

What is application protocol

Lecture 2

Application protocol: E-mail

References:

- CN Section 7.2.
- Tlv1 Chapter 28.
- RFCs: 821, 2821 (SMTP), 822, 2822 (message format), 1341 (MIME), 1939 (POP3), 2060 (IMAP).
All RFCs are freely available on the net, from <http://www.ietf.org>. Our department also has a copy at <ftp://ftp.csis.hku.hk/pub/doc/rfc>.

POS(0234B)

- In last lecture we see that the network software helps us to deal with a lot of unexpected problems of the network.
- Application programs normally **don't** need to deal with such problems, and can thus **expect the network to be reliable**.
- On the other hand, applications solve problems that are specific to the application.
If the problem occurs in all applications, then it should have been implemented in some layer below it.
- So **every application has its own protocol**, reflecting its own requirements (and sometimes history).
- Instead of looking at all possible applications (which is a huge number), we will just look at one to see what kind of problems are there.
- We will look at the complete E-mail system.

POS(0234B)-2.1

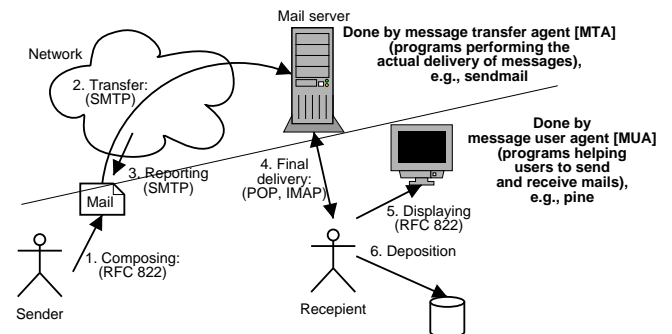
A bit of history

- Key feature: sending and receiving can occur at completely different times (like mail, unlike phone).
- First E-mail system is a **file transfer protocol**, 1st line is the recipient address, and otherwise plain text.
- Simplicity is not superiority here: the lack of structure means the lack of functionalities like group mailing, easy searching, multiple content types, etc.
- Eventually **internet standards** RFC821 (transmission protocol) and RFC822 (message format) are proposed, and later refined to actual standards (RFC 2821, 2822).
RFC = Request For Comment, a stage of standard proposal when it is still not certain it will become standard. But since people read it more than the actual standard, they usually refer to the standard by the RFC numbers.
- CCITT created another standard X.400, but nobody actually use it.

POS(0234B)-2.2

Basic E-mail architecture

To support delayed reading of mails, the E-mail system centers around the mail server. Senders and receivers use it for transferring mails.



POS(0234B)-2.3

Mail content: RFC822

A message is a **stream of bytes**, interpreted as lines separated by the end-of-line character.

- Header fields, one per line: To, Cc, From, Sender (who actually send the message), Received (servers that processed the message), Return-Path, Date, Subject, Reply-To, Keywords, ..., and user-defined.
- Single blank line indicate end of headers.
- Actual mail content.

Note that the specification is **completely ASCII**. This is not an accident: ASCII makes it **easier to test and debug** the protocol (although it sacrifice a bit of efficiency).

When a message is **delivered**, a "From line" indicates the start of a message. The line starts with the five characters "From ", and contain information about the sender and date of message reception.

What if "From " appear in message content?

POS(0234B)-2.4

Example message

```
H| Received: from staff.csis.hku.hk ([147.8.176.12])
H|   by hwpq11.csis.hku.hk with smtp (Exim 3.35 #1 (Debian))
H|   id 17qgGS-00029x-00
H|   for <kkto@csis.hku.hk>; Mon, 16 Sep 2002 15:25:48 +0800
H| Received: from camaro.informatik.uni-freiburg.de (camaro.informatik.uni-
H|   freiburg.de [132.230.167.178])
H|   by staff.csis.hku.hk (8.9.0/8.9.0) with ESMTTP id
H|   JAA20267; Mon, 16 Sep 2002 09:26:18 +0200 (MET DST)
H| To: jckyau@csis.hku.hk
H| Cc: kkto@csis.hku.hk (Isaac To)
H| Subject: Re: Apollo 11....
H| From: Sau Dan Lee <danlee@informatik.uni-freiburg.de>
H| Date: 16 Sep 2002 09:25:39 +0200
H| Message-ID: <xb7znui5rbg.fsf@camaro.informatik.uni-freiburg.de>
B|
C| >>>> "jckyau" == jckyau <jckyau@csis.hku.hk> writes:
C|
C|   jckyau> This is crazy (please visit both of them)....
C|   jckyau> http://www.research.att.com/~smb/imex.gif
C|
C|   jckyau> http://www.independent.co.uk/story.jsp?story=56850
C|
C| God bless America!
```

POS(0234B)-2.5

Simple Mail Transport Protocol (SMTP)

- Once we know what is a mail, we need a mechanism to actually send it. The protocol is specified in RFC 821, named SMTP.
- SMTP is talked over a well-known Internet port, number 25.
- Again the protocol is completely ASCII.
- Primary aim of the protocol is to **transfer the message**, supposed in RFC 822 format. So the important thing here is to specify **who is sending a mail, what is the mail, and who should get that mail**.
Why specify the recipient a second time? The field in a RFC 822 message may contain many recipient, and the specification here is for one server. It also removes the distinction between To, Cc, etc; and also allow the use of Bcc, a recipient who is not in the RFC 822 message.
- The sender repeatedly sends commands to the mail server, the server responds with the result executing that command.

POS(0234B)-2.6

Example

The dialog from sender to mail server (→) and back (←) looks like this:

```
<-- 220 study.csis.hku.hk SMTP service ready
--> HELO myhost.csis.hku.hk
<-- 250 study.csis.hku.hk Hello ktkto-pc [147.8.175.178], pleased to meet you
--> MAIL FROM: <kkto@csis.hku.hk>
<-- 250 2.1.0 <kkto@csis.hku.hk>... Sender ok
--> RCPT TO: <kkto@csis.hku.hk>
<-- 250 2.1.5 <kkto@csis.hku.hk>... Recipient ok
--> DATA
<-- 354 Enter mail, end with "." on a line by itself
--> Test only: this is not even in RFC 822 message format!
--> .
<-- 250 2.0.0 gBJ7j0W28386 Message accepted for delivery
--> QUIT
<-- 221 study.csis.hku.hk closing connection
```

The program reads only the first number written by the server. The string is for human to debug the protocol.

Most MTA accepts messages in non-RFC 822 format, and react by adding just enough headers so that it becomes RFC 822.

And note another thing that is disallowed in RFC 822 data.

POS(0234B)-2.7

Non-text: MIME

So everything is in ASCII: message data, transfer protocol, storage format. What if I have something non-ASCII to send?

- There are two problems here:
 - How to specify **what type of data** we want to send?
Saying that "the mail content should tell the user about it" and therefore "everything is a simple file" would make it very inconvenient to view the message.
 - How to **represent non-ASCII with ASCII** for MTAs?
Some MTA won't work with 8-bits, some break up or even truncate long lines, some have short message length limit, etc.
- The answer: MIME (Multipurpose Internet Mail Extension, RFC1341)
 - New headers to specify what is the type and "encoding" of content.
 - Some encoding which allows arbitrary data to be converted to short lines of ASCII characters containing no "strange" characters.

POS(0234B)-2.8

RFC 822 headers added by MIME

They direct the MUA to display the message in a sensible way.

- **MIME-Version:** Identifies the MIME version, usually 1.0.
For future extension. Most protocols has such a "version" field.
- **Content-Description:** a short description of what is contained. E.g., "Photo of my cat".
- **Content-Id:** a unique identifier. Usually unused.
- **Content-Type:** What is the file type. E.g., audio/wav, text/plain. The first part is the "**type**", the second part is the "**sub-type**".
Solves our first problem. It is also the place to put the coding system used for text, e.g., "text/plain; charset=big5" specifies traditional Chinese text.
- **Content-Transfer-Encoding:** How the content is wrapped, e.g., quoted-printable, 7bit, 8bit, base64.
Indicate what method is used for solving the second problem.

POS(0234B)-2.9

ASCII armor

- **Quoted-printable:** send 7-bit characters as usual. For 8-bit characters, they are **escaped** with the = character, followed by its hex ASCII value (e.g., "=A8").
What to do if the content contains "=" itself? (Hint: it also has an ASCII value.) This technique is called byte-stuffing, and we will meet it again soon.
- **Base64:** Every 3 bytes (i.e., 24 bits) are interpreted as 4 6-bit quantities. **A 6-bit quantity** has only **64 possible values**, so it just use A-Z, a-z, 0-9, + and / (Note: no period and space...).
Newlines are inserted periodically to make lines short enough.
- A quoted-printable message can mostly be viewed **without** a special viewer, but is less efficient when there are a lot of 8-bit characters.
It encodes 1 byte with 3 bytes, while base64 encodes 3 bytes with 4 bytes.
- So sometimes we want quoted-printable and sometimes we want base64. The **choice** is usually **made by MUA**.

POS(0234B)-2.10

Multi-part messages

- Sometimes we want to send a mail **together with some other content**, e.g., another mail, an image, etc. Sometimes the same thing is sent in **different formats**, allowing the receiver to choose between them.
- MIME answers this need by a special content type: **Multipart**. A multipart message contains multiple messages, **each in MIME format**.
Note the recursion. So a multipart message can contain another multipart one.
- **Subtype** allows users to specify whether the contained messages are equivalent (and hence only one need to be shown).
If so, it is multipart/alternative; otherwise it is multipart/mixed.
- The parameter *boundary* specifies a separator to split the parts. E.g., multipart/alternative; boundary="-----"
- A new header *Content-Disposition* allows users to specify whether the part is as part of the enclosing message or an "attachment".
Values: "inline" or "attachment". It also allows suggesting a filename.

POS(0234B)-2.11

Example multipart message

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="===="

-----

The following is my very fragile bash init script. Please use
with a lot of care as it is going to break your system.

-----
Content-Disposition: attachment; filename=.bashrc
Content-Description: My bash init script

HISTSIZE=500
HISTFILE=$HOME/Login/.bash_history
HISTFILESIZE=100
IGNOREEOF=1
MAILCHECK=1
PS1="\h-\u \! \w>"

-----
Note the last separator: two more "-" specifies end of message.
```

POS(0234B)2.12

POS(0234B)2.13

Example POP3 session

POP3 sessions are surprisingly similar to SMTP sessions. The well-known port number is 110 this time.

```
<-- +OK POP3 server ready
--> USER carolyn
<-- +OK
--> PASS vegetables
<-- +OK login successful
--> LIST
<-- 1 2505
<-- 2 14302
<-- 3 8122
<-- .
--> RETR 1
<-- (message 1)
--> DELE 1
--> QUIT
<-- +OK POP3 server disconnecting
```

Note the single period of the response from the LIST command tells that it has no more lines to send.

POS(0234B)2.14

Final delivery

- There is one more problem: how the user gets the message?
- If the user can run commands in the mail server, this is not a problem. The MUA can run in the server directly.
- It is not a problem either if the user has a computer that shares the hard disk with the server. This is the situation of this department.
The problem is solved by another network application: network filesystem.
- But most ISP users are not that lucky. The answer: **Post Office Protocol** (POP3, RFC 1939), allowing users to retrieve mails from remote.
- The answer is adequate for many users, but not so for many others: once you get the mails, you can no longer see it in another computer.
- The answer: **Internet Message Access Protocol** (IMAP, RFC 2060). It allows users to access and manage mails from remote.

Conclusion

We have finished our investigation to the application arguably most familiar to us all. To summarize:

- Application protocols are designed to serve **unique requirements of applications**. E.g., a file transfer protocol will have no need for multipart messages.
- Most application protocols assume underlying network channel to be **reliable**. E.g., there is nothing in the message that allows error checking.
- Network protocols are **heavily influenced by its history**. E.g., life will be a lot easier if the first E-mail system uses 8-bit encoding.
- They are mostly concerned with **moving data from one place to another** in a format that the other side **understands**.
- Much of protocol designs are to cater for cases where **the data contains the symbols reserved by the protocol**.

POS(0234B)2.15