CHAPTER

SMTP and Email 21

What You Will Learn

In this chapter, you will learn about the major architectures used to send and receive email on the Internet. We'll also see the five steps needed to send an email message.

You will learn about the protocols used with email applications, especially SMTP and POP3. We'll also describe MIME messages and discuss the important role of headers in email.

The Internet and TCP/IP are known to the greatest number of people through electronic mail (email) applications. Even those who cannot tell a router from a modem, or a packet from a frame, can check their email and send a message. A certain percentage of users still use the Internet mainly for email.

Email was one of the original applications the Internet was created to support (the others being file transfer and remote computer access). Things have come a long way since the original `mail` application, which is still supported on many Unix boxes:

```
>mail harry
We need to talk.
.
```

The modern email explosion has produced on-line ads, do-not-contact lists, spam, spam blockers, evil attachments, impounded attachments, and dozens of other moves and countermoves that make the email experience at once essential and yet daunting for many. Hardly anyone uses email except through a GUI today, and the mail user agents (MUAs)—the technical term for email client applications—are as varied as they are powerful, allowing users to schedule meetings, reserve conference rooms, or even request a projector for a certain time or place.

Email is a set of related and interconnected protocols that run on clients and servers to provide the global mesh of mailboxes and readers and writers upon which email depends. We'll look at several scenarios for sending and receiving email, using the devices on the network shown in Figure 21.1.

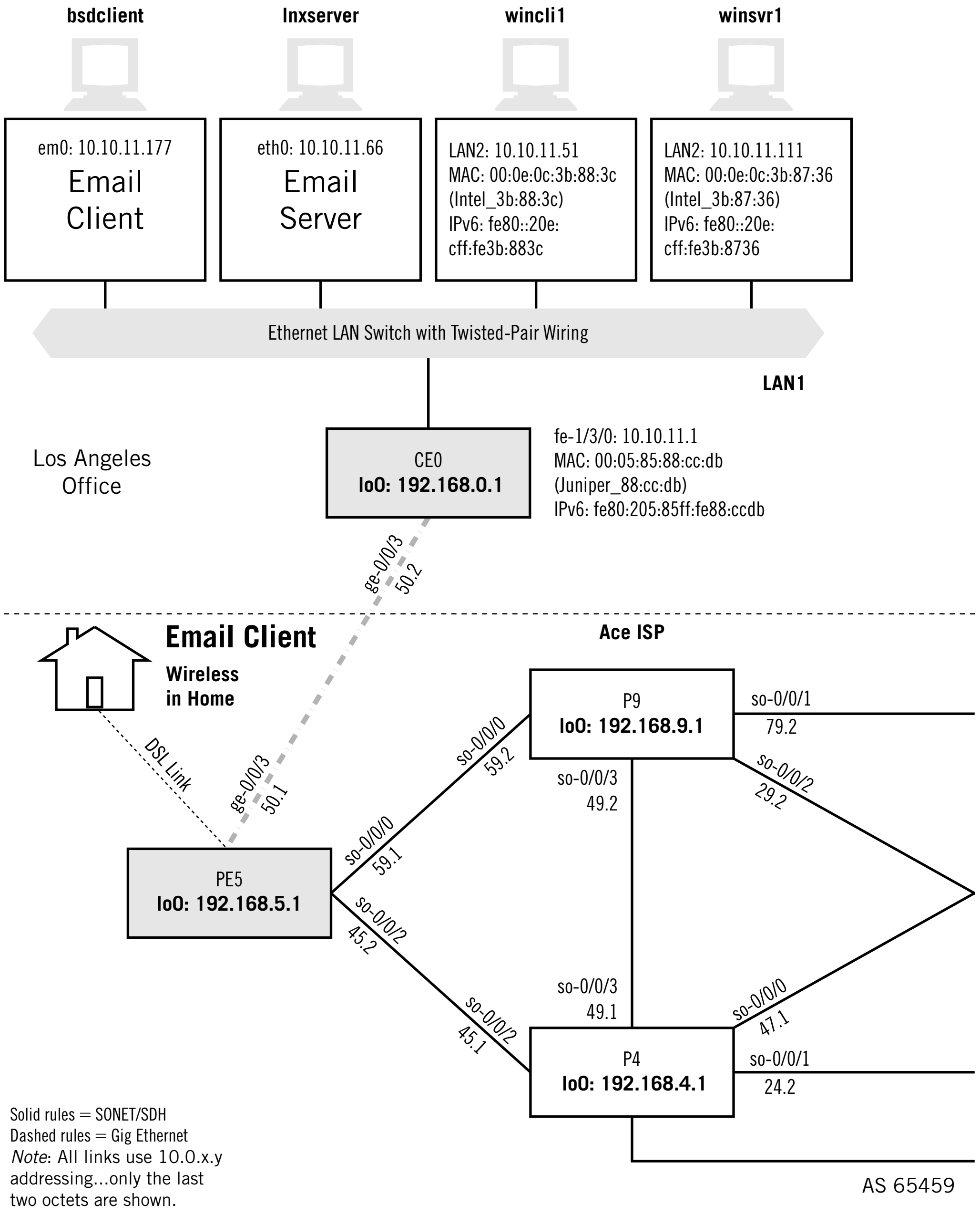
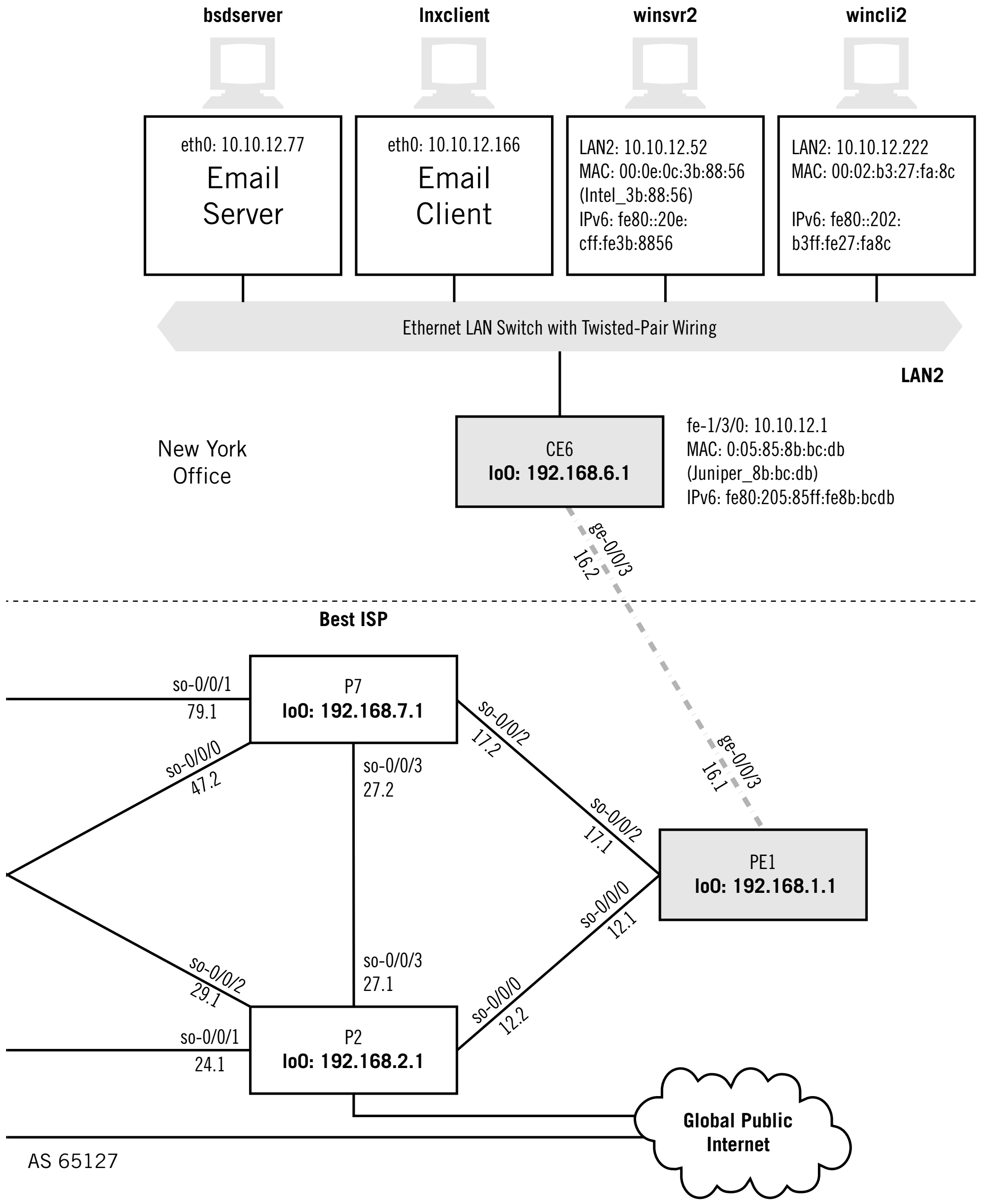


FIGURE 21.1

Email on the Illustrated Network, showing the Unix-based hosts used on email clients and servers.


bsdserver
lnxclient
winsvr2
wincli2
eth0: 10.10.12.77
Email Server
eth0: 10.10.12.166
Email Client
LAN2: 10.10.12.52
MAC: 00:0e:0c:3b:88:56
(Intel_3b:88:56)
IPv6: fe80::20e:cff:fe3b:8856
LAN2: 10.10.12.222
MAC: 00:02:b3:27:fa:8c
IPv6: fe80::202:b3ff:fe27:fa8c
Ethernet LAN Switch with Twisted-Pair Wiring
LAN2
New York Office
CE6
lo0: 192.168.6.1
fe-1/3/0: 10.10.12.1
MAC: 0:05:85:8b:bc:db
(Juniper_8b:bc:db)
IPv6: fe80:205:85ff:fe8b:bcdb
ge-0/0/3
16.2
Best ISP
so-0/0/1
79.1
P7
lo0: 192.168.7.1
so-0/0/2
17.2
ge-0/0/3
16.1
so-0/0/0
47.2
so-0/0/3
27.2
so-0/0/2
17.1
PE1
lo0: 192.168.1.1
so-0/0/0
12.1
so-0/0/2
29.1
so-0/0/3
27.1
so-0/0/0
12.2
so-0/0/1
24.1
P2
lo0: 192.168.2.1
Global Public Internet
AS 65127

In some examples, we'll use the Unix-based host systems as email clients and servers. We won't leave Windows out, however. We'll use the email client at the home to office to show that Windows Outlook works essentially the same as older email systems.

ARCHITECTURES FOR EMAIL

What needs to be added to the network to create the TCP/IP email system shown in the figure? It all depends on the overall architecture used to support email, and these have evolved through three distinct stages, all of which are still supported today. The final stage is the general email architecture for the Internet today, and that's what we will be exploring in this chapter. The three architectures are:

Single shared system—The shared system could be a mainframe or minicomputer that users access. The email administrator creates *mailboxes* (restricted access files on the local hard drive) where received messages are stored. A special user agent (UA) program creates the messages and stores them in the user's mailbox.

Shared systems connected by the Internet—The second architecture takes into account the fact that users might not share the same local system. Another piece was added to the email architecture: the message transfer agent (MTA). The UA still handles mailboxes and messages locally, whereas the MTA handles communications between the two systems in the usual client/server fashion.

Email clients and servers connected by the Internet—The final step is to realize that today most users are connected to their email servers by a LAN or WAN (dial-up or DSL) link. Because receivers are not always present (even on a LAN), users need the services of a message access agent (MAA) to retrieve their email from their local email server. The architecture of this final scenario is shown in Figure 21.2, between typical users we can call "Alice" and "Bob." The flow shown is from Alice to Bob, but when Bob replies to Alice the roles of client and server (as well as MTA and MAA) are reversed.

This architecture shows two systems dedicated to managing users' email mailboxes and delivering email. But how does the sender's email system know which device is acting as the receiver's email system? Today, special DNS records provide this information, but in the early days of the Internet *relaying* was used to deliver email. Email was routed from email system to email system in a fashion similar to forwarding packets. Today, most email travels over the Internet from an originator's email system directly to the recipient's, minimizing complexity and delay.

But email servers are not necessary for the TCP/IP email protocol, the Simple Mail Transfer Protocol (SMTP), to operate. We can still use the original and simple Unix built-in applications (`sendmail` and `mail`) to send and retrieve email from (for example)

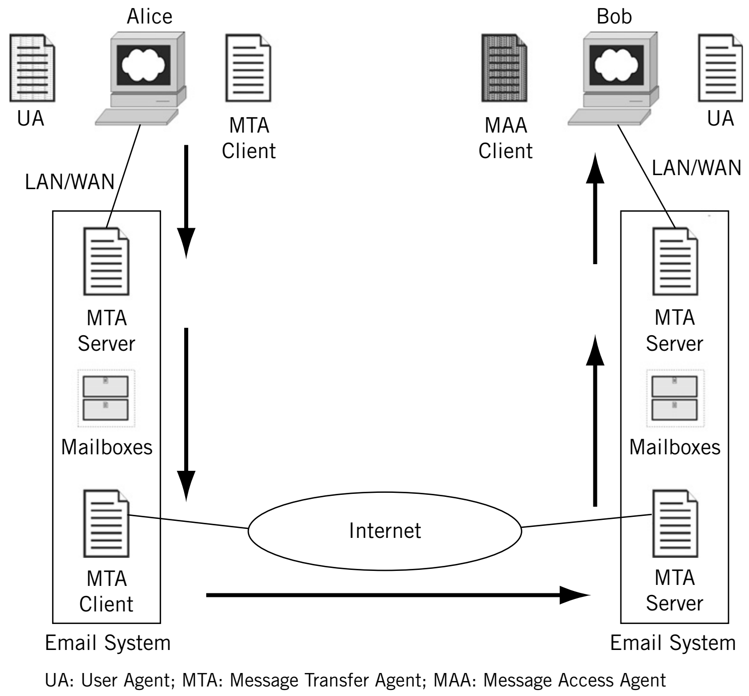


FIGURE 21.2

Email over the Internet, showing the role of client and server components.

`bsdserver` to `bsdclient`. It's nice to know that even today complex GUIs and massive directories are not needed to exchange email messages from the command prompt.

```
bsdserver# sendmail admin@bsdclient.booklab.englab.juniper.net
testing to 10.10.11.177
.
bsdserver#
```

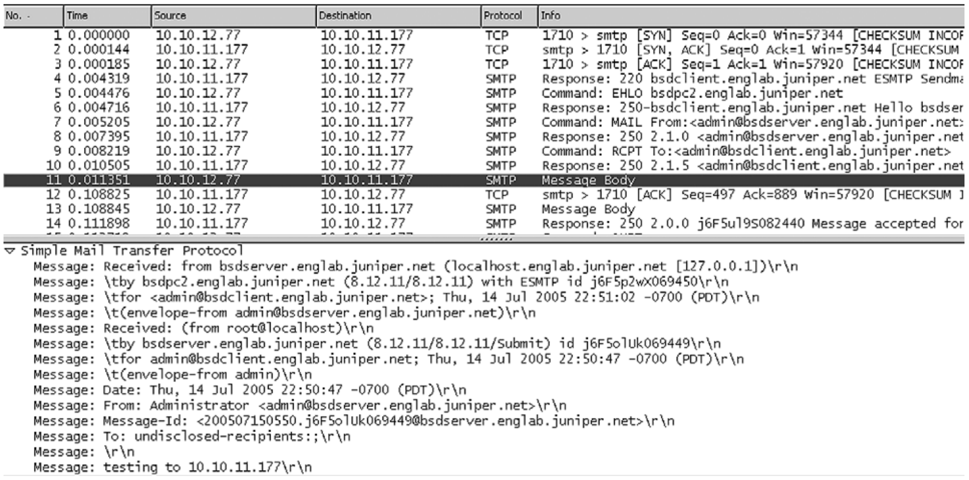
This email is going to the admin user on `bsdclient`. The text of the message is "`testing to 10.10.11.177`" and the text entry ends with a single period on a line by itself. Shown in the following is what happens at the receiver, starting with the prompt indicating that mail has arrived (the period does not appear in the received text).

```
You have new mail.
bsdclient# mail
Mail version 8.1 6/6/93.  Type ? for help.
"/var/mail/admin": 2 messages 1 unread
    1 admin@bsdserver.engl  Fri Jan 18 22:38  22/1153
U   2 admin@bsdserver.engl  Fri Jan 18 22:56  22/1162
& 2
```

```
Message 2:
From admin@bsdserver.booklab.englab.juniper.net Fri Jan 18 22:56:47 2008
Date: Fri, 18 Jan 2008 22:50:47 -0700 (PDT)
From: Administrator<admin @bsdserver.booklab.englab.juniper.net>
To: undisclosed-recipients:;

testing to 10.10.11.177

&
```



```
No. -  Time      Source         Destination    Protocol Info
     1 0.000000  10.10.12.77    10.10.11.177   TCP      1710 > smtp [SYN] Seq=0 Ack=0 Win=57344 [CHECKSUM INCOF
     2 0.000144  10.10.11.177   10.10.12.77    TCP      smtp > 1710 [SYN, ACK] Seq=0 Ack=1 Win=57344 [CHECKSUM
     3 0.000185  10.10.12.77    10.10.11.177   TCP      1710 > smtp [ACK] Seq=1 Ack=1 Win=57920 [CHECKSUM INCOF
     4 0.004319  10.10.11.177   10.10.12.77    SMTP     Response: 220 bsdclient.englab.juniper.net ESMTP Sendma
     5 0.004476  10.10.12.77    10.10.11.177   SMTP     Command: EHLO bsdpc2.englab.juniper.net
     6 0.004716  10.10.11.177   10.10.12.77    SMTP     Response: 250-bsdclient.englab.juniper.net Hello bsdser
     7 0.005205  10.10.12.77    10.10.11.177   SMTP     Command: MAIL From:<admin@bsdserver.englab.juniper.net:
     8 0.007395  10.10.11.177   10.10.12.77    SMTP     Response: 250 2.1.0 <admin@bsdserver.englab.juniper.net
     9 0.008219  10.10.12.77    10.10.11.177   SMTP     Command: RCPT To:<admin@bsdclient.englab.juniper.net>
    10 0.010505  10.10.11.177   10.10.12.77    SMTP     Response: 250 2.1.5 <admin@bsdclient.englab.juniper.net
    11 0.011351  10.10.12.77    10.10.11.177   SMTP     Message Body
    12 0.108825  10.10.11.177   10.10.12.77    TCP      smtp > 1710 [ACK] Seq=497 Ack=889 Win=57920 [CHECKSUM I
    13 0.108845  10.10.12.77    10.10.11.177   SMTP     Message Body
    14 0.111898  10.10.11.177   10.10.12.77    SMTP     Response: 250 2.0.0 j6F5ul9S082440 Message accepted for
▽ Simple Mail Transfer Protocol
    Message: Received: from bsdserver.englab.juniper.net (localhost.englab.juniper.net [127.0.0.1])\r\n
    Message: \tby bsdpc2.englab.juniper.net (8.12.11/8.12.11) with ESMTP id j6F5p2wX069450\r\n
    Message: \tfor <admin@bsdclient.englab.juniper.net>; Thu, 14 Jul 2005 22:51:02 -0700 (PDT)\r\n
    Message: \t(envelope-from admin@bsdserver.englab.juniper.net)\r\n
    Message: Received: (from root@localhost)\r\n
    Message: \tby bsdserver.englab.juniper.net (8.12.11/8.12.11/Submit) id j6F5olUk069449\r\n
    Message: \tfor admin@bsdclient.englab.juniper.net; Thu, 14 Jul 2005 22:50:47 -0700 (PDT)\r\n
    Message: \t(envelope-from admin)\r\n
    Message: Date: Thu, 14 Jul 2005 22:50:47 -0700 (PDT)\r\n
    Message: From: Administrator <admin@bsdserver.englab.juniper.net>\r\n
    Message: Message-Id: <200507150550.j6F5olUk069449@bsdserver.englab.juniper.net>\r\n
    Message: To: undisclosed-recipients:;\r\n
    Message: \r\n
    Message: testing to 10.10.11.177\r\n
```

FIGURE 21.3

Delivery of message using SMTP. Note the embedded control characters (starting with \) in the message body.

In this case, the mail was delivered directly from system to system. Only the SMTP MTA was used, with a minimal UA. Figure 21.3 shows the actual delivery of the message text itself. (Do not be concerned about the "undisclosed-recipients:" in the `To:` field. The `for` field in the message shows that the message is for the `admin` user on `bsdclient`.) Note that there is a lot more information carried in the message and displayed by the receiver than was entered by the sender. We'll talk more about these added email *headers* in detail later in this chapter.

Even when a complex GUI is used as an email front end, the same basic sequence of about 24 packets is used by SMTP to pass a small message off anywhere in the world. However, most people don't use the command prompt for this purpose. Modern email is more complex.

Sending Email Today

Today, there are five basic steps almost everyone uses to send and receive email. Although the procedures are absolutely symmetrical, and everyone is both sender and receiver when it comes to email, we'll follow a message one way from one person to another.

Email Message Composition

The user accesses a GUI email user agent (UA or sometimes MUA) to create the message. The email message contains two major parts: the *header* and the *body*. The header contains a series of fields that describes the message and controls how it is delivered and processed. The body of the message contains the actual information to be sent to the recipient. There can be multiple files accompanying the header and simple text of the message, and these are known as *attachments*. Most users do little more with the header than specify the email addresses of the intended recipients and subject line content. The UA takes care of making sure the entire message is in the correct standard format.

Submission of Email

When the user "sends" the newly created email, the sender's host (in a client role) does not need to set up a TCP connection directly to the receiver's host (in a server role). In fact, the user can compose a message and decide to submit it for delivery later, manually or automatically. Even when the message leaves the sender's host, the message is sent to the local email server using SMTP, and might sit there for a while rather than being forwarded across the Internet immediately. This allows for more efficient use of resources on the local email server. The server might require SMTP authentication of the user before accepting the message (we'll talk more about authentication later).

Delivery of Email

Once the local SMTP server has accepted the email message, the email server of the recipient(s) must be determined. DNS is used for this purpose, and the local email server performs a DNS query to access special Mail Exchanger (MX) records stored on a name server to provide this information. For example, an email sent to `walter@example.com` might be sent to a remote email server known as `pop3.example.com`. DNS provides both the name and IP address of this server.

SMTP also supports the ability to pass email messages through a specified sequence of SMTP servers to reach the destination. The intermediate servers are email *relay agents*. Relay agents are useful when a large organization has a single email server connected to the Internet (perhaps for ease of screening incoming messages) and yet has departments with their own email servers on each LAN. One way or another, the message makes its way to the destination email SMTP server that knows exactly who `walter@example.com` is. If the server cannot be contacted after a certain period of time, the mail is bounced back to the sender as undeliverable.

Email Processing

The receiving STMP server processes the incoming message, and if all seems well, places it into the recipient's mailbox. The message remains until the user retrieves it. If the recipient is unknown to the receiving server, the message is bounced back to the sender (also as undeliverable).

Email Access and Reading

The recipient's email application checks in periodically with the local SMTP server to see if any mail has arrived. This checking can be either automatic or when specifically run. If there is mail, the user can retrieve the mail, open it, and read it, and delete it. Usually, these are all separate steps. This step does not use SMTP, but a special mail access method and protocol such as POP3 or IMAP4 (both are used by TCP/IP MAAs).

All five of these steps are not always necessary. Some hosts act as mail servers all on their own, and the host-local-mail-server communication steps can be bypassed. Dial-in users often compose, send, and receive email all at once when they send mail. But usually all five steps are needed.

Four devices are involved in the five steps. They are the sender's client, the sender's local SMTP mail server, the recipient's local SMTP mail server, and the recipient's client. The relationship they have with one another and the protocols the email uses are shown in Figure 21.4. Note the symmetrical nature of the components so that two-way communication is possible.

Email Protocols

There are three common protocols used to deliver email over the Internet: the Simple Mail Transfer Protocol (SMTP), the Post Office Protocol (POP), and the Internet Message Access Protocol (IMAP). All three use TCP, and the last two are used for accessing electronic mailboxes. Special records stored in DNS servers play a role as well, using

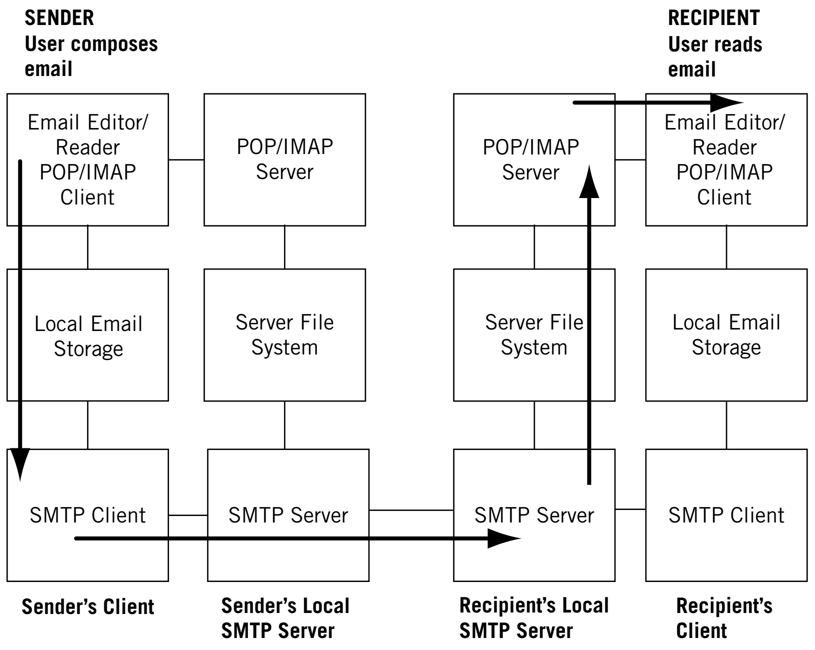


FIGURE 21.4

Email protocols and components, showing the components used to send an email message. Note the symmetrical nature of the sender and recipient so that the receiver can respond.

UDP. The current version of POP is version 3 (POP3) and the current version of IMAP is version 4 (IMAP4).

Although not a protocol, there is a series of Multipurpose Internet Mail Extensions (just MIME, never "MIMEs") for various types of email attachments (not just simple text). Finally, a number of related specifications add authentication to the basic email protocols. The way the protocols fit together is shown in Figure 21.5.

As we have seen, the original SMTP was designed as a simple host-to-host protocol. A user on one host created a message with a program called `sendmail` or `mail` and this text was sent directly to the destination host using SMTP as a Mail Transfer Agent (MTA). Of course, if the remote user was not running an email server process to accept the SMTP session, there was nothing for the sender to do but keep trying.

Modern email systems "decouple" the sender from the receiver so that email still goes through, even when the recipient is away for two weeks (but the messages keep piling up, just like regular mail). In addition, unlike almost every other TCP/IP application email operates *not* from host to host but from user to user. This means that users are not required to receive email on a particular host, nor is a particular host expected to have only one user with email capabilities. (We can even pick up email for a recipient from the *sending* host, and we'll do that later.) This user "mobility" poses special challenges for email addressing, which is why more than just a host name is required for correct email delivery.

The solution, of course, is to add another level to the hostname, this one identifying a particular user. So, for example, `walter@example.com` indicates a different mail destination than `goralski@example.com`. And, in fact, the actual host on which an email user is defined is not always added to the email address (which would yield something like `walter@bsdclient.example.com`). The email protocols all work together to make this work.

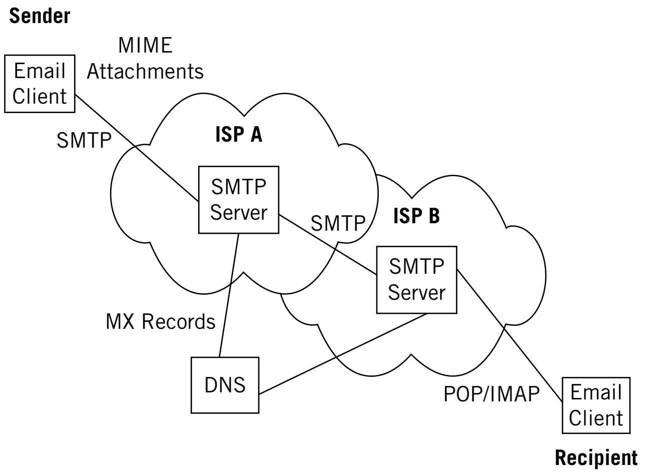


FIGURE 21.5

Email protocols, showing where they fit between sender and recipient.

There are older email address formats—FIDOnet, UUCP, email gateways (distinguished by the use of `user%` notations), and so on—but these are only of historical interest today. This is not to say that the evolution of email is not interesting, just that the history can be given very briefly and the discussion can turn to what is actually done with email on the Internet today.

The Evolution of Email in Brief

As expected with an application that has grown from a simple way to send text messages to an almost universal tool on the Internet, the email RFCs track a long evolutionary path as email changed with the times. In fact, email goes back to the days before TCP/IP and the Internet formally existed—all the way back to ARPAnet. Two very early documents, RFCs 95 and RFC 155, described physical mailing lists for distributing documents. Then the pioneers realized that the network itself could be used to distribute these documents, in the form of an electronic messaging application and associated protocols. In 1971, RFC 196 described the Mail Box Protocol for sending documents for remote printing.

By the mid-1970s, more sophisticated methods were developed, including some based on FTP. Today, the basic protocol for TCP/IP email is defined in RFC 821, and RFC 822 defines the format of the basic email message. RFC 974 added interactions with DNS to email transactions, and RFC 1869 added more capabilities as SMTP Service Extensions (ESMTP). Today, everyone still calls it SMTP, even when ESMTP is a more accurate term. Those same RFCs are still essentially in force today, although heavily added to in a number of ways and currently gathered as RFC 2821 and RFC 2822 (exactly 2000 away from the originals, an intentional numeration).

Email quickly grew to include various types of attachments, and modern users are used to these. RFCs 2045 through 2048 define basic MIME, which allows email to carry various types of email attachments. This series replaced RFCs 1521, 1522, and 1590, which had displaced RFC 1341.

Modern email protocols split the sending and retrieving task. The retrieval protocol POP3 has evolved through five RFCs, from RFC 1081 to RFC 1939. Another method, IMAP4 (often just IMAP), went from RFC 1730 to 2060.

Finally, RFC 2254 extended the SMTP authentication capabilities, and these were based on ESMTP in RFC 1869. Most modern SMTP applications support SMTP authentication, which defines an SMTP authentication server to advertise this function to SMTP clients. Today, the list of RFCs relating to MIME security (S/MIME) is a lengthy one and additional drafts are added all the time. And many RFCs address SMTP authentication.

SMTP Authentication

How do you know that the email you send goes only to the person intended? How do you know that the email you just got, supposedly from the president of your company, really came from that person? SMTP authentication was introduced to help prevent these email abuses, and others. It was based partly on ESTMP, and most implementations support SMTP authentication today. A lot of MUAs, which of course include the SMTP client, make it available. A server can support several forms of authentication, and the client application should pick one to use. The client can request a specific authentication method, but the server is free to reject its use.

SMTP authentication, which is advertised by an SMTP authentication server, requires clients to authenticate themselves, and both parties must mutually accept and support the chosen authentication procedure. Once successfully authenticated, the user can receive and send email.

Unfortunately, SMTP authentication does not fit very well into the SMTP protocol, mainly because it is based on the Simple Authentication and Security Layer (SASL) concept, which is more strictly aimed at direct client–server interactions. And several RFCs are needed to understand how it all works, some of which don't even mention any SMTP extensions, although they require use of the special ESMTP `EHLO` (Extended Hello) command.

The goal of SMTP authentication is to prevent username and password from crossing the network (the Internet) in plain text. A full discussion of STMP authentication depends on an understanding of how encryption provides authentication, topics which have not been covered yet. SMTP authentication is still evolving, and the mechanisms, methods, and procedures used will change as time goes on.

Simple Mail Transfer Protocol

A basic SMTP session between sender and local SMTP server is shown in Figure 21.6.

Like FTP, SMTP uses a system of client commands with parameters and numerical server responses, which is usually accompanied by some basic text as well. Oddly, if you know what you are doing, you can simply use a remote access method to connect to the SMTP server, and simply send the keywords and any parameters by typing them at the command prompt. The basic interaction between client and server when SMTP authentication is used is shown in Figure 21.7.

The client indicates to the server that it knows the server supports ESMTP (and wants to use it) with the SMTP `EHLO` command. The server offers a number of authentication schemes, including simple log-in with password. The client selects this option with the `AUTH` command. The server then uses *base64 encoding* (a special type of character coding) to ask the user for username and password, one at a time. The client replies are also encoded with base64, not encrypted. If the user types in the password incorrectly, the authentication fails, but the user can usually try again before the server drops the connection altogether.

The 11 most common SMTP commands are outlined in Table 21.1. A few others are defined, but they are hardly used anymore.

SMTP reply codes resemble FTP reply codes. The first digit refers to the command status, the second classifies the reply, and the third adds details. The meanings of the first two digits are outlined in Table 21.2.

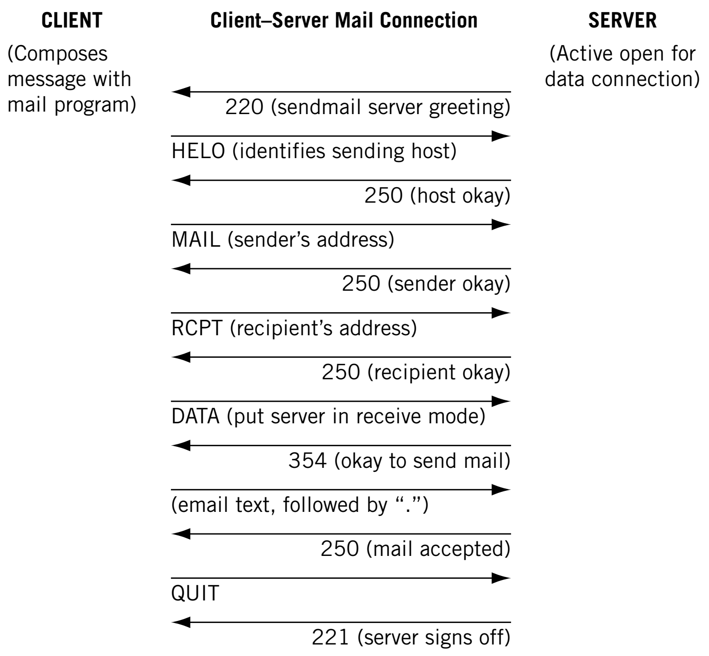

CLIENT
Client–Server Mail Connection
SERVER
(Composes message with mail program)
(Active open for data connection)
220 (sendmail server greeting)
HELO (identifies sending host)
250 (host okay)
MAIL (sender's address)
250 (sender okay)
RCPT (recipient's address)
250 (recipient okay)
DATA (put server in receive mode)
354 (okay to send mail)
(email text, followed by ".")
250 (mail accepted)
QUIT
221 (server signs off)

FIGURE 21.6

Basic STMP email exchange between a client and a server.

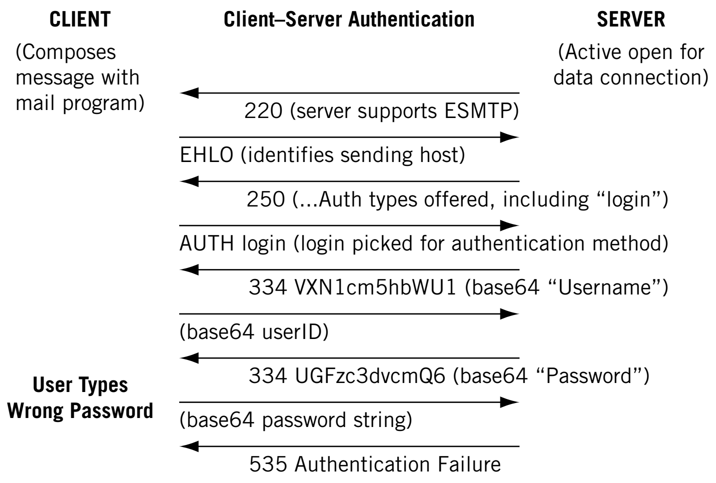

CLIENT
Client–Server Authentication
SERVER
(Composes message with mail program)
(Active open for data connection)
220 (server supports ESMTP)
EHLO (identifies sending host)
250 (...Auth types offered, including "login")
AUTH login (login picked for authentication method)
334 VXN1cm5hbWU1 (base64 "Username")
(base64 userID)
334 UGFzc3dvcmQ6 (base64 "Password")
User Types Wrong Password
(base64 password string)
535 Authentication Failure

FIGURE 21.7

SMTP authentication. Note that SMTP uses a special coding known as base64.

Table 21.1 Common SMTP Commands and Meanings

Command	Meaning
HELO	Identifies the sender to the receiver.
EHLO	Identifies the sender with extended capabilities to the receiver.
MAIL FROM	Identifies the originator and starts a mail transaction.
RCPT TO	Identifies an individual recipient. Repeated for multiple recipients. Receiver, if possible, checks for the validity of the recipient.
DATA	Sender is ready to transmit lines of text. Maximum line length is 1000 characters, including final "new line" character or characters.
RSET	Aborts current mail transaction and clears all information.
NOOP	Asks for a positive reply.
QUIT	Asks for a positive reply to close the connection.
VRFY	Asks the receiver to validate recipient name.
EXPN	Asks the receiver to confirm name in a mailing list, and for list content. For information only (do not change recipient names).
HELP	Asks for implementation details, such as commands supported.

Table 21.2 SMTP Reply Codes and Meanings

Digit and Position	Meaning
1xx	Positive preliminary (not currently used)
2xx	Positive completion
3xx	Positive intermediate result
4xx	Transient negative (okay to try again)
5xx	Permanent negative ("stop doing that!")
x0x	For a problem, syntax error, or unknown command
x1x	Information request reply (such as to HELP)
x2x	Connection reply
x3x	Unspecified
x4x	Unspecified
x5x	Receiver status reply

MULTIPURPOSE INTERNET MAIL EXTENSIONS

MIME is a rather dry subject, but quite important, if for no other reason than that MIME formats are also used in transfer using the protocol of the World Wide Web, the Hypertext Transport Protocol (HTTP), which is examined in the next chapter. So, MIME deserves at least a quick look here.

A MIME message has a set of headers and one or more "body parts." Internet text mail messages also have headers, of course, with fields such as `To:`, `From:`, and `Date:`. MIME messages have additional introductory headers to describe the overall format and content of the message.

MIME Media Types

When there are multiple parts to a MIME message, one introductory header defines a string used to mark the boundaries between parts. After the boundary delimiter, which is chosen by the email application, there are additional headers to describe the part of the MIME message that follows. The overall structure of the information in each part is determined by the `Content-Type` MIME headers. The type can be an image, audio, text, or even a mixture of these.

There are seven standard media types, all of which have a variety of subtypes. Five of them are considered "discrete" (meaning that the format is consistent throughout the part), and two are "composite," meaning that the format changes independently in each component. The discrete types are:

- Text
- Image
- Video
- Audio
- Application

The composite types are:

- *Multipart*—Each component can have a different data type, usually discrete.
- *Message*—Used to "encapsulate" other information, such as a forwarded email message.

Some of the more common subtypes used in these seven major data types are outlined in Table 21.3.

MIME Encoding

The data type and subtype establish the format of the content of a MIME body part. But how should the data in each part be represented for transmission across the Internet? MIME defines a variety of coding methods, allowing hosts and MTAs to be as flexible as possible.

The default coding method is ASCII (as used in the United States). If another method is used, such as for formatted documents, this must be announced in a MIME `Content-Transfer-Encoding` header.

There are six major MIME encoding methods. These are listed in Table 21.4. The `quoted-printable` encoding extends the usual 7-bit ASCII code set to allow a few extra characters. Special hex characters are preceded by an = sign. So, `0x0C` (form feed) is sent in quoted `=printable` as `= 0C`.

`Base64` encoding is very common today. SMTP was originally a text-based transmission system. Yet a lot of email content is sent as simple bytes, such as audio and video,

Table 21.3 MIME Content Types and Subtypes

Type	Subtypes
text	plain, richtext, tab-separated-values, html, sgml
image	jpeg, gif, ief, tiff, g3fax, png
video	mpeg, quicktime, vnd.vivo
audio	basic, 32kadpcm, vnd.vivo
application	octet-stream, postscript, rtf, pdf, zip, macwriteii, msword, remote-printing, EDI-X12, EDIFACT, dec-dx, dca-rft, activemessage, applefile, mac-binhex40, news-message-id, mews-transmission, wordperfect5.1, mathematica, pgp-encrypted, pgp-signature, pgp-keys, andrew-inset, slate, set-payment, set-registration, sgml, wita, lotus-wordpro, lotus-1-2-3, lotus-organizer, ms-excel, powerbuilder-6
multipart	mixed, alternative, digest, parallel, appledouble, header-set, form-data, report, voice-message, signed, encrypted
message	rfc822, partial, external-body, news, http, delivery-status

Table 21.4 MIME Encoding Methods and Meanings

Method	Meaning
7bit	Ordinary ASCII as used in the United States.
quoted-printable	Adds a few special characters and coding to ASCII text.
base64	Content is mapped into a “text” package (very common).
8bit	Similar to 7bit, but can include 8-bit characters.
binary	True binary data.
x-(name)	Experimental encodings must have a name starting with “x”.

and even as executable code (much to the chagrin of network administrators). `Base64` encoding converts a binary data stream to a sequence of “text” characters. This usually results in the size of the binary file growing by about 33% in terms of bytes. This is because 6 bits can indicate the numbers 0 through 63. But bytes are 8 bits, of course, at least where the Internet and TCP/IP are concerned.

An Example of a MIME Message

Consider a writer delivering a short story to an editor as an email attachment (been there, done that). What would the MIME headers that form the overall body of the email message look like? Well, they would resemble the following:

```
Content-Type: multipart/mixed;
     boundary = "--- = _NextPart_000_027HB582.0E7E0F6"
This is a message in MIME format.
--- = _NextPart 000_027HB582.0E7E0F6
Content-Type: text/plain
```

```
Please take a look at the attached short story. Thanks.

W
```

--- = **_NextPart_000_027HB582.0E7E0F6**
Content-Type: application/msword;
name = "new story.doc"
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
filename = "new story.doc"

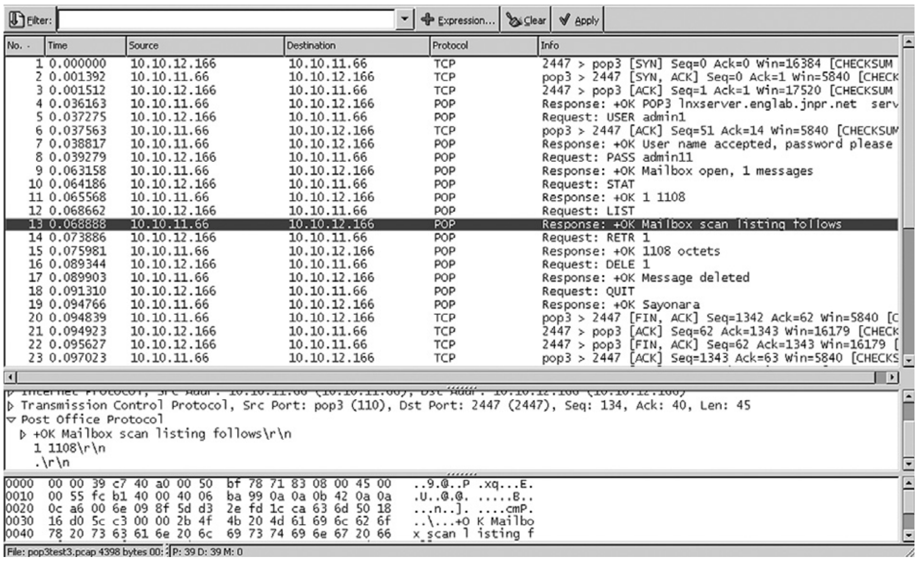
```
(Lots of nonsense characters form the base64 table.)
```

--- = **_NextPart_000_027HB582.0E7E0F6**

The lines in bold are the MIME headers.

USING POP3 TO ACCESS EMAIL

The original host-to-host SMTP did not allow for attachments, limited messages to 1000 bytes, was a purely connection-oriented application, and never imagined a world of personal computers and intermittent email checking. STMP was built for immediate email delivery to a specific host, sort of what we think of as instant messaging (IM) today. Email today is often delivered to mailboxes on mail servers, not directly to the end user, that is, users who might only have dial-up Internet access.



```
No. - Time        Source          Destination     Protocol  Info
    1 0.000000    10.10.12.166    10.10.11.66     TCP       2447 > pop3 [SYN] Seq=0 Ack=0 Win=16384 [CHECKSUM
    2 0.001392    10.10.11.66     10.10.12.166    TCP       pop3 > 2447 [SYN, ACK] Seq=0 Ack=1 Win=5840 [CHECK
    3 0.001512    10.10.12.166    10.10.11.66     TCP       2447 > pop3 [ACK] Seq=1 Ack=1 Win=17520 [CHECKSUM
    4 0.036163    10.10.11.66     10.10.12.166    POP       Response: +OK POP3 lnxserver.englab.jnpr.net  serv
    5 0.037275    10.10.12.166    10.10.11.66     POP       Request: USER admin1
    6 0.037563    10.10.11.66     10.10.12.166    TCP       pop3 > 2447 [ACK] Seq=51 Ack=14 Win=5840 [CHECKSUM
    7 0.038817    10.10.11.66     10.10.12.166    POP       Response: +OK User name accepted, password please
    8 0.039279    10.10.12.166    10.10.11.66     POP       Request: PASS admin11
    9 0.063158    10.10.11.66     10.10.12.166    POP       Response: +OK Mailbox open, 1 messages
   10 0.064186    10.10.12.166    10.10.11.66     POP       Request: STAT
   11 0.065568    10.10.11.66     10.10.12.166    POP       Response: +OK 1 1108
   12 0.068662    10.10.12.166    10.10.11.66     POP       Request: LIST
   13 0.068888    10.10.11.66     10.10.12.166    POP       Response: +OK Mailbox scan listing follows
   14 0.073886    10.10.12.166    10.10.11.66     POP       Request: RETR 1
   15 0.075981    10.10.11.66     10.10.12.166    POP       Response: +OK 1108 octets
   16 0.089344    10.10.12.166    10.10.11.66     POP       Request: DELE 1
   17 0.089903    10.10.11.66     10.10.12.166    POP       Response: +OK Message deleted
   18 0.091310    10.10.12.166    10.10.11.66     POP       Request: QUIT
   19 0.094766    10.10.11.66     10.10.12.166    POP       Response: +OK Sayonara
   20 0.094839    10.10.11.66     10.10.12.166    TCP       pop3 > 2447 [FIN, ACK] Seq=1342 Ack=62 Win=5840 [C
   21 0.094923    10.10.12.166    10.10.11.66     TCP       2447 > pop3 [ACK] Seq=62 Ack=1343 Win=16179 [CHECK
   22 0.095627    10.10.12.166    10.10.11.66     TCP       2447 > pop3 [FIN, ACK] Seq=62 Ack=1343 Win=16179 [
   23 0.097023    10.10.11.66     10.10.12.166    TCP       pop3 > 2447 [ACK] Seq=1343 Ack=63 Win=5840 [CHECKS

▷ Transmission Control Protocol, Src Port: pop3 (110), Dst Port: 2447 (2447), Seq: 134, Ack: 40, Len: 45
▽ Post Office Protocol
  ▷ +OK Mailbox scan listing follows\r\n
    1 1108\r\n
    .\r\n

0000  00 00 39 c7 40 a0 00 50  bf 78 71 83 08 00 45 00   ..9.@..P .xq...E.
0010  00 55 fc b1 40 00 40 06  ba 99 0a 0a 0b 42 0a 0a   .U..@.@. .....B..
0020  0c a6 00 6e 09 8f 5d d3  2e fd 1c ca 63 6d 50 18   ...n..]. ....cmP.
0030  16 d0 5c c3 00 00 2b 4f  4b 20 4d 61 69 6c 62 6f   ..\...+O K Mailbo
0040  78 20 73 63 61 6e 20 6c  69 73 74 69 6e 67 20 66   x scan l isting f

File: pop3test3.pcap 4398 bytes 00:  P: 39 D: 39 M: 0
```

FIGURE 21.8

A POP3 capture, highlighting how the email listing is sent to the user.

These intermittent Internet users log in and access their mailbox with POP3 (commonly just called POP). POP3 does not send email: SMTP does that. But POP3 retrieves the email, and the IMAP4 protocol maintains and controls access to the mailbox accounts.

POP3 uses TCP port 110, and users are authenticated by userID and password. POP3 then places a lock on the mailbox to avoid access conflicts. The POP3 server then enters transaction mode for user access to messages. POP3 features include the ability to view a list of email messages and their sizes and to selectively retrieve or delete messages, but many implementations simply dump all waiting mail to the client. POP3 servers can be the same device as the SMTP mail server, but this is not a requirement.

Let's add POP3 to our network. We used the BSD hosts before, so let's make `lnxserver` (`10.10.11.66`) into our email server for the network. We can then compose a fairly long (1108 bytes) message and send it to user `admin1`. Figure 21.8 shows the sequence of packets used to retrieve the message from host `lnxclient` (`10.10.12.166`).

POP3 employs a characteristic `+OK` and not a code when responding normally to a client. The series of packets shown in Figure 21.8 is boiled down to its POP3 essentials in Figure 21.9.

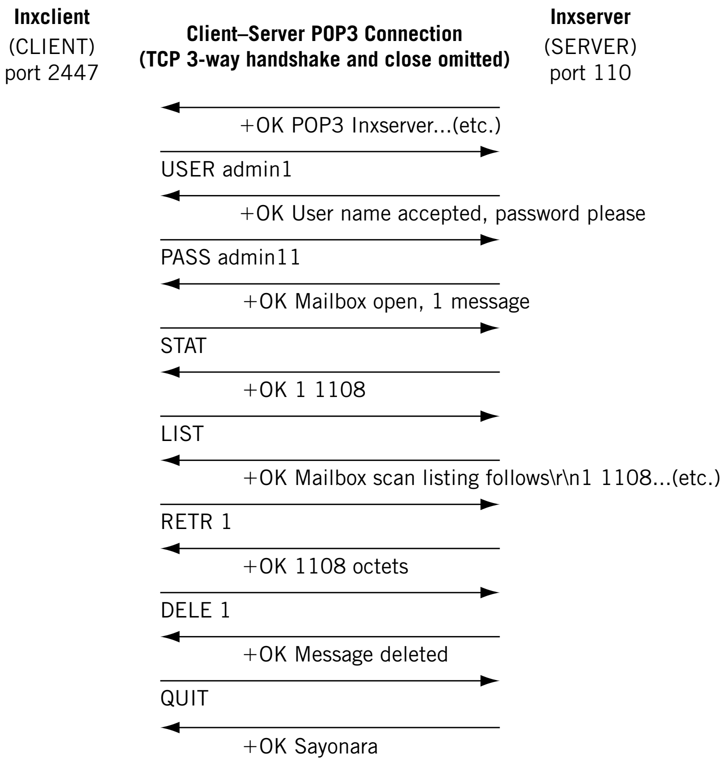

lnxclient
(CLIENT)
port 2447
Client–Server POP3 Connection
(TCP 3-way handshake and close omitted)
lnxserver
(SERVER)
port 110
+OK POP3 lnxserver...(etc.)
USER admin1
+OK User name accepted, password please
PASS admin11
+OK Mailbox open, 1 message
STAT
+OK 1 1108
LIST
+OK Mailbox scan listing follows\r\n1 1108...(etc.)
RETR 1
+OK 1108 octets
DELE 1
+OK Message deleted
QUIT
+OK Sayonara

FIGURE 21.9

A POP3 connection used to fetch email, showing a more schematic view than the capture.

Note that the retrieval of the message (RETR) by the client and its deletion from the server (DELE) are separate steps. You don't have to delete email as you read it, of course. The `+OK Sayonara` is also part of the POP3 protocol implementation.

HEADERS AND EMAIL

We've mentioned email headers already and supplied some details about MIME headers (header extensions). Email has its own proper set of headers as well, and an Internet email message is little more than a sequence of headers and their values, one after the other, from the start of the email message to the end. Table 21.5 outlines the basic email header field names and groups established by RFC 822.

Now we have everything in place to examine the headers created when sending a short email message through our email server (`lnxserver`) from a client host to another user. We'll use the `admin` account on `lnxclient` to send a message to the `admin` user on

Table 21.5 RFC 822 Email Header Fields and Characteristics

Field Group	Field Name	Appearance	Occurrences per Message	Comment
Destination Address Field	To:	Usually present	1	Primary recipient list
	Cc:	Optional	1	Copy recipient
	Bcc:	Optional	1	"Blind" copy
Identification Fields	Message-ID:	Usually present	1	Unique code applied when sent
	In-Reply-To:	Optional, normal for replies	1	Provides method to coordinate responses
	References:	Optional	1	Other documents or message IDs
Informational Fields	Subject:	Usually present	1	Topic of the message
	Comments:	Optional	Unlimited	Describe message
	Keywords:	Optional	Unlimited	Useful search item
Origination Date	Date:	Mandatory	1	Date and time stamp for mail
Originator Fields	From:	Mandatory	1	Source address of "originator"
	Sender:	Optional	1	If different from "originator"
	Reply-To:	Optional	1	If absent, reply goes to "from"

Table 21.5 (*continued*)

Field Group	Field Name	Appearance	Occurrences per Message	Comment
Resent Fields	Resent-Date: Resent-From: Resent-Sender: Resent-To: Resent-Cc: Resent-Bcc: Resent-Message-ID:	Each time message is resent, this block is generated	Resent-Date: and Resent-Sender: are mandatory; all others optional	Special, used for forwarding an email message to others
Trace Fields	Received: Return-Path:	Inserted by email system	Unlimited	Used to trace the message through the email system

`lnxserver` (these are not necessarily the same users: they just share a mailbox name). Then we'll fetch the message from the email server mailbox using the `admin` account, showing that we can fetch our email almost anywhere, even from the sending host.

We can use the same basic `mail` program as we did on the BSD hosts. This time, we'll use the `-s` flag to create a subject for the message. The text is simple, and we end our message with a single dot as before.

```
[admin@lnxclient admin]$ mail –s "Here is another example"
        admin@lnxserver.booklab.englab.juniper.net

This is text...
.
Cc: (enter)
```

Now we'll use fetchmail to "fetch" the mail message with POP3 from the email server (`lnxserver`) and bring it back to `lnxclient`. Note that when we run the program and have email we get a version of the familiar "you've got mail" prompt.

```
[admin@lnxclient admin]$ fetchmail
Enter password for admin@lnxserver.booklab.englab.juniper.net: (not shown)
You have new mail in /var/spool/mail/admin
```

Usually, our complete email application would display the information and the message. But there's nothing magical about that. We can do the same with the command prompt, listing the mailbox content and displaying the email message with normal Unix commands.

```
[admin@lnxclient admin]$ ls –l /var/spool/mail/admin
-rw-------  1 admin    mail        3122 Jan 17 16:42 /var/spool/mail/admin
```

```
[admin@lnxclient admin]$ cat /var/spool/mail/admin
From admin@lnxserver.booklab.englab.juniper.net  Wed Jan 16 13:04:50 2008
Return-Path: <admin@lnxclient.booklab.englab.juniper.net>
Received: from localhost (localhost.localdomain [127.0.0.1])
        by lnxclient.booklab.englab.juniper.net (8.12.9/8.12.8) with ESMTP id
        jBGL4onD026830
        for <admin@localhost>; Wed, 16 Jan 2008 13:04:50 -0800
Received: from lnxserver.booklab.englab.juniper.net
        by localhost with POP3 (fetchmail-6.2.0)
        for admin@localhost (single-drop); Wed, 16 Jan 2008 13:04:50 -0800 (PST)
Received: from lnxclient.booklab.englab.juniper.net ([10.10.12.166])
        by lnxserver.booklab.englab.juniper.net (8.12.8/8.12.8) with ESMTP id
        jBGL4HFa027257
        for <admin@lnxserver.booklab.englab.juniper.net>; Wed, 16 Jan 2008
        13:04:17 -0800 (PST)
Received: from lnxclient.booklab.englab.juniper.net (localhost.localdomain
        [127.0.0.1])
        by lnxclient.booklab.englab.juniper.net (8.12.8/8.12.8) with ESMTP id
        jBGL4HnD026820
        for <admin@lnxserver.booklab.englab.juniper.net>; Wed, 16 Jan 2008
        13:04:17 -0800
Received: (from admin@localhost)
        by lnxclient.booklab.englab.juniper.net (8.12.8/8.12.8/Submit) id
        jBGL4HHf026818
        for admin@lnxserver.booklab.englab.juniper.net; Wed, 16 Jan 2008
        13:04:17 -0800
Date: Wed, 16 Jan 2008 13:04:17 -0800
From: admin@lnxclient.booklab.englab.juniper.net
Message-Id: <200801172104.jBGL4HHf-26818 @lnxclient.booklab.englab.juniper.net>
To: admin@lnxserver.booklab.englab.juniper.net
Subject: Here is another example
X-IMAPbase: 1134766876 8
Status: o
X-UID: 8
X-Keywords:

This is text...
```

The important fields are highlighted. Most of the other headers were added when the email was created, of course. Most useful is the series of `Received:` headers, which allows us to trace the message back to its origin. It might seem odd that there are *five* receiver headers along the trace for a message that has gone from client to email server and then back to client. But the application adds a `localhost` step at each end, at the sender (`admin@localhost`) and receiver (`from localhost`) to the message trace. The complete path of the message recorded in the headers (from "bottom to top") is:

1. The mail application receives the composed message from the local user.
2. The local mailbox receives the message using ESMTP.
3. The email server receives the message using ESMTP.

4. The other client retrieves the message from the email server using POP3 (`fetchmail`).
5. The local host transfers the message to the local mailbox using ESMTP.
6. The use of these protocols is highlighted in the headers.

HOME OFFICE EMAIL

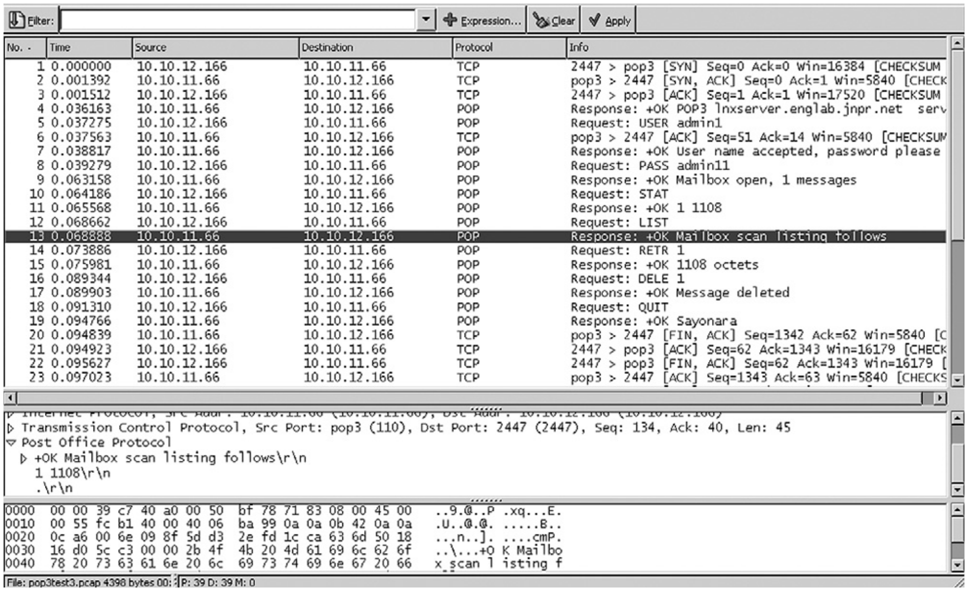
Let's end our email discussion by showing that Windows uses the same protocols and headers to send and receive email over the Internet. This time, we'll send a message from `lnxclient` on the Illustrated Network to my home office host (which uses Outlook).

Almost all email applications have an option to view the complete headers. In Outlook, it's just "Message Header" in the singular, but the following is the result of viewing the message headers in Outlook. Only the headers are displayed, not the message text itself.

```
Microsoft Mail Internet Headers Version 2.0
Received: from beta.jnpr.net ([172.24.18.109]) by positron.jnpr.net with
    Microsoft SMTPSVC(5.0.2195.6713);
    Thu, 17 Jan 2008 07:37:14 -0700
Received: from merlot.juniper.net ([172.17.27.10]) by beta.jnpr.net over TLS
    secured channel with Microsoft SMTPSVC(6.0.3790.1830);
    Thu, 17 Jan 2008 07:37:13 -0700
Received: from lnxclient.englab.juniper.net (lnxclient.englab.juniper.net
   [10.10.12.166])
   by merlot.juniper.net (8.11.3/8.11.3) with ESMTP id k9JEbDH15244
   for <walterg@juniper.net>; Thu, 17 Jan 2008 07:37:13 -0700 (PDT)
   (envelope-from admin@lnxclient.englab.juniper.net)
Received: from lnxclient.englab.juniper.net (localhost.localdomain
   [127.0.0.1])
   by lnxclient.englab.juniper.net (8.12.8/8.12.8) with ESMTP id
   k9JEacUg026193
   for <walterg@juniper.net>; Thu, 17 Jan 2008 07:36:58 -0700
   Received: (from admin@localhost)
   by lnxclient.englab.juniper.net (8.12.8/8.12.8/Submit) id k9JEaSlp026191
   for walterg@juniper.net; Thu, 17 Jan 2008 07:36:28 -0700
Date: Thu, 17 Jan 2008 07:36:28 -0700
From: admin@lnxclient.englab.juniper.net
Message-Id: <200801171436.k9JEaSlp026191@lnxclient.englab.juniper.net>
To: walterg@juniper.net
Subject: here is an email example
Return-Path: admin@lnxclient.englab.juniper.net
X-OriginalArrivalTime: 17 Jan 2008 14:37:13.0230 (UTC) FILETIME=[10F80AE0:
  01C6F38C]
```

QUESTIONS FOR READERS

Figure 21.10 shows some of the concepts discussed in this chapter and can be used to answer the following questions.



```
No. -  Time      Source         Destination    Protocol  Info
     1 0.000000  10.10.12.166   10.10.11.66    TCP       2447 > pop3 [SYN] Seq=0 Ack=0 Win=16384 [CHECKSUM
     2 0.001392  10.10.11.66    10.10.12.166   TCP       pop3 > 2447 [SYN, ACK] Seq=0 Ack=1 Win=5840 [CHECK
     3 0.001512  10.10.12.166   10.10.11.66    TCP       2447 > pop3 [ACK] Seq=1 Ack=1 Win=17520 [CHECKSUM
     4 0.036163  10.10.11.66    10.10.12.166   POP       Response: +OK POP3 lnxserver.englab.jnpr.net  serv
     5 0.037275  10.10.12.166   10.10.11.66    POP       Request: USER admin1
     6 0.037563  10.10.11.66    10.10.12.166   TCP       pop3 > 2447 [ACK] Seq=51 Ack=14 Win=5840 [CHECKSUM
     7 0.038817  10.10.11.66    10.10.12.166   POP       Response: +OK User name accepted, password please
     8 0.039279  10.10.12.166   10.10.11.66    POP       Request: PASS admin11
     9 0.063158  10.10.11.66    10.10.12.166   POP       Response: +OK Mailbox open, 1 messages
    10 0.064186  10.10.12.166   10.10.11.66    POP       Request: STAT
    11 0.065568  10.10.11.66    10.10.12.166   POP       Response: +OK 1 1108
    12 0.068662  10.10.12.166   10.10.11.66    POP       Request: LIST
    13 0.068888  10.10.11.66    10.10.12.166   POP       Response: +OK Mailbox scan listing follows
    14 0.073886  10.10.12.166   10.10.11.66    POP       Request: RETR 1
    15 0.075981  10.10.11.66    10.10.12.166   POP       Response: +OK 1108 octets
    16 0.089344  10.10.12.166   10.10.11.66    POP       Request: DELE 1
    17 0.089903  10.10.11.66    10.10.12.166   POP       Response: +OK Message deleted
    18 0.091310  10.10.12.166   10.10.11.66    POP       Request: QUIT
    19 0.094766  10.10.11.66    10.10.12.166   POP       Response: +OK Sayonara
    20 0.094839  10.10.11.66    10.10.12.166   TCP       pop3 > 2447 [FIN, ACK] Seq=1342 Ack=62 Win=5840 [C
    21 0.094923  10.10.12.166   10.10.11.66    TCP       2447 > pop3 [ACK] Seq=62 Ack=1343 Win=16179 [CHECK
    22 0.095627  10.10.12.166   10.10.11.66    TCP       2447 > pop3 [FIN, ACK] Seq=62 Ack=1343 Win=16179 [
    23 0.097023  10.10.11.66    10.10.12.166   TCP       pop3 > 2447 [ACK] Seq=1343 Ack=63 Win=5840 [CHECKS
```

```
▷ Internet Protocol, Src Addr: 10.10.11.66 (10.10.11.66), Dst Addr: 10.10.12.166 (10.10.12.166)
▷ Transmission Control Protocol, Src Port: pop3 (110), Dst Port: 2447 (2447), Seq: 134, Ack: 40, Len: 45
▽ Post Office Protocol
  ▷ +OK Mailbox scan listing follows\r\n
    1 1108\r\n
    .\r\n
```

```
0000   00 00 39 c7 40 a0 00 50  bf 78 71 83 08 00 45 00   ..9.@..P .xq...E.
0010   00 55 fc b1 40 00 40 06  ba 99 0a 0a 0b 42 0a 0a   .U..@.@. .....B..
0020   0c a6 00 6e 09 8f 5d d3  2e fd 1c ca 63 6d 50 18   ...n..]. ....cmP.
0030   16 d0 5c c3 00 00 2b 4f  4b 20 4d 61 69 6c 62 6f   ..\...+O K Mailbo
0040   78 20 73 63 61 6e 20 6c  69 73 74 69 6e 67 20 66   x scan l isting f
```

File: pop3test3.pcap 4398 bytes 00: P: 39 D: 39 M: 0

FIGURE 21.10

POP3 session capture.

1. Which port does POP3 use?
2. Which password is provided by the user?
3. Was the email message deleted after it was retrieved?
4. How long was the message?
5. How many other messages are in the user's mailbox?