

“Application Layer: HTTP and Cookies”

Table of Contents

- Readings
- Principles of Network Applications
 - Network Application Architectures
 - Processes Communicating
 - Interface between process and network
 - Addressing processes
 - Transport Services
 - TCP services
 - SSL
 - UDP Services
 - Application Layer Protocols
- The Web and HTTP
 - HTTP: HyperText Transfer Protocol
 - Non-persistent and persistent connections
 - HTTP Request Message
 - HTTP Response Message
 - Cookies
 - Web Caching
 - Static Web Documents
 - Dynamic content
- File Transfer Protocol: FTP
 - FTP Commands and Replies
- Email
 - SMTP
 - SMTP Commands
 - SMTP vs HTTP
 - Mail message header
 - MIME - Multipurpose Internet Mail Extensions
 - Mail access protocols
- DNS

- DNS Components
- Domain name characteristics
- Database: Resource Records
- Inserting records into DNS
- Types of name servers
- Resolving queries
- DNS Messages
- DNS Caching
- DNS Security

Readings

- ☒ K&R 2.1
- ☒ K&R 2.2
- ☒ K&R 2.3
- ☒ K&R 2.4
- ☐ K&R 2.6

Principles of Network Applications

Network Application Architectures

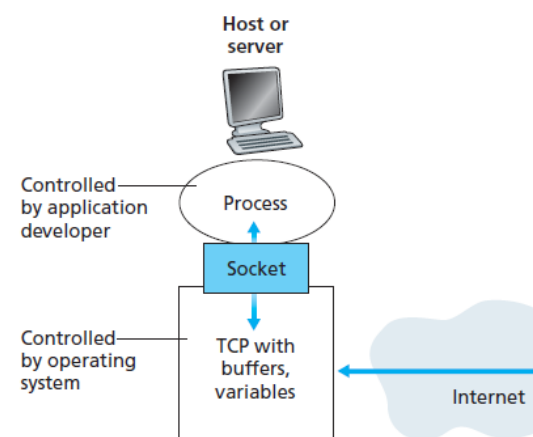
- **client-server:** always on *server* host services requests from many other *client* hosts
 - clients do not directly communicate
 - server has static IP address
 - e.g. Web, FTP, Telnet, email
 - data centres with multiple hosts provide powerful server to handle large volume of requests
- **peer-to-peer (P2P):** direct communication between intermittently connected *peer* hosts
 - peers are not owned by service providers but by end users
 - e.g. BitTorrent, Skype
 - self scalable: peers introduce workload with requests but also add service capacity through file distribution etc
 - cost effective: minimal server infrastructure/bandwidth

Processes Communicating

- **process**: instance of a program running in an end system
- processes on different hosts communicate by exchanging **messages** across the network
- in context of communication session between two processes:
 - *client process* initiates communication
 - *server process* waits to be contacted

Interface between process and network

- **socket**: software interface that allows a process to send/receive messages from the network
 - aka API between application and network, as socket is programming interface with which network apps are built
 - app developer has control over everything app side of socket, very little control transport-layer side:
 - * may have a choice of protocol (UDP/TCP)



- * may have ability to set some transport-layer parameters

Addressing processes

To communicate with a process on a remote host, you need: - **IP address** identifying a host - IPv4: 32-bit - **port number** specifies receiving process in destination host - e.g. HTTP: 80, SMTP: 25

Transport Services

Can be classified by these dimensions: - reliable data transfer - guaranteed data delivery - without it app needs to be loss-tolerant - provided by TCP - throughput - specification of bits/sec required - elastic applications make use of whatever throughput is available - not provided by Internet transport protocols - security - encryption/decryption - data integrity - end-point authentication - TCP with SSL - timing - e.g. guarantee that every bit pumped into the socket is received within 100ms - e.g. applications: telephony, gaming - not provided by Internet transport protocols

TCP services

- **connection oriented service:** handshake to set up connection
 - **full-duplex:** two processes can send messages over the connection simultaneously
 - connection must be torn down once finished
- **reliable data transfer service:** communicating processes can rely on TCP to deliver all data sent without error and in proper order
- **congestion control:** serves Internet as a whole rather than communicating processes
 - throttles a sending process when network is congested
 - attempts to allocate fair share of bandwidth

SSL

- TCP and UDP have no built-in encryption
- **Secure Sockets Layer (SSL):** TCP enhancement providing encryption, data integrity, end-point authentication
 - not a transport protocol, but an enhancement in residing in application layer
 - to use SSL, you need to include the code in your application
 - similar API to TCP, but before the transmission occurs it is first encrypted, then passed to TCP socket

UDP Services

- lightweight transport protocol with minimal services
- connectionless
- unreliable data transfer service:

- no guarantee of delivery
- messages may arrive out of order
- no congestion control

Application Layer Protocols

- **application-layer protocol:** defines how application's processes on different hosts pass messages, in particular
 - type of messages e.g. requests/responses
 - syntax of message types: fields, delimiters
 - semantics: what values of fields means
 - rules determining when/how process sends/responds to messages
- application layer protocol \neq network application; the protocol is one part of the application
 - e.g. the Web
 - * application: includes standard for document formats (HTML), web browsers, web servers, application-layer protocol
 - * protocol: HTTP

The Web and HTTP

- before the Web, the Internet was used primarily by researchers, academics, university students primarily to transfer files, receive news, send email
- early 90s saw introduction of the Web, and general public was now using the Internet
 - on demand content
 - easy/low cost to publish content

HTTP: HyperText Transfer Protocol

- HTTP 1.0 RFC1945
- HTTP 1.1 RFC2616
- HTTP 2.0 RFC7540
- Web's application-layer protocol
- implemented on client program and server program, on distinct hosts, which communicate via HTTP messages

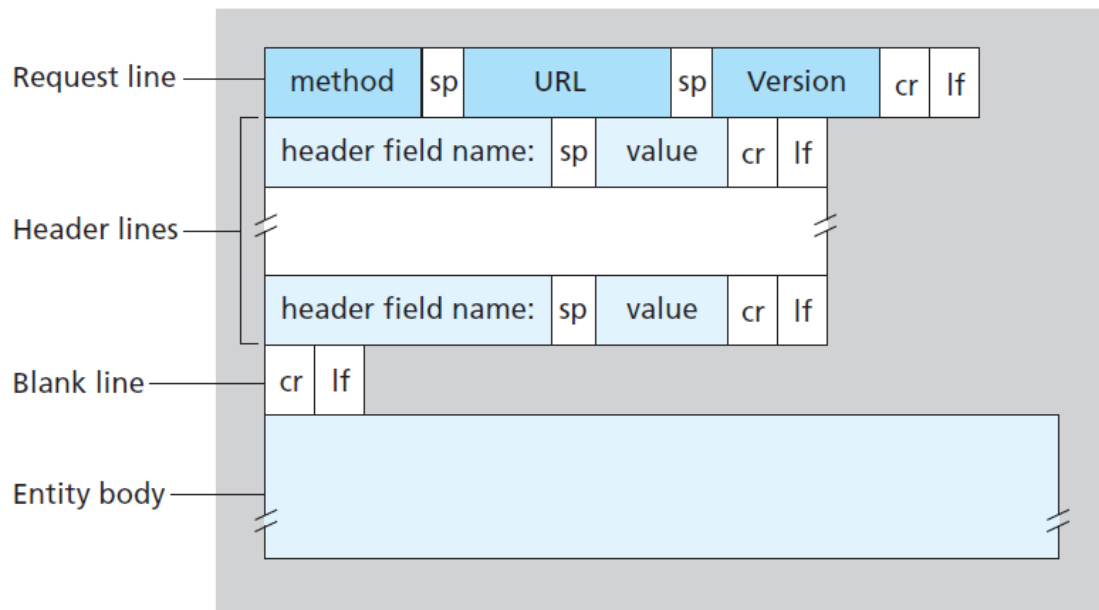
- HTTP defines structure of the messages and how the client/server exchange messages
- **web page** consists of **objects** - a file addressable by a single URL
- most web pages: **base HTML file** + several objects (images, CSS, javascript, ...) referenced by base HTML file
- once HTTP client sends message into socket interface, it is out of hands of client and in hands of TCP
- **stateless protocol**: HTTP server maintains no info about clients

Non-persistent and persistent connections

- decision by app developer whether to use **persistent/non-persistent connection**
- **non-persistent connections**
 - TCP connection needs to be established for each object: lots of overhead and significant burden on the server
 - could use parallel TCP connections (typically up to 5-10) handling individual request-response transaction
- **three-way handshake**: each step involves transfer of TCP segment
 - client: requests connection
 - server: responds with acknowledgement
 - client: acknowledges connection + HTTP request
- **persistent connection**: server leaves TCP connection open after sending response
 - **pipelining**: back-to-back requests made without waiting for replies to pending requests; speeds up transfer
 - multiple web pages residing on the same server can be sent via a single persistent TCP connection
 - server closes connection after a timeout interval
 - default mode: persistent connections with pipelining

HTTP Request Message

- **User-agent**: specifies browser type making request, allowing server to provide different versions of the same object depending on the user agent
- **Host**: necessary for Web proxy caches

**Figure 1:** http_request

HTTP Response Message

Consists of - status line - header lines - entity body: requested object itself

Cookies

- **cookies** allow sites to keep track of users to identify users, either to restrict access or serve tailored content
- place small amount of info (<4kB) on user's computer,
- fields: domain: server the cookie belongs to, path, content, expiry, security
- HTTP messages carry state
- components
 - cookie header line in HTTP response message
 - cookie header line in HTTP request message
 - cookie file kept on user's end system, managed by the browser
 - back-end database server-side
- when you access a site, it may respond with a Set-cookie: , with that id and the server hostname being appended to a cookie file. When you make HTTP requests this id is added to the header,

and the server uses it for some cookie-specific action, such as maintaining intended purchases.

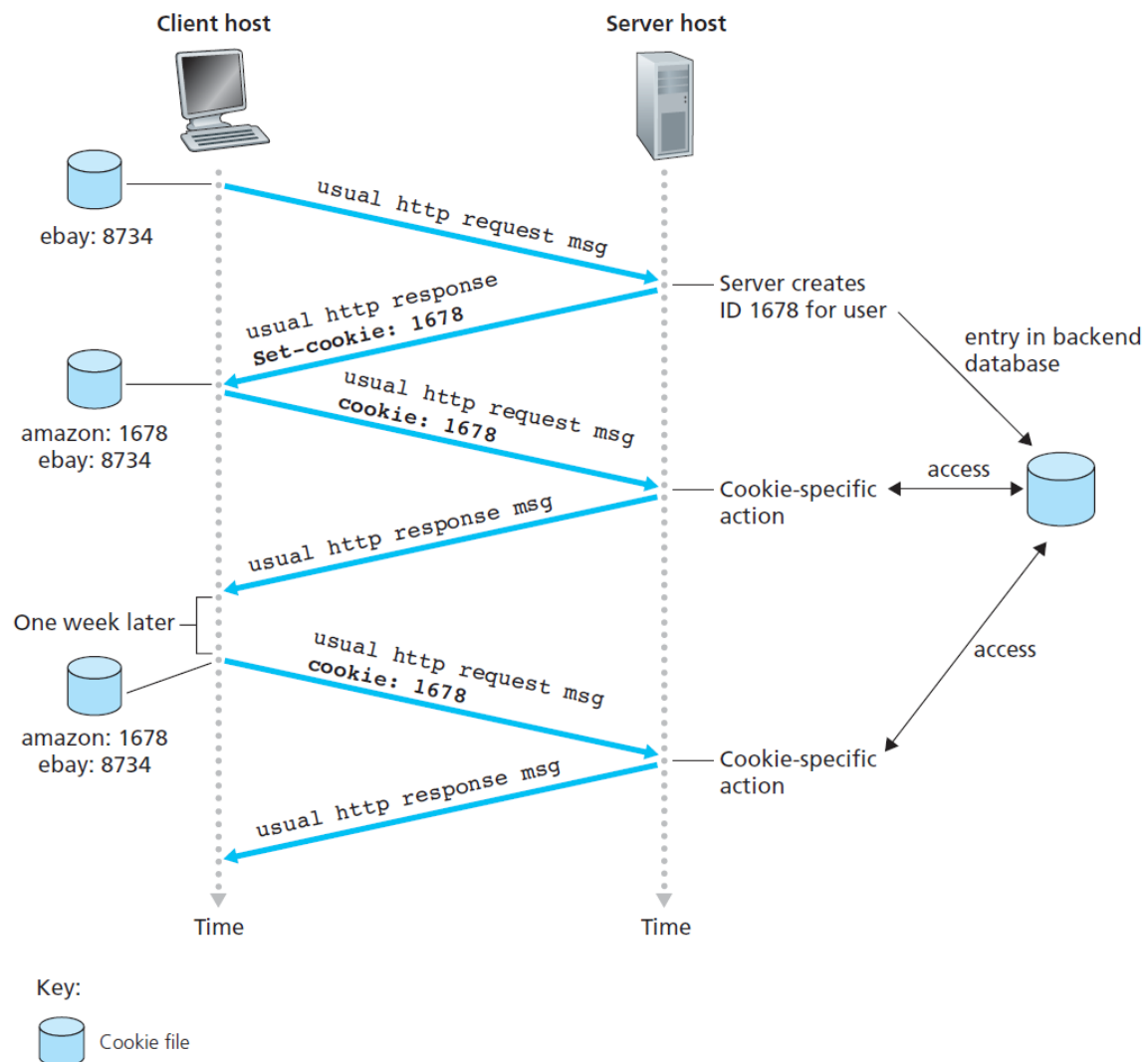


Figure 2.10 ♦ Keeping user state with cookies

Figure 2: cookies

Web Caching

- **web cache/proxy server:** network entity satisfying HTTP requests on behalf of origin web server
 - has storage on which it caches recently requested objects

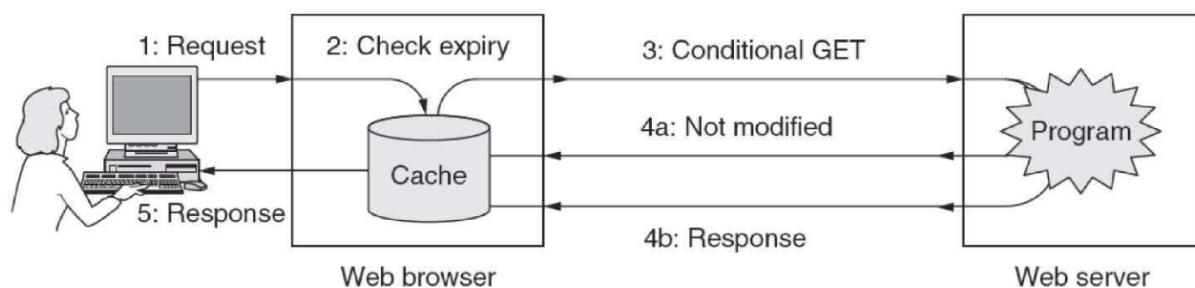
- browsers can be configured so that HTTP requests are first directed to the web cache
- if the web cache doesn't have a copy of that object it requests it from the origin server
- typically installed by an ISP or e.g. university campus

- **benefits:**

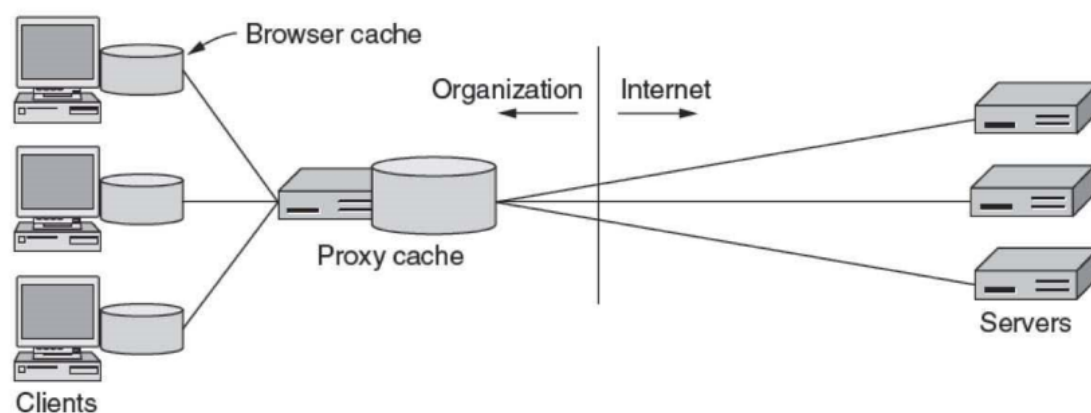
- reduces response times for client requests if there is a high speed connection from client-cache c.f. client-origin
- substantial reduction in traffic on access link (e.g. within campus), using less bandwidth
- substantial reduction in Internet traffic, improving performance for all applications

- **conditional GET:** HTTP mechanism to verify objects are up to date

- adds an **If-Modified-Since:** header line to a GET request
- cache will issue this request if it has a cached object, using the **Last-Modified:** header line from the response header
- if the object has not been modified, the server responds with a 304 **Not Modified** and an empty entity body
- conditional GET saves bandwidth and increases end user response times



Conditional get from browser cache



Proxy server for cacheing, security, IP address sharing

Static Web Documents

- **hypertext markup language (HTML)**
 - plain text encoding, browser rendering
- components of a web page
 - Head
 - ...
 - Body
 - ...
 - Attributes and values
 - hyperlinks/anchors
 - cannot nest tags, < and > can be in argument strings
 - cannot mis-nest text \Rightarrow text

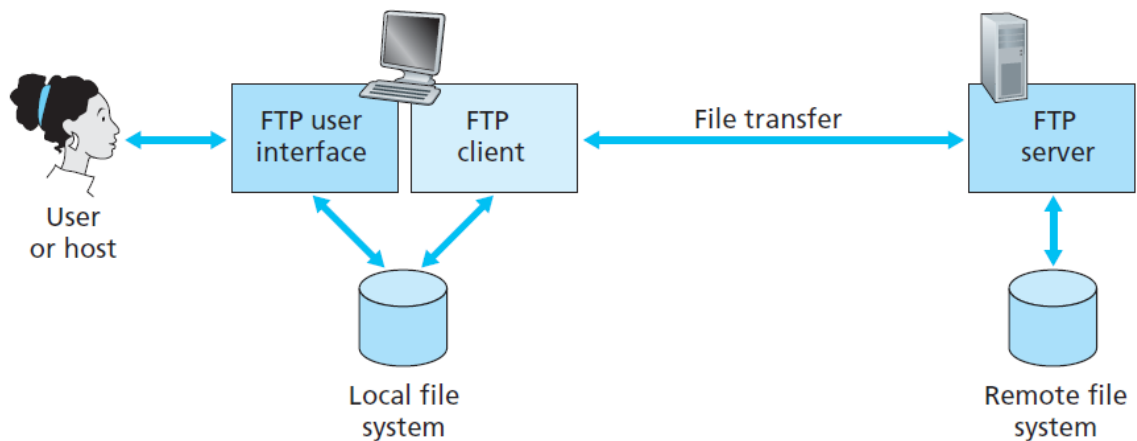
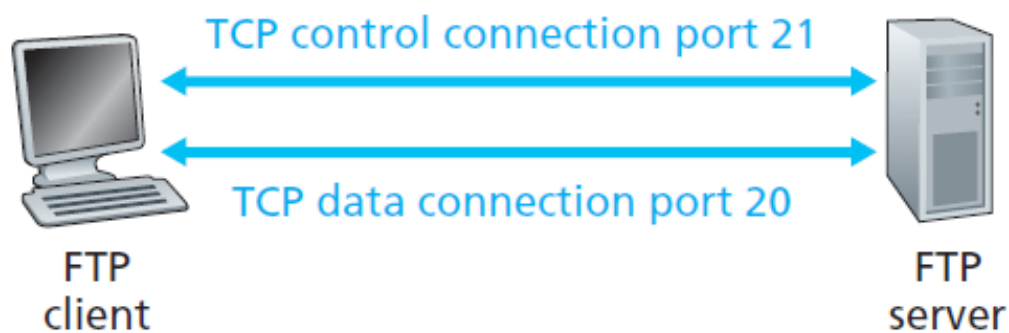
Dynamic content

- server side: PHP script
- client side: e.g. JavaScript, AJAX
 - script sent to client

File Transfer Protocol: FTP

- local host wants to transfer files to/from remote host
- user provides remote hostname + authentication and then can transfer files using an FTP user agent
- runs on TCP, but uses two parallel connections to transfer a file
 - **control connection**: sending information e.g. credentials, put/get commands
 - * this is **out-of-band**
 - * port 21
 - **data connection**: send actual files
 - * this is **in-band**
 - * port 20
 - * one connection per file transfer
- user **state** maintained:

- control connection associated with an account
- current directory
- need to keep track of state info constrains total number of simultaneous sessions

**Figure 3:** ftp**Figure 4:** ftp_connections

FTP Commands and Replies

- RFC 959
- commands/replies sent across control connection in 7-bit ASCII format

- successive commands delimited by
- commands are 4 uppercase characters, some with arguments
 - USER `username`
 - PASS `password`
 - LIST: file listing in current directory
 - RETR `filename`: retrieve file from current directory of remote host
 - STOR `filename`: put file
- replies: 3 digit numbers with optional message
 - 331 Username OK, password required
 - 125 Data connection already open; transfer starting
 - 425 Can't open data connection
 - 452 Error writing file

Email

- high level view
 - **user agents**: e.g. Microsoft Outlook, allows users to read/compose/etc. email
 - **mail servers**: user **mailboxes** stored on the mail server
 - * manages and maintains messages sent to him
 - * authenticates users
 - * attempts to deliver message to recipient's mail server. This goes in **message queue** and stays on the senders queue until successfully sent. User is notified if there is no success after several days
 - **Simple Mail Transfer Protocol (SMTP)**: principal application-layer protocol for Internet e-mail
 - * uses TCP for reliable data transfer between mail servers

SMTP

- defined in RFC5321, first published 1982, but protocol was around much earlier
- port 25
- much older than HTTP, has archaic characteristics
 - e.g. restricts message body to 7-bit ASCII: binary media data has to be encoded in ASCII then decoded back to binary

- does not normally use intermediate mail servers for sending mail: TCP connection is direct between mail servers
- when sending mail:
 - user submits mail to client mail server
 - client mail server establishes TCP connection with recipient mail server
 - SMTP handshake: identifies sender address and recipient address
 - client sends message
 - process is repeated if there are other messages to send to that server, otherwise the con-

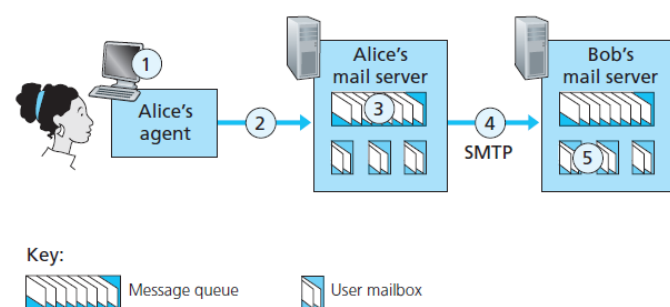


Figure 2.15 ♦ Alice sends a message to Bob

nection is closed (persistent connections)

SMTP Commands

- **HELO:** Hello; initiate handshake
- **MAIL FROM:** sender email
- **RCPT TO:** recipient email
- **DATA:** email message, terminated with period .
- **QUIT:** close connection

SMTP vs HTTP

- HTTP transfers objects (files) from Web server to Web client
- SMTP transfers messages (files) from mail server to mail server
- HTTP mainly **pull protocol**: someone loads information, HTTP used to pull at convenience
 - TCP connection initiated by machine that *wants to receive* file
- SMTP mainly **push protocol**: sending mail server pushes file to receiving mail server
 - TCP connection initiated by machine that *wants to send* file

- SMTP requires ASCII message, HTTP does not
- HTTP encapsulates each object in an individual HTTP response message
- SMTP combines all message objects into one message

Mail message header

- distinct from SMTP handshake
- headers are separated by <CR><LF>
- header ends with a blank line i.e. <CR><LF>
- mandatory header lines:

```
1 From: abc@xyz.com
2 To: ijk@bbb.com
3 Subject: searching for the meaning of life
```

- other header lines:

```
1 CC:
2 Bcc:
3 Message-Id:
4 In-Reply-To: <- ID of message you are replying to
5 Reply-To:
6 ...
```

MIME - Multipurpose Internet Mail Extensions

- originally email messages only used ASCII. Mime introduced to support:
 - other languages
 - alternative message content types (audio, images)
- 5 additional message headers:
 - MIME-Version
 - Content-Description: human readable description
 - Content-Id
 - Content-Transfer-Encoding: how body is wrapped for transmission
 - Content-Type: type/format of content

Mail access protocols

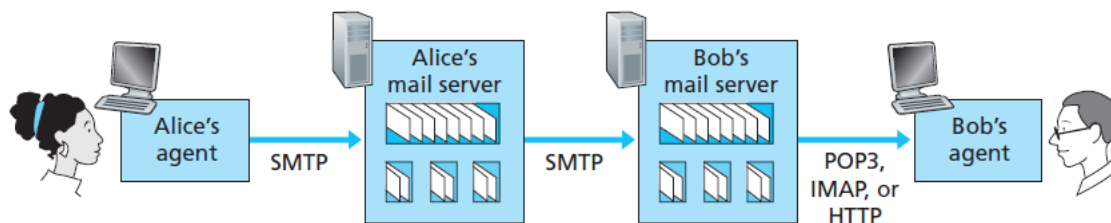


Figure 5: mail_access_protocols

- used to access mailbox from user's mail server
- cannot use SMTP to pull mail as it is a push protocol
- options:
 - **POP3 (Post office protocol - version 3):** simple, limited functionality
 - * user agent opens TCP connection on port 110 of mail server
 - * 3 phases:
 - authorisation phase
 - transaction phase: user agent retrieves messages, mark messages for deletion etc.
 - update phase: ends POP3 session, then mail server deletes messages marked
 - * does not maintain state between POP3 sessions
 - **IMAP: Internet Mail Access Protocol**
 - * nomadic user wants to maintain state e.g. folders of mail across different devices
 - * IMAP provides ability to
 - add folders
 - move mail to folders
 - search remote folders
 - obtain component of a message (e.g. message header) saving bandwidth
 - * IMAP server maintains state across folders
 - HTTP
 - * Hotmail introduced Web based email access in 1990s
 - * user agent is an ordinary browser
 - * messages retrieved from mail server via HTTP

DNS

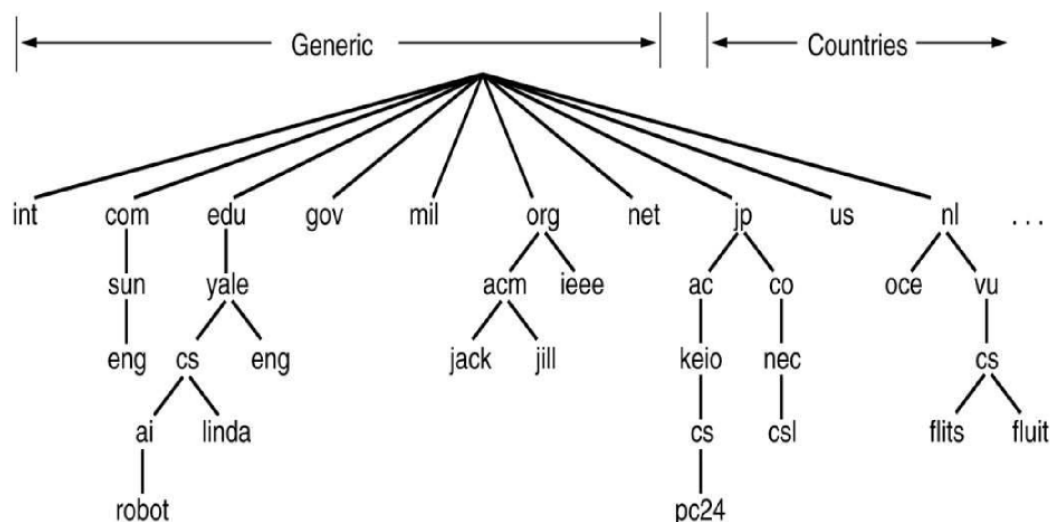
- **hostname:** human-readable identifier of Internet host
 - provides little info on location of host on Internet
 - variable length strings: difficult to process by routers
- **IP addresses:** “machine-readable” host identifier
 - IPv4 address: 4 bytes $a.b.c.d$, $a, b, c, d \in [0, 255]$
 - hierarchical: address from left to right is increasingly specific about where host resides in Internet
- **domain name system (DNS):** directory service translating hostnames to IP addresses
 - distributed database implemented in a hierarchy of DNS servers
 - application-layer protocol that allows hosts to query this database
 - servers are typically UNIX machines running Berkeley Internet Domain Name (BIND) software
 - uses **UDP, port 53**
 - commonly used by other application-layer protocols (e.g. HTTP, SMTP) to translate user-supplied hostnames to IP addresses
 - IP addresses are often cached in nearby servers to reduce DNS traffic and latency
 - unlike HTTP, FTP, SMTP; DNS is not intended for end user use
- other services DNS provides
 - **host aliasing:** resolve to **canonical hostname** and corresponding IP address from multiple aliases
 - **mail server aliasing:** useful to have mnemonic mail server address, which can also be identical to Web server hostname
 - **load distribution** among replicated servers by providing round robin response of IP address
 - why DNS is distributed not centralised:
 - * single point of failure: if DNS failed entire Internet would cease working
 - * traffic volume
 - * distant centralised database: e.g. Australian queries directed to America would introduce significant latency
 - * maintenance
 - * in summary: it doesn't scale

DNS Components

[TODO] - domain name space: - DNS database - name servers - resolvers:

Domain name characteristics

- case insensitive
- ≤ 63 chars per constituent
- ≤ 255 chars per path
- can be internationalised: introduces security problems as people can repeat domain name



- opened up in 2014 to allow e.g. [.accenture](#)

Database: Resource Records

- resource records carried by DNS replies
 - 4-tuple: ([Name](#), [Value](#), [Type](#), [TTL](#))

| Type | Value |
|------|--|
| A | IPv4 address for hostname Name |
| AAAA | IPv6 address for hostname Name |
| NS | Hostname of authoritative DNS server for domain Name |

| Type | Value |
|-------|---|
| CNAME | Canonical hostname for alias hostname Name |
| MX | Mail exchange. Canonical name of a mail server. Allows company to have same aliased name for mail and Web |

- Authoritative DNS server for a particular hostname contains corresponding A record
- Non-authoritative server for a given hostname: contains a NS record for domain that includes the hostname
 - also contains A record that provides IP address of the DNS server referenced in the NS record
- Can use multiple A records for a single domain name to balance traffic across multiple servers

Inserting records into DNS

- **DNS registrar** provided with names, IP addresses of authoritative name server
 - ensures uniqueness of the domain name
 - inserts two resource records into TLD server
- create authoritative server: [TODO]

Types of name servers

Hierarchy - **Root DNS servers**: managed by 13 different organisations, with ~1100 (at 2020-03-12) distinct server instances around the world - provide IP address of TLD servers - IANA list of root servers - a.root-servers.net - 13 root servers due to old DNS infrastructure + IPv4: IP addresses needed to fit in single packet of 512 bytes = $13 * (32 \text{ bytes}) + (96 \text{ bytes for protocol info})$ - **Top-level domain DNS servers**: server clusters for - top-level domains e.g. com, edu, org, net - country top-level domains e.g. au, fr - provide IP address of Authoritative DNS servers - **Authoritative DNS servers**: houses DNS records that map hostnames to IP addresses - publicly accessible hosts e.g. web servers, mail servers, provide these records - organisations can implement their own or pay to host records with a service provider that implements authoritative DNS server - **Local DNS server**: not part of hierarchy - each ISP has a local DNS server (default name server) - when host connects to an ISP, ISP provides IP addresses of 1+ local DNS servers - local DNS server is typically close to the host - local DNS server acts as a proxy, forwarding DNS queries into server hierarchy - ISP has default name server that handles DNS queries - local DNS server acts as proxy

Resolving queries

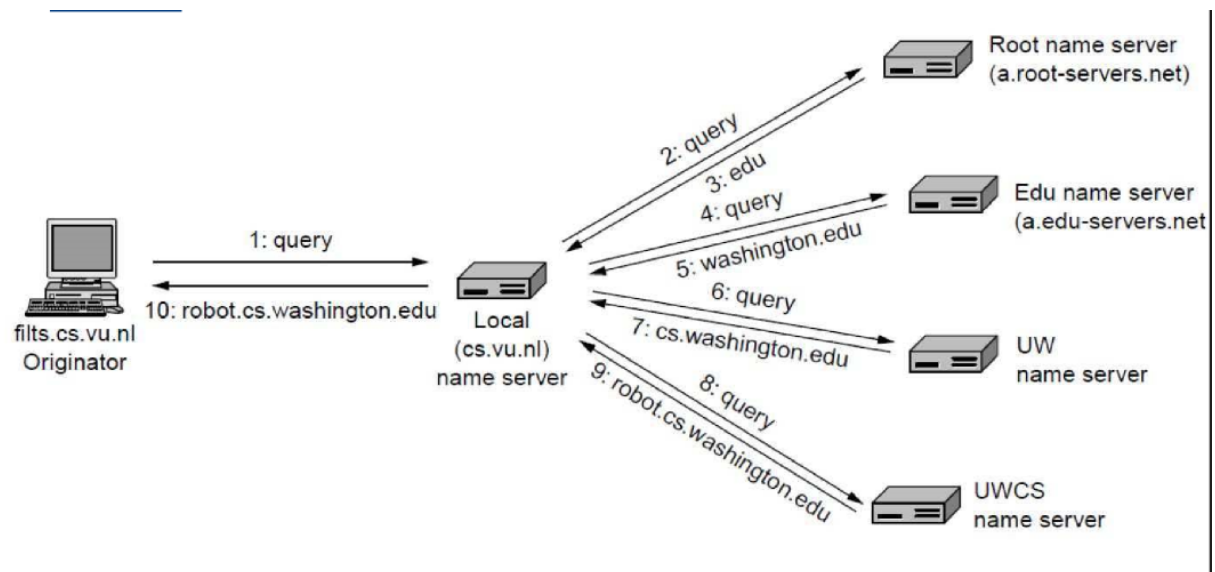


Figure 6: `dns_resolve_query`

- **recursive query** e.g. 1 in image, as `dns.nyu.edu` obtains mapping on behalf of `cse.nyu.edu`
- **iterative query** e.g. 2, 3, 4 in image, as replies are directly returned to `dns.nyu.edu`
- in theory: any query could be recursive or iterative, but usually follow pattern
 - requesting host → local DNS server: recursive
 - remaining queries: iterative

DNS Messages

| Identification | Flags | |
|---|--------------------------|-----------|
| Number of questions | Number of answer RRs | 12 |
| Number of authority RRs | Number of additional RRs | |
| Questions (variable number of questions) | | Na a c |
| Answers (variable number of resource records) | | RR |
| Authority (variable number of resource records) | | Re au |
| Additional information (variable number of resource records) | | Ac in |

- DNS has only query and reply messages

DNS message format

- to see this in action, use `nslookup`:

```

1 # query A record for google.com
2 $ nslookup
3 > google.com
4 Server:          192.168.20.1
5 Address:         192.168.20.1#53
6
7 Non-authoritative answer:
8 Name:   google.com
9 Address: 216.239.32.117
10 Name:   google.com
11 Address: 216.239.34.117
12 Name:   google.com
13 Address: 216.239.38.117
14 Name:   google.com
15 Address: 216.239.36.117
16 Name:   google.com
17 Address: 2001:4860:4802:38::75
18 >^C
19 # look up NS record of registermachine.com with google name server
    8.8.8.8
20 $ nslookup -type=NS registermachine.com 8.8.8.8
21 Server:          8.8.8.8
22 Address:         8.8.8.8#53

```

```
23
24 Non-authoritative answer:
25 registermachine.com      nameserver = ns-1362.awsdns-42.org.
26 registermachine.com      nameserver = ns-1556.awsdns-02.co.uk.
27 registermachine.com      nameserver = ns-363.awsdns-45.com.
28 registermachine.com      nameserver = ns-894.awsdns-47.net.
29
30 Authoritative answers can be found from:
```

DNS Caching

- DNS caching used extensively to
 - improve delay performance
 - reduce number of DNS messages travelling through the Internet
- when DNS server receives a DNS reply, it caches the mapping in local memory, and returns this for any future queries
- DNS servers discard cached information after time period (typically 2 days)
- due to caching, root servers are largely bypassed

DNS Security

- no security in original design
 - DNS spoofing: e.g. “I’m google”
 - DNS flooding: DNS critical to Internet so DOS on DNS could break a huge amount
- solutions to make DNS secure
 - DNSSEC: digitally signed answers to DNS queries
 - * not yet fully deployed
 - root signing