# Computer Networks

## Chapter 07

prof. dr ir Maarten van Steen

Vrije Universiteit Amsterdam
Faculty of Science
Dept. Mathematics and Computer Science
Room R4.20. Tel: (020) 444 7784

steen@cs.vu.nl

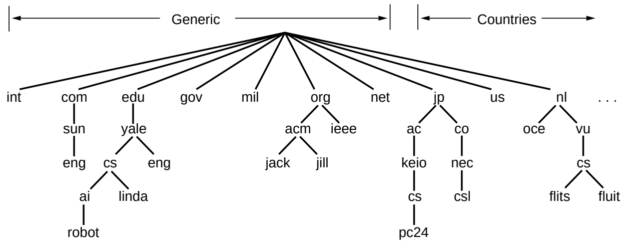
## Contents

# Application Layer

- Domain Name System
- Electronic Mail
- World Wide Web
- Multimedia

# The Domain Name System

**Basic idea:** Every host has a worldwide unique name that is bound to an IP address. DNS provides name-lookup facilities: when given a hostname, it returns that host's IP address.

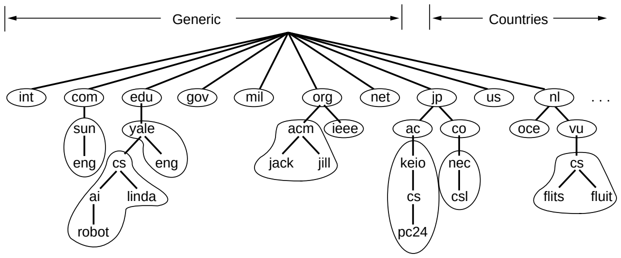


A **domain name** is a path from a leaf node up to the root. A **domain** is a subtree in the domain name space.

Each domain can have a set of **resource records** which are stored (in a file) at name servers.

# DNS Name Servers

**Basic idea:** Divide the name space into a collection of non-overlapping **zones**, and let each zone be taken care of by one or more **name servers**:



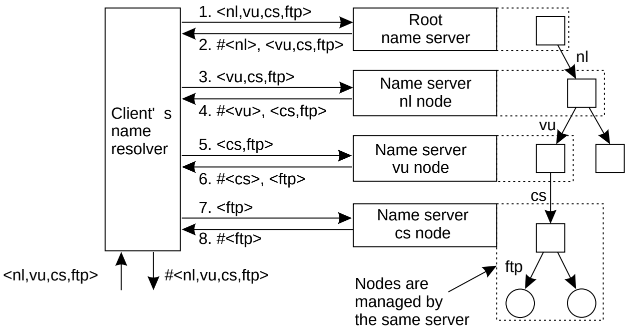
**Note:** There can be several servers per zone. Most of the time, *secondary masters* pull in their information from a *primary master*. The latter gets its info from domain administrators.

A **resolver** is capable of sending DNS queries to a name server. A resolver is often just a collection of library routines that can be linked to an application.

# Resource Records

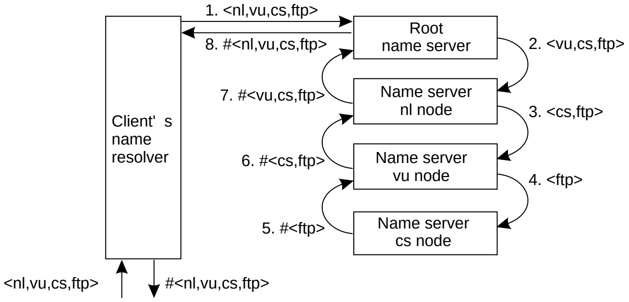
| **Type** | **Associated entity** | **Description** |
|---|---|---|
| SOA | Zone | Holds information on the represented zone |
| A | Host | Contains an IP address of the host this node represents |
| MX | Domain | Refers to a mail server to handle mail addressed to this node |
| SRV | Domain | Refers to a server handling a specific service |
| NS | Zone | Refers to a name server that implements the represented zone |
| CNAME | Node | Symbolic link with the primary name of the represented node |
| PTR | Host | Contains the canonical name of a host |
| HINFO | Host | Holds information on the host this node represents |
| TXT | Any kind | Contains any entity-specific information considered useful |

# DNS Iterative Name Resolution



**Note:** Name resolution can be **iterative**, in which the client repeatedly asks name servers to resolve part of a name.

# DNS Recursive Name Resolution



**Note:** With recursive resolution, a higher level server passes the query to a lower one instead of passing it back to the querying server.

**Question:** What is the main drawback of recursion?

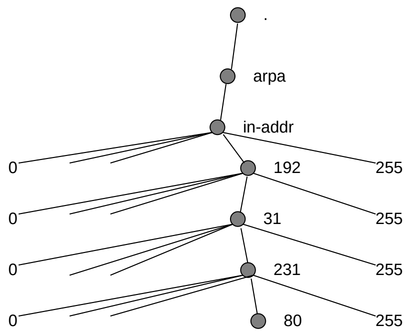
**Important:** The main reason why DNS works is that it assumes that name–to–address bindings hardly change. This means that we can effectively cache bindings locally, saving the trouble of having to go to the actual name server.

# DNS Address Resolution

**Guess what:** You can also find the name of a host when given its address:

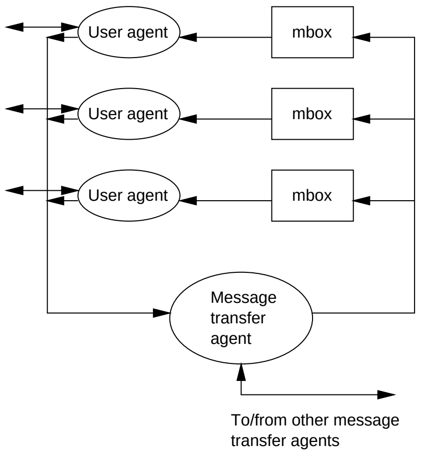
```
“% host  192.31.231.80
“% 80.231.31.192.in-addr.arpa domain name pointer veldersschuit.cs.vu.nl.
```

**Solution:** IP addresses are stored in the special in-addr.arpa domain:



# Electronic Mail

**Main issue:** Mail is really not that interesting: you just send mail to a mail server who takes care about delivery:



# Email: Message Transfer

**Basic idea:** The message transfer agent extracts the destination host from the message, and queries DNS to obtain the destination address. DNS keeps track of mailers in MX records:

```
cs.vu.nl mail is handled (pri=10) by zephyr.cs.vu.nl
cs.vu.nl mail is handled (pri=10) by tornado.cs.vu.nl
cs.vu.nl mail is handled (pri=20) by top.cs.vu.nl
cs.vu.nl mail is handled (pri=30) by solo.cs.vu.nl
```

**Example:** mail for steen@cs.vu.nl is first sent to the message transfer agent on zephyr.cs.vu.nl, then tornado, etc. Of course, zephyr is looked up as well.

The message transfer agent extracts the user and makes an attempt to deposit the incoming message into the user's mailbox. The user may then be notified.

**Note:** We're assuming that the user's mailbox is accessible for the agent, not necessarily the user. Protocols exist that allow a user to remotely access the mailbox.

# Email: Message Format (1/2)

**Basic idea:** Don't prescribe anything concerning the **content** of a message, but specify only the header:

| | |
|---|---|
| To: | e-mail address(es) main destination |
| Cc: | e-mail address(es) to send copies |
| Bcc: | e-mail address(es) to send blind copies |
| From: | name of sender(s) |
| Sender: | e-mail address sender |
| Received: | line added by each intermediate transfer agent |
| Return-path: | return address |

**Note:** The *From:* field is often the same as *Sender:*, so that the latter can be left out.

# Email: Message Format (2/2)

**Problem:** Too many e-mail messages have specific content that requires special applications to process. We therefore need information on content (meta-data).

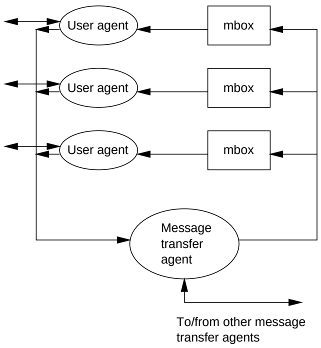
**Solution:** Extend header info with **MIME** fields (Multipurpose Internet Mail Extensions).

| | | |
|---|---|---|
| Text | Plain<br>Richtext | unformatted<br>formatted RTF |
| Image | Gif<br>Jpeg | GIF still<br>JPEG still |
| Audio | Basic | |
| Video | Mpeg | MPEG video |
| Application | Octet-stream<br>Postscript | binaries<br>printable doc |
| Message | RFC822<br>Partial<br>External-body | Embedded rfc822 msg<br>more to follow<br>provide URL |
| Multipart | Mixed<br>Alternative<br>Parallel<br>Digest | independent parts<br>same in diff. formats<br>view all at once<br>set of rfc822s |

# Email: Message Transfer

**SMTP:** (Simple Mail Transfer Protocol) *Really* simple: (1) set up TCP/IP connection between client and server; (2) client requests server to accept its messages; (3) server responds, so that client can send.

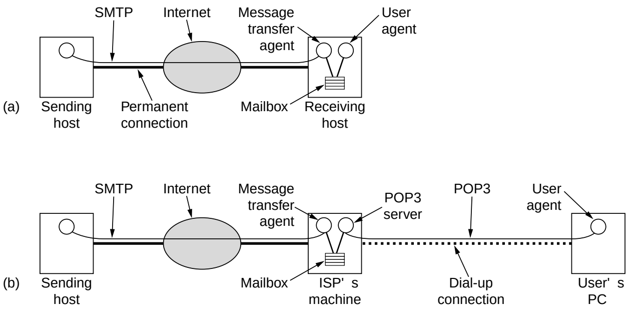
**Note:** Client is often a message transfer agent, but could also be the user agent.



**Question:** What's the drawback of letting the user agent contact a remote mail server?

# POP3 (1/2)

**Problem:** a user’s mailbox may be stored on a different machine than the user agent. We need remote access to incoming (and actually also outgoing) messages ⇒ **Post Office Protocol**:



# POP3 (2/2)

| Client | Server |
|---|---|
| | +OK POP3 server ready |
| USER carolyn | |
| | +OK |
| PASS vegetables | |
| | +OK login successful |
| LIST | |
| | 1 2505 |
| | 2 14302 |
| | 3 8122 |
| | . |
| RETR 1 | |
| | (sends message 1) |
| DELE 1 | |
| RETR 2 | |
| | (sends message 2) |
| DELE 2 | |
| RETR 3 | |
| | (sends message 3) |
| DELE 3 | |
| QUIT | |
| | +OK POP3 server disconnecting |

# IMAP

**Observation:** POP3 (implicitly) assumes that retrieved mail is deleted at the server. Not a good idea for people wanting to access mail from different computers. The **Internet Message Access Protocol** solves this problem.

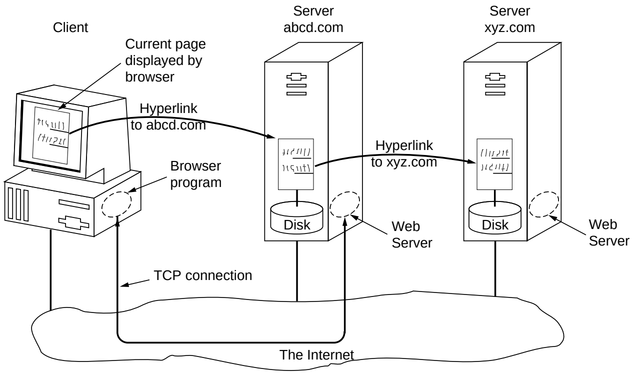
| **Feature** | **POP3** | **IMAP** |
|---|---|---|
| Where is e-mail stored | User’s PC | Server |
| Where is e-mail read | Off-line | On-line |
| Connect time required | Little | Much |
| Use of server resources | Minimal | Extensive |
| Multiple mailboxes | No | Yes |
| Who backs up mailboxes | User | ISP |
| Good for mobile users | No | Yes |
| User control over downloading | Little | Great |
| Partial message downloads | No | Yes |
| Are disk quotas a problem | No | Could be in time |
| Simple to implement | Yes | No |
| Widespread support | Yes | Growing |

# World Wide Web

**Basic model:** Users and organizations maintain pages of information that contain references to each other as **hyperlinks**. Selecting a link has the effect of pulling in the referenced page.

- The **client** has a Web browser that can display Web pages. Web pages are formatted in a special **markup language** which is interpreted by the browser. This allows for fancy typefonts and the like.
- A hyperlink identifies a (remote) Web server that has access to the referred Web page. When selecting a link, the browser establishes a TCP connection to the server, and the page is transferred to the user.
- A Web server listens for connection requests, accepts one request, returns the page over the connection, and closes it again. A connected client and server speak HTTP (HyperText Transfer Protocol).

# WWW – Client/Server Interaction

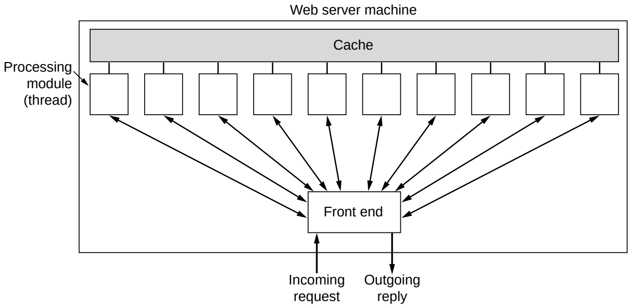


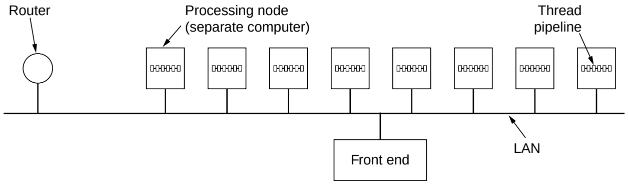
**Note:** When a client receives a new Web page, it can access other servers through the links contained in that new page.

**Note:** Each special feature in a page (i.e. in-line images) is transferred *separately* after the page has been copied to the client. This also means establishing and releasing several connections to the original server. Alternative solutions also exist.

# Web Servers

**Issue:** Most servers need to process many incoming requests. An often applied design is to use multithreaded servers, and sometimes even **server farms**:





# URL – Uniform Resource Locators

**Essence:** A **URL** contains three informative parts: (1) the name of a page, (2) the name of its location, (3) the access protocol:

| Name | Usage | Example |
|---|---|---|
| http | Hypertext | http://www.cs.vu.nl/˜steen/cn/ |
| ftp | File transfer | ftp://ftp.cs.vu.nl/pub/minix/README |
| file | Local file | file:/home/steen/www/cn/index.html |
| news | News group | news:comp.os.minix |
| news | News article | news:AA0134223112@cs.utah.edu |
| gopher | Gopher | gopher://gopher.tc.umn.edu/11/Libs |
| mailto | Sending email | mailto:kim@acm.org |
| telnet | Remote login | telnet://www.w3.org:80 |

**Disadvantage:** URLs contain location information: they refer to the location where a page is found. This makes it much harder to move pages around and to replicate them. In both cases, you don't care where the page is, but just that it has a worldwide unique name.

# Cookies

**Issue:** The Web is stateless: servers do not keep track of their clients. However, this may be (mis)useful in many cases. Solution: drop a **cookie** at the client side containing server state relevant for that client.

| Domain | Content |
|---|---|
| toms-casino.com | CustomerID=497793521 |
| joes-store.com | Cart=1-00501;1-07031;2-13721 |
| aportal.com | Prefs=Stk;SUNW+ORCL;Spt:Jets |
| sneaky.com | UserID=3627239101 |

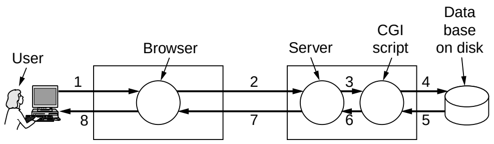
**Note:** There is also an *Expires* field; The *Secure* field indicates that a cookie may be returned only to a secure server.

**Basic idea:** When a browser contacts a server, a related cookie is sent to the server, after which a page can be displayed relevant to that cookie/client.

**Question:** How secure are cookies?

# Dynamic Web Pages

**Essence:** Instead of storing and returning statically defined Web pages, servers often generate pages on-the-fly:

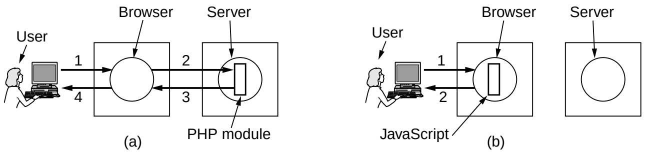


1. User fills in form
2. Form sent back
3. Handed to CGI
4. CGI queries DB
5. Record found
6. CGI builds page
7. Page returned
8. Page displayed

**Common Gateway Interface:** CGI essentially allows you to identify a program and its parameters in a URL. The server will start a process to execute that program, which, in turn, will return its results (if any) as a regular Web page.

# Scripting and Web Pages

**Alternative approach:** Let Web pages incorporate interepretable code; when a page is being processed, the embedded script is simply executed. Distinguish server-side and client-side solutions:



# HTTP (1/2)

**Essence:** Communication in the Web is generally based on the **HyperText Transfer Protocol**; a relatively simple client-server transfer protocol having the following request messages:

| Operation | Description |
|---|---|
| Head | Request to return the header of a document |
| Get | Request to return a document to the client |
| Put | Request to store a document |
| Post | Provide data that are to be added to a document (collection) |
| Delete | Request to delete a document |

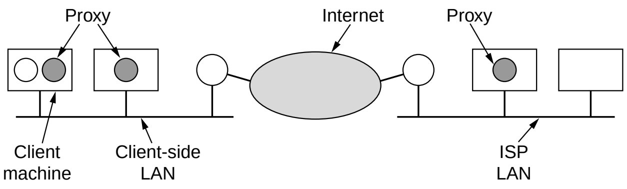
# HTTP (2/2)

| Header | C/S | Contents |
|---|---|---|
| Accept | C | The type of documents the client can handle |
| Accept-Charset | C | The character sets are acceptable for the client |
| Accept-Encoding | C | The document encodings the client can handle |
| Accept-Language | C | The natural language the client can handle |
| Authorization | C | A list of the client's credentials |
| Date | C+S | Date and time the message was sent |
| ETag | S | The tags associated with the returned document |
| Expires | S | The time for how long the response remains valid |
| From | C | The client's e-mail address |
| Host | C | The server's DNS name |
| If-Match | C | The tags the document should have |
| If-None-Match | C | The tags the document should not have |
| If-Modified-Since | C | Tells the server to return a document only if it has been modified since the specified time |
| If-Unmodified-Since | C | Tells the server to return a document only if it has not been modified since the specified time |
| Last-Modified | S | The time the returned document was last modified |
| Location | S | A document reference to which the client should redirect its request |
| Referer | C | Refers to client's most recently requested document |
| Upgrade | C+S | The application protocol sender wants to switch to |
| Warning | C+S | Information about status of the data in the message |

# WWW – Proxy Servers

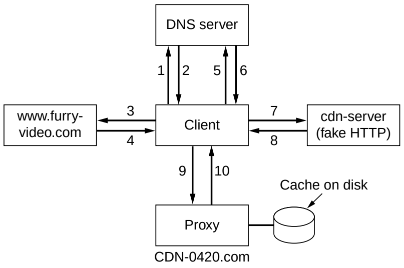
**Original idea:** It is also possible to transfer information referenced by a different transfer protocol (notably FTP). If the browser does not speak that protocol, a proxy was used.

**Now:** Proxies are mainly used to enhance Web performance by caching responses. The proxy cache simply keeps previously requested pages and returns them on the next request.



# Content Delivery Networks

**Basic idea:** Install a bunch of servers across the Internet and simply replicate Web pages on those servers. Be sure to redirect clients to the nearest replica server.

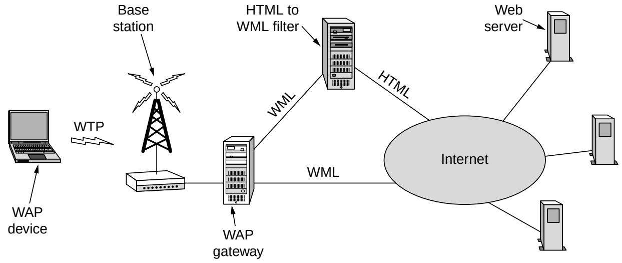


1. Look up www.furryvideo.com
2. Furry' s IP address returned
3. Request HTML page from Furry
4. HTML page returned
5. After click, look up cdn-server.com
6. IP address of cdn-server returned
7. Ask cdn-server for bears.mpg
8. Client told to redirect to CDN-0420.com
9. Request bears.mpg
10. Cached file bears.mpg returned

**Example:** looking up *www.furry-video.com* which is a page containing references to replicated web pages (identified as *http://cdn-server.com/...*).

# Wireless Application Protocol (1/2)

**Essence:** A simple protocol that allows mobile devices to talk to Web servers over a low-bandwidth connection. The original version assumed a circuit-switched connection to a server (i.e., call your server using you GSM).



**Problems:** low bandwidth, costly connections, too much conversions necessary on the server side (pages need to be in WML/XML).

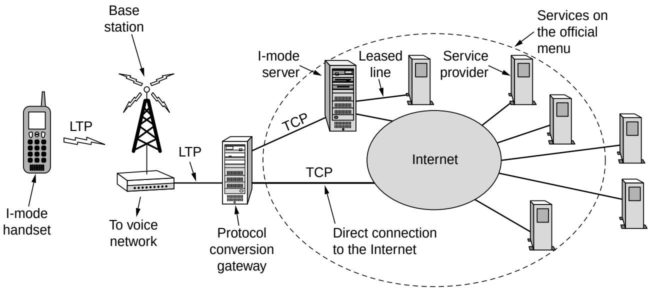
# Wireless Application Protocol (2/2)

However, Things are going to change:

| **WAP 1.0** | **WAP 2.0** |
|---|---|
| WSP (sessiom) | HTTP |
| WTP (transactions) | |
| WTLS (security) | TLS |
| | TCP |
| WDP (datagram) | IP |
| Bearer layer | Bearer layer |

# I-mode

**Essence:** Rather than making use of wireless telephone, I-mode transfers data packets across (new) wireless packet-switching networks (128-byte packets at 9600 bps). Voice goes over the usual circuit-switched wireless connections. Existing networks (GPRS) are to be supported as well. The I-mode devices are pretty sophisticated.



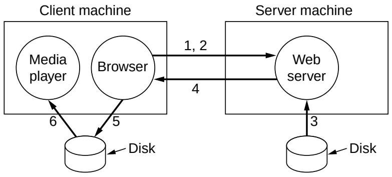
# Comparison

| **Feature** | **WAP** | **I-mode** |
|---|---|---|
| What it is | Protocol stack | Service |
| Device | Handset, PDA, etc. | Handset |
| Access | Dial up | Always on |
| Underlying network | Circuit-switched | Circuit+packet |
| Data rate | 9600 bps | 9600 bps |
| Screen | Monochrome | Color |
| Markup lang. | WML | cHTML |
| Scripting | WMLscript | None |
| Usage charges | Per minute | Per packet |
| Pay for shopping | Credit card | Phone bill |
| Pictograms | No | Yes |
| Typical user | Businessman | Young woman |

# Multimedia

**Issue:** The Internet is by-and-large turning partly into an infrastructure for broadcasting multimedia streams. Streams consist of packets containing samples of audio and video, possibly augmented with data (such as used for subtitles or meta-information).

**Streaming audio:** The simplest way to handle this type of streaming:

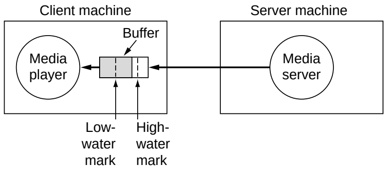


1. Establish TCP connection
2. Send HTTP GET request
3. Server gets file from disk
4. File sent back
5. Browser writes file to disk
6. Media player fetches file block by block and plays it

**Problem:** There’s no real streaming; instead a (potentially very large) file is sent to the client for playback.

# Real Time Streaming Protocol

**Better solution:** Let the browser start an **media player** application that sets up a connection to the server, which in turn starts streaming packets to the player. **Note:** we are using RTP on top of UDP. To reduce jitter, buffers are used:



| Command | Description |
|---|---|
| DESCRIBE | List media parameters |
| SETUP | Establish channel |
| PLAY | Start sending data to client |
| RECORD | Start accepting data from client |
| PAUSE | Temporarily stop sending data |
| TEARDOWN | Release channel |

# Voice over IP

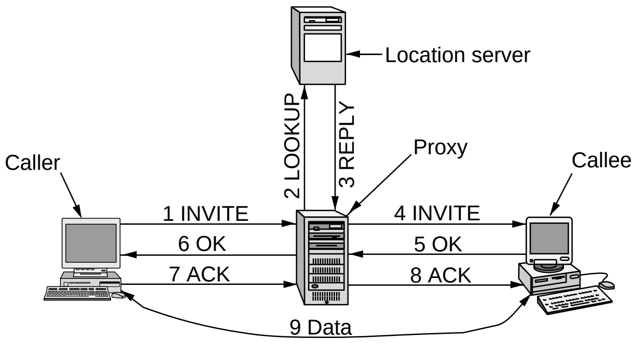
**Essence:** Instead of using traditional circuit-switching technology, it may pay off to send digitized voice over a packet-switching network. Two groups defined a standard. ITU (the telcos) invented H.323. It's big, as all other ITU protocols are. The IETF invented SIP. It's simpler, like most IETF protocols.

**Session Initiation Protocol:** It deals only with setup, management, and termination of sessions. The data transport should use other protocols like RTP.

| **Method** | **Description** |
|---|---|
| INVITE | Request initiation of a session |
| ACK | Confirm initiation of session |
| BYE | Request termination of session |
| OPTIONS | Query a host about capabilities |
| CANCEL | Cancel a pending request |
| REGISTER | Inform redirection server about current location |

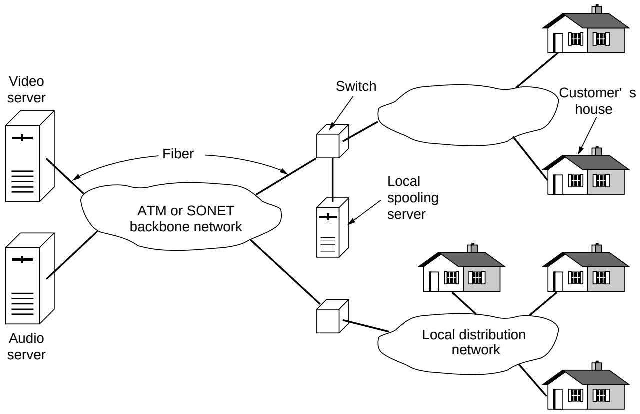
SIP is text-based and runs on UDP as well as TCP. Users are addressed through, for example, *telephone* URLs.

# SIP in Practical Use



# Video-on-Demand (1/2)

**Two models:** (1) The user can request any video, and stop the incoming stream at any time. (2) The provider broadcasts popular videos, but starts several at short intervals (near video on demand).



# Video-on-Demand (2/2)

**Note:** The video server can be pretty hot stuff: it requires *a lot* of storage, which will include tape archives, optical disks (as in a juke box), and arrays of magnetic disks (RAIDs).

**Note:** The client will have a **set-top box**, which is just a simple computer that does the (MPEG) decoding, has an interface to the network, and controls the TV set (monitor & remote control).

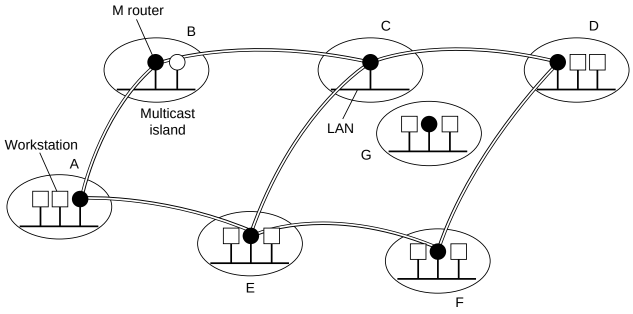
**Big problem:** How are we ever going to get the bandwidth to our homes (the local loop problem).

**Really big problem:** There are so many possible solutions, and all of them cost a lot. What we'll need is a convergence of solutions to standards. This means a lot of experimentation in the coming years, much of which is already going on (ADSL, Fiber-to-the-house, etc.)

# MBone: Multimedia on the Internet

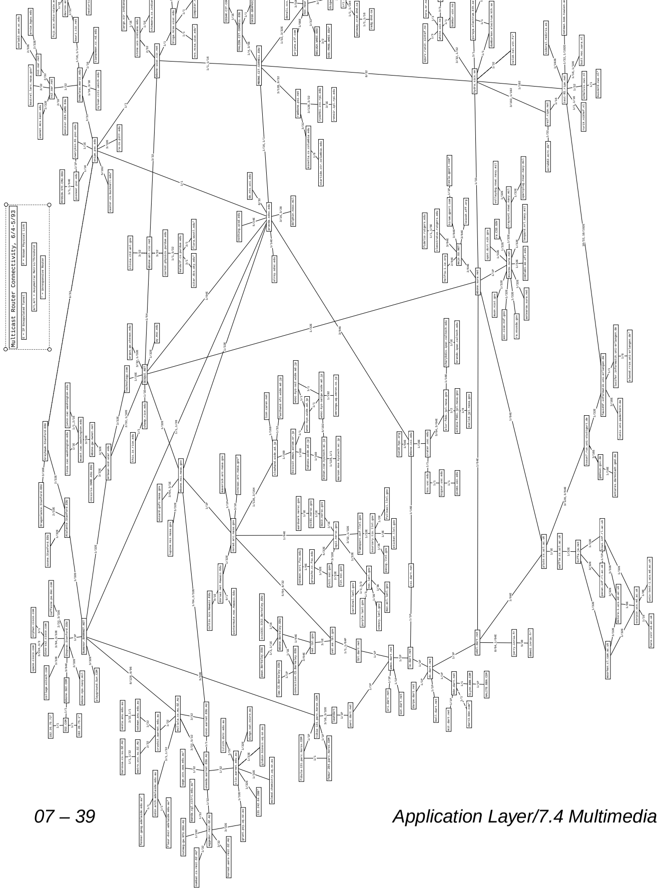
**Back to basics:** Let's do it the Internet way and just a hack a simple multimedia subsystem on top of what we already have: a worldwide data network.

**Basic idea:** Construct an **overlay network** on top of the Internet, where each node constitutes a **multicast island**. Islands communicate by tunneling packets. Construct multicast trees on the overlay network:



# MBone: Routing

- Each island has one or more **multicast routers**, which form the end points of the tunnels.
- We start a broadcast by picking a class D IP address (reserved for multicasting) and multicast it on our island. The mrouter picks it up and forwards it onto its tunnels.
- Incoming packets at an mrouter are forwarded along a spanning tree using reverse shortest path first. This means that only those packets are forwarded that came in along the best route to the source.
- From time to time, mrouters query the hosts on their island to figure out which broadcasts they are interested in. This information is also used to prune multicast trees on the MBone.
- **Data:** On 23.02.1999, there were a total of 4178 mrouters.


Multicast Router Connectivity, 6/4-5/93
E = IP Encapsulated Tunnel
P = Known Physical Link
m/t,m/t = Assymetric Metric/Threshold
* = Unresponsive Router