

Email

TCP/IP Electronic Mail Message Communication Model and Device and Protocol Roles

The purpose of the electronic mail system as a whole is to accomplish the transmission of electronic mail messages from a user of a TCP/IP internetwork to one or more recipients. In order to accomplish this, a special method of communication is required that makes the electronic mail system quite different from that used by most other protocols. To understand what I mean by this, you need only recognize the difference in communication between sending a letter and making a phone call.

Most TCP/IP protocols are analogous to making a phone call in this respect: the sender and the receiver must both be on the network at the same time. You can't call someone and talk to them if they aren't around to answer the phone. (Yeah, yeah, answering machines and voice mail. Stop interrupting, will you? ☺) Most TCP/IP protocols are like this: to send a file using [FTP](#), for example, you must make a direct connection from the sender's machine to the recipient's machine. If the recipient's machine is not on the network at the exact time that sender's is, no communication is possible.

For electronic mail, this type of communication is simply unacceptable. Electronic mail is like its real-world counterpart; Joe wants to be able to put a message into the system at a time that is convenient for him, and Ellen wants to be able to receive Joe's mail at a time that works for her. For this to work, e-mail must use a "send and forget" model, just like real mail, where Joe drops the "envelope" into the e-mail system and it eventually gets to its destination.

This *decoupling* of the sender and receiver is critical to the design of the electronic mail system. This is especially true because many of the users of Internet electronic mail are not on the Internet all the time. Just as you wouldn't want real mail to be rejected if it arrived when you are not home, you don't want electronic mail to not be delivered if you are not on the Internet when it arrives. Similarly, you may not want to be connected to the Internet for the entire time it takes to write a message, especially if you have access to the Internet for only a limited amount of time each day.

Also critical to the entire electronic mail system is that idea that communication is between specific **users**, and not between particular machines. This makes e-mail inherently different than many other types of communication on TCP/IP

internetworks. We'll see more of why this is important when we look at [e-mail addressing](#).

The Delayed Delivery Model and Device Roles

To allow the type of communication needed for electronic mail, the entire system is designed to facilitate the *delayed delivery* of e-mail messages from one user to another. To see how this works, let's look again at the example communication we discussed in [the previous topic](#), but this time, considering the roles of the different **devices** in the exchange (as shown in Figure 301):

- **Sender's Client Host:** The sender of the mail composes an electronic mail message, generally using a mail client program on his or her local machine. The mail, once composed, is not immediately sent out over the Internet; it is held in a buffer area called a *spool*. This allows the user to be “detached” for the entire time that a number of outgoing messages are created. When the user is done, all of the messages can be sent at once.
- **Sender's Local SMTP Server:** When the user's mail is ready to be sent, he or she connects to the internetwork. The messages are then communicated to the user's designated local [SMTP](#) server, normally run by the user's Internet Service Provider (ISP). The mail is sent from the client machine to the local SMTP server using SMTP. (It is possible in some cases for the sender to be working directly on a device with a local SMTP server, in which case sending is simplified.)
- **Recipient's Local SMTP Server:** The sender's SMTP server sends the e-mail using SMTP to the recipient's local SMTP server over the Internetwork. There, the e-mail is placed into the recipient's incoming mailbox (*inbox*). This is comparable to the outgoing spool that existed on the sender's client machine. It allows the recipient to accumulate mail from many sources over a period of time, and retrieve them when it is convenient.
- **Recipient's Client Host:** In certain cases the recipient may access his or her mailboxes directly on the local SMTP server. More often, however, a mail access and retrieval protocol, such as POP3 or IMAP, is used to read the mail from the SMTP server and display it on the recipient's local machine. There, it is displayed using an e-mail client program, similar to the one the sender used to compose the message in the first place.

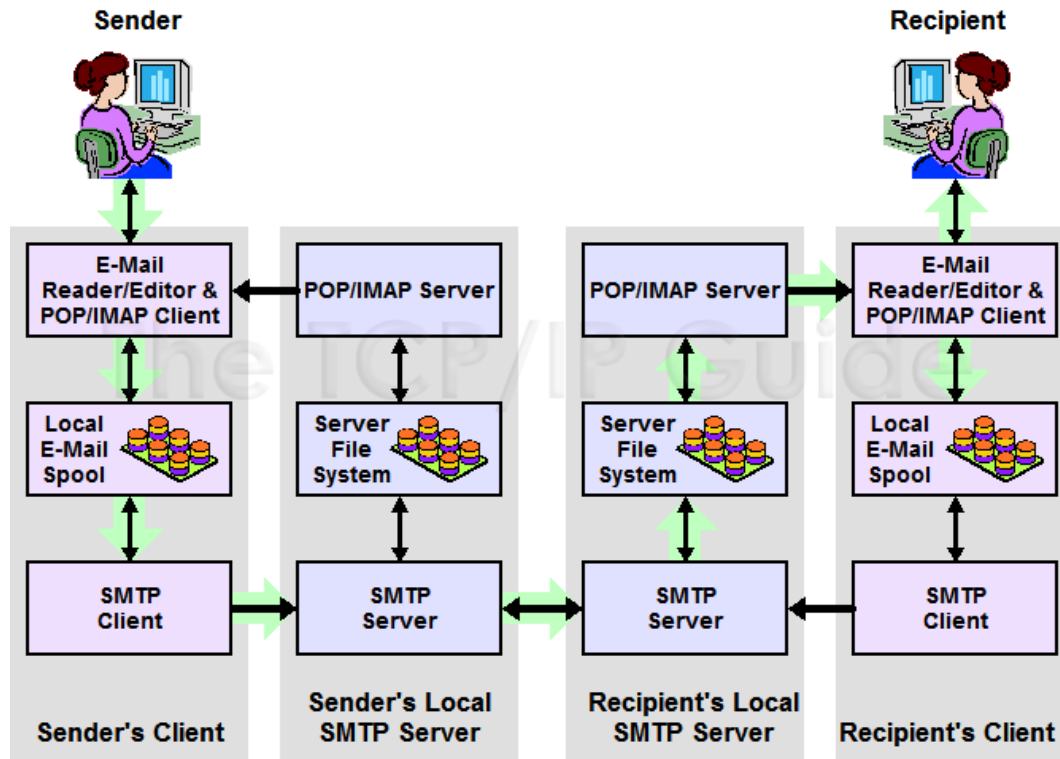


Figure 301: Electronic Mail (E-Mail) Communication Model

This diagram shows the four devices that are involved in a typical e-mail communication between two users. (Yes, they *are* identical twins, imagine that! ☺) Each device consists of a number of different elements, which communicate as indicated by the black arrows. Note the inherent asymmetry, because the method used to send an e-mail from a user is not the same as that used to retrieve it from the server. The large green arrows show a typical transaction: the sender composes mail and it goes to her local e-mail spool. It is sent to the sender's local SMTP server using SMTP, and then to the recipient's SMTP server, where it goes into that user's inbox. It is then retrieved, usually using a protocol such as POP or IMAP.

Protocol Roles In Electronic Mail Communication

You may have noticed that SMTP is used for most of this communication process. In fact, if the recipient uses a machine that runs SMTP software, which is common for those using dialup UNIX shell Internet access, the process of sending e-mail uses SMTP exclusively. SMTP servers must, however, always be available on the Internet and ready to accept mail. Most people access the

internetwork using devices that aren't always online or that don't run SMTP software. That is why the last step, mail access and retrieval, is usually required.

It might have been possible to define the electronic mail system so that this last step of communication was carried out using SMTP as well, which would mean the entire system used the same protocol. However, SMTP was tailored for the specific purpose of transporting and delivering e-mail, not for remote mailbox access. It made more sense to leave the function of mailbox access to dedicated, separate protocols. This not only allows these protocols to be tailored to the needs of e-mail recipients, but provides flexibility by giving users more than one option for how e-mail is retrieved. I discuss mail access protocols and methods in a [separate section of the Guide](#), highlighting the two most common protocols: the [Post Office Protocol \(POP\)](#) and the [Internet Message Access Protocol \(IMAP\)](#).

The three protocols discussed above—SMTP, POP3 and IMAP—get the “lead billing” on the TCP/IP electronic mail stage, but they rely on two other elements to play “supporting roles”. The first is a [method of addressing e-mail messages](#) to ensure that they arrive at their destinations. The second is the set of [message formats](#) used to encode messages and control how they are delivered and used. These don't usually get as much attention as they deserve, but they do here, as I have devoted the next two sections to them.



Key Concept: One of the critical requirements of an electronic mail system is that the sender and receiver of a message not be required to both be on the system at the time mail is sent. TCP/IP therefore uses a communication model with several devices that allow the sender and recipient to be *decoupled*. The sender's client device spools mail and moves it to the sender's local SMTP server when it is ready for transmission; the e-mail is then transmitted to the receiver's SMTP server using SMTP. The e-mail can remain on the recipient's server for an indefinite period of time. When the recipient is ready to read it, he or she retrieves it using one or more of a set of mail access protocols and methods, the two most popular of which are POP and IMAP.

SMTP Mail Transaction Process

The delivery of e-mail message begins with the [establishment of an SMTP session](#) between the devices sending and receiving the message. The SMTP sender initiates a TCP connection to the SMTP receiver, and then sends a *HELO* or *EHLO* command, to which the receiver responds. Assuming there are no problems, the session is then established and ready for actual e-mail message transactions.

SMTP Mail Transaction Overview

The SMTP mail transaction process itself consists of three steps:

1. **Transaction Initiation and Sender Identification:** The SMTP sender tells the SMTP receiver that it wants to start sending a message, and gives the receiver the e-mail address of the message's originator.
2. **Recipient Identification:** The sender tells the receiver the e-mail address(es) of the intended recipients of the message.
3. **Mail Transfer:** The sender transfers the e-mail message to the receiver. This is a complete e-mail message meeting the [RFC 822 specification](#) (which may be in [MIME](#) format as well).

That's it! So you can see that the word “Simple” in “Simple Mail Transfer Protocol” definitely has at least **some** merit. Especially when compared with other protocols that claim to be simple, such as [SNMP](#). ☺

The Rationale for A Separate E-Mail Message and Envelope

In fact, one question that sometimes comes up when examining SMTP is why couldn't this process be even **simpler**? The first two steps identify the sender of the e-mail and the intended recipient(s). But all of this information is already contained in headers in the message itself. Why doesn't SMTP just read that information from the message, which would in fact make the mail transaction a *one-step* process?

The explanation for this isn't specifically addressed in the SMTP standards, but I believe there are several reasons:

- Specifying the sender and recipients separately is more efficient, as it gives the SMTP receiver the information it needs “up front” before the message itself is transmitted. In fact, the SMTP receiver can decide whether or not to accept the message based on the source and destination e-mail addresses.
- Having this information specified separately gives greater control on how e-mail is distributed. For example, an e-mail message may be addressed to two recipients, but they may be on totally different systems; the SMTP sender might wish to deliver the mail using two separate SMTP sessions to two different SMTP receivers.

- In a similar vein, there is the matter of delivering blind carbon copies. Someone who is “BCC'ed” on a message must receive it without being mentioned in the message itself.
- Having this information separate makes implementing security on SMTP much easier.

For these reasons, SMTP draws a distinction between the message itself, which it calls the *content*, and the sender and recipient identification, which it calls the *envelope*. This is of course consistent with our running analogy between regular mail and e-mail. Just as the postal service delivers a piece of mail using only the information written on the envelope, SMTP delivers e-mail using the envelope information and not the content of the message. It's not quite the case that the SMTP server doesn't look at the message itself, just that this is not the information it uses to manage delivery.



Note: It is possible for the sender of a message to generate envelope information based on the contents of the message, but this is somewhat “external” to SMTP itself. It is described in the standard but caution is urged in exactly how this is implemented.

SMTP Mail Transaction Details

Let's take a more detailed look at the SMTP mail transaction process, using as aids the process diagram in Figure 305 and the example transaction of Table 250 (which has commands highlighted in bold and replies in italics). The first two steps in the mail transaction are responsible for providing the receiving SMTP server with the envelope information just discussed. The transaction begins by the SMTP sender issuing a **MAIL** command. This serves to inform the receiver that a new transaction is commencing, and also to tell it the “from” information on the “envelope”. An example:

MAIL FROM:<joe@someplace.org>

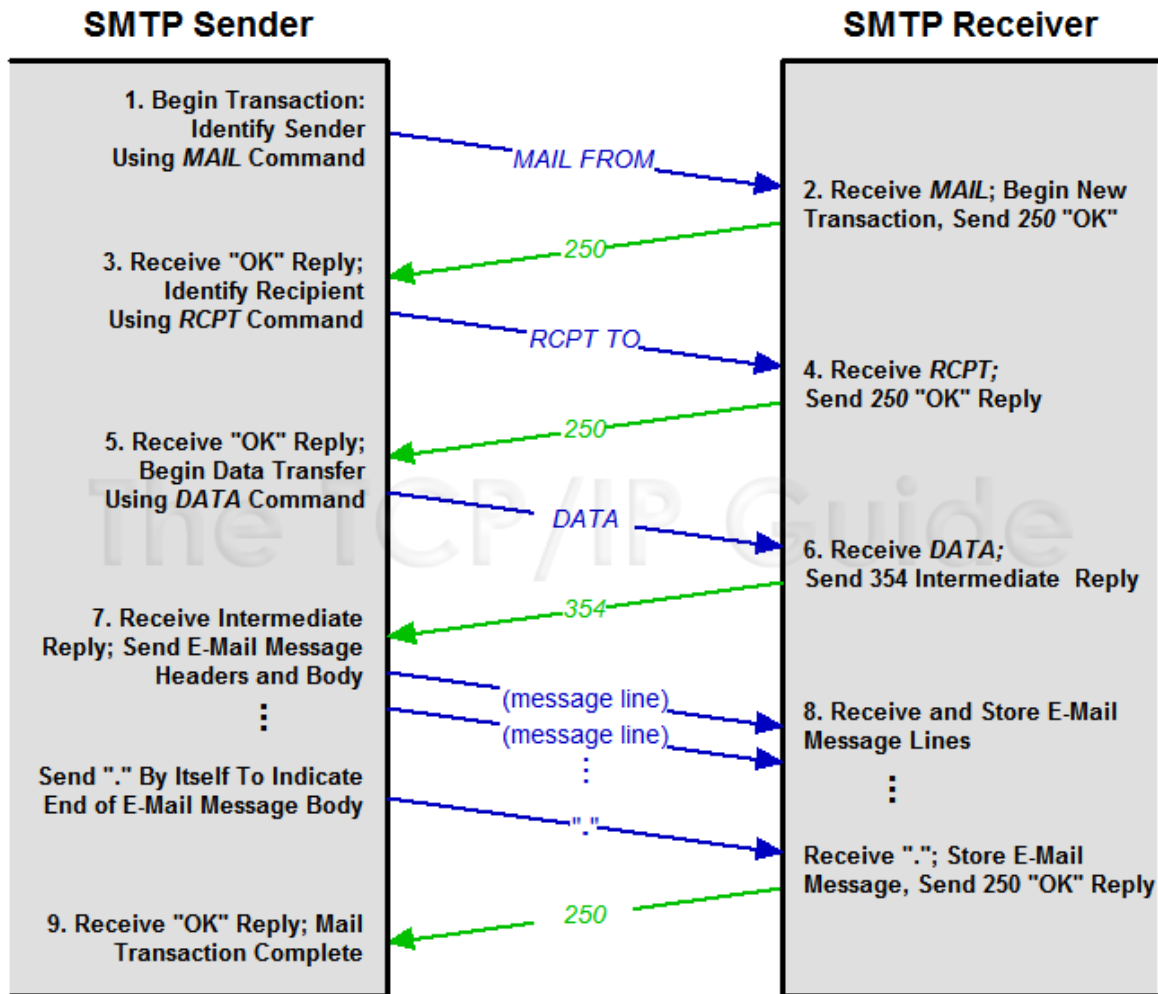


Figure 305: SMTP Mail Transaction Process

Once an SMTP session is established between a sender and receiver, each mail transaction consists of a set of three command/reply sequences. The sender is first identified using the *MAIL* command and the recipients are specified using one or more *RCPT* commands. The actual mail message is then transferred using the *DATA* command, which involves a preliminary reply before the actual message is sent, and a completion reply when it has been fully received.

The e-mail address of the originator is always enclosed in angle brackets ("*<*" and "*>*"). The SMTP receiver acknowledges the command with a 250 ("OK") reply message, sometimes sending back the address as a confirmation. For example:

250 <joe@someplace.org>... Sender ok

Next, the SMTP sender uses *RCPT* commands to specify the intended recipients of the e-mail that is being sent. Each *RCPT* line can contain only one recipient, so if multiple recipients are indicated, two or more *RCPT* commands must be issued. Each one normally specifies an e-mail address, but if relaying is being used, the command may contain routing information as well. (As described in [the SMTP communication topic](#), this is not as commonly done as it was in the past.) For example:

```
RCPT TO:<jane@somewhereelse.com>
```

Assuming the server accepts the e-mail, it will give a 250 “OK” reply again, such as this:

```
250 <jane@somewhereelse.com>... Recipient ok
```

The SMTP sender then issues the *DATA* command, which tells the SMTP receiver that the message is coming:

```
DATA
```

The SMTP receiver responds with a 354 “intermediate” reply message, such as this:

```
354 Enter mail, end with "." on a line by itself
```

The SMTP sender then sends the e-mail message, one line at a time, with a single “.” on a line to terminate it. The server confirms the receipt of the message with another 250 “OK” reply, and the transaction is done.

Table 250: Example SMTP Mail Transaction

```
MAIL FROM:<joe@someplace.org>
250 <joe@someplace.org>... Sender ok
RCPT TO:<jane@somewhereelse.com>
250 <jane@somewhereelse.com>... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
From: Joe Sender <joe@someplace.org>
To: Jane Receiver <jane@somewhereelse.com>
Date: Sun, 1 Jun 2003 14:17:31 -0800
Subject: Lunch tomorrow


Hey Jane,
```



```

It's my turn for lunch tomorrow. I was thinking we could
[rest of message]
Hope you are free. Send me a reply back when you get a
chance.
Joe.
.
250 OK

```

 **Key Concept:** After an SMTP session is established, e-mail messages are sent using the SMTP *mail transaction process*. The SMTP sender starts the transaction by identifying the sender of the e-mail, and then specifying one or more recipients. The e-mail message itself is then transmitted to the SMTP receiver. Each e-mail to be sent is a separate transaction.

Potential Mail Transaction Complications

While this indeed is quite simple, I should point out that I have only shown an e-mail from a sender to one recipient, and the case where there are no problems or complications in the transaction. Due to either command syntax or server issues, it is possible for various types of errors to occur at different stages of the process, which may result in the transaction failing. There are also [security concerns](#) that may come into play, that may lead to restrictions in what transactions a server may allow.

TCP/IP Electronic Mail Mailbox Access Model, Method and Protocol Overview

In an ideal world... we would all be born knowing everything there is to know about computers and networking, and I'd have become a famous novelist instead of writing thousands of pages of this stuff. ☺ Well, that may be asking a bit **too** much, but wouldn't it have been nice if every device on the Internet simply ran SMTP server software? If so, then that one protocol would be sufficient to implement the entire TCP/IP e-mail system. You would just compose e-mail on your machine, and your SMTP software would send it to your recipient's, and he or she would read it. Nice and simple.

Back here in the real world, however, this is really not possible in general terms. As I explained in [the overview section on e-mail](#) and [the discussion of SMTP](#), an SMTP server must be connected to the Internet and available around the clock to receive e-mail sent at any time by any of the millions of other computers in the

world. Most of us either cannot or do not want to run machines continuously connected to the Internet, nor do we want to configure and maintain potentially-complex SMTP software. This is the reason why a complete e-mail exchange normally involves not two devices but four: a message is composed on the sender's client machine, transferred to the sender's SMTP server, then to the recipient's SMTP server, and finally, to the recipient's machine.

The Advantages of Specialized Mail Access and Retrieval Protocols

The communication between SMTP servers is of course done with SMTP. So is the initial step of sending the e-mail from the sender's machine to the sender's SMTP server. However, SMTP is **not** used for the last part of the process, accessing the recipient's mailbox. Instead, a set of *mailbox access and retrieval* protocols and methods were devised.

A fair question is... why was this done? Why not simply have mail sit “pending” on the recipient's SMTP server and then have it send the mail to the recipient client device when it comes online, using SMTP? There are two main reasons for this. The first is that SMTP was designed for the specific purpose of transporting e-mail only. Having it responsible for client mailbox access would require adding more functionality, making it difficult to keep SMTP “simple”. Also, SMTP works on a “push” model, with transactions being initiated by the sender. It would need changes to allow it to respond to requests from a client device that is only online intermittently.

But the second reason is probably more important: *flexibility* in how electronic mail is accessed. If we used SMTP, all we would be able to do is transfer e-mail to the recipient's client machine. This would be functional, but would greatly limit the capabilities of how e-mail is used. For example, some users might wish to access mail directly on the server and manipulate it there. Also consider the problem of people with special requirements, such as those who travel and may need to access e-mail from a number of different client devices.

E-Mail Access and Retrieval Models

For the reasons just examined, there is an advantage to providing more than one way to access a mailbox. RFC 1733, *Distributed Electronic Mail Models In IMAP4*, describes three different paradigms or models for mail access and retrieval:

- **Online Access Model:** This is the mode of access that we would all be using in my “ideal world” scenario, where every machine was always connected to the Internet running an SMTP server. You would have

constant, direct online access to your mailbox. In the real world, this model is still used by some Internet users, especially those who have UNIX accounts or run their own SMTP servers. I call this *direct server access*.

- **Offline Access Model:** In this paradigm, a user establishes a connection to a server where his or her mailbox is located. The user downloads received messages to the client device, and then deletes them from the server mailbox. All reading and other activity performed on the mail can be done “offline” once the mail has been retrieved.
- **Disconnected Access Model:** This is a hybrid of online and offline access. The user downloads messages from the server, so he or she can read or otherwise manipulate them without requiring a continuous connection to the server. However, the mail is *not* deleted from the server, like in the offline model. At some time in the future, the user connects back with the server and synchronizes any changes made on the local device with the mailbox on the server.

What sort of changes? Examples include marking whether or not a message has been read, to keep track of unread mail, and marking messages to which the user has already replied. These are important tools to help those with busy mailboxes keep track of what they need to do.

Comparing E-Mail Access and Retrieval Models

Of the three, which is best? You should know better than to ask me that question. ☺ Each has advantages and disadvantages, which is why it is good that we have these options rather than the single SMTP protocol for mail access.

[Direct server access](#) has the main benefits of instant speed and universal access from any location. It has the disadvantage that you must be online to read mail, and that it usually requires you to use UNIX e-mail clients that most people are not familiar with. However, the [Internet Message Access Protocol \(IMAP\)](#) can also be used for online access.

Offline access has the main advantages of simplicity and short connection time requirements; you can easily connect to the mailbox, download messages and then read them locally. But that makes this method somewhat inflexible, and poorly-suited to access from different machines. Still, it is right now the most popular access method, because simplicity is important; it is best typified by the popular [Post Office Protocol \(POP\)](#).

Disconnected access attempts to combine the advantages of offline and online access without combining their disadvantages, and does a pretty good job. The advantages are significant: the ability to quickly access mail and use it offline, while retaining and updating the mailbox on the server to allow access from different client machines. [IMAP](#) is the protocol popularly used for disconnected access. In the [IMAP overview](#) I explore its advantages over offline access, as well as its main disadvantages: complexity and far less universal support than POP (though acceptance of IMAP is slowly increasing).

Finally, in recent years, a somewhat new mailbox access method has become popular: [e-mail access using the World Wide Web](#). This technique allows a user to access his or her mailbox from any computer with an Internet connection and a Web browser. It is a good example of “line blurring”, not only between the access models discussed here, but between TCP/IP applications, in this case [the Web](#) and e-mail.



Key Concept: For flexibility, TCP/IP uses a variety of *mailbox access and retrieval protocols and methods* to allow users to read e-mail. Three different models describe how these different methods work: the *online* model, in which e-mail is accessed and read on the server; the *offline* model, in which mail is transferred to the client device and used there; and the *disconnected* model, where mail is retrieved and read offline but remains on the server with changes synchronized for consistency.

TCP/IP World Wide Web Electronic Mail Access

I don't know about you, but I was pretty darned glad when bell bottoms went out of style... and then, rather mortified when they came **back** in style a few years ago! That's the way the world of fashion is, I suppose. And sometimes, even in networking, “what's old is new again”. In this case, I am referring to the use of the online TCP/IP e-mail access model.

Most e-mail users like the advantages of online access, especially the ability to read mail from a variety of different machines. What they don't care for is [direct server access](#) using protocols like [Telnet](#) (“Tel-what?”), UNIX (“my father used to use that I think...” ☺) and non-intuitive, character-based e-mail programs. They want online access but they want it to be simple and easy to use.

In the 1990s, the [World Wide Web](#) was developed and grew in popularity very rapidly, due in large part to its ease of use. Millions of people became accustomed to firing up a Web browser to perform a variety of different tasks, to the point where using the Web became almost “second nature”. It didn't take very

long before someone figured out that using the Web would be a natural way of providing easy access to e-mail on a server. This is now sometimes called *Webmail*.

How Web-Based E-Mail Works

This technique is straight-forward: it exploits the flexibility of the [Hypertext Transfer Protocol \(HTTP\)](#) to informally “tunnel” e-mail from a mailbox server to the client. A Web browser (client) is opened and given a URL for a special Web server document that accesses the user's mailbox. The Web server reads information from the mailbox and sends it to the Web browser, where it is displayed to the user.

This method uses the [online access model](#) like direct server access, because requests must be sent to the Web server, and this requires the user to be online. The mail also remains on the server as when [NFS](#) or Telnet are used.

Pros and Cons of Web-Based E-Mail Access

The big difference between Web-based mail and the UNIX methods is that the former is much easier for non-experts to use. Since the idea was first developed, many companies have jumped on the Web-mail bandwagon, and the number of people using this technique has exploded into the millions in just a few years. Many free services even popped up in the late 1990s as part of the “dot com bubble”, allowing any Internet user to send and receive e-mail using the Web at no charge (except perhaps for tolerating advertising). Many Internet Service Providers (ISPs) now offer Web access as an option in addition to conventional POP/IMAP access, which is useful for those who travel. Google’s new Gmail service is the latest entrant into the sweepstakes, offering users 1 GB of e-mail storage in exchange for viewing Google’s text ads on their site.

There are drawbacks to the technique, however, which as you might imagine are directly related to its advantages. Web-based mail is easy to use, but inflexible; the user does not have direct access to his or her mailbox, and can only use whatever features the provider's Web site implements. For example, suppose the user wants to search for a particular string in his or her mailbox; this requires that the Web interface provide this function. If it doesn't, the user is out of luck.

Web-based mail also has a disadvantage that is an issue for some people: performance. Using conventional UNIX direct access, it is easy to quickly read through a mailbox; the same is true of access using POP3, once the mail is downloaded. In contrast, Web-based mail services mean each request requires

another HTTP request/response cycle. The fact that many Web-based services are free often means server overload that exacerbates the speed issue.

Note that when Web-based mail is combined with other methods such as POP3, care must be taken to avoid strange results. If the Web interface doesn't provide all the features of the conventional e-mail client, certain changes made by the client may not show up when Web-based access is used. Also, mail retrieval using POP3 by default removes the mail from the server. If you use POP3 to read your mailbox and then later try to use the Web to access those messages from elsewhere, you will find that the mail is “gone”—it's on the client machine where you used the POP3 client. Many e-mail client programs now allow you to specify that you want the mail left on the server after retrieving it using POP3.



Key Concept: In the last few years a new method has been developed to allow e-mail access using the *World Wide Web (WWW)*. This technique is growing in popularity rapidly, because it provides many of the benefits of direct server access, such as the ability to receive e-mail anywhere around the world, while being much simpler and easier than the older methods of direct access such as making a Telnet connection to a server. WWW-based e-mail can in some cases be used in combination with other methods or protocols, such as POP3, giving users great flexibility in how they read their mail.