CSE 461 - Module 2: Protocols

(Computer Communication) Protocol

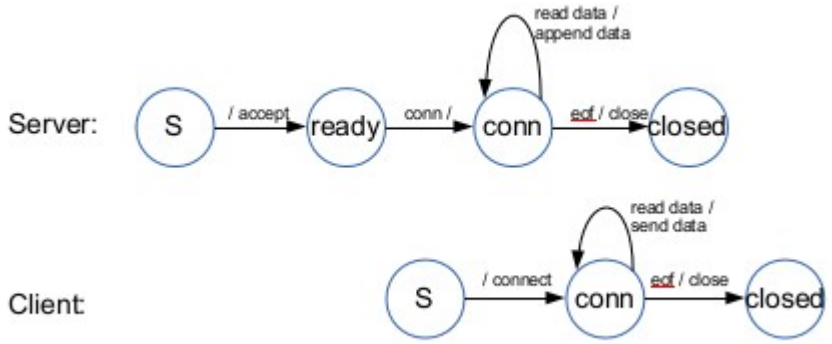
- Dictionary definition: Protocol (noun)
  - *a system of rules that explains the correct conduct and procedures to be followed in formal situations*
- Communication protocol
  - rules: who says what, when
  - syntax: how do you say what you want to say
    - Example (HTTP): `GET /index.html HTTP/1.1`
  - framing: delimiting logical entities
    - Example (HTTP):
      - "lines" end with CR LF
      - HTTP header ends with a null line
  - encoding:
    - Example (hypothetical):
      - strings are consecutive sequences of non-space characters
        - embedded spaces need to be escaped, as "\ "
        - backslash needs to be escaped, as "\\"
      - numbers are given as decimal strings
      - characters are encoded using ASCII

Example 0: Logger

- Application:
  - "client" reads line from keyboard and sends them to the "server"
  - server appends them to a log file
- Example questions related to designing a suitable protocol:
  - What log file?
  - Can more than one client append to the log file at a time?
  - What transport protocol should be used?
  - How will the client "discover" the server?
  - Is there any sort of authentication / authorization?
  - Is there an restriction on the data to be logged? Is binary data ok, for instance?

- What error cases can occur?
- How does the server tell the client an error has occurred?
- Does the server provide “positive acknowledgements”?
- What is being logged? Lines? Paragraphs? Documents?

- For our example, the simplest possible answers:
  - 1 client; statically determined log file name; logging a stream of characters; no ACKs; no indication that an error has occurred; client “just knows” the server's IP:port address; no auth/auth
- Protocol “specification”: client opens a TCP connection to the server and writes data to it; the server reads data from the connection and appends it to the log file
  - What's missing from that?
- State machine representation



Example 2: Telnet

- The **telnet application** allowed remote login (much like ssh does)
  - RFC 854, May 1983
- The **telnet protocol** supports implementing the client and server portions of the telnet application.
  - TCP-based
  - Is primarily about data encoding
- The heart of the state machines for both the client and the server look the same, and basically just send locally obtained input to the other side and print whatever the either side sends
  - That's done concurrently

- There are many “options” that the client can negotiate with the server
- Options (control) is sent in the same stream as the data (characters)
  - A byte with value 0xFF indicates the next byte is a command
  - The command type describes what bytes follow
  - Have to “byte stuff” data bytes with value 0xFF
- Option commands are:
  - Will xxx
    - Means sender wants to use option xxx
    - Responses are Do and Don't
  - Do xxx
    - Means sender wants the other party to start using option xxx
    - Responses are Will and Won't
  - Why?
    - *The syntax of option negotiation has been set up so that if obth parties request an option simultaneously, each will see the other's request as the positive acknowledgement of its own.*
  - Why doesn't it causes response loops?
    - *If a party receives what appears to be a request to enter some mode it is already in, the request should not be acknowledged. It is required that a response be sent to requests for a change of state – even if the mode is not changed.*
- You can see the protocol in action in a small `wireshark` trace
  - Machine 192.168.0.109 uses telnet to login to machine 192.168.0.105
  - What the client saw in the login window was this:

```
Ubuntu 13.10
jz-desktop login: zahorjan
Password:
Last login: Tue Apr  1 19:09:42 PDT 2014 from j2desktop on pts/8
Welcome to Ubuntu 13.10 (GNU/Linux 3.11.0-19-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

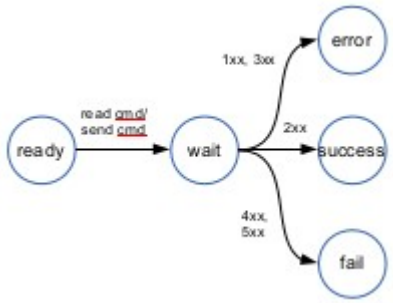
zahorjan@jz-desktop:~
$ logout
```

  - The trace file is at `attu:/cse/courses/cse461/14sp/telnetLogin.pcap`
- Telnet includes an “out of band” signal by utilizing TCP's urgent data facility
  - Put a marker into the data stream
  - Send a TCP urgent segment, which is noticed by the receiver ahead of processing all the data
  - Receiver ignores all data until it encounters the mark

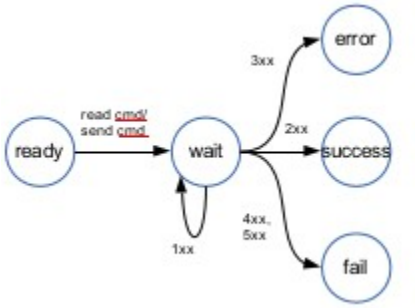
Example 3: FTP

- The **FTP application** provided file transfer, like `scp`, and some aspects of remote shell, like `ssh`.
  - RFC 959, October 1985
- The **FTP protocol** supports that application
- Highlights:
  - FTP uses two separate TCP connections between client and server
    - one for control and one for data transfer
    - why?
  - The control connection uses the Telnet protocol to send command strings
    - Commands are strings; telnet transfers strings
    - Commands are framed as lines; telnet has a line mode option.
  - The data connection attempts to deal with heterogeneity in systems
    - File structure, e.g., stream vs. record
    - Data encoding
      - e.g., binary vs. character content
        - 32-bit vs. 36-bit word machines
      - e.g., character enoding; 36-bit word machines; LF vs CR LF end-of-line; ...
    - FTP transfers files, not bits
  - Dealing with errors
    - FTP uses TCP, so transmission is reliable so long as the connection stays up
    - The connection doesn't stay up
      - FTP includes a facility to restart a transfer when the connection comes back
        - What is required to do that?
  - Strong client-server orientation
    - Client initiates exchanges
  - Strong request-response orientation
    - Example: DELE <pathname> =>
      <= OK
    - Responses are numeric codes (transmitted as characters)
      - 1xx => positive preliminary
      - 2xx => positive completion
      - 3xx => positive intermediate
      - 4xx => transient negative
      - 5xx => permanent negative

- Simple command handling (e.g., DELE):



- Commands that use the data channel:



- Command sequences
  - Example: `rename-from` must be immediately followed by `rename-to`
  - Add states to the state machine
  - The success path is:
    - ```
      rename-from =>
                  <= positive intermediate
      rename-to   =>
                  <= positive complete
      ```
- Discovery
  - Well known port: 21
    - Default data port: control port number – 1
  - FTP includes a PORT command, so client can communicate a non-standard port number
    - In fact, it's possible for the client to transmit an IP for the client side of transfer that is not its own, to control a transfer between two other machines

Example 4: HTTP

- Originally: used to support web browser applications
  - RFC 2616 (HTTP 1.1), June 1999
  - Now: Used for a wide variety of things
- Error handling: based on TCP, plus error message responses
  - Responses include 1xx, 2xx, 3xx, 4xx, 5xx codes
- Pure request-response protocol
      ```

    - Only client can initiate a request
- Framing
    - Header is composed of lines
        - Lines end with CRLF
        - Header ends with a blank line
- Data Encoding / Format
    - Header
        - All data sent as strings
        - Each line of header is of form tag : value
        - Tags can be anything
            - There are some standardized tags, but its legal to make up new tags
    - There is a lot of flexibility in how to encode data
        - Client sends to server list of acceptable encoding, server picks one that it knows how to produce
- Example http request (produced by Chrome):

```
GET /foo.bar.html HTTP/1.1
Host: localhost:44444
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/33.0.1750.152 Safari/537.36
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
```