TCP/IP and OSI Reference Model

- Individual layer of the protocol stack perform a set of functions
- The layer is implemented as software or hardware using a set of protocols.

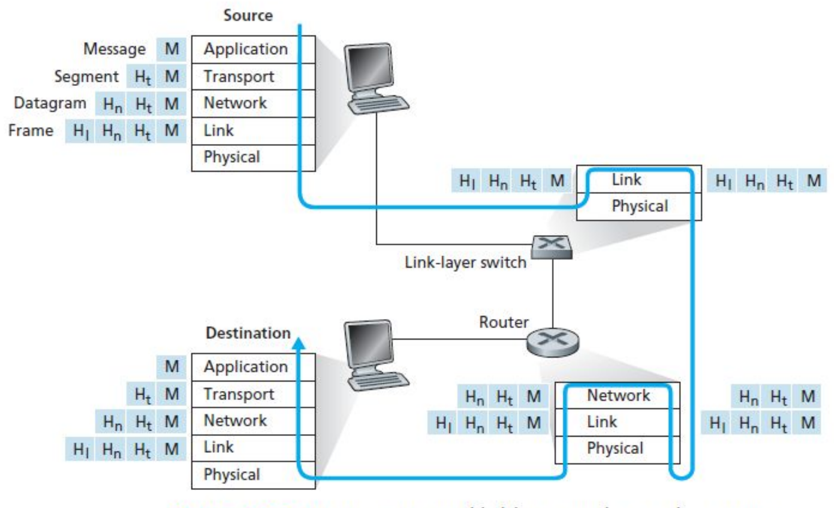


Figure 1.24 ♦ Hosts, routers, and link-layer switches; each contains a different set of layers, reflecting their differences in functionality

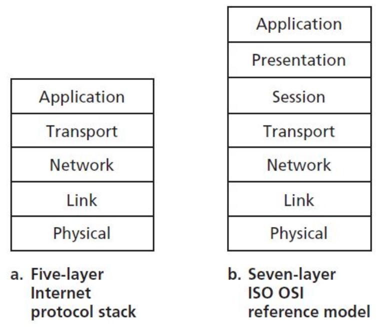
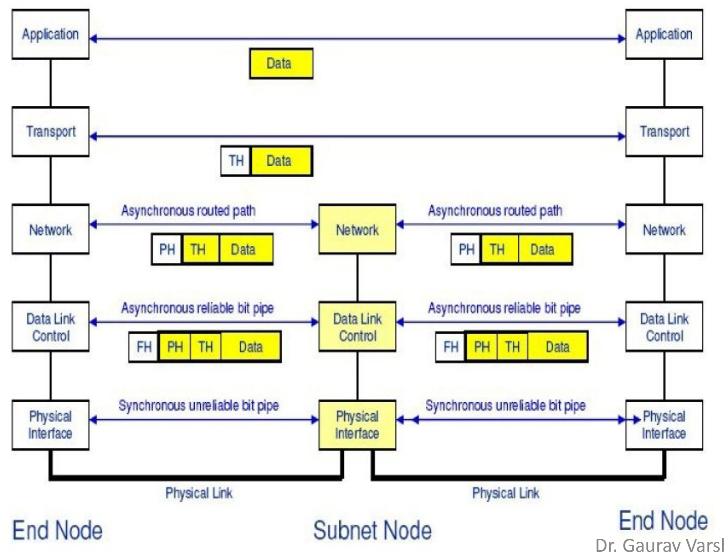


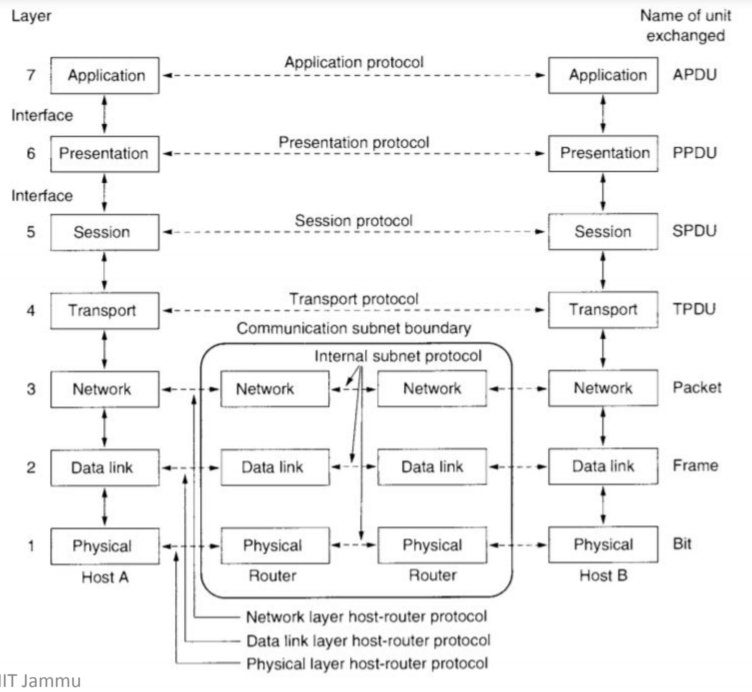
Figure 1.23 ♦ The Internet protocol stack (a) and OSI reference model (b)

TCP/IP and OSI Reference Model

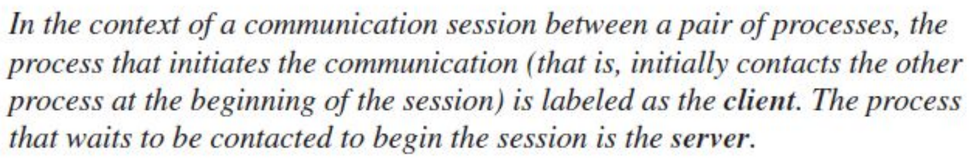
- Individual layer of the protocol stack perform a set of functions
- The layer is implemented as software or hardware using a set of protocols.

Encapsulation

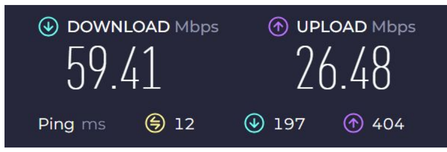




Network Application Architecture

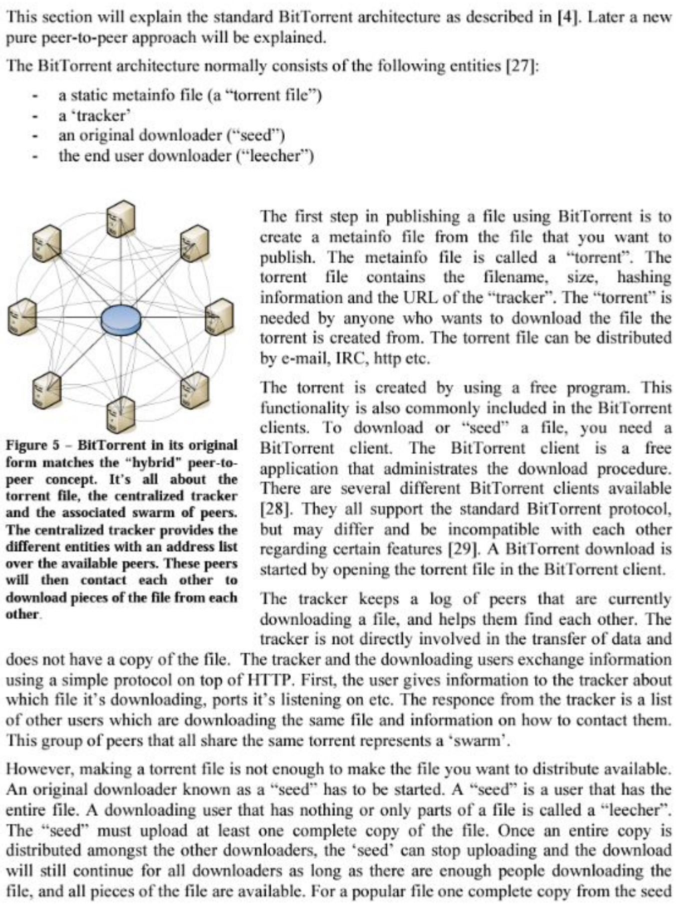

*In the context of a communication session between a pair of processes, the process that initiates the communication (that is, initially contacts the other process at the beginning of the session) is labeled as the **client**. The process that waits to be contacted to begin the session is the **server**.*

- **Client Server Architecture**
  - Server has a well known IP address and well known port number
  - Clients connect to servers for access to services
  - HTTP, FTP support client server application architecture
- **Peer to Peer Applications**
  - Peers communicate directly with each other without a central web server
  - In this way all peers are client and server at the same time.
  - Bit Torrent, Skype used to have Hybrid P2P protocol.
  - Advantage: Self Scalability
  - Disadvantage: ISP [upstream vs downstream], Security, Incentives [convince users to volunteer storage, bandwidth].


DOWNLOAD Mbps 59.41
UPLOAD Mbps 26.48
Ping ms 12 197 404

https://web.cs.ucla.edu/classes/cs217/05BitTorrent.pdf

https://novelbits.io/bluetooth-low-energy-protocol-stack-layers/


This section will explain the standard BitTorrent architecture as described in [4]. Later a new pure peer-to-peer approach will be explained.

The BitTorrent architecture normally consists of the following entities [27]:

- a static metainfo file (a "torrent file")
- a 'tracker'
- an original downloader ("seed")
- the end user downloader ("leecher")

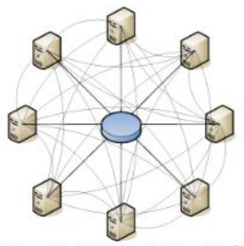


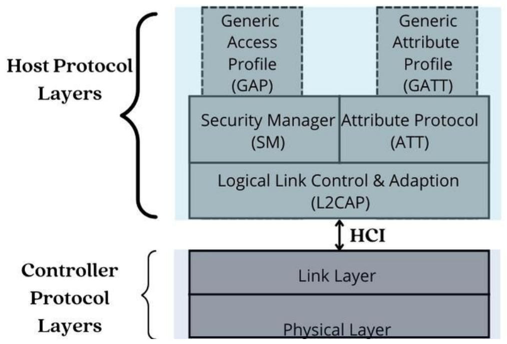
Figure 5 – BitTorrent in its original form matches the "hybrid" peer-to-peer concept. It's all about the torrent file, the centralized tracker and the associated swarm of peers. The centralized tracker provides the different entities with an address list over the available peers. These peers will then contact each other to download pieces of the file from each other.

The first step in publishing a file using BitTorrent is to create a metainfo file from the file that you want to publish. The metainfo file is called a "torrent". The torrent file contains the filename, size, hashing information and the URL of the "tracker". The "torrent" is needed by anyone who wants to download the file the torrent is created from. The torrent file can be distributed by e-mail, IRC, http etc.

The torrent is created by using a free program. This functionality is also commonly included in the BitTorrent clients. To download or "seed" a file, you need a BitTorrent client. The BitTorrent client is a free application that administrates the download procedure. There are several different BitTorrent clients available [28]. They all support the standard BitTorrent protocol, but may differ and be incompatible with each other regarding certain features [29]. A BitTorrent download is started by opening the torrent file in the BitTorrent client.

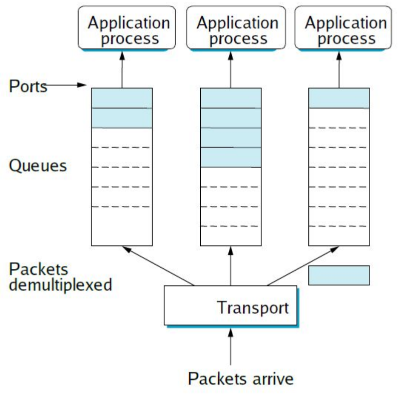
The tracker keeps a log of peers that are currently downloading a file, and helps them find each other. The tracker is not directly involved in the transfer of data and does not have a copy of the file. The tracker and the downloading users exchange information using a simple protocol on top of HTTP. First, the user gives information to the tracker about which file it's downloading, ports it's listening on etc. The responce from the tracker is a list of other users which are downloading the same file and information on how to contact them. This group of peers that all share the same torrent represents a 'swarm'.

However, making a torrent file is not enough to make the file you want to distribute available. An original downloader known as a "seed" has to be started. A "seed" is a user that has the entire file. A downloading user that has nothing or only parts of a file is called a "leecher". The "seed" must upload at least one complete copy of the file. Once an entire copy is distributed amongst the other downloaders, the 'seed' can stop uploading and the download will still continue for all downloaders as long as there are enough people downloading the file, and all pieces of the file are available. For a popular file one complete copy from the seed


Host Protocol Layers
Generic Access Profile (GAP)
Generic Attribute Profile (GATT)
Security Manager (SM)
Attribute Protocol (ATT)
Logical Link Control & Adaption (L2CAP)
HCI
Controller Protocol Layers
Link Layer
Physical Layer

Sockets

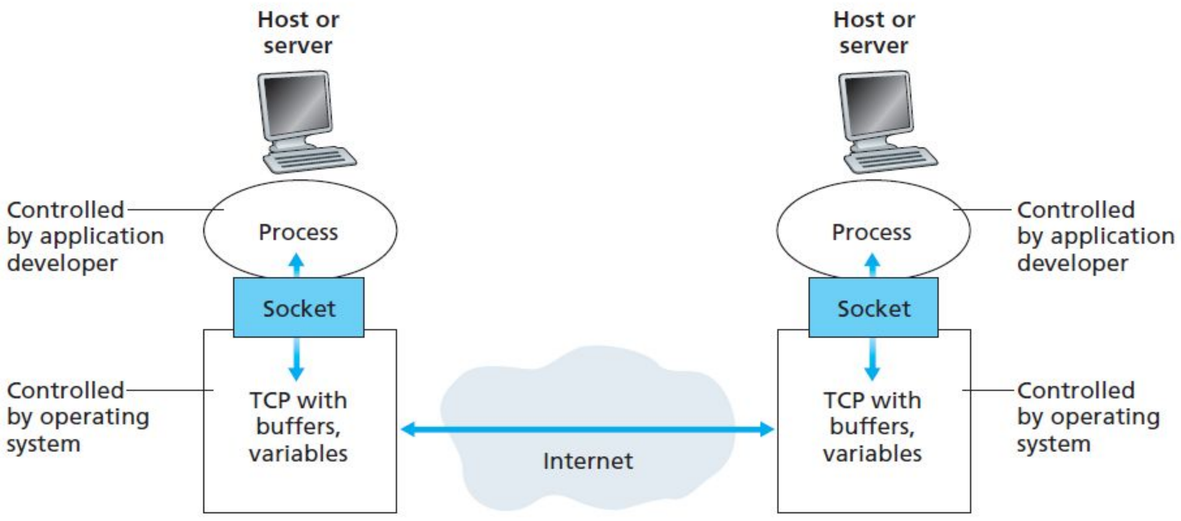
- Application generate data but how to push that data to Transport layer controlled by operating system.
- Application create socket to push the data to the below layer. A socket connect two endpoints in a network.
- An end point is defined by an IP address and a port.
    - Port is unique for every process and it provides multiplexing and demultiplexing at the Transport layer.



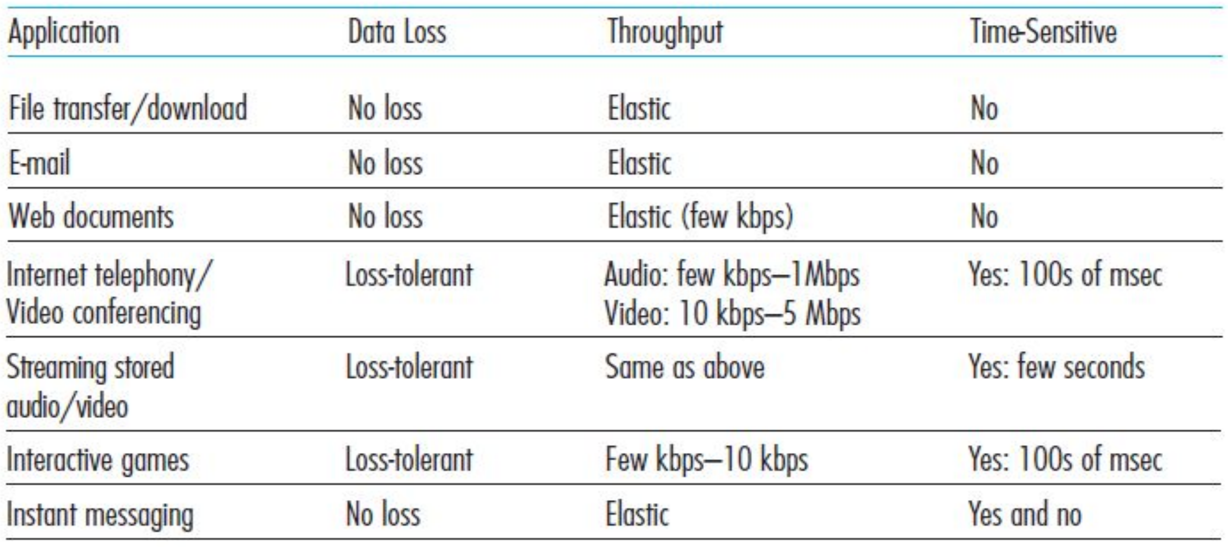
- Socket: An interface between an application process and transport layer
    - The application process can send/receive messages to/from another application process (local or remote)via a socket
- In Unix jargon, a socket is a file descriptor – an integer associated with an open file

Kameswari Chebrolu
Dept. of Electrical Engineering, IIT Kanpur

Application Layer to Transport Layer


Host or server
Host or server
Controlled by application developer
Process
Socket
Controlled by operating system
TCP with buffers, variables
Internet
Process
Socket
TCP with buffers, variables
Controlled by application developer
Controlled by operating system

Transport Services Application

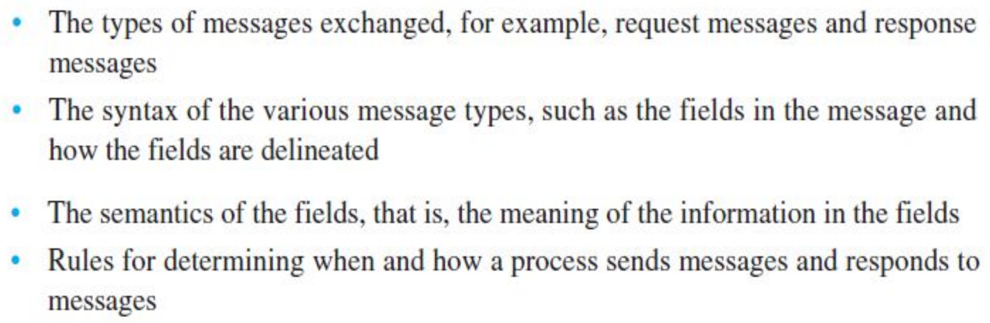


Application	Data Loss	Throughput	Time-Sensitive
File transfer/download	No loss	Elastic	No
E-mail	No loss	Elastic	No
Web documents	No loss	Elastic (few kbps)	No
Internet telephony/ Video conferencing	Loss-tolerant	Audio: few kbps—1Mbps Video: 10 kbps—5 Mbps	Yes: 100s of msec
Streaming stored audio/video	Loss-tolerant	Same as above	Yes: few seconds
Interactive games	Loss-tolerant	Few kbps—10 kbps	Yes: 100s of msec
Instant messaging	No loss	Elastic	Yes and no

Figure 2.4 ♦ Requirements of selected network applications

Application Layer Protocols

- An application-layer protocol defines:



  - The types of messages exchanged, for example, request messages and response messages
  - The syntax of the various message types, such as the fields in the message and how the fields are delineated
  - The semantics of the fields, that is, the meaning of the information in the fields
  - Rules for determining when and how a process sends messages and responds to messages

- Network Applications and **Application Layer Protocol**
  - HTML, Web Browser, Web Servers and **HTTP**

Application Layer Protocol HTTP RFC 1945 2616

Invented by Timothy John Berners Lee in early 90s

•Historical Insights

- The term hypertext was coined by Ted Nelson in 1965 in the Xanadu Project.
- Tim Berners-Lee and his team at CERN are credited with inventing the original HTTP along with HTML and the associated technology for a web server and a text-based web browser.
  The first version of the protocol had only one method, namely GET, which would request a page from a server. The response from the server was always an HTML page.

Application Layer Protocol HTTP RFC 1945 2616

- HTTP 0.9 - Only method GET and resource name
- HTTP 1.0 - POST and HEAD and HTTP headers functionality [request headers and response status codes]
- HTTP 1.1 - Persistent Connections and Method PUT, PATCH, DELETE, CONNECT, TRACE, and OPTIONS
- HTTP 2.0 - Request multiplexing, request prioritization, automatic compressing [binary framing]
- HTTP 3.0 - QUIC- Quick UDP Internet connections

Web 1.0 - 1989 - Read only, Mono Directional, Information Sharing Web

Web 2.0 - 2004 - Read/Write web, People centric web, participative web

Web 3.0 - 2006 - Semantic Web, Machine Understandable Web, Efficient indexing and searching

Web 4.0 - In Ideation - Symbiotic Web - AI, Natural Language Processing, Intelligent Web

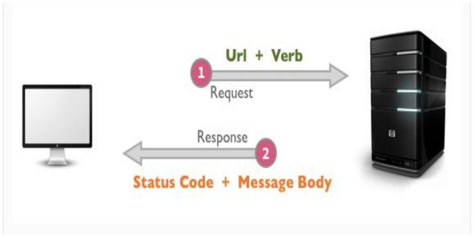
Application Layer Protocol **HTTP** RFC 1945 2616

Invented by Timothy John Berners Lee in early 90s

- HTTP is an application layer protocol for distributed and collaborative hypermedia information system.
- HTTP is foundation for communication of hypertext documents over World Wide Web which may include hyperlinks to other resources which users can easily access through HTTP.
- Tim Berners-Lee initiated the development of HTTP in 1989 and which is standardized by Internet Engineering Task Force and World Wide Consortium.

•HTTP functions as a request–response protocol in the client–server computing model.

- A web browser, for example, may be the client and an application running on a computer hosting a website may be the server.
- The client submits an HTTP request message to the server.
- The server, which provides resources such as HTML files and other content, or performs other functions on behalf of the client, returns a response message to the client.
- The response contains completion status information about the request and may also contain requested content in its message body.



Application Layer Protocol **HTTP** RFC 1945 2616

Invented by Timothy John Berners Lee in early 90s

- A Web Page consists of Objects
  - An Object is addressable by a Single URL. Image, Script, HTML Page
  - All webpages are
    - Base HTML and reference objects
  - Each URL that reference a object has a hostname and a pathname

```
http://www.someSchool.edu/someDepartment/picture.gif
```

  - Browser implement the client side of HTTP
  - Web Servers such as Apache implements the server side of HTTP.
  - HTTP defines how web client such as browsers must request web pages from a web server.

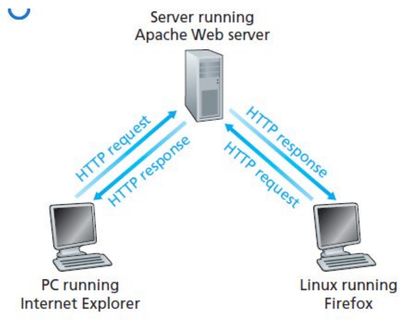


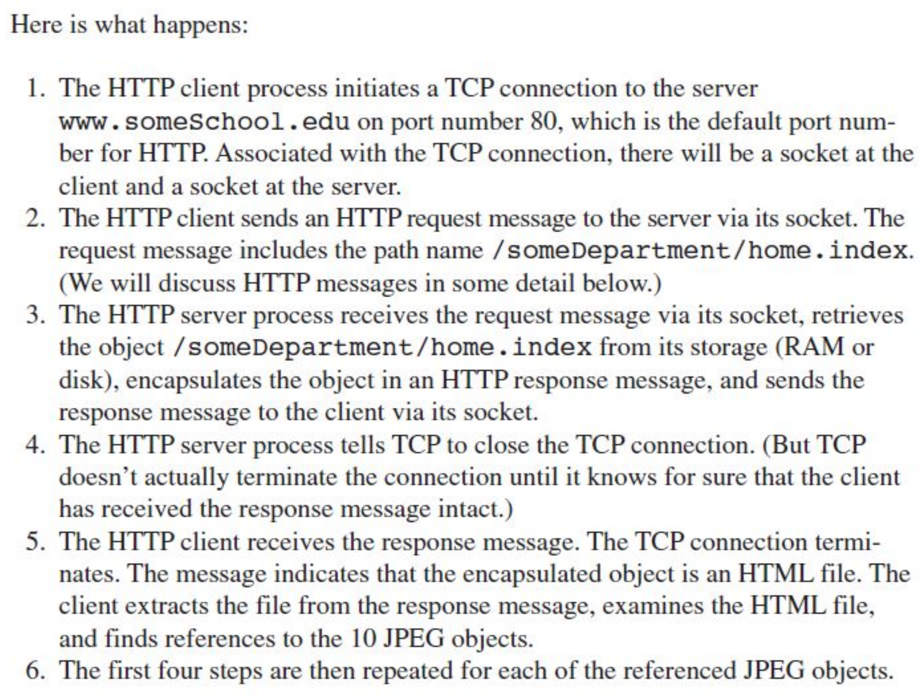
Figure 2.6 ♦ HTTP request-response behavior

Application Layer Protocol **HTTP** RFC 1945 2616

Invented by Timothy John Berners Lee in early 90s

- **User Agents**
  - Web Browsers, web crawlers, mobile apps, or other software that accesses, consumes or displays web content using HTTP.
- **Web caching servers**
  - High-traffic websites often benefit from web cache servers that deliver content on behalf of upstream servers to improve response time.
- **Protocol Suite**
  - HTTP is designed to work under IP protocol suite.
    - HTTP by definition works under a reliable transport layer protocol such as TCP but can also work with UDP for example HTTPU.
- **HTTP/1.1 is a revision of the original HTTP (HTTP/1.0).**
    - In HTTP/1.0 a separate connection to the same server is made for every resource request.
    - HTTP/1.1 can reuse a connection multiple times to download images, scripts, stylesheets, etc. after the page has been delivered. HTTP/1.1 communications therefore experience less latency as the establishment of TCP connections presents considerable overhead

HTTP: Non Persistent Connections



Here is what happens:

1. The HTTP client process initiates a TCP connection to the server `www.someSchool.edu` on port number 80, which is the default port number for HTTP. Associated with the TCP connection, there will be a socket at the client and a socket at the server.
2. The HTTP client sends an HTTP request message to the server via its socket. The request message includes the path name `/someDepartment/home.index`. (We will discuss HTTP messages in some detail below.)
3. The HTTP server process receives the request message via its socket, retrieves the object `/someDepartment/home.index` from its storage (RAM or disk), encapsulates the object in an HTTP response message, and sends the response message to the client via its socket.
4. The HTTP server process tells TCP to close the TCP connection. (But TCP doesn't actually terminate the connection until it knows for sure that the client has received the response message intact.)
5. The HTTP client receives the response message. The TCP connection terminates. The message indicates that the encapsulated object is an HTML file. The client extracts the file from the response message, examines the HTML file, and finds references to the 10 JPEG objects.
6. The first four steps are then repeated for each of the referenced JPEG objects.

HTTP is a stateless protocol

HTTP 1.1 uses persistent connections in default mode.

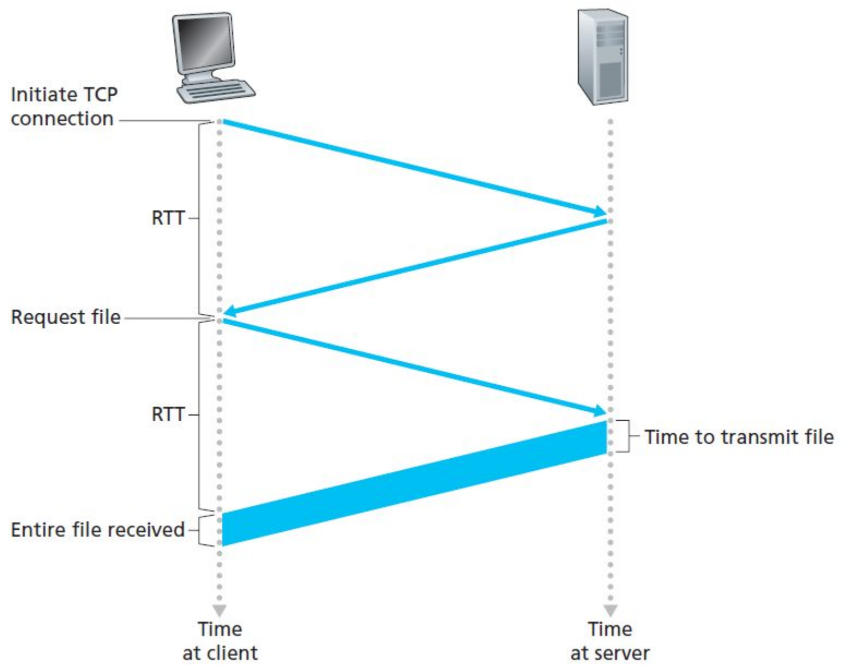
HTTP run over TCP at the Transport Layer via sockets at client and server

HTTP has nothing to do with how the webpage is interpreted by the web clients.

In **non persistent** connections the underlying TCP connection is closed after server sends the object.

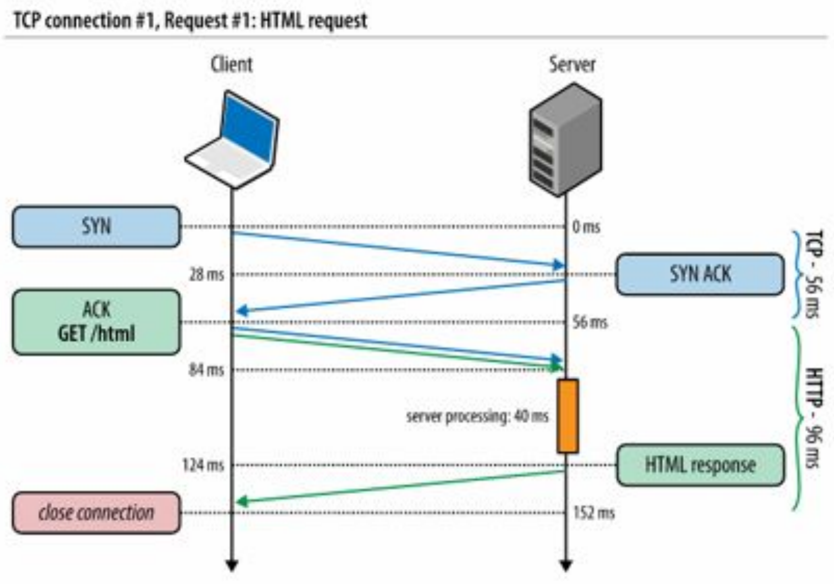
In **persistent** connections all request and responses between the client and server are sent on the same TCP connection.

HTTP: Request Response Time



HTTP: Non Persistent Connections

https://hpbn.co/http1x/



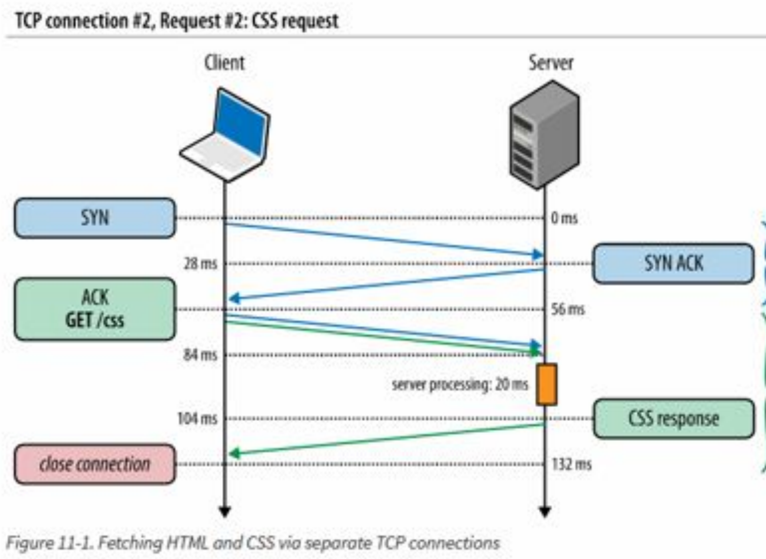


Figure 11-1. Fetching HTML and CSS via separate TCP connections

HTTP: Persistent Con

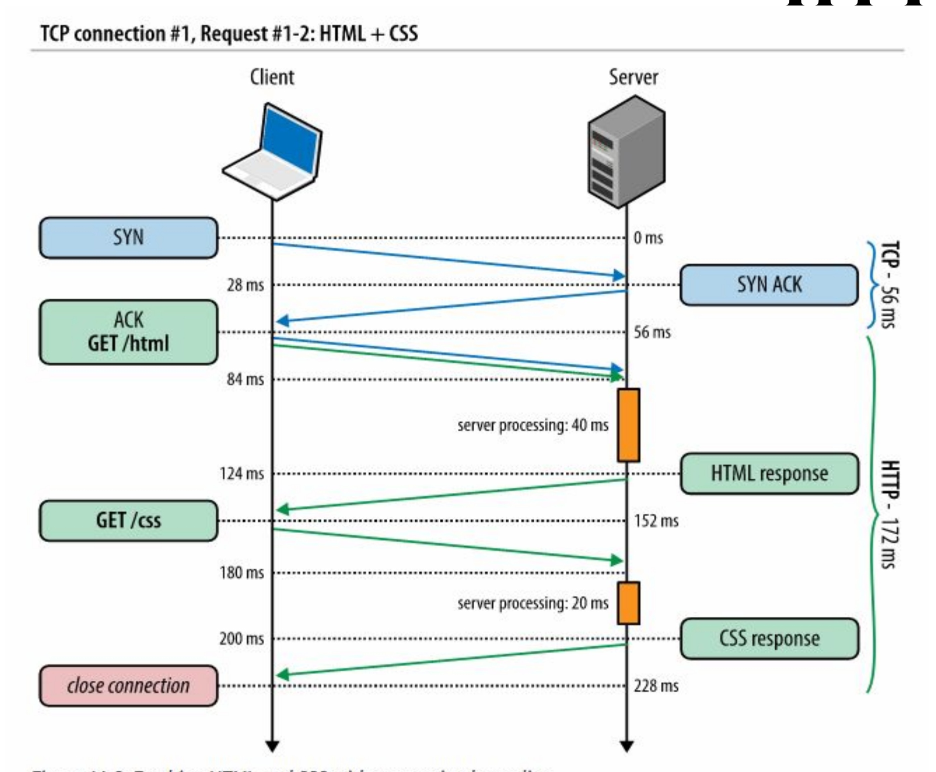


Figure 11-2. Fetching HTML and CSS with connection keepalive

https://hpbn.co/http1x/

The total latency savings for *N* requests is *(N–1)* × *RTT* when connection keepalive is enabled.

HTTP: Pipelining in Persistent Con

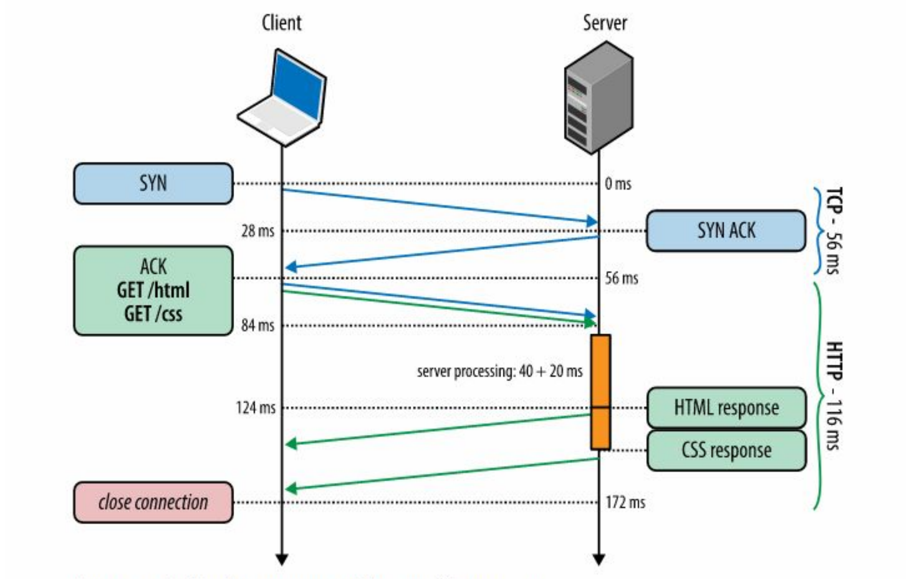


Figure 11-3. Pipelined HTTP requests with server-side FIFO queue

Note

Eliminating the wait time imposed by response and request propagation latencies is one of the primary benefits of HTTP/1.1 pipelining. However, the ability to process the requests in parallel can have just as big, if not bigger, impact on the performance of your application.

HTTP: Message Format: Request Message

- The HTTP specification states that a request or response message has the following generic structure:

```
message = <start-line>
          (<message-header>)
          CRLF
          [<message-body>]
```

<start-line> = Request-Line | Status-Line

<message-header> = Field-Name : Field-Value

Request-Line = Method SP URI SP HTTP-Version CRLF

Method = "OPTIONS"

| "HEAD"

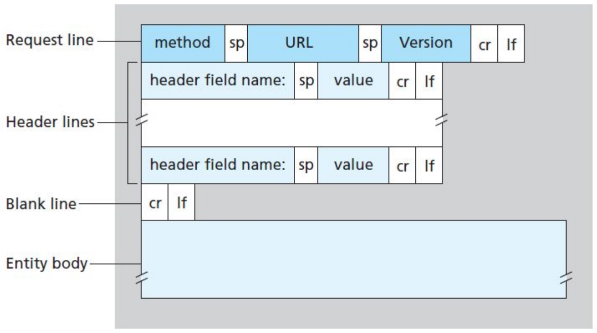
| "GET"

| "POST"

| "PUT"

| "DELETE"

| "TRACE"



```
GET /articles/http-basics HTTP/1.1
Host: www.articles.com
Connection: keep-alive
Cache-Control: no-cache
Pragma: no-cache
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

HTTP: Message Format: Response Message

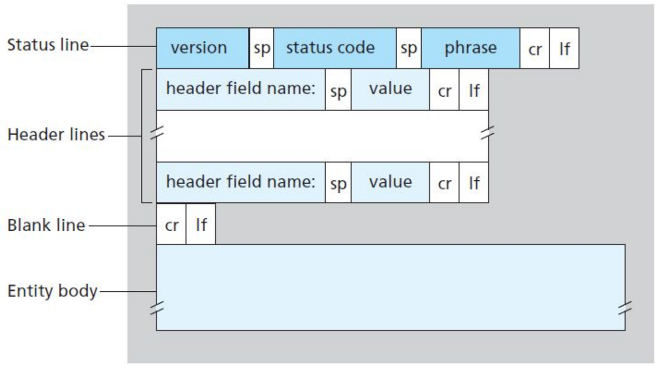
- **Response Format**
- The response format is similar to the request message, except for the status line and headers. The status line has the following structure:

Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF

- HTTP-Version is sent as HTTP/1.1
- The Status-Code is one of the many statuses discussed earlier.
- The Reason-Phrase is a human-readable version of the status code.
- A typical status line for a successful response might look like so:

HTTP/1.1 200 OK

```
HTTP/1.1 200 OK
Connection: close

Date: Tue, 09 Aug 2011 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 09 Aug 2011 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html

(data data data data data ...)
```



HTTP: Message Format: Response Message

- **Status Codes**
  - With URLs and verbs, the client can initiate requests to the server. In return, the server responds with status codes and message payloads. The status code is important and tells the client how to interpret the server response.
    - **1xx: Informational Messages**
      - This class of codes was introduced in HTTP/1.1 and is purely provisional. The server can send a Expect: 100-continue message, telling the client to continue sending the remainder of the request, or ignore if it has already sent it.
    - **2xx: Successful**
      - This tells the client that the request was successfully processed. The most common code is 200 OK. For a GET request, the server sends the resource in the message body. There are other less frequently used codes:

•Status Codes

- 202 Accepted: the request was accepted but may not include the resource in the response. This is useful for asynchronous processing on the server side. The server may choose to send information for monitoring.
- 204 No Content: there is no message body in the response.
- 205 Reset Content: indicates to the client to reset its document view.
- 206 Partial Content: indicates that the response only contains partial content. Additional headers indicate the exact range and content expiration information.
- **3xx: Redirection**
  - This requires the client to take additional action. The most common use-case is to jump to a different URL in order to fetch the resource.
  - 301 Moved Permanently: the resource is now located at a new URL.
  - 303 See Other: the resource is temporarily located at a new URL. The Location response header contains the temporary URL.
  - 304 Not Modified: the server has determined that the resource has not changed and the client should use its cached copy. This relies on the fact that the client is sending ETag (Entity Tag) information that is a hash of the content.

HTTP: Message Format: Response Message

•Status Codes

- **4xx: Client Error**
  - These codes are used when the server thinks that the client is at fault, either by requesting an invalid resource or making a bad request.
  - 400 Bad Request: the request was malformed
  - 401 Unauthorized: request requires authentication. The client can repeat the request with the Authorization header.
  - 403 Forbidden: server has denied access to the resource
  - 405 Method Not Allowed: invalid HTTP verb used in the request line, or the server does not support that verb.
  - 409 Conflict: the server could not complete the request because the client is trying to modify a resource that is newer than the client's timestamp. Conflicts arise mostly for PUT requests during collaborative edits on a resource.

•Status Codes

- **5xx: Server Error**
- This class of codes are used to indicate a server failure while processing the request. The most commonly used error code is 500 Internal Server Error. The others in this class are:
- 501 Not Implemented: the server does not yet support the requested functionality.
- 503 Service Unavailable: this could happen if an internal system on the server has failed or the server is overloaded. Typically, the server won't even respond and the request will timeout.

HTTP: Message Format: Response Message

- **HTTP Request Methods**
  - HTTP defines methods (sometimes referred to as verbs) to indicate the desired action to be performed on the identified resource.
  - The HTTP/1.0 specification defined the GET, POST and HEAD methods and the HTTP/1.1 specification added 5 new methods: OPTIONS, PUT, DELETE, TRACE and CONNECT.
  - Any client can use any method and the server can be configured to support any combination of methods.
- GET
  - The GET method requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect.
- HEAD
  - The HEAD method asks for a response identical to that of a GET request, but without the response body

- **HTTP Request Methods**

https://http.dev/patch

- **POST**
  - The POST method requests that the server accept the entity enclosed in the request as a new subordinate of the web resource identified by the URI.
  - The data POSTed might be, for example, an annotation for existing resources; a message for a bulletin board, newsgroup, mailing list, or comment thread; a block of data that is the result of submitting a web form to a data-handling process; or an item to add to a database.
- **PUT**
  - The PUT method requests that the enclosed entity be stored under the supplied URI. If the URI refers to an already existing resource, it is modified; if the URI does not point to an existing resource, then the server can create the resource with that URI
- **DELETE**
  - The DELETE method deletes the specified resource.

HTTP: Cookies

- Remember that HTTP is stateless.

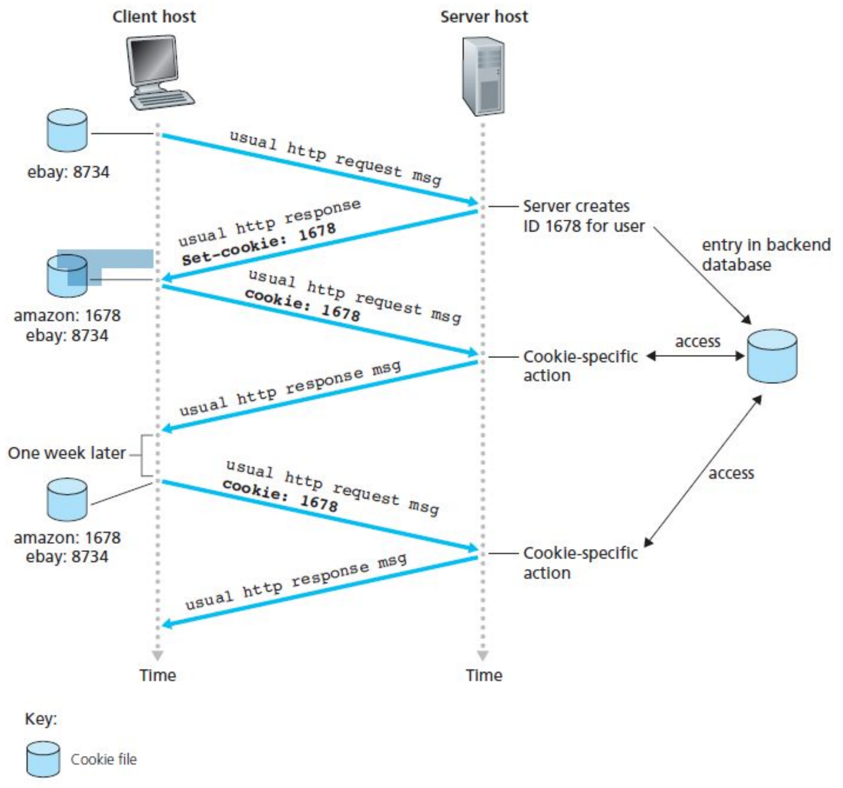


Figure 2.10 ♦ Keeping user state with cookies

HTTP: Cookies: Web Caching and Types

Browser cache

Proxy cache

Transparent proxy cache

Reverse (inverse) proxy cache

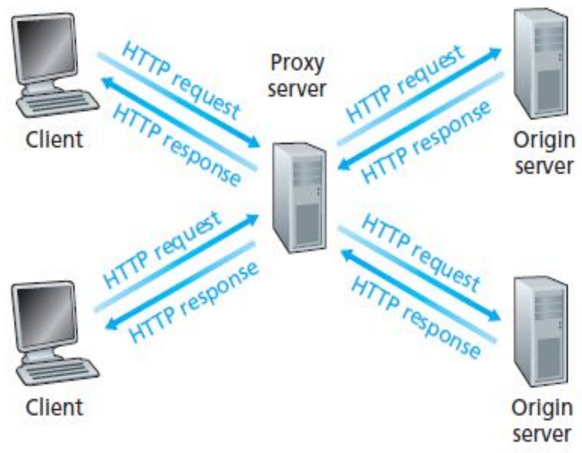


Figure 2.11 ♦ Clients requesting objects through a Web cache

HTTP: Benefits of Caching

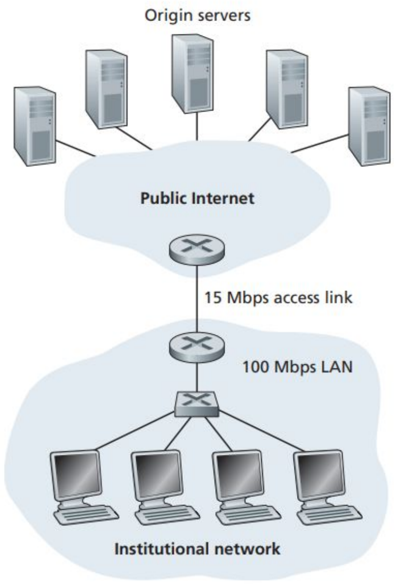
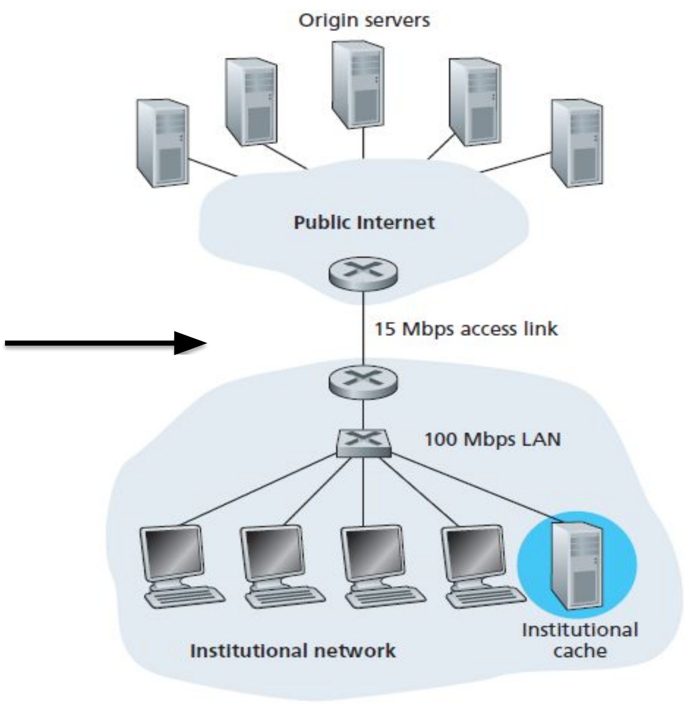


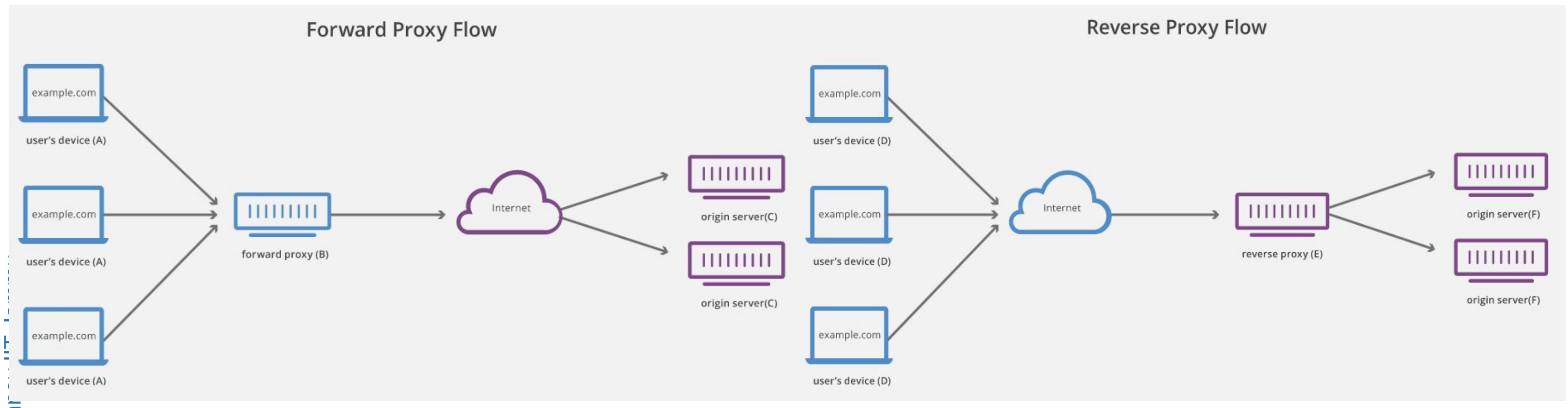
Figure 2.12 ♦ Bottleneck between an institutional network and the Internet



There are two main reasons that Web caches are used:

To **reduce latency**
To **reduce network traffic**

Forward and Reverse Proxy



Benefits of forward proxy:

To avoid state or institutional browsing restrictions

To block access to certain content

To protect their identity online

Benefits of reverse proxy:

Load Balancing

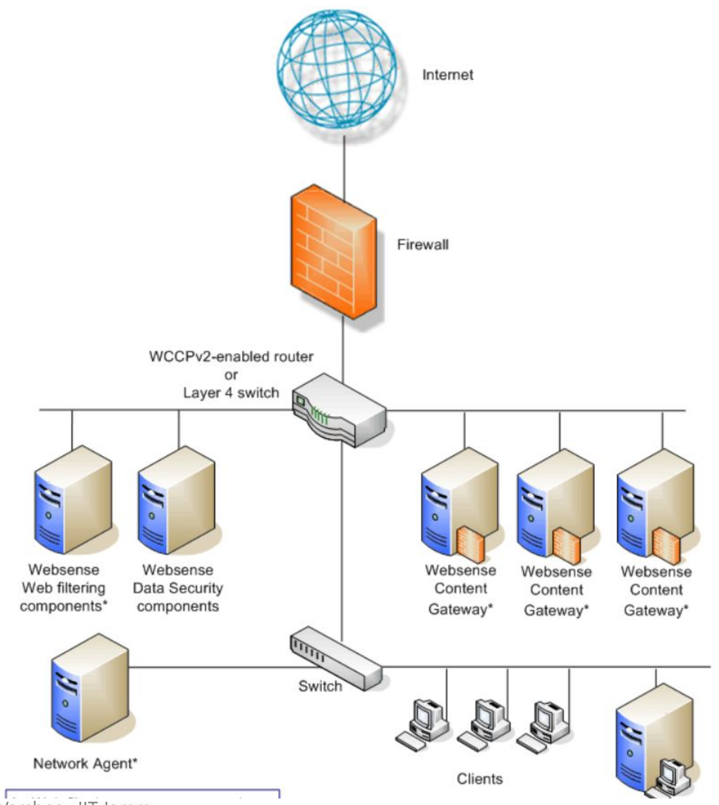
Protection from DDoS attacks [Address of the server is not known]

caching

SSL Encryption [proxy can do encryption and decryption and forward the request to the origin server]

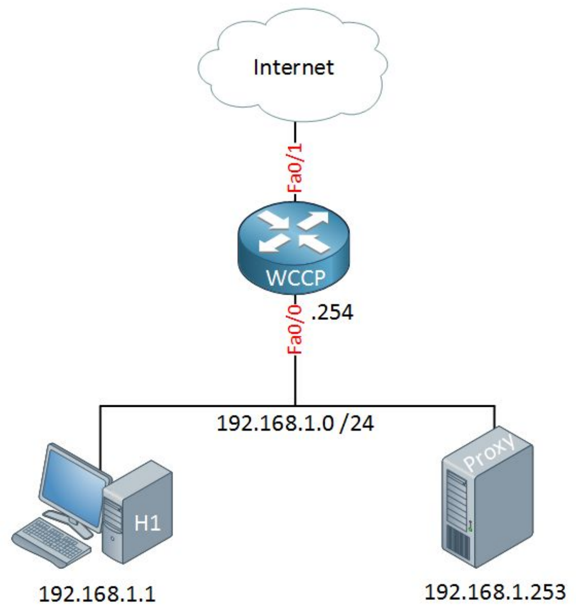
HTTP: Cookies: Transparent Proxy

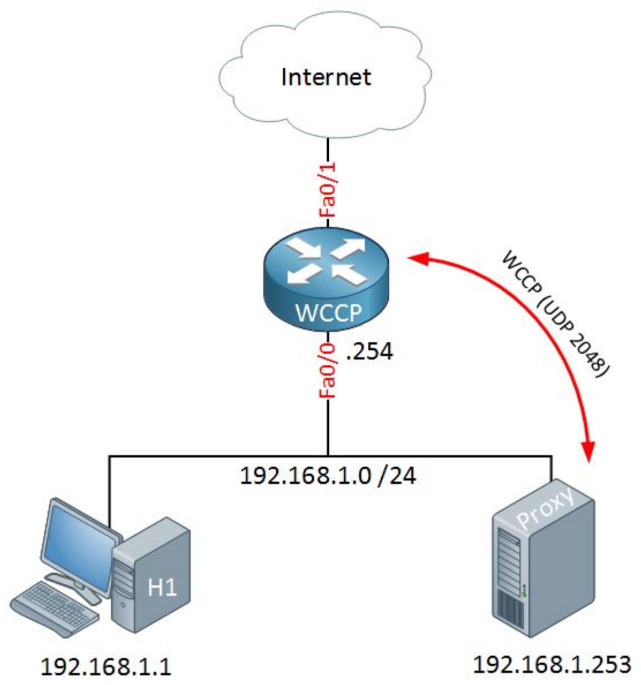
Transparent proxy cache


Internet
Firewall
WCCPv2-enabled router or Layer 4 switch
Websense Web filtering components*
Websense Data Security components
Websense Content Gateway*
Websense Content Gateway*
Websense Content Gateway*
Switch
Network Agent*
Clients

HTTP: Cookies: Web Caching: Transparent Proxy

To demonstrate this I will use the following topology, a small network that has a Cisco router, a host that will browse the Internet and a Squid proxy server:



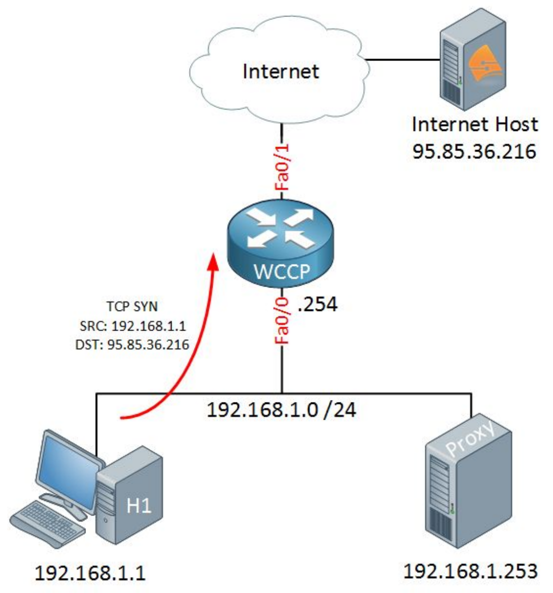


The router and proxy server are both running WCCP. The squid proxy server will announce itself to the router using UDP port 2048 and the router will respond:

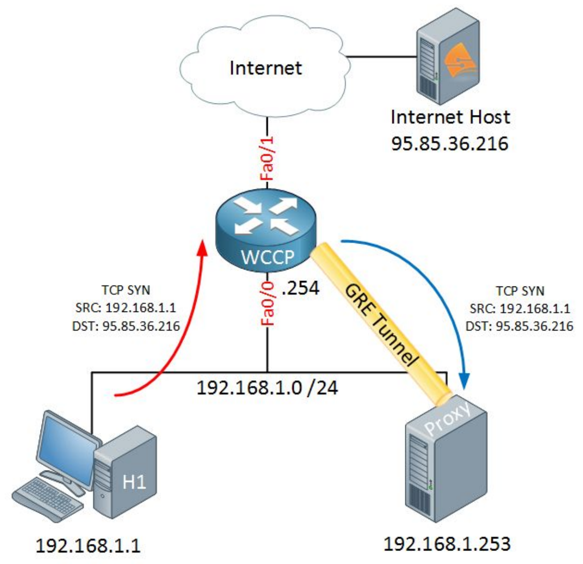
https://networklessons.com/cisco/ccie-routing-switching-written/cisco-wccp-squid-transparent-proxy

HTTP: Cookies: Web Caching: Transparent Proxy

When the host wants to browse the Internet it will create a TCP SYN packet for the destination which will end up at the router:

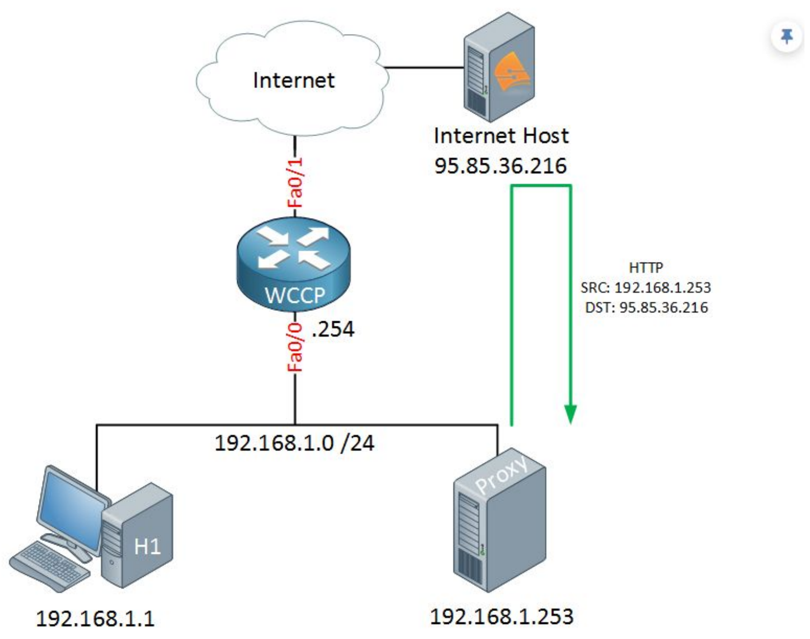


Without WCCP, the TCP SYN would be forwarded to the Internet because we are running WCCP, something else will happen. The router will forward the TCP SYN from the host to the proxy server:

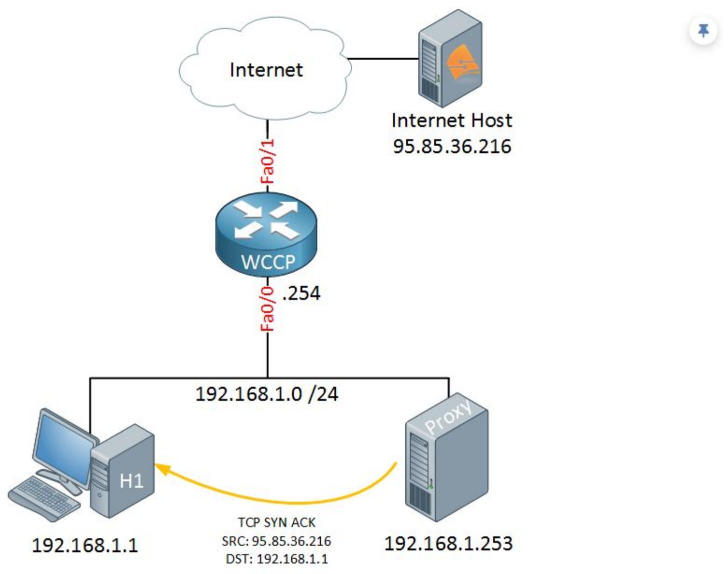


To make sure that the original packet **does not change** it will use a GRE tunnel for this. Normally GRE is used when the proxy server is on another subnet, when the proxy server is on the same subnet you can also use layer 2 redirection. The proxy server will check its cache and see if it has served the requested webpage before:

HTTP: Cookies: Web Caching: Transparent Proxy



When it doesn't have anything in its cache, the proxy server will contact the webserver on the Internet and request the webpage that the host was looking for. Once this process is done, it can serve the webpage to the host:



Once the proxy server has the webpage it will send a TCP SYN ACK to the host that is still waiting for the webpage. The important part here is that the source IP address of this packet is the **IP address of the host on the Internet, NOT the proxy server!** From the host's perspective, it thinks that it is talking directly to the webserver and it has no idea that there is a proxy server in the network. This is what we call a **transparent proxy**.

HTTP: Cookies: Web Caching and Conditional GET

Request 1

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
```

Response 1

```
HTTP/1.1 200 OK
Date: Sat, 8 Oct 2011 15:39:29
Server: Apache/1.3.0 (Unix)
Last-Modified: Wed, 7 Sep 2011 09:23:24
Content-Type: image/gif

(data data data data data ...)
```

Request 2

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
If-modified-since: Wed, 7 Sep 2011 09:23:24
```

Response 2

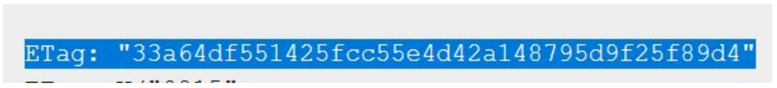
```
HTTP/1.1 304 Not Modified
Date: Sat, 15 Oct 2011 15:39:29
Server: Apache/1.3.0 (Unix)

(empty entity body)
```

ETAG

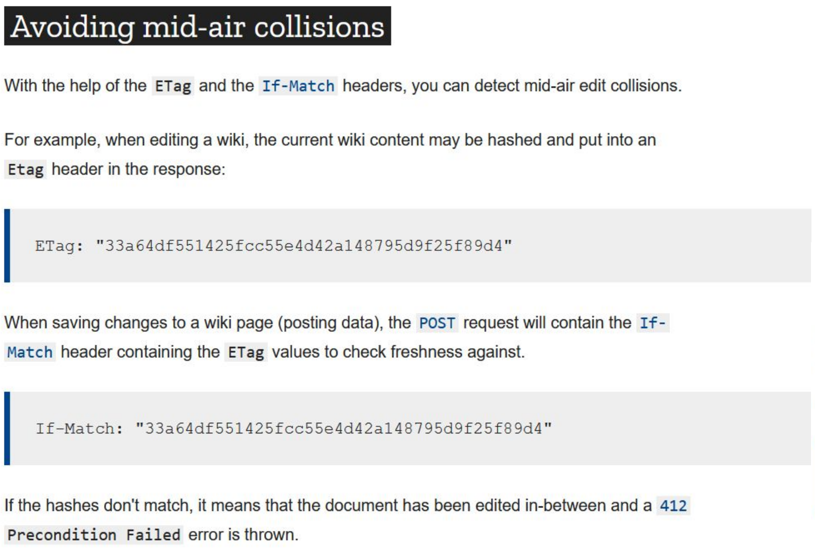
The **ETag** HTTP response header is an identifier for a specific version of a resource

"<etag_value>"

Entity tag uniquely representing the requested resource. They are a string of ASCII characters placed between double quotes, like `"675af34563dc-tr34"`. The method by which `ETag` values are generated is not specified. Often, a hash of the content, a hash of the last modification timestamp, or just a revision number is used. For example, MDN uses a hexadecimal hash of the wiki article content.



```
ETag: "33a64df551425fcc55e4d42a148795d9f25f89d4"
```



Avoiding mid-air collisions

With the help of the `ETag` and the `If-Match` headers, you can detect mid-air edit collisions.

For example, when editing a wiki, the current wiki content may be hashed and put into an `Etag` header in the response:

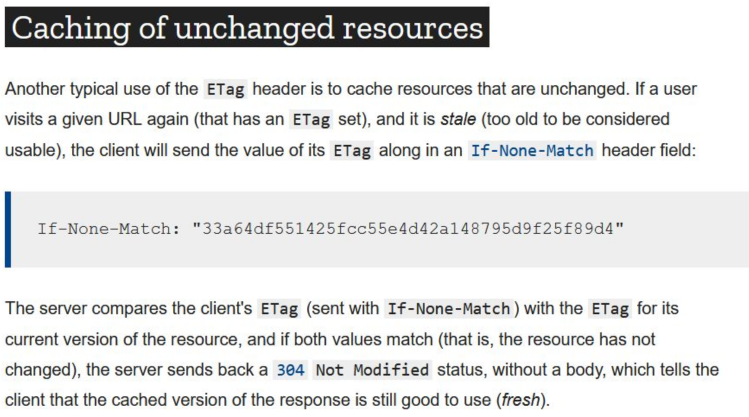
```
ETag: "33a64df551425fcc55e4d42a148795d9f25f89d4"
```

When saving changes to a wiki page (posting data), the `POST` request will contain the `If-Match` header containing the `ETag` values to check freshness against.

```
If-Match: "33a64df551425fcc55e4d42a148795d9f25f89d4"
```

If the hashes don't match, it means that the document has been edited in-between and a `412 Precondition Failed` error is thrown.

Also Study the Cache-Control HTTP header...



Caching of unchanged resources

Another typical use of the `ETag` header is to cache resources that are unchanged. If a user visits a given URL again (that has an `ETag` set), and it is *stale* (too old to be considered usable), the client will send the value of its `ETag` along in an `If-None-Match` header field:

```
If-None-Match: "33a64df551425fcc55e4d42a148795d9f25f89d4"
```

The server compares the client's `ETag` (sent with `If-None-Match`) with the `ETag` for its current version of the resource, and if both values match (that is, the resource has not changed), the server sends back a `304 Not Modified` status, without a body, which tells the client that the cached version of the response is still good to use (*fresh*).

Read about HTTP Connect- https://support.kaspersky.com/KWTS/6.1/en-US/188634.htm
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/ETag

DNS: Domain Name System

- People prefer to use easy-to-remember names instead of IP Addresses.
- The domain name system (DNS) is an Internet-wide distributed database that translates between domain names and IP addresses.
- Domain names are alphanumeric names that corresponds to an IP address.
- An application program on a host accesses the domain system through a DNS client, called the resolver.
- Resolver contacts DNS server, called name server.
- DNS server returns IP address to resolver which passes the IP address to application

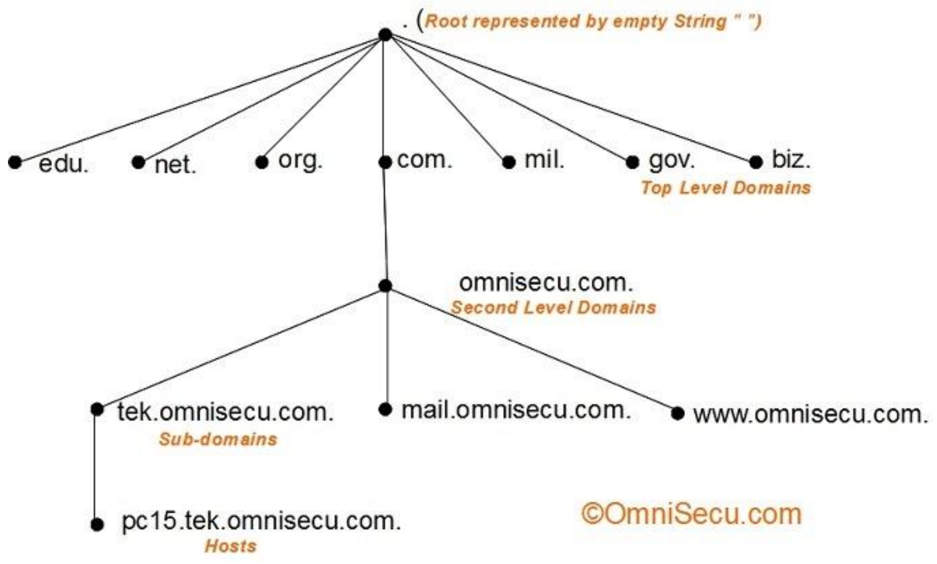
Interactive Video: https://youtu.be/72snZctFFtA

Paul Mockapetris expanded the Internet beyond its academic **origins** by **inventing** the Domain Name Sy (**DNS**) in 1983.

Source: cs.virginia.edu

DNS: Domain Name System

- DNS hierarchy can be represented by a tree
- Root and top-level domains are administered by an Internet Central Name Registration Authority (ICANN)
- Below top-level domain, administration of name space is delegated to organizations
- Each organization can delegate further.



DNS: Domain Name System **cs.virginia.edu**

- Every node in the DNS domain tree can be identified by a unique **Fully Qualified Domain Name (FQDN).**
  - *The FQDN gives the position in the DNS tree.* A FQDN consists of labels (“cs”,“virginia”,”edu”) separated by a period (“.”) cs.virginia.edu.
- Each label can be up to 63 characters long
- FQDN contains characters, numerals, and dash character (“-”)
- FQDNs are not case-sensitive

DNS: Domain Name System

Internal Namespace (**news.nic.in**)

- Top-level domains:
  - *Organizational: 3-character code indicates the function of the organization*
    - Examples: gov, mil, edu, org, com, net
  - *Geoographical: 2-character country or region code*
    - Examples: us, va, jp, de
  - *Reverse domains: A special domain (in-addr.arpa) used for IP address-to-name mapping*

http://www.tcpipguide.com/free/t_DNSReverseNameResolutionUsingtheINADDRARPADomain-2.htm

news.nic.in

Top-level domain (**in**)

Registered Internet Name (external DNS name) **nic.in**

com	Commercial organizations
edu	Educational institutions
gov	Government institutions
int	International organizations
mil	U.S. military institutions
net	Networking organizations
org	Non-profit organizations

DNS - Hierarchy

- The resolution of the hierarchical name space is done by a hierarchy of name servers
- Each server is responsible (authoritative) for a contiguous portion of the DNS namespace, called a zone.
- Zone is a part of the subtree
- DNS server answers queries about hosts in its zone

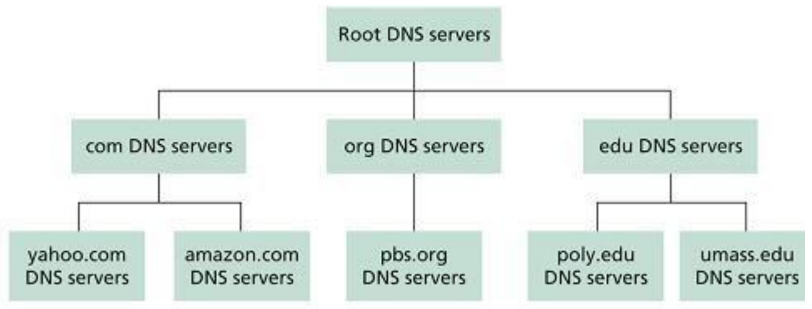
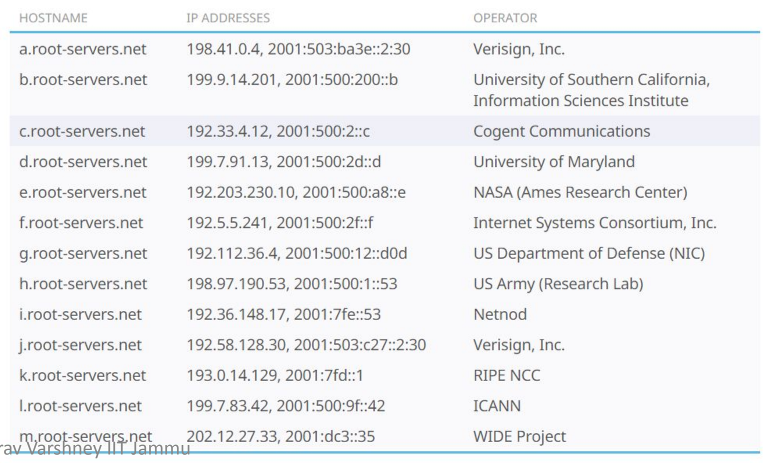


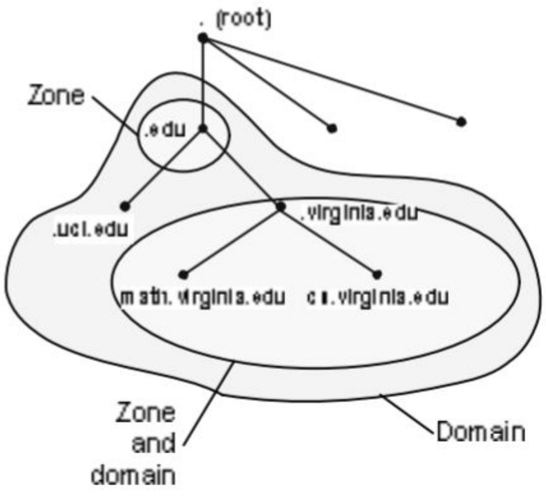
Figure 2.18 ♦ Portion of the hierarchy of DNS servers

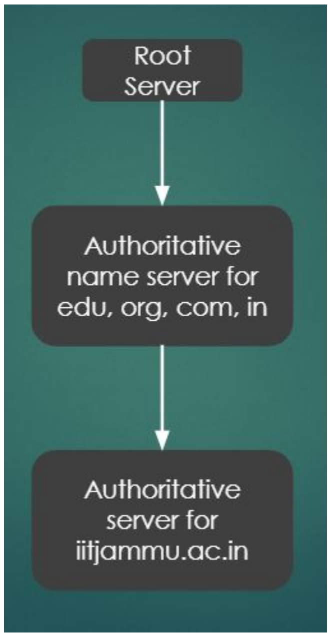


HOSTNAME	IP ADDRESSES	OPERATOR
a.root-servers.net	198.41.0.4, 2001:503:ba3e::2:30	Verisign, Inc.
b.root-servers.net	199.9.14.201, 2001:500:200::b	University of Southern California, Information Sciences Institute
c.root-servers.net	192.33.4.12, 2001:500:2::c	Cogent Communications
d.root-servers.net	199.7.91.13, 2001:500:2d::d	University of Maryland
e.root-servers.net	192.203.230.10, 2001:500:a8::e	NASA (Ames Research Center)
f.root-servers.net	192.5.5.241, 2001:500:2f::f	Internet Systems Consortium, Inc.
g.root-servers.net	192.112.36.4, 2001:500:12::d0d	US Department of Defense (NIC)
h.root-servers.net	198.97.190.53, 2001:500:1::53	US Army (Research Lab)
i.root-servers.net	192.36.148.17, 2001:7fe::53	Netnod
j.root-servers.net	192.58.128.30, 2001:503:c27::2:30	Verisign, Inc.
k.root-servers.net	193.0.14.129, 2001:7fd::1	RIPE NCC
l.root-servers.net	199.7.83.42, 2001:500:9f::42	ICANN
m.root-servers.net	202.12.27.33, 2001:dc3::35	WIDE Project

DNS

- Each zone is anchored at a specific domain node, but zones are not domains.
- *A DNS domain* is a branch of the namespace
- A zone is a portion of the DNS namespace generally stored in a file (It could consists of multiple nodes)
- A server can divide part of its zone and delegate it to other servers




Root Server
Authoritative name server for edu, org, com, in
Authoritative server for iitjammu.ac.in

DNS: Name Resolution

slashroot.in

- User program issues a request for the IP address of a hostname
- Local resolver formulates a DNS query to the local name server configured for the host
- Name server checks if it is authorized to answer the query.
  - *a) If yes, it responds.*
  - *b) Otherwise, it will query other name servers, starting at the root tree*
- When the name server has the answer it sends it to the resolver.

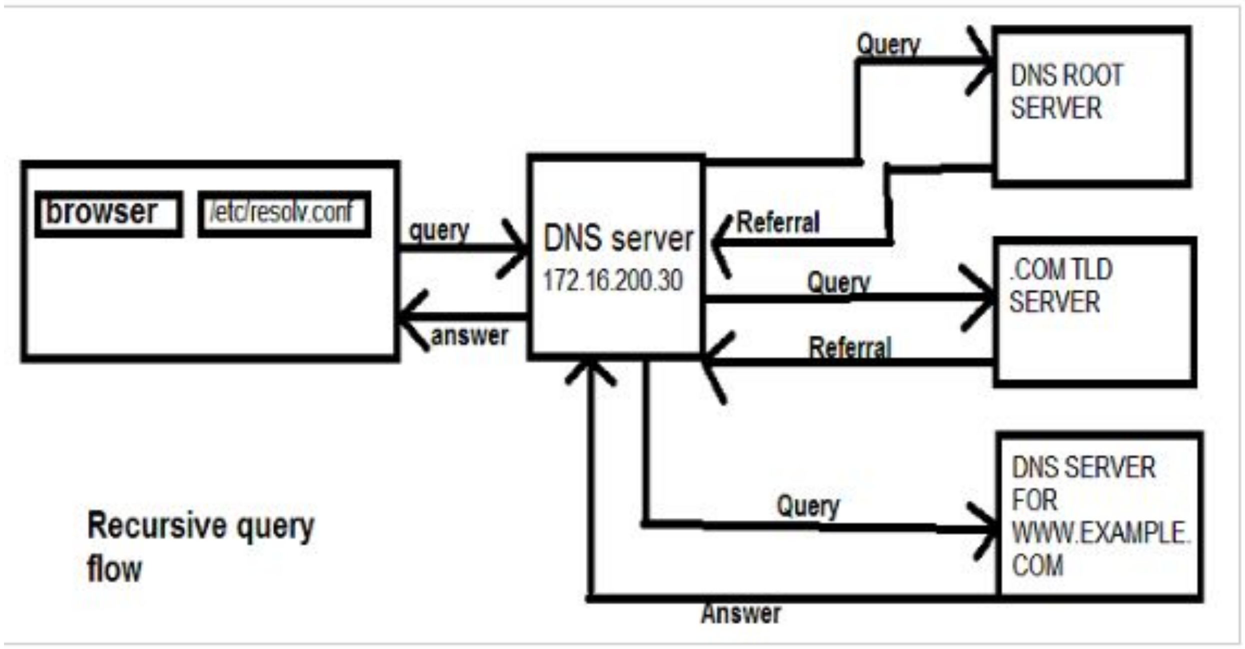
Recursive Queries

- **STEP 1:** You enter www.example.com in the browser. So the operating system's resolver will send a DNS query for the A record to the DNS server 172.16.200.30
- **STEP 2:** The DNS server 172.16.200.30 on receiving the query, will look through its tables(cache) to find the IP address(A record) for the domain www.example.com. But it does not have the entry.
- **STEP3:** As the answer for the query is not available with the DNS server 172.16.200.30, this server sends a query to one of the DNS root server, for the answer.

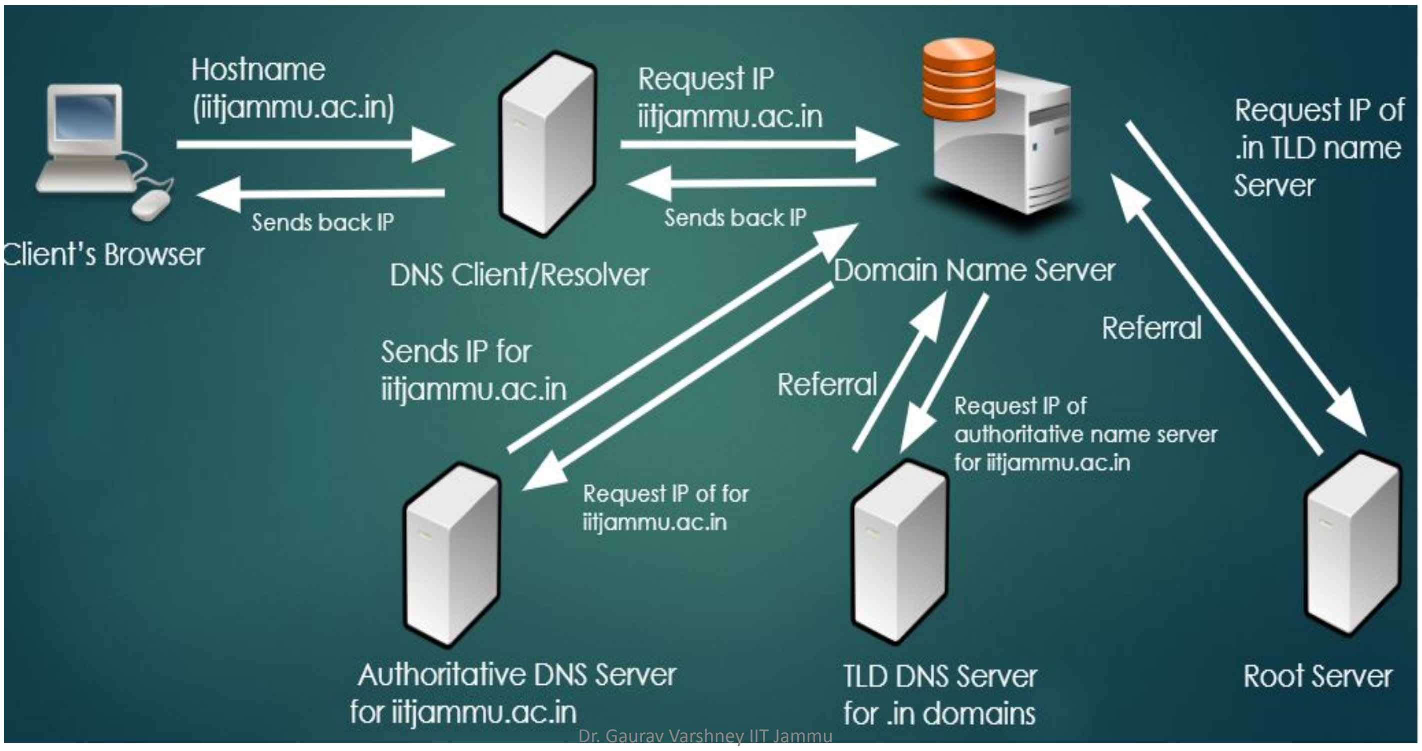
DNS: Recursive Resolution

- **STEP 4:** The DNS root server's will reply with a list of server's (referral) that are responsible for handling the **.COM** gTLD's.
- **STEP 5:** Our DNS server 172.16.200.30 will select one of the .COM gTLD server from the list given by the root server, to query the answer for www.example.com
- **Step 6:** Similar to the root server's , the gTLD server's are also iterative in nature, so it replies back to our DNS server 172.16.200.30 with the list of IP addresses of the DNS server's responsible for the domain(authoritative name server for the domain) www.example.com.
- **STEP 7:** This time also our DNS server will select one of the IP from the given list of authoritative name servers, and queries the A record for www.example.com. The authoritative name server queried, will reply back with the A record as below.
- *www.example.com = <XXX:XX:XX:XX> (Some IP address)*
- **STEP 8:** Our DNS server 172.16.200.30 will reply us back with the IP domain pair(and any other resource if available). Now the browser will send request to the IP given, for the web page www.example.com.

DNS: Recursive Resolution Flow



DNS: Resolution


Hostname (iitjammu.ac.in)
Sends back IP
Client's Browser
DNS Client/Resolver
Request IP iitjammu.ac.in
Sends back IP
Domain Name Server
Request IP of .in TLD name Server
Referral
Sends IP for iitjammu.ac.in
Referral
Request IP of authoritative name server for iitjammu.ac.in
Request IP of for iitjammu.ac.in
Authoritative DNS Server for iitjammu.ac.in
TLD DNS Server for .in domains
Root Server

DNS Resolution:

If ROOT and
TLD name servers support Recursion

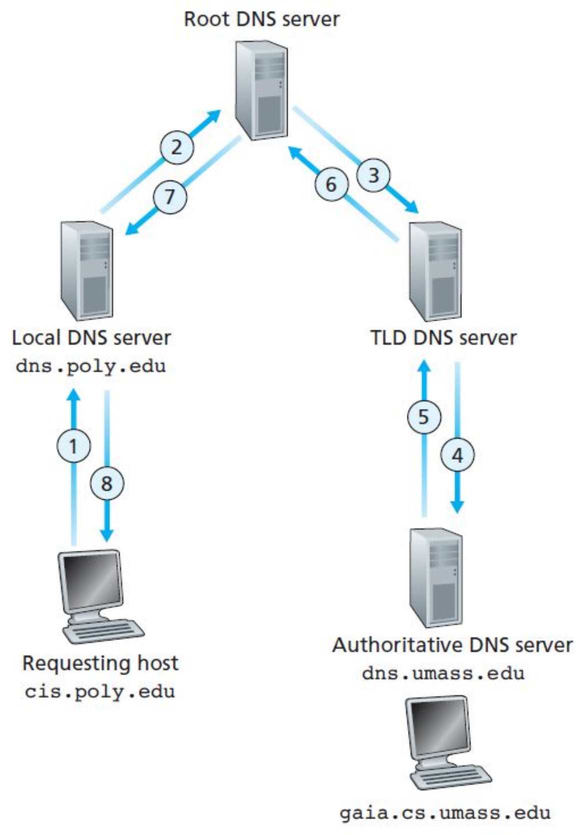


Figure 2.22 ♦ Recursive queries in DNS