Application Layer

The application layer consists of what most users think of as programs. The application does the actual work at hand. Although each application is different, some applications are so useful that they have become standardized.

The Internet has defined standards for:

File transfer (FTP): Connect to a remote machine and send or fetch an arbitrary file. FTP deals with authentication, listing a directory contents, ascii or binary files, etc.

Remote login (telnet): A remote terminal protocol that allows a user at one site to establish a TCP connection to another site, and then pass keystrokes from the local host to the remote host.

Mail (SMTP): Allow a mail delivery agent on a local machine to connect to a mail delivery agent on a remote machine and deliver mail.

News (NNTP): Allows communication between a news server and a news client.

Web (HTTP): Base protocol for communication on the World Wide Web.

Electronic Mail

Electronic mail is a popular network service. There are two parts to electronic mail systems:

User agent: The user interface to the mail system. The user agent accesses messages stored in a system mailbox, provides ways to view, edit, and reply to messages, etc. The user agent can be as simple as having the user use a text editor to create a file that the user agent hands to the message transfer agent.

Sample agents in Unix: */bin/Mail*, *mh*, *elm*, and the *emacs rmail* package.

Message transfer agent (MTA): Software that transports messages created by a user to destination mailboxes, possibly on remote machines.

The MTA's job is more complex than other applications:

1. It must handle temporary failures; if a destination machine is temporarily unavailable, it must *spool* the message on the local machine for later delivery. Thus, the User Agent typically just deposits messages into a spool area.
2. It must distinguish between local and remote recipients.
3. It may have to deliver copies of a message to several machines.
4. It may allow mixing text, voice, and video in a message as well as appending documents and files to a message.

Mail addresses consist of:

mailbox names: The name of a specific mailbox. Usually, a mailbox is associated with one login id.

symbolic names: The name of a service rather than a specific user. For instance, "postmaster" is universally recognized as an address for mail problems. Symbolic names are aliases for specific mailbox(s).

group names (mail exploders): An alias for a set of recipients.

How does an MTA know what to do with mail? It consults an internal database that specifies how the mail address should be interpreted.

The Internet includes a standard for mail delivery called the *Simple Mail Transfer Protocol* (SMTP). The protocol itself is surprisingly simple; because it uses TCP, much of the hard work is handled by lower-level protocols.

To deliver mail, the MTA opens a TCP connection to a destination site, and sends it the message. The remote MTA deposits the message in its spool area, and returns an acknowledgment ONLY after it has saved the message in stable storage. The sender removes its copy ONLY after it has received the acknowledgment.

If the destination is unavailable, the MTA tries again later. If a message cannot be delivered in (say) 3 days, an error is returned to the user.

Internet mail has an important advantage over other mail systems (e.g., *uucp* or *bitnet*): Because it is an end-to-end delivery system, it is inherently more reliable than other systems.

In contrast, systems utilizing *message switching* can only guarantee that a message gets to an intermediate destination. In uucp, one specifies a list of machines (e.g., "mcvax!enidbo!cxr"). The local MTA delivers the message to machine *mcvax* and deletes it from the local spool area.

Unfortunately, we now no longer have a copy of the message, and it has not yet reached the destination machine *enidbo*. In short, anything can happen:

- Machine *mcvax* might crash and lose the message (without reporting an error).
- The message could be delayed hours or days.
- Worst of all, neither sender nor recipient can find out what the status of the message is.

Within the Internet, all mail addresses have the same form: *local-part@domain-name*. Is *domain-name* a machine name? No. Rather than referring to a machine name, *domain-name* refers to an MX record in the DNS. *Local-part* refers to anything to the left of the "@" and can be the name of a user, the name of a mailing list, etc. Internet addresses only specify the *syntax* of addresses, not their *semantics*. For example:

cew@cs.wpi.edu: Specifies a domain name of *cs.wpi.edu* and local-part *cew*.

cew%cs.wpi.edu@cs.purdue.edu: Specifies a domain name of *cs.purdue.edu*, and a local part of *cew%cs.wpi.edu*. The MTA would deliver the mail to purdue, which would process the mail based on the local-part.

mcvax!enidbo!cxr@uunet.UU.NET: The MTA delivers the message to *uunet.UU.NET*, *uunet* then inspects the local part. *Uunet.UU.NET* uses the convention that "!" characters in the local part refer to uucp path names. Note that this is a convention only — many sites do not interpret "!" in this way, and there is no requirement that they do so.

The mail to a bitnet site demonstrates the need for *mail gateways*, gateways that accept mail messages from one network type (e.g. Internet) and send them across another (e.g. bitnet). Mail gateways:

1. Are used when the sender and recipient cannot communicate directly using common transport protocols.
2. Hide the details of multiple incompatible networks from users.
3. Increase the complexity of an MTA, which now has to be able to translate selected addresses into other forms. An MTA may need to translate an internet address into a Bitnet address, for instance.

SMTP deals with transferring mail from one MTA to another. It only concerns itself with transferring mail from one machine to another. Surprisingly, the SMTP protocol is quite simple. It uses the query response model, and only a few message types are defined. The hard work is handled by TCP.

SMTP commands consist of human-readable ascii strings.

Responses begin with a 3-digit number (used by the SMTP program), followed by a string (used as comments for users), and a terminating newline character. Each command has a corresponding response:

HELO: Initiate a mail transaction, identifying the sender to the recipient.

Typical response: "250 OK".

MAIL FROM: <reverse-path>: Tells the remote SMTP that a new mail transaction is beginning. A single TCP connection can be used to serially process a set of message exchanges between a pair of hosts.

Reverse-path specifies where errors should be sent to, allowing error messages to be forwarded to a system administrator rather than the sender. This feature is especially useful in large mailing lists, where errors about undeliverable mail should go to the list maintainer rather than a sender.

Typical response: "250 OK".

RCPT TO:<forward-path>: *Forward-path* specifies a single recipient of the message. *Forward-path* is the "local part" of a mail address, and can be a user, mail list, etc. The sending SMTP sends a *RCPT* command for each intended recipient.

Typical responses: "250 OK", "251 User not local; will forward to <forward-path>", or "550 No such user".

DATA: If accepted, the sender transfers the actual message. End-of-message is indicated by sending a "." on a line by itself.

Typical response: "250 OK". When can sender delete message from its queue? The sender does not delete its copy of the message until *after* receiving the 250 reply.

QUIT: Terminate the connection.

The following example shows mail sent by *walnut* through *owl* to *cs.purdue.edu*.

```
220-owl.WPI.EDU Sendmail 8.6.12 ready at Wed, 24 Apr 1996 11:10:06 -0400
220 ESMTP spoken here
>>> EHLO walnut.WPI.EDU
250-owl.WPI.EDU Hello walnut.WPI.EDU [130.215.8.94], pleased to meet you
250-EXPN
250-SIZE
250 HELP
>>> MAIL From:<cew@walnut.WPI.EDU> SIZE=91
250 <cew@walnut.WPI.EDU>... Sender ok
>>> RCPT To:<cew@cs.purdue.edu>
250 <cew@cs.purdue.edu>... Recipient ok
>>> DATA
354 Enter mail, end with "." on a line by itself
>>> .
250 LAA27806 Message accepted for delivery
cew@cs.purdue.edu... Sent (LAA27806 Message accepted for delivery)
Closing connection to owl.wpi.edu.
>>> QUIT
221 owl.WPI.EDU closing connection
```

Miscellaneous SMTP commands:

TURN: Turn the line around. When the client has finished sending mail, the TURN command switches the sender and receiver, allowing the receiver to start sending messages without having to terminate and reestablish a connection.

EXPN: expand a mailbox name into its aliases.

Note: Is mail authenticated? No. It is trivial to fool SMTP into accepting forged messages. SMTP makes no attempt to authenticate a sender.

Remote Login

A remote login facility allows a user to establish a login session to a remote machine and then execute commands. Implementing a remote login facility is not as trivial as might seem:

- Some operating systems were built before networks existed and assume that login sessions can come only from terminal lines.
- Some systems allow special keys (e.g., CTRL-C) to abort the current process. In some systems it may be impossible to send the CTRL-C to the remote server.

  On the other hand, sending a CTRL-C to a remote server may prevent a user from aborting the local end of the session.
- A remote login facility should provide interoperability between *heterogeneous* machines. Should a program output CR-NL or just a NL? Individual machines have differing interpretations for how to specify functions to perform such logical functions as indicating end of record/line.

Telnet is an Internet standard remote login protocol that connects a local terminal with a remote login session. It copies keystrokes to the remote machine and copies output from the remote machine to the local terminal. *Telnet* provides three services:

1. Telnet defines a *Network Virtual Terminal* (NVT) standard that describes a standard terminal. Client programs interact with the NVT, and the server translates NVT operations into ones specific to the actual hardware/operating system.
2. Telnet allows the two ends of the connection to negotiate options with one another. Option negotiation allows both ends to agree on a common level of service.
3. Telnet treats both ends of a connection symmetrically allowing both ends to be programs.

To handle heterogeneity, *telnet* defines how data and command sequences are represented. The client translates keystrokes into NVT format and sends them the server. The server translates NVT operations into the appropriate local representation.

All telnet operations are sent as 8-bit bytes:

- Bytes with a first bit of zero are treated as normal 7-bit ascii character data.
- Printable ascii characters have standard meanings, and the meaning of control characters is specified to eliminate ambiguities.

- Bytes with the high order bit set are used for command sequences.
- The two-character sequence CR-LF delimits records.

Telnet defines the following commands:

Interrupt process (IP): Terminate the running program.

Abort output (AO): Discard any buffered output.

Are you there (AYT): Allows client to send and out-of-band query to verify the remote end is still there.

Erase character (EC): Erase the previous character.

Erase line (EL): Delete the entire current line.

Synchronize: Clear data path to remote party.

Break: Equivalent of the BREAK or ATTENTION key.

To send a command, *telnet* sends the escape sequence *Interpret As Command* (IAC) followed by the 1-byte command. The IAC has a value of 0xff (and must be escaped when it appears in the data).

Sending control functions is not always enough to guarantee the desired results. Consider the following scenario:

- If the remote program is in a loop, it may never read input or generate output.
- If the client continues sending output, the server's receive buffer will fill, and the server will eventual advertise a TCP flow-control window of size 0. Further data sent to the server will remain in buffered at the sender.
- What if the client now sends an "interrupt process" command? It will never be delivered because the client is no longer allowed to send new data. The command stays in the sender's buffer.

Telnet solves this problem by using the TCP *urgent pointer* mechanism to send an *out-of-band* signal. When it sends the *interrupt process* command, it directs TCP to set the urgent pointer to the IAC command. TCP then sends a segment (that doesn't necessarily have data in it) with the urgent pointer set. The remote TCP then alerts the remote *telnet* of the urgent data, which proceeds to process it.

Note: The TCP urgent mechanism specification requires that the operating system provide a mechanism for asynchronously notifying a process of the presence of urgent data; Unix uses the SIGURG signal.

Telnet allows either end of a connection to negotiate the use of options. Option processing is interesting because either end can request the use of an option. To request an option, side *A* sends a "Will X" option request. The remote side responds with "Do X" or "Don't X", indicating that it will or will not participate in the requested option. If an option is accepted, it become effective immediately after the "DO X" in the data stream.

Actual option negotiation takes place in two steps. At the top level, two telnet processes negotiate as to whether they are willing to negotiate an option. If the answer is yes, further option-specific subnegotiation takes place to exchange the actual option information.

Example options:

1. Use an 8-bit instead of 7-bit data path. This allows the transmission of non-ascii data.
2. Full duplex vs. half duplex operation.
3. Local editing. That is, have the local telnet perform echoing, erase, etc., and only send complete lines. This can save a tremendous amount of bandwidth.

   Another option allows the remote side to specify what characters indicate "end of command".
4. Specify mappings of characters to interrupt functions. For example, map ctrl-C into the abort command.
5. Determine the size (rows and columns) of the terminal.

Note: *Telnet* servers using newer options coexist with older servers; they negotiate the use of new options with newer servers, while older server respond "Won't X" for any option X they do not understand.

Berkeley Rsh commands

Berkeley Unix includes a set of network applications that support the idea of a set of trusted hosts. System administrators define a set of machines that can be trusted, allowing users within the set of machines to dispense with explicit authentication (e.g., typing passwords).

In the CS Department, for instance, all Sun machines belong to one set, allowing users to *rlogin* from one workstation to another without providing passwords.

Unix includes several *rsh* ("r-shell") programs:

rlogin: Similar to *telnet*, but tailored to the Unix environment. For instance:

1. It doesn't require use of password when logging into another machine in the set of trusted hosts.
2. Users can augment the default set of trusted hosts by adding (machine name, user id) pairs to the file *.rhosts*. For example, the *.rhosts* file for the cs513 account on *wpi* could contain the entry: "sequoia.wpi.edu cew".
3. It exports the terminal type (e.g., the TERM variable) to the remote machine. (Note: telnet only recently added the option necessary to achieve this.)
4. Handles CTRL-S and CTRL-Q locally, so that output is stopped immediately. Recall that with *telnet*, CTRL-S is not processed by the local client at all; it is sent to the remote *telnet*, introducing a significant delay between entering CTRL-S and having the output stop.

Many of the weaknesses in *telnet* relative to *rlogin* have been fixed in the past several years.

rsh: Execute a remote command. For example, "rsh eve ls" executes the "ls" command on eve.

rcp: Copy a file from one machine to another. Analogous to *cp*, but filenames can be prepended with "machine:" to specify files on remote machines. One advantage that *rcp* has over FTP is that since *rcp* assumes both machine run Unix, it can set the permissions of the copied file properly. FTP is unable to do that because permissions generally have differing interpretations on different vendor's machines.

Secure Shell commands

New set of commands: slogin, ssh, scp to support secure r-shell programs.

Security methods;

- .rhosts/.shosts file—considered insecure and often not allowed by remote server.
- Client hosts maintains a key for each host and supplies it when logging on to the remote host. Remote host verifies it has correct key from the client host. Closes security holes due to IP spoofing, DNS spoofing and routing spoofing. The "`$HOME/.ssh/known_hosts`" file can be insecure.
- Use RSA encryption. On server machine maintain `$HOME/.ssh/authorized_keys`, which lists public keys for machines in which remote login is permitted. On login, server sends random number encrypted with user's public key. User maintains private key in *.ssh/identity.pub*, which is used to decrypt. Client can obtain public/private key pair using *ssh-keygen*. After setting up public and private keys, the user can remote login without a password.
- Use TIS authentication server.
- Use encrypted password entered by the user.

Remote File Access

There are two basic approaches to remote file access:

On-line access: Multiple programs running on different machines concurrently access a single file located on a file server. When accessing a file, an application operates on pieces of the file rather than the entire file. In addition, the server (and its associated protocols) may be able to guarantee each client sees the same version of a file.

This approach is typified by Sun's Network File System (NFS).

Whole file copying: An application fetches a copy of the entire file from a remote server, and then operates on the local copy.

If the operating system provides access to remote files exactly like it provides access to local files, we say that the remote files are *integrated* with local files and that the file system provides *transparent* access. The advantages of transparent on-line access should be obvious:

1. Users use the same programs to access local and remote files.
2. Files needed by many machines can be shared, reducing the amount of disk space needed. For example, binaries to popular programs such as *emacs*.
3. Users can access their files, regardless of which particular machine they log into.

However, there also disadvantages:

1. If an application uses both local and remote files, an application may not work, even if the local machine is running. That is, we still have a central point of failure.
2. Performance degrades when the network or remote server becomes overloaded.

The alternative to transparent access is to provide application programs to copy files between local and remote machines. The Internet defines one such application, *File Transfer Protocol* (FTP). FTP provides:

Interactive access: Users interact with a remote server, issuing commands such as "ls", "chdir", "put", "get", etc.

Format: The User specifies the type of file being transferred, allowing the transfer of ASCII or EBCDIC text files, binary data, etc.

Authentication: User must present account and password before being given access to the remote system.

Unlike other applications, FTP uses two TCP connections to do its work:

Control connection: The first connection is used by an interactive front end for command processing.

Moreover, rather than defining a new protocol, FTP uses a restricted form of *telnet* (e.g., no option negotiation) for the control connection.

Transfer connection: A second connection is used for the actual transfer of data (e.g., files, output from "ls" command, etc.).

One interesting aspect of FTP is its selection of TCP port numbers for the data connection. For the control connection, a client picks any unused local port number and connects to the well-known FTP port on the remote system. The server accepts control connection requests from *any* machine.

For the transfer connection, however, the server should only accept a connection from the user at the other end of the control connection. Thus, the remote FTP client picks a local port number and sends it to the server via the control connection. The server than accepts a connection request, but specifies that the remote address must match that received over the control port.

Typically, a user connecting to an FTP server must specify an account and password. Often, it is convenient to set up a special account in which no password is needed. Such systems provide a service called *anonymous FTP*:

1. When asked for an account and password, the user uses an id of "anonymous" and password "guest". (Or any password for that matter. The user's Internet mailing address is the preferred password.)
2. The account generally has restricted privileges; in Unix systems, for instance, anonymous FTP users can only access files within the restricted directory of the FTP account.
3. Anonymous ftp is a convention and must be setup by a system administrator.

The following shows a sample FTP session used to fetch X software from MIT:

```
< sequoia >ftp ftp.cs.usask.ca
Trying 128.233.130.57...
```

```
Connected to clotho.usask.ca.
220 clotho FTP server (UNIX(r) System V Release 4.0) ready.
Name (ftp.cs.usask.ca:cew): anonymous
331 Guest login ok, send ident as password.
Password:
230 Guest login ok, access restrictions apply.
ftp> ls
200 PORT command successful.
150 ASCII data connection for /bin/ls (130.215.8.87,1467) (0 bytes).
bin
dev
etc
ftpd_setup
pub
usr
226 ASCII Transfer complete.
37 bytes received in 0.016 seconds (2.3 Kbytes/s)
ftp> cd pub
250 CWD command successful.
ftp> ls -CF
200 PORT command successful.
150 ASCII data connection for /bin/ls (130.215.8.87,1469) (0 bytes).
111/           cmpt250/       eric/          pc99/          yang/
115/           combi/         geometry/      publications/
UofS_local/    defaults/      grassman/      rsmith.tar
aries/         discus/        keil/          sailor/
carey/         envlop/        microweb/      software/
226 ASCII Transfer complete.
remote: -CF
294 bytes received in 0.016 seconds (18 Kbytes/s)
ftp> binary
200 Type set to I.
ftp> get rsmith.tar
200 PORT command successful.
150 Binary data connection for rsmith.tar (130.215.8.87,1470) (55296 bytes).
226 Binary Transfer complete.
local: rsmith.tar remote: rsmith.tar
55296 bytes received in 3.8 seconds (14 Kbytes/s)
ftp> quit
221 Goodbye.
```

The *Trivial File Transfer Protocol* (TFTP) is another Internet standard for transferring files. It differs from FTP in the following ways:

1. It is based on UDP rather than TCP and is a basic stop-and-wait protocol with acknowledgments and retransmissions.
2. It has no authentication.
3. It is small enough that it fits in workstation and PC read-only memories (ROMs). Thus, workstations can use standard Internet protocols to *bootstrap* off of file servers.

Indeed, Sun diskless workstations boot as follows:

1. Use RARP to obtain their Internet addresses.
2. Use TFTP to download a copy of the operating system into memory.
3. Start execution.

The protocol itself is exceedingly simple:

1. The first datagram identifies the name of a file, and specifies whether the file is to be fetched or sent.
2. Datagrams are numbered sequentially, and each datagram containing 512 bytes of data.

   End-of-data is signaled by a datagram containing less than 512 data bytes.
3. Datagrams contain a sequence number and are either ACK or DATA packets.

Sun Network File System (NFS)

Sun's NFS has become a de facto standard for remote file systems. It provides online, transparent, integrated file access. Because it is RPC-based, users access remote files in the same way as local files. Clients invoke operations on the file server solely through RPCs.

For instance, to open a file, an application issues an *open* system call specifying the file to be opened. If the file is local, the operating system processes the open request as normal. If the file is located on a remote another server, the operating system issues an RPC to the server.

A *read* system call is translated into an RPC request for the relevant data blocks of the file.

Some interesting issues in building NFS:

1. NFS is an interface to a *virtual file system*. Although it is used primarily in Unix, it can also be used to access files in other operating systems.
2. RPC and XDR had to be moved from user-level library routines into the operating system kernel, because any file-related system calls might translate into a server request.
3. For fault tolerance, file servers should be *stateless*. That is, the server shouldn't maintain any information between subsequent client calls needed to satisfy a request — if the server were to crash, the state would be lost and the request could not be satisfied.

   In other words, an RPC's arguments should contain all information needed to satisfy a request.
4. Client RPC calls were designed to be idempotent. Is this possible given a stateless server? No. For example, what happens if a client invokes an RPC to delete a file, gets no response, and retransmits the request. If the request is executed once, the file will be deleted. If executed twice, the second request will result in an error — non-existent file. Without maintaining state, the server has no way of knowing whether a request is new or a retransmission.
5. Clients must first *mount* a remote file system onto the local file tree before accessing remote files. The mount operation allows the server to authenticate the client. Note, however, that authentication is based on the IP address of the client, a weak form of authentication.
6. The first version of NFS was very slow. Adding a small cache of recently used file pages to the client speeded up execution. (Files being written are flushed to the server when a file is closed.)

7. Further study showed that 90% of the client-server RPCs were for the *getattr* routine (e.g., Unix *stat*). Adding a client cache of recent responses reduced traffic to 10%.

Like many industry-originated de facto standards, NFS exhibits significant technical problems:

Cache coherency: Using caches to improve performance is a well known technique. A big problem is *cache coherence.*

What happens if two machines access the same file, one writing it, and the other reading it? They interfere with each other, because NFS's caching strategy isn't sophisticated enough to turn off caching.

Authentication: NFS performs authentication based on the sender's IP address (servers maintain a list of "trusted" machines), and each RPC packet contains an authentication field. In the Unix environment, the authentication field contains the sender's machine name, user id, group id, and a list of the groups the user is a member of. When a server gets an RPC request, it (naively?) assumes that the information is valid. Thus, in theory, anyone with a PC or workstation that can send arbitrary packets is able to impersonate a legitimate user. For instance, an imposter could claim to be some other user, and then send a sequence of packets to open a file and then read arbitrary data from the file.

NFS does include hooks for using DES encryption, but special hardware is needed.

UDP-based transport protocol: RPC uses UDP rather than a true transport protocol. One result has been implementations that work across LANs, but not WANs.

Yellow pages: Sun workstations use a distributed database system called *yellow pages.* The idea is to have a single server manage such information as passwords, and translating host names to host addresses.

Unfortunately, yellow pages is unable to return temporary failure indications, such as when using the DNS.

Despite its technical problems, NFS is tremendously popular and useful.

World Wide Web

Built on the Hypertext Transport Protocol (HTTP).

Addressing a key. Done through a Uniform Resource Locator (URL).

Form of a URL: protocol://hostname/file

e.g. `http://cs.wpi.edu/~cew/index.html`

Look at sample of what happens.

Other protocols can be used (see Fig. 7-63).

HTML

Language used is HTML (Hypertext Markup Language). Can include graphics/images.

Forms, scripts (commands to generate a page).

Secure Sockets Layer

With increased use of electronic commerces, secure transactions are important. SSL protocol developed by Netscape for encrypted communication between clients and servers.

SSL sits between TCP/IP and HTTP layers. Invoked by browsers using the protocol "https".

Uses public-key cryptography. Features:

- Clients can authenticate servers thrugh a certificate authority. Important as a client should not send credit card information to an unknown server.
- Servers can authenticate clients in the same way.
- Encrypted SSL connection (no plain text).

Two sub-protocols: SSL record protocol and SSL handshake protocol. SSL record format defines the format used to transmit data. SSL handshake protocol uses SSL record protocol to establish connection between SSL-enabled server and SSL-enabled client. Exchange of messages does the following:

- authenticate the server to the client
- allow client and server to select the cryptographic algorithms, or ciphers, they both support. Choices include DES, Digianl signature algorithm (DSA), MD5, RSA, RSA key exchange. Most common is RSA key exchange developed for SSL. Also FORTEZZA Cipher Suites for SSL 3.0.
- Optionally authenticate client to the server.
- Use public-key encryption techniques to generate shared secrets.
- Establish encrypted SSL connection.

SSL Handshake

Outline using one of Cipher suites previously given.

1. Client sends SSL version number, cipher settings, randomly generated data and other information server needs.
2. Server sends server SSL version number, cipher settings, randomly generated data, other information and servers own certificate. Server may optionally request client's certificate. Client authenticates server certificate by using public key of public key of certificate authority (CA). See Figure 2.
3. Client creates *premaster key* for session and encrypts it with servers public key (obtained from server's certificate) and sends to server.
4. Client sends encrypted data based on own private key if client needs authentication.
5. Server generates *master secret* after optional client authentication.
6. Both client and server use master secret secret to generate *session keys*, which are symmetric keys for encryption/decryption of exchanged information during SSL session.
7. Client and server inform each other session key has been created.
8. SSL handshake is complete.

Look at timings for cached and uncached key from IBM technical report.

Client needs to verify server's Domain Name (DN) to make sure a "man-in-the-middle" does not intercept messages and substitute its own keys.

Multimedia

"Holy grail of computing"—lots of technical challenges, lots of potential profits.

Generally concerned about *continuous media*—audio and video.

Particular characteristics of each in the text.

Data compression is important to reduce the amount of data sent.

Standards:
JPEG—used to compress still pictures.
MPEG—used to compress videos. Four kinds of frames (MPEG-1):

1. I (intracoded) frames: self-contained JPEG images
2. P (predictive) frames: block-by-block difference with last frame
3. B (bidirectional) frames: difference with last and next frame
4. D (DC-coded): block averages used for fast forward