Internet and Intranet Protocols and Applications

Lecture 4: General Characteristics of Internet Protocols; the Email Protocol

February 12, 2003
Arthur Goldberg
Computer Science Department
New York University
artg@cs.nyu.edu
Tonight

- Introduction to Internet applications and their protocols
- How to read RFCs
- Internet Mail Architecture
  - SMTP (RFC 821)
  - Messages (RFC 822)
  - POP3
- Discussion of approach to assignment 1
What do the most widely used Internet applications have in common?

• Client/Server paradigm
• Request/Response Protocol
• Typically use ASCII protocol
• TCP or UDP
How to Read RFCs

• ID: RFC number
  – About 3400 total

• Obsoletes: previous RFCs superceeded

• Category: Standard, Informational, Experimental (see below)

• Status: see maturity levels below
Typical RFC Contents

- Protocol structure
  - E.g., section 2.1 of 2821
- Objects being processed
- Protocol commands
- Reply codes
- Security issues
Requirements Terminology
(From 2119)

- **MUST, REQUIRED, or SHALL:**
  - Functionality that all conforming implementations *must* implement

- **MUST NOT, or SHALL NOT**
  - Functionality that all conforming implementations *must not* implement

- **SHOULD, or RECOMMENDED:**
  - Functionality that conforming implementations need good reasons to not implement

- **SHOULD NOT, or NOT RECOMMENDED:**
  - Functionality that conforming implementations need good reasons to implement

- **MAY, or OPTIONAL**
  - Functionality that is truly optional
Goals of the Internet Standards Process (from RFC 2026, Bradner)

- technical excellence;
- prior implementation and testing;
- clear, concise, and easily understood documentation;
- openness and fairness; and
- timeliness.
The Internet Standards Track
Maturity Levels

• Proposed Standard
  • Generally stable, has resolved known design choices, has received significant community review
  • But further experience might result in a change or even retraction of the specification

• Draft Standard
  • at least two independent and interoperable implementations from different code bases have been developed, and tested

• Internet Standard
  • generally held belief that the specified protocol or service provides significant benefit to the Internet
Non-standards Track Specifications

- Experimental
  - R&D

- Informational
  - General information—not a consensus

- Historic
  - An obsolete RFC
Applications and application-layer protocols

Application: distributed processes
– running in network hosts in “user space”
– exchange messages to implement application

Application-layer protocols
– define messages exchanged by apps and actions taken
– user services provided by lower layer protocols
Client-server paradigm

Typical network app has two pieces: client and server

Client:
• initiates contact with server ("speaks first")
• typically requests service from server,
• for Web, client is implemented in browser; for e-mail, in mail reader

Server:
• provides requested service to client
• e.g., Web server sends requested Web page, mail server delivers e-mail
Transport service requirements of common apps

<table>
<thead>
<tr>
<th>Application</th>
<th>Data loss</th>
<th>Bandwidth</th>
<th>Time Sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>file transfer</td>
<td>no loss</td>
<td>elastic</td>
<td>no</td>
</tr>
<tr>
<td>e-mail</td>
<td>no loss</td>
<td>elastic</td>
<td>no</td>
</tr>
<tr>
<td>Web documents</td>
<td>loss-tolerant</td>
<td>elastic</td>
<td>no</td>
</tr>
<tr>
<td>real-time audio/video</td>
<td>loss-tolerant</td>
<td>audio: 5Kb-1Mb</td>
<td>yes, 100’s msec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>video: 10Kb-5Mb</td>
<td></td>
</tr>
<tr>
<td>stored audio/video</td>
<td>loss-tolerant</td>
<td>same as above</td>
<td>yes, few secs</td>
</tr>
<tr>
<td>interactive games</td>
<td>loss-tolerant</td>
<td>few Kbps up</td>
<td>yes, 100’s msec</td>
</tr>
<tr>
<td>financial apps</td>
<td>no loss</td>
<td>elastic</td>
<td>yes and no</td>
</tr>
</tbody>
</table>
Services provided by Internet transport protocols

TCP service:
• connection-oriented: setup required between client, server
• reliable transport between sending and receiving process
• flow control: sender won’t overwhelm receiver
• congestion control: throttle sender when network overloaded
• does not provide: timing, minimum bandwidth guarantees

UDP service:
• unreliable data transfer between sending and receiving process
• does not provide: connection setup, reliability, flow control, congestion control, timing, or bandwidth guarantee

Q: Why bother? Why is there a UDP?
## Internet apps: their protocols and transport protocols

<table>
<thead>
<tr>
<th>Application</th>
<th>Application layer protocol</th>
<th>Underlying transport protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>e-mail</td>
<td>SMTP [RFC 821]</td>
<td>TCP</td>
</tr>
<tr>
<td>remote terminal access</td>
<td>telnet [RFC 854]</td>
<td>TCP</td>
</tr>
<tr>
<td>Web</td>
<td>HTTP [RFC 2068]</td>
<td>TCP</td>
</tr>
<tr>
<td>file transfer</td>
<td>ftp [RFC 959]</td>
<td>TCP</td>
</tr>
<tr>
<td>streaming multimedia</td>
<td>proprietary (e.g. RealNetworks)</td>
<td>TCP or UDP</td>
</tr>
<tr>
<td>remote file server</td>
<td>NSF</td>
<td>TCP or UDP</td>
</tr>
<tr>
<td>Internet telephony</td>
<td>proprietary (e.g., Vocaltec)</td>
<td>typically UDP</td>
</tr>
</tbody>
</table>
Electronic Mail

Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP

**User Agent**

- a.k.a. “mail reader”
- composing, editing, reading mail messages
- outgoing, incoming messages stored on server
Mail System Example
Electronic Mail: mail servers

Mail Servers

- mailbox contains incoming messages (yet to be read) for user
- message queue of outgoing (to be sent) mail messages
- SMTP protocol between mail servers to send email messages
  - client: sending mail server
  - “server”: receiving mail server
Electronic Mail: SMTP [RFCs 821, 2821 (obsoletes 821)]

• uses TCP to reliably transfer email msg from client to server, port 25
• direct transfer: sending server to receiving server
• three phases of transfer
  – handshaking (greeting)
  – transfer of messages
  – closure
• command/response interaction
  – commands: ASCII text
  – response: status code and phrase
• messages must be in 7-bit ASCII
SMTP

• Minimum Implementation Commands

**HELO**  <SP>  <domain>  <CRLF>
**MAIL**  <SP>  FROM:<reverse-path>  <CRLF>
**RCPT**  <SP>  TO:<forward-path>  <CRLF>
**DATA**
**QUIT**
**RSET**
**NOOP**
Sample SMTP interaction

S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
You can try SMTP interaction for yourself

- `telnet servername 25`
- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

above lets you send email without using email client (reader)
SMTP: Key Points

- SMTP uses persistent connections (WHY?)
- SMTP pushes email
- SMTP requires that message (header & body) be in 7-bit ASCII
- certain character strings are not permitted in message (e.g., CRLF . CRLF). Thus message has to be encoded (usually into either base-64).
- SMTP server uses CRLF . CRLF to determine end of message
How to send crlf.crlf ??

• If CRLF.CRFL ends the message, how to send this as part of your message?

• Sender:
  – if line starts with period, insert period at the beginning of the line.

• Receiver:
  – If the line is composed of a single period it is the end of mail.
  – If the first character is a period and there are other characters on the line, the first character is deleted.
Mail message format

RFC 822: standard for text message format:

- header lines, e.g.,
  - To:
  - From:
  - Subject:
    different from SMTP commands!

- body
  - the “message”, ASCII characters only
Message format: multimedia extensions

- MIME: multipart Internet mail extension, RFC 2045, 2056
- additional lines in msg header declare MIME content type

From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data ......
..........................
......base64 encoded data
MIME types

Text: plain, html
Image: jpeg, gif
Audio: basic, 32kadpcm
Video: mpeg, quicktime
Application: other data that must be processed by reader before “viewable”. Example subtypes: msword, octet-stream.

Multipart: used when sending objects of different types
Multipart Type

• Multipart separation
  – boundary="------------C725BD4EC548463C"

• Potentially recursive
  – Each part described by a MIME type
Multipart Example

From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=98766789

--98766789
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain

Dear Bob,
Please find a picture of a crepe.
--98766789
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data ..... 
.........................
.......base64 encoded data
--98766789--
What is base64?

• Problem: how to send binary data?

• Base64 is hashing method to map 8 bit to 6 bit codes that define a subset of the ASCII Character space.
• Convert each 24 bits to four 6 bit codes, pad trailing bits with 0’s.
• Use 6 bit code to index table for mapped ASCII character
• Note: base64 map table does not include CRLF or other RFC822 special characters (or =, which is a base64 pad character).

• The Base 64 Conversion Table
Mail access protocols

- Mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]
    - authorization and download
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - more features (more complex)
    - manipulation of stored messages on server
  - HTTP: Hotmail, Yahoo! Mail, etc.
POP3 protocol

authorization phase
• client commands:
  – **user**: declare username
  – **pass**: password
• server responses
  – +OK
  – -ERR

transaction phase
• **list**: list msg numbers
• **retr**: retrieve msg by number
• **dele**: delete msg by number
• **quit**

```plaintext
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```
POP3 Commands, by State

• AUTHORIZATION

  USER name
  PASS password
  APOP name digest - ‘digest' is MD5 of the server’s timestamp and a shared SECRET known only to the client and server
POP3 Commands, by State, continued

- **TRANSACTION**
  - **STAT** – status
  - **LIST** – list messages
  - **UIDL [msg]** – unique-id listing
  - **RETR msg** – retrieve msg
  - **DELE msg** – delete msg
  - **QUIT** – enter UPDATE state

- **UPDATE**
  - **QUIT** – remove messages marked as deleted; release any lock on the mailldrop and close the connection
Secure Email

• How to implement?
  – In SMTP or in Message?
  – Remember layering concept!
  – What would happen if Mail Servers were required authenticate one another?

• What is it that we want protect?
Secure Email: Approaches

- **OpenPGP** (RFC 2440) based on PGP
- **S/Mime** (RFC 2311-2315, 2268) based on RSA

Both provide:
- digital signing (authentication)
- encryption (privacy)

Implementation is in the **MUA**

**OpenPGP** is further along in IETF standards track and is available for most User Agents.

Or, simply, IMAP or POP over secure channel