Plan

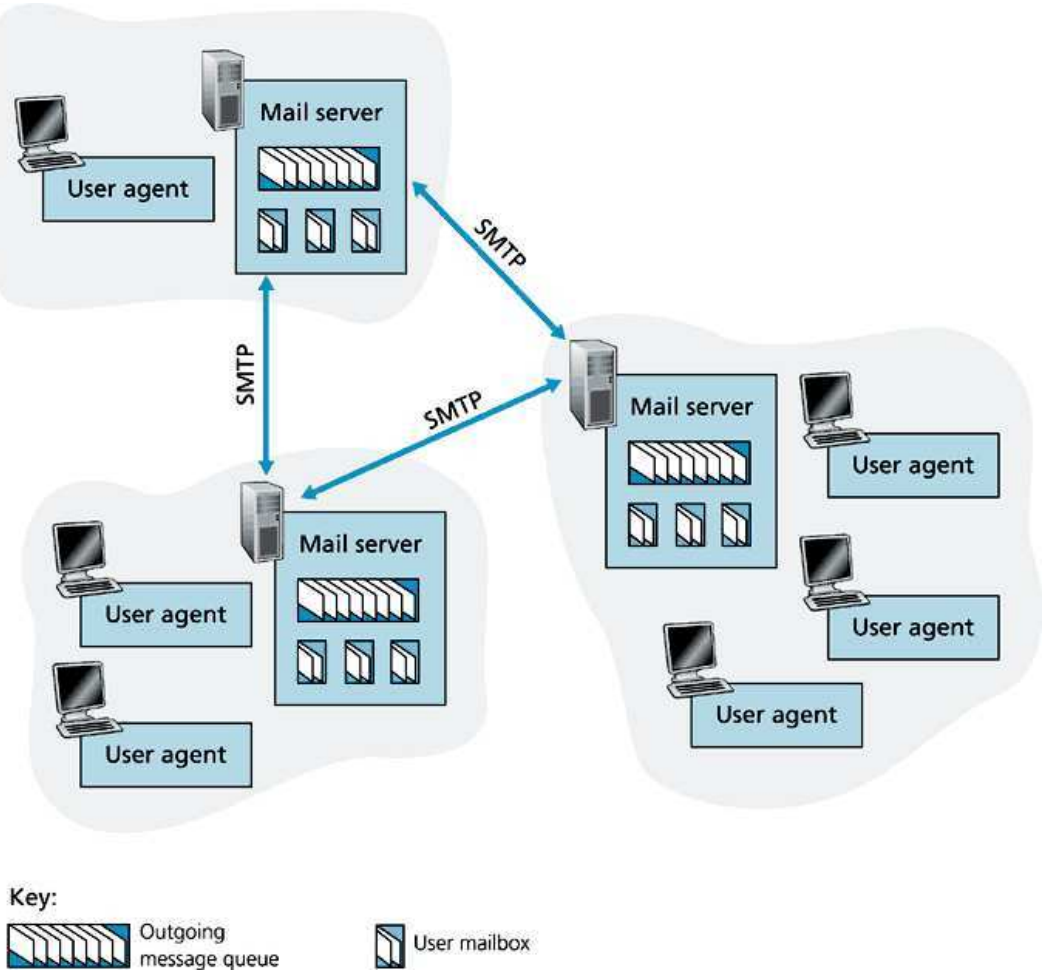
- Application layer
  - Principles of application layer protocols
  - The Web and HTTP
  - File transfer: FTP
  - **Electronic mail in the Internet**
  - DNS — The Internet's directory service
  - Socket programming with TCP and UDP
  - Content distribution

Electronic mail in the Internet

Electronic mail (or **e-mail**) is one of the first application of the Internet, and the most successful.
It is an **asynchronous** communication means, i.e. the two correspondent do not need to be using the e-mail application at the same moment.
The next figure provides a high-level view of the e-mail system. The three major components are

- **user agents** (also called in this context **mail readers**),
- **mail servers**,
- **Simple Mail Transfer Protocol** (**smtp**).

Electronic mail in the Internet (cont)



Electronic mail in the Internet (cont)

We will illustrate how the e-mail system works through the example of a user Alice which sends an e-mail to its recipient Bob.

1. When Alice is finished composing her message, her user agent sends it to its mail server, where it is put in the outgoing **message queue**.
2. In turn, Alice's mail server will send it to Bob's mail server, where it will be stored in Bob's incoming message queue (also called **mailbox**).
3. When Bob will check his e-mails, his user agent will download Alice's message from his mailbox in his mail server and display it.

Electronic mail in the Internet (cont)

Mailboxes

Each recipient, such as Bob, has a mailbox located in one of the mail servers. Bob's mailbox manages the messages that have been sent to him.

When Bob wants to read his new messages, he has to connect to his mailbox and the hosting mail server will authenticate him, by requesting a user name and a password.

Delivery failures

In case Alice's mail server fails to deliver her e-mail to Bob's mail server, the message is kept in Alice's mail server outgoing queue and the server will try to re-send it every 30 minutes or so. If there is no success after several days (often 5 days), the server removes the messages and informs Alice of the situation.

SMTP

The Simple Mail Transfer Protocol (SMTP) is the main application-layer protocol for Internet e-mail. It relies on TCP to transfer the e-mails. SMTP is divided into a **client** and a **server** entity: the client executes on the sender's mail server and the server runs on the recipient's mail server. Both client and server run on every mail server. A mail server becomes an SMTP client when he sends some e-mail, and when it receives some e-mail it behaves as an SMTP server.
Contrary to HTTP, SMTP does require the mail content to be encoded in ASCII. This implies that multimedia content must be encoded by the sender's agent and decoded by the recipient's agent.

SMTP (cont)

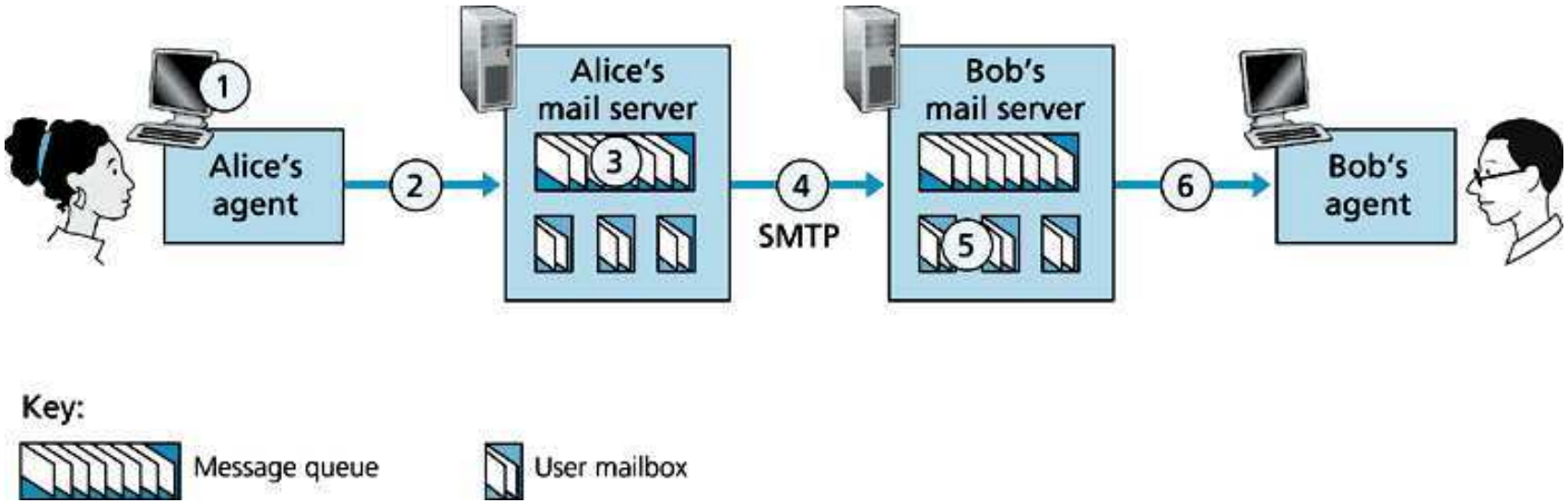
Let us illustrate a mail exchange between Alice and Bob in a more detailed way.

- Alice invokes her mail agent, provides Bob's **e-mail address**, composes a message and order the agent to send the message.
- Alice's agent sends the message to her mail server, where it is placed in a queue.
- The client side of SMTP, running on Alice's mail server, detects the message in the queue. It opens a TCP connection to an SMTP server running on Bob's mail server, on port 25.
- After some SMTP handshaking, the SMTP client sends Alice's message into the TCP connection.
- At Bob's mail server, the server side of SMTP receives Alice's message and places it in Bob's mailbox.

SMTP (cont)

- When Bob decides to read his messages he runs his mail agent, which connects to his mail server.
  Bob's mail server authenticates him and delivers to his mail agent Alice's message.

This situation is summarized in the following picture:



Note that there is a direct TCP connection between the two mail servers.

SMTP (cont)

Let us detail the SMTP session between Alice's client and Bob's server. In the following transcript, lines prefixed by **C:** are sent verbatim by the client agent to its TCP socket, and lines starting with **S:** are exactly the lines send to the TCP connection by the server. Client hostname is `crepes.fr` and server hostname `hamburger.edu`.

```
S:  220 hamburger.edu
C:  HELO crepes.fr
S:  250 Hello crepes.fr, pleased to meet you
C:  MAIL FROM: <alice@crepes.fr>
S:  250 alice@crepes.fr ... Sender ok
C:  RCPT TO: <bob@hamburger.edu>
S:  250 bob@hamburger.edu ... Recipient ok
```

SMTP (cont)

C: `DATA`
S: `354 Enter mail, end with "." on a line by itself`
C: `Do you like ketchup?`
C: `How about pickles?`
C: `.`
S: `250 Message accepted for delivery`
C: `QUIT`
S: `221 hamburger.edu closing connection`

Note that SMTP uses persistent TCP connections, therefore if the sending mail server has several messages to send to the same receiving mail server, it can send all the messages over the same TCP connection.

SMTP (cont)

It is recommended to use Telnet to send some e-mail:

```
$ telnet mail.konkuk.ac.kr 25
```

where `mail.konkuk.ac.kr` is the name of the *remote* server. This command establishes a TCP connection between the local host and the remote mail server.
After invoking Telnet at the prompt, the client receives:

```
Trying 202.30.38.143...
Connected to mail.konkuk.ac.kr.
Escape character is '^]'.
220 konkuk.ac.kr ESMTP Postfix
```

and can type now SMTP commands.

Comparison with HTTP

Both SMTP and HTTP transfer files (objects vs. mails) from one host to another use persistent TCP connections (mandatory for SMTP). The differences are:

1. an HTTP client pulls files (objects) from a server, whereas an SMTP pushes them (mails) to a server: we say that HTTP is a **pull protocol** whereas SMTP is a **push protocol**;
2. SMTP requires the body of each message to be encoded in ASCII, contrary to HTTP, thus accentuated characters or binary data must be encoded and decoded;
3. when a file consists of several parts (like text and images), SMTP places all of them in the same message, contrary to HTTP which puts them in separate messages.

Mail message formats and MIME

The body of an e-mail can be preceded by a blank line and a **header** made of header lines. Each of these lines is made of a keyword followed by a value.
Header lines starting with keywords `From:` and `To:` are mandatory. The header line starting with `Subject:` is optional. For instance:

```
From: Alice
To: bob@hamburger.edu
Subject: Some topics
```

A blank line follows the header and then the contents.
Warning! The header lines are **not** SMTP commands, but are part of the mail body. The header is interpreted by the *receiving mail agent* (not the SMTP server), which then displays it.

The MIME extension for non-ASCII data

The header can be used to tell the receiving mail agent how to handle multimedia (i.e. non-ASCII) and multi-part (i.e. attached files) e-mails. These extra header lines and their interpretation constitute the **Multipurpose Internet Mail Extensions** (**MIME**).
The two header keywords enabling MIME are

- `Content-Transfer-Encoding:`
- `Content-Type:`

The first field instructs the receiving mail agent what kind of encoding to ASCII has been used, so it can reverse the process. Then, the second field enables the receiving mail agent to take an appropriate action on the message, as launching an image viewer if the content is JPEG.

The MIME extension for non-ASCII data (cont)

Let us take an example. Imagine Alice wants to send a JPEG image to Bob. Her mail agent generates a MIME message, which might look like this:

```
From: Alice
To: bob@hamburger.edu
Subject: Picture
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

... base64 encoded data ...
```

Alice's mail agent encoded the JPEG file using base64, which is one possible technique to produce ASCII from binary.

The MIME extension for non-ASCII data (cont)

When Bob's mail agent reads this message, it reads the

```
Content-Transfer-Encoding: base64
```

header, so it proceeds to decode the message body with a base64-decoder.
The header line

```
Content-Type: image/jpeg
```

tells Bob's mail agent that the message should be interpreted (*after decoding*) as a JPEG image. Thus, it may propose to Bob to launch a JPEG viewer if he would like to.
The following header line tells the version of MIME used by Alice's mail agent:

```
MIME-Version: 1.0
```

The MIME extension for non-ASCII data (cont)

There are two formats of MIME types:

`Content-Type:` *`type/subtype`*
`Content-Type:` *`type/subtype`*`;` *`parameters`*

In the previous example, the type was `image` and the subtype was `jpeg`.
The subtype restricts the meaning of the type, thus allowing a more precise interpretation of the content.
Most parameters are associated with a specific subtype and are similar to a variable definition.
The set of types and subtypes, as well as parameter, is open and increasing. New types should be registered at the Internet Assigned Numbers Authority (IANA)[2].

[2] `http://www.iana.org/`

The MIME extension for non-ASCII data (cont)

Currently there are seven types defined. The following are common.

- **text** tells the receiving mail agent that the body contains text. If the subtype is `plain`, the mail agent does not need any extra software to display the body, except font support. The character set can be specified using a parameter, like `charset=euc-kr`. Another popular pair is `text/html`, which informs the receiving agent that it can display the content as a web page.
- **image** tells the receiving mail agent that the body is an image. Two pairs of type and subtype are popular: `image/jpeg` and `image/gif`.

The MIME extension for non-ASCII data (cont)

- **application** is used when the content does not fit any other type. It is often used for data that must be processed by an application before it can be viewed. For instance, `application/msword` instructs the receiving mail agent to launch Microsoft Word or OpenOffice to read the mail.
- **multipart** is used when the mail contains several parts because the sender wanted to send some other files together with his main message. If the attached files are not text, the pair `multipart/mixed` is often used.

The MIME extension for non-ASCII data (cont)

Let us say a little more about multi-part e-mails.
The mail agent needs to determine where each part starts and ends inside the body, how each non-ASCII part was encoded and what kind of content is it.
This is simply achieved by

- placing *boundary characters* between each part,
- repeating `Content-Transfer-Type:` and `Content-Type:` header line at the beginning of each part.

The MIME extension for non-ASCII data (cont)

Assume Alice wants to send a message to Bob consisting of some ASCII text and a JPEG image. After she types the text and attach the image, the agent produces

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: My picture
MIME-Version: 1.0
Content-Type: multipart/mixed; Boundary=StartOfNextPart

--StartOfNextPart
Dear Bob, look at my picture:
--StartOfNextPart
Content-Transfer-Encoding: base64
Content-Type: image/jpeg
... base64 encoded data ...
```

Received Message header

The *receiving* SMTP server can also insert a specific line at the top of an e-mail: the `Received:` header line. This line specifies the originating SMTP server, the receiving SMTP server and the reception time. Like:

```
Received: from crepes.fr by hamburger.edu;
          12 Oct 98 15:27:39 GMT
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: My picture
.....
```

Received Message header (cont)

A user can instruct his mail server to forward his e-mail to another mail server. In this case each receiving mail server, i.e. the forwarding one and the final one, will add a `Received:` line to the original mail.
For instance, if Bob forwards his e-mails to the mail server `kimchi.kr`, the final messages will look like

```
Received: from hamburger.edu by kimchi.kr;
          3 Jul 01 15:30:01 GMT
Received: from crepes.fr by hamburger.edu;
          3 Jul 01 15:17:39 GMT
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: My picture
.....
```

Mail Access Protocols

We implicitly assumed until now that Bob reads his e-mails by directly logging in his mail server.
For a long time, this was the only way but since the mid 90's users have a mail agent that connects to the user's mail server, downloads the e-mails on the local disk and displays their contents to the user.
Using a mail agent allows to store locally the e-mails but also to easily visualise their possible multimedia attachments (images, videos, music etc.) — the purpose of a mail server is to serve mails, not to provide fancy features for display.
Note: the idea to run the recipient's mail server on the same machine as the mail agent is not good. Indeed, the user may want to turn his machine off and so the mail server would be, leading to the delivery failure of subsequent incoming mails.

Mail Access Protocols (cont)

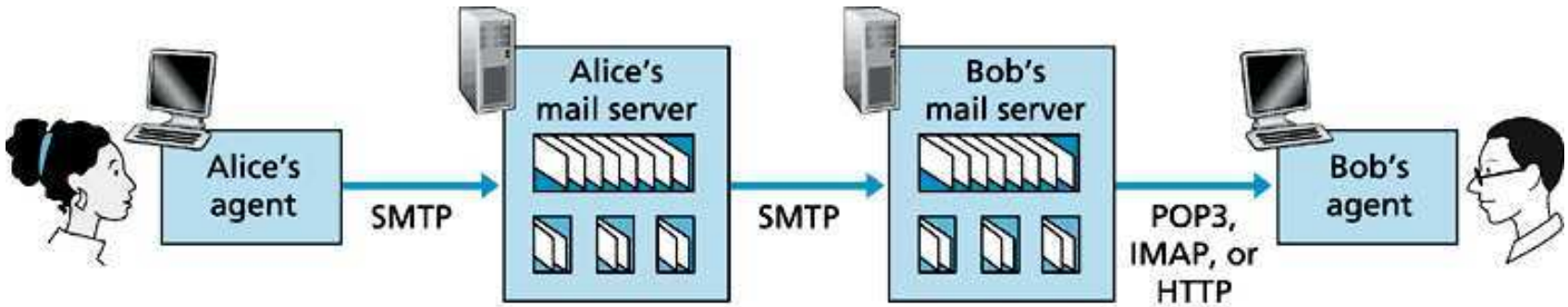
Remember that a mail server is always on, it is usually shared by several users and is maintained by the ISP.
So

- Alice's mail agent has *to push* her e-mail to her mail server,
- Bob's user agent has *to pull* his e-mails to his local disk.

The first task can be done using SMTP because it is a push protocol by design. Why a two-step procedure (from Alice's mail agent to her mail server, then from her mail server to Bob's mail server)? Because otherwise, if Bob's mail server is down, Alice's mail agent should keep trying again to send the e-mails for days, impeding Alice to log out meanwhile.

Mail Access Protocols (cont)

The second tasks cannot be achieved by SMTP: we need a pull protocol, or **Mail Access Protocol**.
There are several possibilities: **pop3** (**Post Office Protocol** version 3), **imap** (**Internet Mail Access Protocol**) and HTTP:



Mail Access Protocols/POP3

POP3 is an extremely simple mail access protocol.
The session starts with the recipient's mail agent opens a TCP connection to its mail server on port number 110.
Then three phases occur in turn:

1. **Authorisation**: the user agent sends the user's name and password to the mail server, which authenticates him;
2. **Transaction**: the user agent retrieves the messages and the user can: mark the mails to be deleted on the server, unmark them and get statistics about them;
3. **Update**: the mails marked for deletion on the server are actually deleted.

Mail Access Protocols/POP3 (cont)

In a POP3 transaction, the user agent issue textual commands and the server responds to each command with a textual reply.
There are two possible replies:

- `+OK` (sometimes followed by server-to-client data) meaning the last client command was fine;
- `-ERR` meaning that something was wrong with the last command.

Mail Access Protocols/POP3 (cont)

Authorisation

There are two principal commands: `user` *name* and `pass` *password*. Use telnet to try these commands:

```
$ telnet mail.konkuk.ac.kr 110
Trying 202.30.38.143...
Connected to mail.konkuk.ac.kr.
Escape character is '^]'.
+OK <11115.1117125523@konkuk.ac.kr>
user rinderkn
+OK
```

Mail Access Protocols/POP3 (cont)

Example of failed authentication:

```
$ telnet mail.konkuk.ac.kr 110
Trying 202.30.38.143...
Connected to mail.konkuk.ac.kr.
Escape character is '^]'.
+OK <15247.1117126049@konkuk.ac.kr>
user foo
+OK
pass bar
-ERR authorization failed
Connection closed by foreign host.
```

Mail Access Protocols/POP3 (cont)

Transaction

The users can configure his mail agent to **download and delete** or **download and keep**, depending whether they wish to keep a copy of their mails on the server or not.

According to this configuration, the user agent sends different commands to the mail server, e.g. in the "download and delete" mode, the user agent will issue the `list`, `retr` and `dele` commands.

Imagine a user has two messages in his mailbox. In the following transcript, lines prefixed by **C:** are sent verbatim by the client agent to its TCP socket, and lines starting with **S:** are exactly the lines sent to the TCP connection by the server.

Mail Access Protocols/POP3 (cont)

```
C:  list
S:  1 498
S:  2 912
S:  .
C:  retr 1
S:  ... blah blah ...
S:  .
C:  dele 1
C:  retr 2
S:  ... blah blah ...
S:  .
C:  dele 2
C:  quit
S:  OK POP3 server signing off
```

Mail Access Protocols/POP3 (cont)

A problem with this "download and delete" approach is that the recipient, Bob, may be nomadic and want to access his mail messages from multiple machines, e.g. his office computer, his home computer and his portable computer.
If Bob reads his messages from his office machine, he will not be able to reread them from his home machine.
In the "download and keep" mode, the user agent leaves the messages on the server after downloading (a copy of) them.
During a POP3 session, the server keeps a client state, but not across sessions — which greatly simplifies the server implementation.

Mail Access Protocols/IMAP

With POP3, once the recipient has downloaded his messages, he can store them in a hierarchy of folders and later search information in it (e.g. by sender's name or subject).
But a nomadic user would get a hierarchy with different messages *for each machine* from where he downloads the mails. One centralised hierarchy would be preferable in this case. This is not possible with POP3.
To solve this problem, **imap** (**Internet Mail Access Protocol**) provides more features than POP3, at the cost of more complex implementations.

Mail Access Protocols/IMAP (cont)

An IMAP server will associate each message with a folder (initially, it is the "Inbox" folder).
The user can create and change folders on the server, as well as searching the messages they contain according to specific criteria.
This means that IMAP, contrary to POP3, maintain user state information *across sessions*.
Another useful IMAP feature is the possibility for the client to obtain only some components of a message. For instance, a user agent can request just the body of a message or one part of a multi-part MIME message. This is very useful in case of a low-bandwidth connection.

Mail Access Protocols/Web mail

A lot of users today access their e-mails through their Web browsers.
This technique was introduced in the mid-1990's by the company Hotmail.
In this case, Alice's mail agent sends her messages using HTTP requests (like forms) to her web server which, in turn, uses SMTP to push the mail to the recipient's mail server.
Symmetrically, Bob retrieves his messages using HTTP requests to his web server which, in turn uses IMAP to get the messages from the sender's mail server.
This web-based e-mail system is very useful for mobile users since it only requires a web browser and an internet connection.