
    read from network socket
    process acc to telnet NVT
    write to terminal

server-side UNIX architecture



server-side architecture

- UNIX telnet server - started from inetd
- uses **pty** device pair, master/slave paired device
- two drivers with shared i/o queues (pseudo-terminals)
- master driver emulates serial interrupt side
- allows shell to run on what seems to be serial console
- pro: nasty code in telnetd, not O.S., only needs ptys
- con: characters from client must be **echoed** back through server and master pty device over network
- telnetd will 1st run login and then shell, etc., on slave tty

NVT

- data == 7-bit U.S. ASCII. 8th hi-order bit = 0.
- EOL is sent as \r\n
- FTP, SMTP, finger, all use NVT ASCII
- **in-band signalling** used for both 1. **options** and 2. runtime commands
- options are negotiated at startup by client and server.
- escape sequence used for **in-band** signals starts with IAC 255 “interpret as command”

NVT - runtime commands

- interrupt process - IP (244), send IAC IP
- are you there - AYT (246), send IAC AYT
- erase line - EL (247), send IAC EL
- erase char - EC (247), send IAC EL
- urgent mode (OOB) used for command processing
- data byte 255, send IAC IAC
  - rfc 856 allows 8-bit data transmission

option negotiation at startup

- first exchange
- option negotiation is symmetric, either side can initiate
- 4 kinds of requests:
  - WILL - sender wants to enable option
  - DO - sender wants recv to enable option
  - WONT - sender wants to disable option
  - DONT - sender wants recv to disable option
- can accept/reject enable; always do disable

option negotiation - 6 scenarios

- sender sends WILL, recv sends DO (OK)
- sender sends WILL, recv sends DONT
- sender sends DO, recv sends WILL
- sender sends DO, recv sends WONT
- sender sends WONT, recv sends DONT
- sender sends DONT, recv sends WONT

option negotiation

- sender typically puts 3 bytes on wire:
  IAC WILL <option id>
- except for more complex options (like terminal type which is sent as string)
- example: IAC WILL 1 (1 is echo)
- in truth, typically clients does or asks for certain options, server does/asks for certain options
- see RFC<assigned numbers> for options and RFCs that describe them

synch signal - urgent (OOB) data

- one or the other side may send the other side urgent data (e.g., with an interrupt)
- when recv. gets urgent data “signal”, it will read input stream until it see DM or data mark
- DM signifies the final byte of urgent data
- data up to DM ignored, commands processed
- rationale: if data is clogging buffers, we can still get commands across the one pipe
- BSD sockets have one char “urgent data” queue!

telnet modes of operation

- **half-duplex** - original default, but rarely used, server must say GO AHEAD before input accepted
- **character at a time** (default) - server echos characters, entered if server does SUPPRESS GO AHEAD, WILL ECHO
- **line at a time (kludge mode)** - assumes that if 1 of ECHO and SUPPRESS not done, can do line at a time
- **linemode** (RFC 1184) - correct deficiencies in the kludge mode implementation (4.4 BSD systems support it)
- BSD only systems that try to negotiate kludge mode if linemode not supported

summary

- both default to char at a time, remote echo
- both use tcp
- rlogin assumes unix to unix, simpler overall
- rlogin has weak authentication, some versions support kerberos login
- telnet around longer, more versatile
- telnet line at a time mode apparently slowly gaining as opposed to char at a time
- telnet encryption/authentication supposedly on the way - sorely needed