Internet Technology

03r. Application layer protocols: email

Paul Krzyzanowski

Rutgers University

Spring 2016

Email: SMTP
(Simple Mail Transfer Protocol)

Simple Mail Transfer Protocol (SMTP)

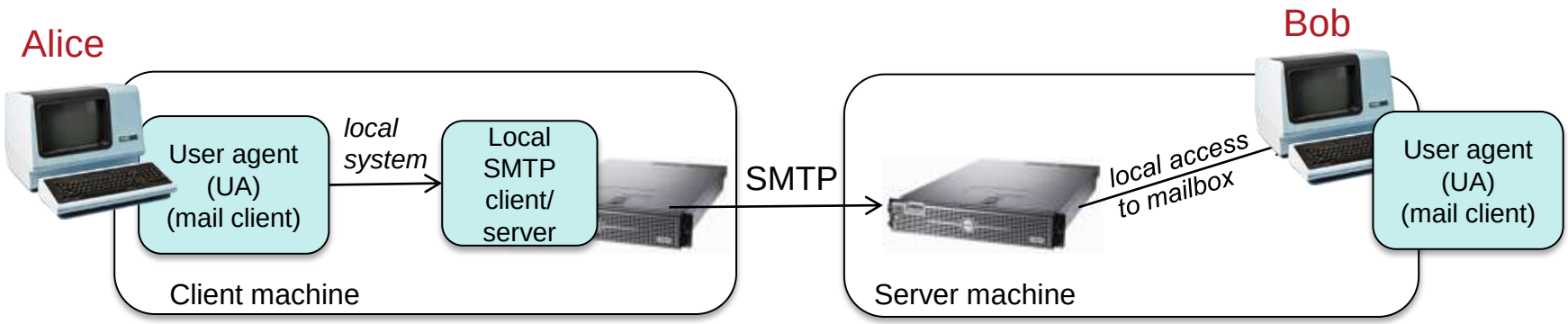
- Protocol defined in RFC 2821 (April 2001)
  - Original definition in RFC 821 (August 1982)
- Designed for:
  - Direct transfer of email from the sender to the receiver (rather then go through a set of relays)
  - Destination system is always up and connected
  - Use TCP to transfer email from client to destination server

Note: There are a lot of variations on email delivery, transmission, and routing. We'll look at a basic model here. Our interest is the app-layer protocol and we'll avoid the terminology of mail submission agents, mail transfer agents, mail exchangers, and mail delivery agents. In most cases one program serves the role all of these.

Original model: Alice sends message to Bob

In the early days of email, users were logged into one system. A mail application would send a message to a local mail server. The local mail server would maintain a queue and send messages to their destinations. Receiving users would run a mail application on the server that would open the mail file – their queue of received messages.

1. Alice uses a mail application (User Agent, UA) to compose a message to Bob
2. Alice's UA sends the message to her local mail server – message is in the outbound queue
3. The local server (acting as an SMTP client) uses DNS to look up the MX record (Mail Exchanger) for Bob's domain; opens TCP connection with Bob's mail server
4. Alice's local mail server (acting as a client) sends Alice's message over the TCP connection using SMTP
5. Bob's mail server receives it and places the message in Bob's mailbox
6. Bob runs his UA, which can access his mailbox (by opening the file)



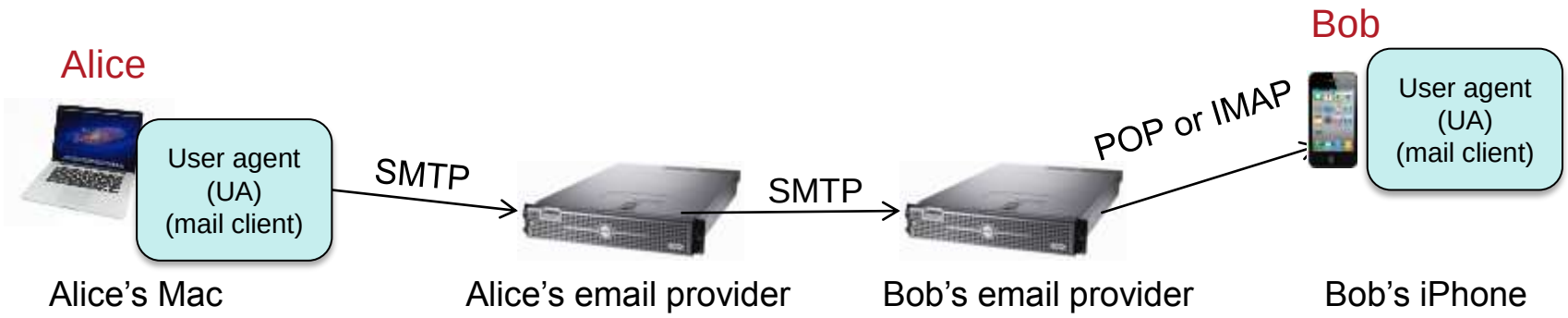
Small enhancements to the model

Mail delivery

- Alice will usually *not* be on the same machine as the local SMTP server
- Alice's UA is a email app running on her phone or laptop.
- It sends the message to her email provider's SMTP server using SMTP
- The email provider uses SMTP to talk to the destination server

Mail receipt

- Bob is usually *not* on the same machine as his mail server
- His mail program (UA) cannot access his mailbox directly
- Bob's UA needs to use a network mailbox access protocol such as IMAP or POP to get the message



Simple Mail Transfer Protocol (SMTP)

Three phases:

1. Handshake (greeting)
2. Transfer of message
3. Close

Command/response interaction

- The SMTP protocol is conversational text
  - A sequence of one-line messages & one-line responses
  - Then the message followed by a single line containing a period (.)
  - Finally, a QUIT command
- All transactions in 7-bit ASCII text
- Responses contain a status code & phrase (like HTTP and FTP)

The basic protocol

The following sequence of commands are used to send email

HELO cs.rutgers.edu	The client identifies itself as cs.rutgers.edu. This is often ignored now since the server may do a reverse DNS lookup on the IP address.
MAIL FROM: <pxk@cs.rutgers.edu>	Identify the address of the mail sender. Note that the domain here may be different from that in the HELO message.
RCPT TO: <testuser@pk.org> RCPT TO: <anotheruser@pk.org>	Identify the destination(s) for this message. You can have a list of these – one line per destination.
DATA	Now give the server the mail message. All the lines after this are the message. A line containing a single period ends it.
QUIT	We're done!

Sample Interaction

This is a sample interaction with me connecting to a Rutgers SMTP server via the telnet program and typing in SMTP commands. My typing is in blue.

```
$ telnet cs.rutgers.edu 25
Trying 128.6.4.2...
Connected to cs.rutgers.edu.
220 aramis.rutgers.edu ESMTP Sendmail 8.11.7p3+Sun/8.8.8; Tue, 9 Feb 2016 14:45:04 -0500 (EST)
HELO cs.rutgers.edu
250 aramis.rutgers.edu Hello aramis.rutgers.edu [128.6.4.2], pleased to meet you
MAIL FROM: <pxk@cs.rutgers.edu>
250 2.1.0 <pxk@cs.rutgers.edu>... Sender ok
RCPT TO: <testuser@pk.org>
250 2.1.5 <testuser@pk.org>... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
From: Paul Krzyzanowski <pxk@cs.rutgers.edu>
Subject: test message
Date: Tue, 9 Feb 2016 14:46:14 -0500
To: Whomever <testuser@pk.org>

Hi,
This is a test
.
250 2.0.0 r1BLxln29829 Message accepted for delivery
quit
221 2.0.0 aramis.rutgers.edu closing connection
```

This is the message body.
Headers may define the structure of the message but are ignored for delivery.

Comparison with HTTP

HTTP

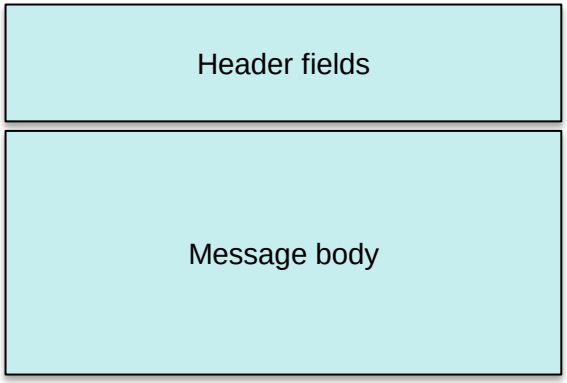
- Pull: you request content
- ASCII command/response interaction
  - Including status codes & messages
- Multiple objects
  - Each object (HTML, CSS, JavaScript, image files) is requested separately and encapsulated in its own message

SMTP

- Push: you provide content
- ASCII command/response interaction
  - Including status codes & messages
- Multiple objects
  - Multiple objects (attachments) are sent in one multipart message

Mail message format

- RFC 822 defines the basic format for text messages
- Header lines:
  - To:
  - From:
  - Subject:
  - Each line contains
    field_name: field_value
  - Terminated by a blank line
- Body
  - The actual message
- All this is treated as the message by SMTP
  - It's up to the user agents to interpret those headers



See RFC 822 for details

Mail headers versus data

- SMTP is interested in delivering the message
- The crucial data is in the “RCPT TO:” commands
- All mail headers are ignored by the SMTP protocol
  - From:, To:, Subject:, Cc:, Bcc:, etc.
  - These are strictly for the mail apps (user agents) to use
  - Mail is delivered exactly the same way whether a recipient is specified as a To, Cc, or Bcc:
    they will always end up as RCPT TO: commands in SMTP
- The User Agent can determine what to display

Problem: We want to send more than text

- Originally – we used email to just send plain text (ASCII)
- Later – we wanted to send:
  - Rich text or HTML text (formatted)
  - One or more Images
  - One or more attached files

Multimedia extensions: MIME

- MIME: multimedia mail extensions (RFC 2045, 2056)
- Lines in the message header define the content type

Type of data

Method used to encode the data

Version of the format

Other stuff, such as how to present the data and what the filename is

Base64 encoded data (allows us to send arbitrary binary data using 7-bit ascii

```
From: Paul Krzyzanowski <p@pk.org>
Content-Type: image/jpg
Content-Transfer-Encoding: base64
Subject: test mime
Date: Tue, 9 Feb 2016 13:09:50 -0500
To: Paul Krzyzanowski <p@pk.org>
Mime-Version: 1.0
Content-Disposition: inline; filename=test-
photo.jpg

/9j/4RZpRXhpZgAATU0AKgAAAAgADAEAAAMAAAABAcIAA
AEBAAMAAAABAUEAAAECAAMAAAADAAAA
ngEGAAMAAAABAAIAAAESAAMAAAABAAEAAAEVAAMAAAABA
AMAAAEaAAUAAAABAAAApAEbAAUAAAAB
```

Multimedia extensions: multipart MIME

Support multiple items in one message

Boundary definition

Start of content 1

Start of content 2

```
From: Paul Krzyzanowski <p@pk.org>
Content-Type: multipart/mixed;
 boundary="Split-96E43D94-57D4"
Subject: multipart mime demo
Date: Tue, 9 Feb 2016 13:09:50 -0500
To: Paul Krzyzanowski <p@pk.org>
Mime-Version: 1.0

--Split-96E43D94-57D4
Content-Disposition: INLINE;
    filename=test_image.jpg
Content-Type: IMAGE/JPG;
    name=test_image.jpg; x-unix-mode=0644
Content-Transfer-Encoding: BASE64

/9j/4RZpRXhpZgAAT .... <rest of content>

--Split-96E43D94-57D4
Content-Disposition: ATTACHMENT;
    filename=test.skp
Content-Type: APPLICATION/OCTET-STREAM;
    name=test.skp; x-unix-mode=0644
Content-Transfer-Encoding: BASE64

//7/DlMAawBl ... <rest of content>
```

Mail access protocols: POP & IMAP

Mail access

- SMTP deals with mail delivery (sending)
  - Sending messages to their destination
- When people ran mail apps on machines other than the mail server:
  - The app didn't have direct (file system) access to the mailbox
  - A protocol was needed to interact with the user's mailbox on the server
- Two protocols were developed
  - POP3: Post Office Protocol version 3 (RFC 1939)
  - IMAP: Internet Mail Access Protocol (RFC 1730)

POP3

- Client (mail application) connects to TCP port 110
- Three phases:
  1. Identification & authentication
     - Send user name and password
  2. Transaction
     - Mail access commands
  3. Update & exit

POP3: Authentication

- User login commands
  - `user` *`username`*
  - `pass` *`password`*

```
$ telnet pk.org 110
Trying 192.168.60.130...
Connected to pk.org.
+OK Dovecot ready.
user paul
+OK
pass mypassword
+OK Logged in.
```

- All commands are sent in plain text
  - This is not secure since internet traffic can be intercepted
  - It is a fundamental weakness here and with other protocols (HTTP, telnet, ftp)
  - POP3 is usually run over an encrypted session

POP3: Transaction

After authentication, the mail app (user agent) sends a series of commands to fetch or delete mail messages

- **stat** show the number of messages in the mailbox and total size

```
stat
+OK 3 5467
```

- **list** show a list of messages with the size of each message. A line containing a period indicates end of data

```
list
+OK Mailbox scan listing follows
1 1823
2 1825
3 1819
```

- **retr** retrieve a specific message

```
retr 2
+OK 1823 octets
--- all message headers and message
message content, including headers
.
```

POP3: Transaction

After authentication, the mail app (user agent) sends a series of commands to fetch or delete mail messages

- **dele** delete a specific message

```
Dele 1
+OK Message deleted
```

- **rset** reset the session: undo all deletes

```
rset
+OK Reset state
```

POP3: Update & exit

When done with the session, make changes permanent

- **quit** commit changes and exit

```
quit
+OK Goodbye
```

POP3 behaviors

- "download-and-delete" behavior
  - A mail client connects to a server
  - Download messages into its local mailbox
  - Delete them from the server
  - This isn't useful when you access mail from multiple devices
    - Once a message goes to a device, it's no longer on the server

- "download-and-keep" behavior
  - A mail client client connects to a server
  - Downloads messages but does not delete them
  - User may delete specific messages
  - A user can access messages from another device
  - But POP3 does not keep session state
    - It does not know if a user marked a message for deletion during a previous session

IMAP: Internet Message Access Protocol

- With POP3
  - Folders, moving messages, & search are all handled at the client with client downloads of the messages
- IMAP was created for users who access mail from multiple clients
  - IMAP keeps all messages on the server
  - User can organize messages in folders
  - State is stored on the server & available across sessions
    - Names of folders
    - Message ID mappings
    - Mark for deletion
  - Messages reside in folders
  - Commands allow users to
    - Create/delete folders, move messages between folders, search for specific messages, mark messages for deletion, delete messages, fetch headers only

IMAP Commands

- We won't cover the IMAP protocol
  - It's a lot uglier and a lot more verbose
  - See RFC 3501
- TCP connection
- All commands are sent as lines of ASCII text
  - Commands are more verbose – spelled out, not abbreviated
  - Each command is prefixed with a unique tag (unique per session)
  - A sequence of commands can be sent without waiting for a response before sending the next one

IMAP Protocol Example

- Login

```
a01 login paul mypassword
RESPONSE: a01 OK User logged in
```

- List folders

```
a02 list "Mail" "*"
* LIST (\NoSelect) "/" Mail
* LIST (\NoInferiors \Marked) "/" Mail/Trash
* LIST (\NoInferiors \Marked) "/" Mail/Sent
* LIST (\NoInferiors \UnMarked) "/" Mail/Drafts
* LIST (\NoSelect) "/" Mail/inbox
     <stuff deleted>
a02 OK LIST complete
```

- Fetch message 1 with full headers

```
a03 fetch 1 full
* 1 FETCH (FLAGS (\Seen) INTERNALDATE ”13-Feb-2013 14:46:22 -0500"
RFC822.SIZE 1
553 ENVELOPE ("Wed, 13 Feb 2013 14:46:22 -0400 “Test Message”
     <stuff deleted>
a03 OK FETCH completed
```

The end