1. What happens when you type a URL like www.google.com in a browser?

Step 1. URL is typed in the browser. “www.google.com”

Step 2. If requested object is in browser cache and is fresh, just move on to Step 8.

Step 3. DNS lookup to find the IP address of the server

1. *Check browser cache*: browsers maintain cache of DNS records for some fixed duration. So, this is the first place to resolve DNS queries.
2. *Check OS cache*: if browser doesn't contain the record in its cache, it makes a system call to underlying Operating System to fetch the record as OS also maintains a cache of recent DNS queries.
3. *Router Cache*: if above steps fail to get a DNS record, the search continues to your router which has its own cache.
4. *ISP cache*: if everything fails, the search moves on to your ISP. First, it tries in its cache, if not found - ISP's DNS *recursive search* comes into picture. DNS lookup is again a complex process which finds the appropriate IP address from a list of many options available for websites like Google. (Check the following DNS graph)

Step 4. Browser initiates a TCP connection with the server. (Check the following TCP graph 3-hand shake)

Step 5. Browser sends a HTTP request to the server.

Browser sends a *GET* request to the server according to the specification of HTTP(Hyper Text Transfer Protocol) protocol.

```
GET http://google.com/ HTTP/1.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:29.0) Gecko/20100101 Firefox/29.0
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
Host: google.com
Cookie: datr=1265876274-[...]; locale=en_US; lsd=WW[...]; c_user=2101[...]
```

Here, browser passes some meta information in the form of headers to the server along with the URL - "http://google.com". *User-Agent* header specifies the browser properties, *Accept-Encoding* headers specify the type of responses it will accept. *Connection* header tells the server to keep open the TCP connection established here. The request also contains *Cookies*, which are meta information stored at the client end and contain previous browsing session information for the same website in the form of key-value pairs e.g. the login name of the user for Google.

Step 6. Server handles the incoming request

HTTP request made from browsers are handled by a special software running on server - commonly known as *web servers* e.g. *Apache*, *IIS* etc. Web server passes on the request to the proper request handler - a program written to handle web services e.g. *PHP, ASP.NET, Ruby, Servlets* etc.

For example URL- *http://edusagar.com/index.php* is handled by a program written in PHP file - *index.php*. As soon as GET request for index.php is received, Apache (our webserver at edusagar.com) prepares the environment to execute index.php file. Now, this php program will generate a response - in our case a HTML response. This response is then sent back to the browser according to HTTP guidelines.

Step 7. Browser receives the HTTP response

```
HTTP/1.1 200 OK
Cache-Control: private, no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Pragma: no-cache
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Connection: Keep-Alive
Content-length: 1215
Date: Fri, 30 May 2014 08:10:15 GMT

.....<some blob> .....
```

HTTP response starts with the returned status code from the server. Following is a very brief summary of what a status code denotes:

- 1xx indicates an informational message only
- 2xx indicates success of some kind
- 3xx redirects the client to another URL
- 4xx indicates an error on the client's part
- 5xx indicates an error on the server's part

Server sets various other headers to help browser render the proper content. *Content-Type* tells the type of the content the browser has to show, *Content-length* tells the number of bytes of the response. Using the *Content-Encoding* header's value, browsers can decode the blob data present at the end of the response.

Step 8. Browsers displays the html content

Rendering of html content is also done in phases. The browser first renders the bare bone html structure, and then it sends multiple GET requests to fetch other hyper linked stuff e.g. If the html response contains an image in the form of img tags such as **, browser will send a HTTP GET request to the server to fetch the

image following the complete set of steps which we have seen till now. But this isn't that bad as it looks. Static files like *images, javascript, css* files are all cached by the browser so that in future it doesn't have to fetch them again.

Step 9. Client interaction with server

Once a html page is loaded, there are several ways a user can interact with the server. For example, he call fill out a login form to sign in to the website. This also follows all the steps listed above, the only difference is that the HTTP request this time would be a *POST* instead of GET and along with that request, browser will send the form data to the server for processing (username and password in this case).

Once server authenticates the user, it will send the proper HTML content(may be user's profile) back to the browser and thus user will see that new webpage after his login request is processed.

Step 10. AJAX queries

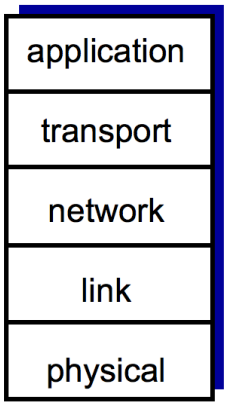
Another form of client interaction with server is through AJAX(Asynchronous JavaScript And XML) requests. This is an asynchronous GET/POST request to which server can send a response back in a variety of ways - *json, xml, html* etc. AJAX requests doesn't hinder the current view of the webpage and work in the background. Because of this, one can dynamically modify the content of a webpage by calling an AJAX request and updating the web elements using Javascript.

2. DNS TCP IP UDP

A. Internet Layer

Internet protocol stack

- *application:* supporting network applications
  - FTP, SMTP, HTTP
- *transport:* process-process data transfer
  - TCP, UDP
- *network:* routing of datagrams from source to destination
  - IP, routing protocols
- *link:* data transfer between neighboring network elements
  - Ethernet, 802.111 (WiFi), PPP
- *physical:* bits "on the wire"



B. How DNS works (iterated & recursive)

Local DNS name server

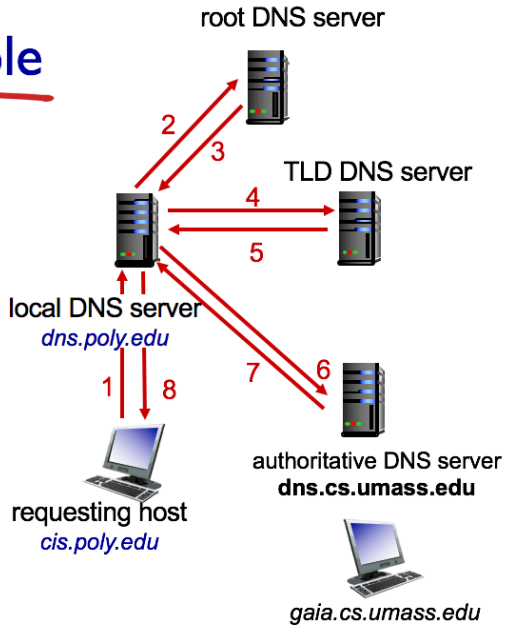
- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
  - also called "default name server"
- when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts as proxy, forwards query into hierarchy

DNS name resolution example

- host at cis.poly.edu wants IP address for gaia.cs.umass.edu

iterated query:

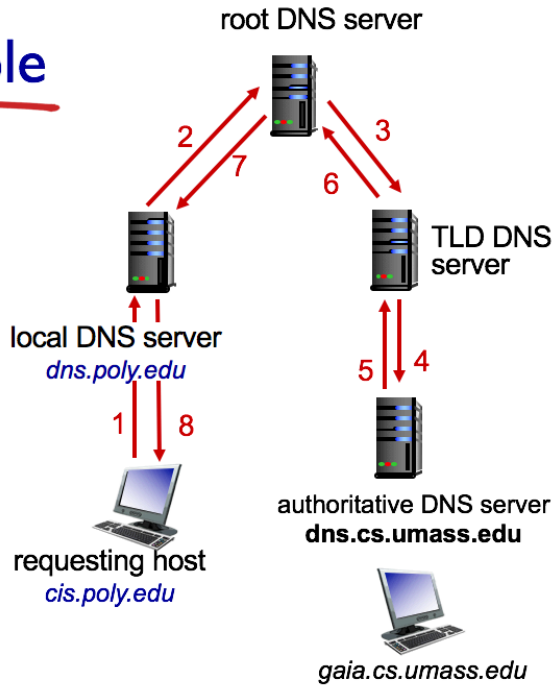
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



DNS name resolution example

recursive query:

- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy?



DNS records

DNS: distributed database storing resource records (RR)

RR format: `(name, value, type, ttl)`

type=A

- **`name`** is hostname
- **`value`** is IP address

type=NS

- **`name`** is domain (e.g., foo.com)
- **`value`** is hostname of authoritative name server for this domain

type=CNAME

- **`name`** is alias name for some "canonical" (the real) name
- **`www.ibm.com`** is really **`servereast.backup2.ibm.com`**
- **`value`** is canonical name

type=MX

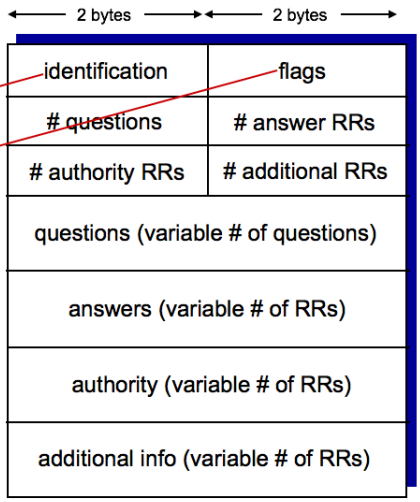
- **`value`** is name of mailserver associated with **`name`**

DNS protocol, messages

- *query* and *reply* messages, both with same *message format*

message header

- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



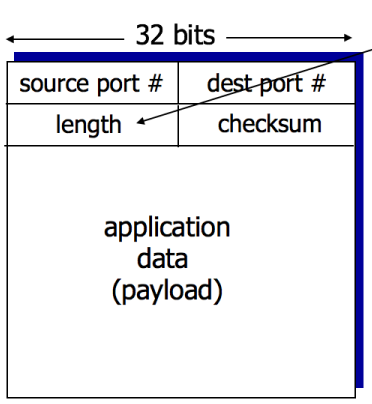
C. UDP

UDP: User Datagram Protocol [RFC 768]

- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out-of-order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

- UDP use:
  - streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS
  - SNMP
- reliable transfer over UDP:
  - add reliability at application layer
  - application-specific error recovery!

UDP: segment header



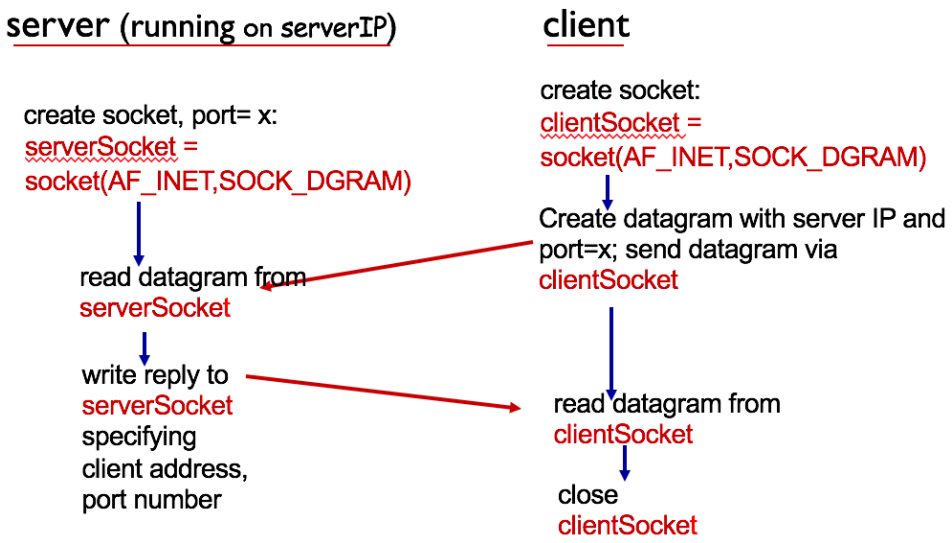
UDP segment format

length, in bytes of UDP segment, including header

why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small header size
- no congestion control: UDP can blast away as fast as desired

Client/server socket interaction: UDP

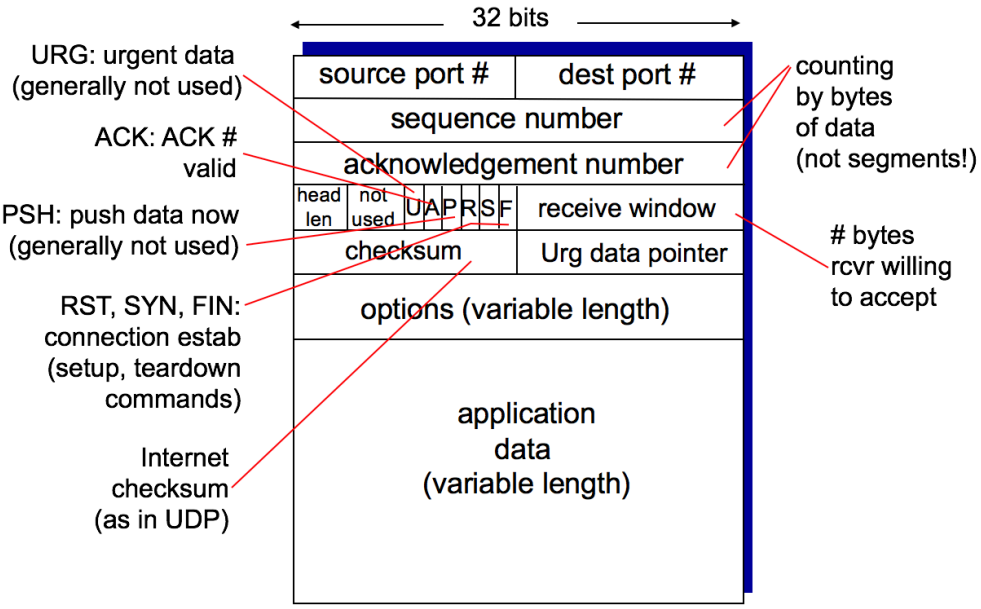


D. TCP

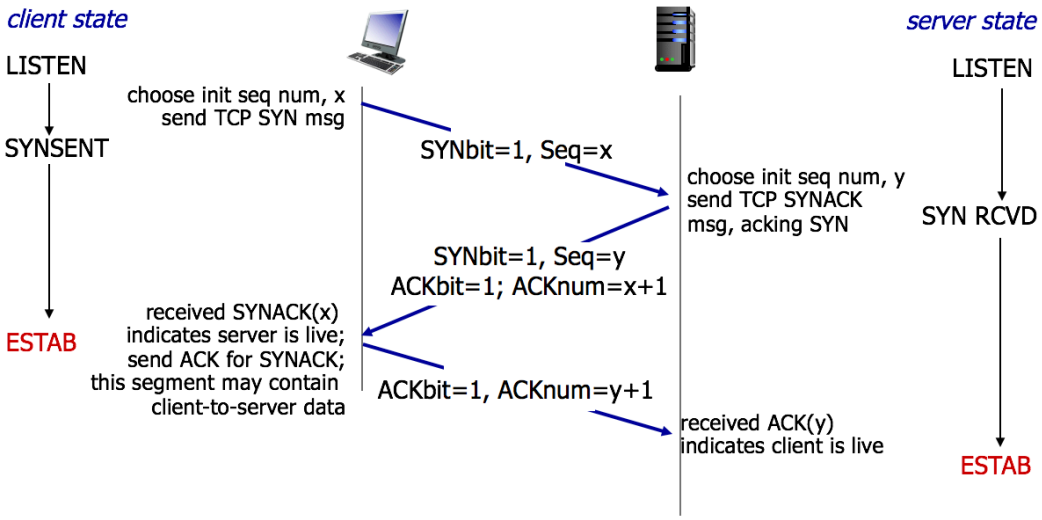
TCP: Overview RFCs: 793,1122,1323, 2018, 2581

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte steam:*
  - no "message boundaries"
- pipelined:
  - TCP congestion and flow control set window size
- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- connection-oriented:
  - handshaking (exchange of control msgs) inits sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

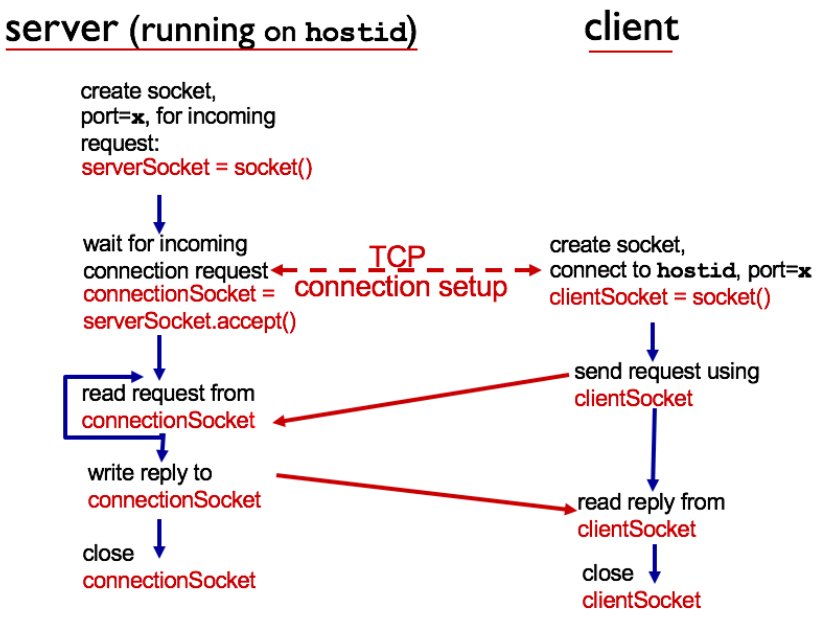
TCP segment structure



TCP 3-way handshake



Client/server socket interaction: TCP



3. HTTP and HTTPS

1) HyperText Transfer Protocol (HTTP),

   The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web. Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text.

2) HyperText Transfer Protocol Secure (HTTPS).

   Using HTTPS, the computers agree on a "code" between them, and then they scramble the messages using that "code" so that no one in between can read them. This keeps your information safe from hackers. They use the "code" on a Secure Sockets Layer (SSL), sometimes called Transport Layer Security (TLS) to send the information back and forth.

附录：

输入 URL 到页面返回的全过程

1. 在 URL 栏里输入 www.google.com 点击确认
2. 浏览器查找浏览器缓存，如果有域名的 IP 地址则返回，如果没有继续查找。
3. 系统查找系统缓存，如果有域名的 IP 地址则返回，如果没有继续查找。
4. 路由器查找路由器缓存，如果有域名的 IP 地址则返回，如果没有继续查找。
5. 本地域名服务器采用迭代查询，它向下一个根域名服务器查询。
6. 根域名服务器告诉本地域名服务器，下一次查询的顶级域名服务器 dns.com 的 IP 地址。
7. 顶级域名服务器 dns.com 告诉本地域名服务器，下一次查询的权限域名服务器 dns.google.com 的 IP 地址。
8. 本地域名服务器 dns.google.com 告诉本地域名服务器，所查询的主机 www.google.com 的 IP 地址。
9. 本地域名服务器最后把查询结果告诉主机。
10. 主机浏览器获取到 Web 服务器的 IP 地址后，与服务器建立 TCP 连接。
11. 浏览器所在的客户机向服务器连接请求报文。
12. 服务器接受报文后，同意建立连接，向客户机发出确认报文。
13. 此处客户机与服务器之间 TCP 连接建立完成，开始通信。
14. 浏览器发出取文件命令：GET。
15. 服务器给出响应，将指定文件发送给浏览器。
16. 浏览器释放 TCP 连接。
17. 浏览器所在的主机向服务器发出释放报文，然后停止发送数据。
18. 服务器接收到释放报文后发出确认报文，然后将服务器上未传输完的数据发送完。
19. 服务器数据传输完毕后，向客户机发送释放报文。
20. 客户机接收到报文后，发出确认，然后等待一段时间后，释放 TCP 连接；
21. 浏览器显示页面中所有文本。

Reference

[1] http://blog.csdn.net/mevicky/article/details/46789381

[2] http://edusagar.com/articles/view/70/What-happens-when-you-type-a-URL-in-browser