Part 5

PROTOCOL DESIGN

AGENDA

- Networking Protocol Design Principles
- Common Networking Protocol Techniques
- Learn from old and highly used internet protocols
- Introducing SMTP, POP3, and IMAP by examples

Principles of Protocol Design

- Reference: http://nerdland.net/2009/12/designing-painless-protocols

Protocol Design: Principle 1

Do not re-invent the Wheel!

- Try first to use existing protocols, or at least to imitate them as much as possible
- Protocols which survived many years are probably good and well thought
- They passed a lot of storms and fire tests and they are still here!
- For this, we need to get to know at least the most popular ones first

Protocol Design: Principle 2

KISSD - Keep It Simple Stupid and Deterministic

- Complicated protocols are doomed to cause chaos, complications, and eventually die!
- At every stage it should be completely clear what can happen next!
- Situations in which anything can happen lead to "code pollution“ and later to horrible bugs and eventually to “protocol death”

Protocol Design: Principle 3

Prefer Human Readability

- Prefer plain simple text on short cryptic codes
- Unless speed is truly the most important factor in your system!
- Always better to sacrifice speed for readability
  - "less is more" principle
- Commands like LOGIN, GOODBYE, HELLO, QUIT are much clearer than codes like: 031, 404, 502, etc.
- If your protocol is going to contain free-form text then your protocol really should use Unicode!
- English is most definitely not the only language on the Internet!

Protocol Design: Principle 4

Make Magic Numbers Meaningful

- In many cases, numeric status codes can be useful and even human readable
- Make sure to use meaningful numbers with clear structure
- For example every HTTP response comes with a numeric status code prefix
- Everyone is familiar with:
  HTTP 404 code ("File Not Found" error code)
- In most cases, it's just enough to see the number and immediately understand what happened
- The meaning embedded in this code is the first digit: 4
- User quickly catch the “400” response family

Protocol Design: Principle 4 Example

Make magic numbers meaningful

Architecture:

1xx information
2xx content
3xx redirection
4xx client error
5xx server error

Details:

200 Request was accepted and fulfilled
301 Page moved
400 Bad request
402 Payment required
403 Forbidden request
404 File not found
500 Server Error
501 Not implemented

Protocol Design: Principle 5

Scalability: Design for Expansion!

- If your protocol is good, it will be revised and extended later on (again and again!). Prepare for this from the start!
- Assign meaningful numbers or bit masks as described in principle 4, and reserve bits and fields for future use
- Indicate your protocol version immediately after handshaking (like: "HTTP/1.0")
- Force both connections to announce and match their protocol versions immediately after handshaking
- Thus if a fatal design flaws are found after a year or two, upgrade your protocol to next version and slowly deprecate the old version
- The backbone protocol of the Internet, IP, does exactly this! and that helps makes IPv6 possible! (the IP version is an integral part of the IP header!)

Protocol Design: Principle 6

Don’t be stingy with information

- never hide relevant information from the other side (unless there is a security concern)
- Practically it means: each end of the connection should be able to query the other side for any relevant information
- Example: In the **BFTP** server/client project
  - the client should be able to query the server if a file exists before attempting to retrieve it, or get a list of files in a directory
  - Otherwise, we will never be able to know if a file cannot be retrieved due to server error connection problem? or it simply does not exist?
  - could be very frustrating or lead to inefficient actions

Protocol Design: Principle 7

Document your protocol precisely !!!

- Write a clear and full design specification of your protocol before you implement it
- You cannot implement a protocol which was not clearly designed and well thought
- For example, it is a bad idea to have a “restart connection” command without documenting what exactly should happen when this command is issued? What to do with partial buffers? Late packets? How many consecutive restarts are ok? etc.

Protocol Design: Principle 8

Postel's Law: "be conservative in what you do, be liberal in what you accept from others."

- This was originally coined in RFC 761, the document specifying TCP
- This is a very important, and widely known principle, yet also widely misunderstood
- The most notorious misapplication of this principle was in the implementation of early HTML parsers.
- Based on this idea, the parsers would take in any old junk that vaguely resembled HTML and try as hard as possible to display something on the browser
- The result of this extreme laxity was more than a decade of the nightmare known as "tag soup" which is only now beginning to heal from

Protocol Design: Postel's law

Postel's Law: "be conservative in what you do, be liberal in what you accept from others."

- The real meaning of the Robustness Principle is not that erroneous input should be accepted as valid, but that erroneous input should not cause catastrophic failure!
- Valid parts of a partially-erroneous input should be accepted if possible, and that diagnostics should be given for erroneous input when feasible
- An HTML parser implementation that properly followed this rule would, upon receiving "tag soup" HTML
  - produce a warning message that the HTML was invalid
  - hopefully display some information about what was wrong (e.g. unclosed anchor tag, missing doctype, etc)
  - and only then try to (or give the option to) display the parser's best approximation of what the author meant

Protocol Design: Principle 9

Design for security from the start

- Security is a common problem to many of the standard protocols, which we live with its detrimental effects every day
- These protocols, designed when the Internet was in its infancy as an academic and governmental experiment, were not designed with security in mind
- This is what facilitates spam, denial-of-service, phishing, privacy invasion, and all other sorts of Internet security problems
- Today, however, it is unacceptable to design a new protocol without giving it serious thought from the start
- Experience shows that if it is not done at the start, it may become too hard to do after a protocol has been widely deployed
- Encryption should be a layer: once the encryption layer is removed, the protocol should continue to adhere to the design principles articulated above

Learn From Examples:

Common Internet Protocols

SMTP – Simple Mail Transport Protocol

Described by RFC 2821 (RFC = Request For Comments)

```
CLIENT:   <<client connects to service port 25>>     # HANDSHAKING
CLIENT:   HELO shark.braude.ac.il                    # Sending host identifies itself
SERVER:   250 OK Hello shark, glad to meet you       # Server acknowledges
CLIENT:   MAIL FROM: <dan@braude.ac.il>              # Identify sending user/domain
SERVER:   250 <dan@braude.ac.il>... Sender ok        # Server acknowledges
CLIENT:   RCPT TO: ran@stimpy.com                    # Identify target user
SERVER:   250 root... Recipient ok                   # Server acknowledges
CLIENT:   DATA
SERVER:   354 Enter mail, end with "." on a line by itself
CLIENT:   Hi Fred: Frenchy called. He wants to share
CLIENT:   options, cards,
CLIENT:   and a large collection of old baseball bats
CLIENT:   Lehitraot,
CLIENT:   Dan
CLIENT:   .                                          # End of multiline send
SERVER:   250 WAA01865 Message accepted for delivery
CLIENT:   QUIT                                       # Client (email sender) signs off
SERVER:   221 stimpy.com closing connection          # Server disconnects
CLIENT:   <<client hangs up>>
```

SMTP: Protocol Design

- SMTP is used for uploading mail to a mail server
- Client requests have a simple command line format:
  - **HELO ...**
  - **MAIL ...**
  - **DATA ...**
  - **RCPT ...**
- Server responses consisting of a status code followed by an informational message:

```
250 <dan@braude.ac.il>... Sender ok
221 stimpy.com closing connection
```

- Server response consists of a status code and a human message
- Protocol software uses the status code and usually ignores the human part
- The **DATA** command sends the mail body, terminated by a line consisting of a single dot

SMTP: Main Commands

- SMTP is one of the oldest application layer protocols which is still in high use on the Internet today
- It is simple, effective, and has withstood the test of time

```
HELO <sendinghostname>
     This command initiates the SMTP conversation.
     The host connecting to the remote SMTP server identifies itself
     by it's fully qualified DNS host name.

MAIL From:<source email address>
     This is the start of an email message.
     The source email address is what will appear in the
     "From:" field of the message.

RCPT To:<destination email address>
     This identifies the receipient of the email message.
     This command can be repeated multiple times for a given
     message in order to deliver a single message to multiple recepients.

For more details look at: http://the-welters.com/professional/smtp.html
```

POP3 – Retrieve mail from server

```
CLIENT:   <<client connects to service port 110>>
SERVER:   +OK POP3 server ready <1896.6971@mailgate.dobbs.org>
CLIENT:   USER bob
SERVER:   +OK bob
CLIENT:   PASS redqueen
SERVER:   +OK bob's maildrop has 2 messages (320 octets)
CLIENT:   STAT
SERVER:   +OK 2 320
CLIENT:   LIST
SERVER:   +OK 2 messages (320 octets)
SERVER:   1 120
SERVER:   2 200
SERVER:   .
CLIENT:   RETR 1
SERVER:   +OK 120 octets
SERVER:   <the POP3 server sends the text of message 1>
SERVER:   .
CLIENT:   DELE 1
SERVER:   +OK message 1 deleted
CLIENT:   RETR 2
SERVER:   +OK 200 octets
SERVER:   <the POP3 server sends the text of message 2>
SERVER:   .
CLIENT:   DELE 2
SERVER:   +OK message 2 deleted
CLIENT:   QUIT
SERVER:   +OK dewey POP3 server signing off (maildrop empty)
CLIENT:   <<client hangs up>>
```

POP3 – Client Commands

- Client commands always start with a 4 characters code

```
USER <username>
PASS <password>
STAT
LIST
RETR <message-id>
DELE <message-id>
QUIT
```

POP3 – Server Commands

- Server has only two response modes: **+OK**, **-ERR**
- Which are essentially "+" and "-", where "**OK**" and "**ERR**" are the "human parts"
- For some client commands, the server status line is followed by data which ends with a single "**.**" line

```
+OK POP3 server ready <1896.6971@mailgate.dobbs.org>
+OK bob
+OK bob's maildrop has 2 messages (320 octets)
+OK 2 320
-ERR never heard of jim
```

http://www.pnambic.com/Goodies/POP3Ref.html

IMAP - Internet Message Access Protocol

- A newer post office protocol designed in a slightly different style
- **IMAP** was designed to replace **POP3**
- Excellent example of a mature and powerful design worth studying and following its principles
- In the next example, user **ilanitk** is logging to a mail server to **retrieve** her email

  (well, it's not Ilanit who is doing it, it's outlook or gmail client without her knowing about it)

IMAP - Internet Message Access Protocol

```
CLIENT:   <<client connects to service port 143>>
SERVER:   * OK iserver.com IMAP4rev1 v12.264 server ready
CLIENT:   A001 USER "ilanitk" "june1987"
SERVER:   * OK User ilanitk authenticated
CLIENT:   A002 SELECT INBOX
SERVER:   * 1 EXISTS
SERVER:   * 1 RECENT
SERVER:   * FLAGS (\Answered \Flagged \Deleted \Draft \Seen)
SERVER:   * OK [UNSEEN 1] first unseen message in /var/spool/mail/dan
SERVER:   A002 OK [READ-WRITE] SELECT completed
CLIENT:   A003 FETCH 1 RFC822.SIZE                    Get message sizes
SERVER:   * 1 FETCH (RFC822.SIZE 2545)
SERVER:   A003 OK FETCH completed
CLIENT:   A004 FETCH 1 BODY[HEADER]                   Get first message header
SERVER:   * 1 FETCH (RFC822.HEADER {1425}
          <<server sends 1425 octets of message payload>>
SERVER:   )
SERVER:   A004 OK FETCH completed
CLIENT:   A005 FETCH 1 BODY[TEXT]                     Get first message body
SERVER:   * 1 FETCH (BODY[TEXT] {1120}
          <<server sends 1120 octets of message payload>>
SERVER:   )
SERVER:   * 1 FETCH (FLAGS (\Recent \Seen))
SERVER:   A005 OK FETCH completed
CLIENT:   A006 LOGOUT
SERVER:   * BYE iserver.com IMAP4rev1 server terminating connection
SERVER:   A006 OK LOGOUT completed
CLIENT:   <<client hangs up>>
```

IMAP - Internet Message Access Protocol

- The standard IMAP procedure is to leave messages on the server instead of retrieving copies
- Email is only accessible when "on-line" (from different locations, and different devices)
- Suited to a world of "always-on/anywhere" connections
- Messages remain on the server, until deleted by the user
- Messages can be accessed by multiple client computers
- Clear advantage when you use more than one computer to check your email (laptop, tablet, smartphone)
- Microsoft "MAPI" is a proprietary variation for their outlook/exchange client/server model (does not work for anything else)

IMAP - Internet Message Access Protocol

- IMAP uses the "Message Length in Advance Technique":
- instead of ending the payload with a **dot**, the payload length is sent in advance
- This makes life harder on the server a little bit:
  - messages have to be composed ahead of time
  - messages cannot be streamed after the send initiation
- But makes life easier for the client
  - Client can know in advance storage and buffer sizes it will need to process the message

IMAP - Internet Message Access Protocol

- Each response is tagged with a **sequence label** supplied by the client
- In the example above they have the form **A000n**, but the client could have generated any token into that slot
- This feature makes it possible for **IMAP** commands to be streamed to the server without waiting for the responses
- A state machine in the client can then simply interpret the responses and payloads as they come back
  - This technique cuts down on latency

RFC – Request For Comments

- Protocol design life cycle starts with an RFC
- RFC's are publications made by Internet Engineering Task Force (IETF)
- IETF develops and promotes Internet standards
- Founded by the US government around 1969 (part of the ARPANET project), but is now a very large international organization with many sub-organizations (acm, IEEE)
- Official RFC's database: http://www.rfc-editor.org/rfc.html
- For example, here is RFC 3501 (March 2003) for the IMAP specifications:
  http://www.rfc-editor.org/rfc/rfc3501.txt
  http://www.rfc-editor.org/rfc/rfc4978.txt
- (read it and write a similar doc for BFTP …)