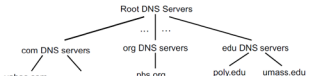
DNS helps to resolve domain names (web address, human readable) into IP addresses (machine readable) - *so that you can be found in the internet.*

DNS: a distributed, hierarchical database



DNS: root name servers

- contacted by local name server that can not resolve name
- root name server: (only 13 in the world)
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server

TLD, authoritative servers

top-level domain (TLD) servers:

- responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp, sg
- Verisign Global Registry Services maintains servers for .com TLD
- Educause for .edu TLD

authoritative DNS servers:

- organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
- can be maintained by organization or service provider

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  - has local cache of recent name-to-address translation pairs (but may be out of date!)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: caching, updating records

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
    - thus root name servers not often visited
- cached entries may be *out-of-date* (best effort name-to-address translation!)
  - if name host changes IP address, may not be known Internet-wide until all TLDs expire
  - who (client or server?) sets the TTL?
- update/notify mechanisms proposed IETF standard
  - RFC 2136

DNS RECORDS

The "data structure" of a DNS resource record (RR). RR is used by clients who query hostname-IP resolution

RR = (name, value, type, TTL)

Name	Value	Type	TTL
hostname, e.g en.wikipedia.com, or mywebsite.net	IP address	**A (Authoritative)**	TTL
A domain or hostname in question, e.g wikipedia.com (for wiki domain), or a gtld.net (for .net domain)	A nameserver that serves this domain, e.g: en.wikipedia.com, ns1.mywebsite.net – they MIGHT know the IP of this hostname	**NS (Name Server)**	TTL
alias name for some canonical (real) hostname, e.g: alias.mywebsite.net	canonical hostname name, e.g: mywebsite.net	**CNAME (Canonical Name)**	TTL
domain, e.g: example.com	mailserver name, e.g: mail.example.com	**MX (Mail)**	TTL

Inserting records into DNS

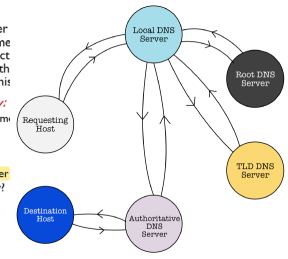
- example: new startup "Network Utopia"
- register name networkuptopia.com at *DNS registrar* (e.g.,Verisign registry)
  - provide names, IP addresses of authoritative name server (primary and secondary)
  - registrar inserts two RRs into .com TLD server: `(networkutopia.com, dns1.networkutopia.com, NS)` `(dns1.networkutopia.com, 212.212.212.1, A)`
- create authoritative server type A record for www.networkuptopia.com; type MX record for networkutopia.com

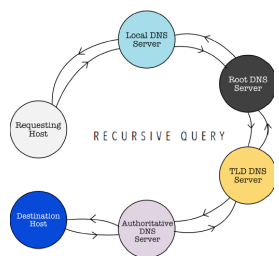
iterated query:

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

recursive query:

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





DNS RECORDS

The "data structure" of a DNS resource record (RR). RR is used by clients who query hostname-IP resolution

RR = (name, value, type, TTL)

Types:

1. **A: Authoritative,** it contains the IP address of the hostname in question. No longer delegates you to further probing the query and you should go to that IP address
2. **NS: Name Server,** it tells which nameservers are authoritative for a **zone (a collection of computer network, e.g: SUTD.com zone, Wiki zone.com, etc).** You can have multiple NS records for load distribution (increasing availability)
3. **CNAME: Canonical Name**, just another alias for a hostname. Requires more probing (queries) to resolve to an IP eventually. It is like the same website with two or more URLs.
4. **MX: Mail server**, points you to the mailserver name responsible for your domain.

DNS RECORDS

An example: You want to know the IP of **b.example.com**. You ask **localDNSserver.com** (A local DNS nameserver whose IP you know, usually a public knowledge), it has the following records:

Name	Value	Type	TTL
b.example.com	NS2.server.com	**NS**	TTL
NS2.server.com	111.222.125.124	**A**	TTL

So your **localDNSServer.com** do not know what is **b.example.com**'sIP, but it has an NS entry regarding it. It means that we should ask **NS2.server.com** on where is **b.example.com**. It has an A record of **NS2.server.com** so we (or the localDNSServer) can go and ask **NS2.server.com** if it has the A record of **b.example.com**. Now suppose **NS2.server.com** has the following record:

Name	Value	Type	TTL
a.example.com	10.12.14.145	**A**	TTL
b.example.com	a.example.com	**CNAME**	TTL

An example (continued): Now the first query to **NS2.server.com** shows that **b.example.com** is a CNAME record, so it is simply an alias of another hostname called **a.example.com**

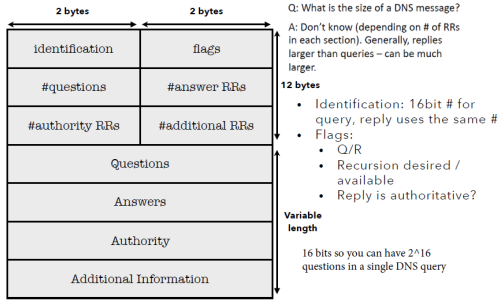
Name	Value	Type	TTL
a.example.com	10.12.14.145	**A**	TTL
b.example.com	a.example.com	**CNAME**	TTL

Hence, you, or the local DNS server has to continue probing and send another DNS query asking where is a.example.com, and finally NS2.server.com returns the A record with IP of 10.12.14.145.

DNS PROTOCOL

This is the protocol to make a DNS query or reply.

Both query and reply has the same message format



Q: What is the size of a DNS message?

A: Don't know (depending on # of RRs in each section). Generally, replies larger than queries – can be much larger.

- Identification: 16bit # for query, reply uses the same #
- Flags:
  - Q/R
  - Recursion desired / available
  - Reply is authoritative?

16 bits so you can have 2^16 questions in a single DNS query

Attacking DNS

DDoS attacks

- Bombard root servers with traffic
  - Not successful to date
  - Traffic Filtering
  - Local DNS servers cache IPs of TLD servers, allowing root server bypass
- Bombard TLD servers
  - Potentially more dangerous

Redirect attacks

- Man-in-middle
  - Intercept queries
- DNS poisoning
  - Send bogus replies to DNS server, which caches

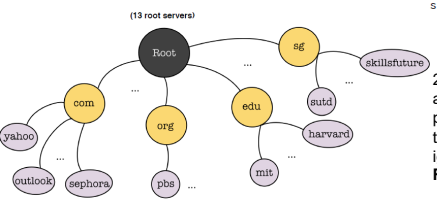
Exploit DNS for DDoS

- Send queries with spoofed source address: target IP
- Requires amplification

DNS CACHING

Once local name servers learns hostname-ip mapping, it **caches** the mapping

- TLD servers are typically cached in DNS local name servers, **hence faster resolution**
- Cached entries may be **out of date**
- **DNS authoritative name server (the one that hosted the website) decides the DNS record TTL**
- DNS local name server re-queries when TTL expires



1. **DNS Namespace:** The entire tree (all nodes)
2. **Domain:** represents a the entire set of names / machines that are contained under an organizational domain name (subtree), eg: ".com" websites are part of the "com" domain, **en.wikipedia.com** and **id.wikipedia.com** websites are part of the "**wikipedia.com**" domain.
3. **Zone:** DNS namespace can be broken into zones for which individual DNS servers are responsible. Unlike "domain" that's a subtree, a zone can just be a bunch of sibling nodes. A DNS zone is not necessarily associated with one server. A DNS server can also contain multiple zones.

Reflections on DNS

- Protection domains to the next level (cf. OS processes)
  - Different *physical machines*: strong modularity, strong fault isolation
- Distributed (hierarchical) servers: scalability
- *Indirection* (name to IP addr) as design principle has many virtues
  - Late binding: runtime — Server can move
  - Many-to-one mappings — Aliasing
  - One-to-many mappings — Load balancing
- Cost of indirection?
  - Delay (overhead), security (poisoned/intercepted mappings)

2. Suppose a Web browser visits an HTML page with four embedded JPEG objects. At least how many RTTs are needed for the full page to load assuming (i) non-persistent HTTP with the browser configured for three parallel connections, and (ii) persistent HTTP (over a TCP connection) without any pipelining. You can assume that all the HTML and JPEG objects are small so that you can ignore their transmission time. You can also ignore TCP close. **(i) 6 RTTs (2 RTTs for the base page, then 2 RTTs for 3 JPEG objects in parallel, then 2 RTTs for the 4th JPEG object). (ii) 6 RTTs (2RTTs for the base page, then 1 RTT for each JPEG object).**

(a) Was this query iterative or recursive? **Recursive [2pts]. Because both the rd (recursion desired) and ra (recursion available) flags are set.**

(b) As far as you can tell from the output, what functions do the machines sutd1.eo.olook.com and nsrv.sutd.edu.sg serve respectively? **sutd1.eo.olook.com is an email server for the sutd.edu.sg domain [2pts]. nsrv.sutd.edu.sg is an authoritative DNS name server for the email server**

2. I ran the command **dig MX sutd.edu.sg** and got the following (edited and excerpted) output:

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 21444
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 3, ADDITIONAL: 6

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;sutd.edu.sg.                    IN      MX

;; ANSWER SECTION:
sutd.edu.sg.            3600    IN      MX      5 post.global.frontbridge.com.
sutd.edu.sg.            3600    IN      MX      0 sutd1.eo.olook.com.

;; AUTHORITY SECTION:
sutd.edu.sg.            2350    IN      NS      nsrv.sutd.edu.sg.
sutd.edu.sg.            2350    IN      NS      dnssec1.singnet.com.sg.
sutd.edu.sg.            2350    IN      NS      dnssec3.singnet.com.sg.

;; ADDITIONAL SECTION:
dnssec1.singnet.COM.sg. 2328    IN      A       165.21.83.11
dnssec1.singnet.COM.sg. 2328    IN      AAAA    2001:c20:18:a::36
dnssec3.singnet.COM.sg. 2328    IN      A       165.21.100.11
dnssec3.singnet.COM.sg. 2328    IN      AAAA    2001:c20:10:a::37
ns.sutd.edu.sg.         13487   IN      A       202.94.70.254

;; Query time: 15 msec
;; SERVER: 192.168.0.1#53(192.168.0.1)
```

Suppose that your department has a local DNS server for all the computers in the department. You are an ordinary user (i.e., not a system administrator). Can you determine if an external web site was likely accessed recently from a computer in your department? Explain. **Yes [1pt]. One possible method is to do a DNS lookup for the web server in question and note the time taken for getting the answer. If the time was small (e.g., LAN delay of a say few milliseconds), the answer was likely cached by the local server due to a recent access.**

2. What is a nonce? Explain why the use of a nonce in an authentication protocol can help defend against the playback attack. **A nonce is an identifier that is guaranteed to be fresh (i.e., never used before). Because a nonce is fresh, whoever replies to it must also be fresh and can't be a previously recorded version of the replier**

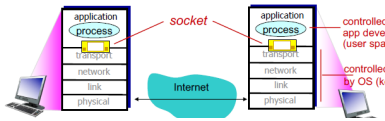
3. How does DNS, in usual practice, mitigate the problem of outdated answers being cached and used? Does this mechanism solve the problem completely? Why**? DNS lookup results are subject a specified time-to-live (TTL) that limits how long the results can be cached [2pts]. This increases the chance that the cached results will be refreshed before a user queries their outdated versions.**

No [1pt]. Since a DNS name-to-value mapping can change at any time, after the change, it is still possible for a user to query the name before the TTL expires [2pts].

Recap: DNS helps to resolve domain names (web address, human readable) into IP addresses (machine readable) - *so that you can be found in the internet.*

Issue: Your machine runs several programs to the same time. We need a way to pass the packet from the internet to the application that needs it, e.g: Steam, Web Browser, Telegram, etc.

Solution: SOCKET - a combination of IP plus port



multiplexing at sender: handle data from multiple sockets, add transport header (later used for demultiplexing)

demultiplexing at receiver: use header info to deliver received segments to correct socket

MULTIPLEXING at sender, handling data from multiple sockets to **send out**. Add transport packet header (for the layer below app layer)

DEMULTIPLEXING at receiver, handling data from multiple sockets to **receive in**. Use transport packet header to deliver to the right socket number

COMMUNICATION PROTOCOLS

DNS is used to translate between web address to IP address. Then, we use socket to direct the packet to the right app in the computer.

Now we need **communication protocols** between two applications over the internet.

UDP socket identified by **2-tuple**: (destination IP, destination port #)

UDP

TCP

TCP socket identified by **4-tuple**: (source IP, source port #, destination IP, destination port #)

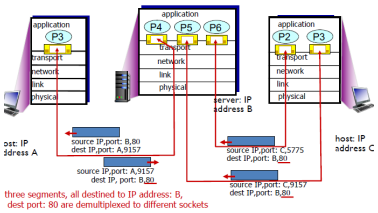
Socket programming in **application layer** relies on **TCP and UDP** services in **transport layer**

Connection-oriented (TCP) demux

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- demux: receiver uses *all four* values to direct segment to appropriate socket
- server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple

Connection Oriented (TCP) vs. Connection Less (UDP):
- connection setup
- reliability
- flow control
- congestion control

Connection-oriented demux: example



Socket programming *with TCP*

client must contact server
- server process must first be running
- server must have created socket (door) that welcomes client's contact

client contacts server by:
- Creating TCP socket, specifying IP address, port number of server process
- *when client creates socket:* client TCP establishes connection to server TCP

Heavy on server side, need create sockets

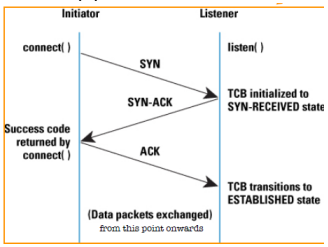
- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients

application viewpoint:

TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server

TCP: This protocol provides a reliable, in-order byte stream transfer (called pipe) between client and server.

TCP Connection establishment: known as handshake, its a 3-step protocol.



UDP: This protocol provides a unreliable-but-fast, may-be-out-of-order transfer of bytes between client and server

UDP socket programming

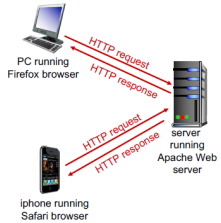
UDP: no "connection" between client & server
- no handshaking (to open connection) before sending data
- sender explicitly attaches IP destination address and port # to each packet
- rcvr extracts sender IP address and port# from received packet

UDP: transmitted data may be lost or received out-of-order

Application viewpoint:
- UDP provides *unreliable* transfer of groups of bytes ("datagrams") between client and server

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - *server:* Web server sends (using HTTP protocol) objects in response to requests



HTTP overview (continued)

uses TCP:
- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

non-persistent HTTP

(HTTP/1.0)
- at most one object sent over TCP connection
  - connection then closed
- downloading multiple objects required multiple connections

Basic HTTP is "stateless"
- server maintains no information about past client requests
- Simple: no need to maintain history and no need to clean up state if server/client crashes
- But *cookies* can be added to keep state

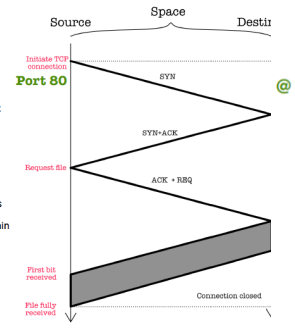
persistent HTTP

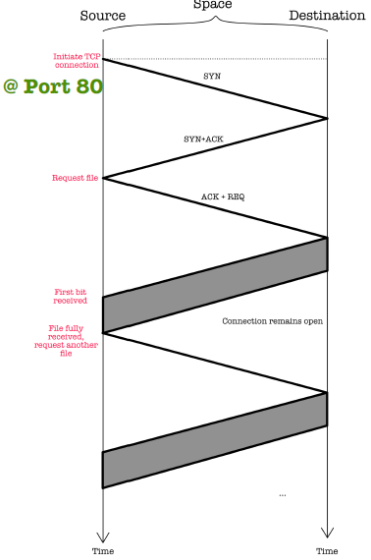
(HTTP/1.1)
- server leaves connection open after sending response
- multiple objects can be sent over single TCP connection between client, server

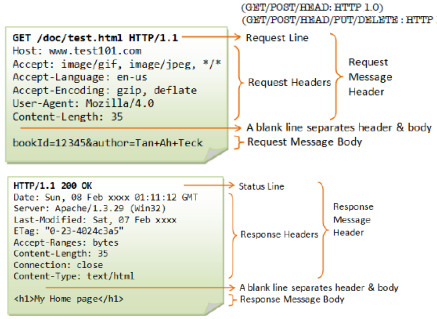
Persistent HTTP

non-persistent HTTP issues:
- requires 2 RTTs per object
- OS overhead for ***each*** TCP connection
- browsers often open *parallel* TCP connections to fetch referenced objects (degree of parallelism can be controlled, e.g., up to 5 parallel connections allowed at a time)
  - i.e., start retrieving another object before completing retrieval of a previous object
  - Usually faster than serial retrievals (by increasing utilization of the network)

persistent HTTP:
- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- multiple objects can be sent over single TCP connection between client, server
- client can send requests as soon as it encounters a referenced object, no need to wait for previous request to finish (this is called *pipelining*, i.e., multiple outstanding HTTP requests within *same* TCP connection)
- as little as one RTT for all the referenced objects

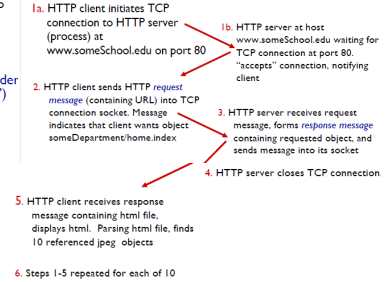






Non-persistent HTTP

suppose user enters URL: `www.someSchool.edu/someDepartment/home.index` (contains text, references to 10 jpeg images)



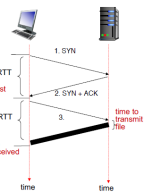
NB. TCP uses *3-way handshake* to open a connection. The 3 messages are: SYN, then SYN + ACK, then ACK. Application (e.g., HTTP) data can be "piggy-backed" onto the last ACK. (See next slide for illustration.)

Non-persistent HTTP: response time

RTT = round trip time (time for a small packet to go to another machine & come back)

HTTP response time:
- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time
- non-persistent HTTP response time (for one URL object retrieved as a file) =

2RTT+ file transmission time



1. If a Web server is to support N simultaneous HTTP connections, each from a different client browser, how many sockets will the Web server need to open during its whole execution? **N + 1. [One welcome socket for accepting connections, then one socket for the established connection with each client.**
2. Assume that you have base HTML file with 30 embedded images, images & base file are small enough to fit in one TCP segment. How many RTT are required to retrieve base file & images under-following condition :

(i) Non-Persistent connection without parallel connection

For each image, 2 RTT are required -- one for TCP connection and one for image to send. So transmit time for 30 images = 2*(30 RTT) = 60 RTT| Total time = 2 RTT+60 RTT = **62RTT**

(2RTT is the initial required connection one for TCP connection and one for HTML base file)

(ii) Non-persistent connection with 10 parallel connection

Here 10 images can be send simultaneously. So for 30 images it required -> 2*(30/10) = 6RTT |Total time = 2 RTT + 6 RTT = **8RTT**

(iii) Persistent connection without pipe-lining

Here TCP connection is required again and again.

So for 30 images it requires -> 30 RTTs | Total time = 2 RTT + 30 RTT = **32RTT**

(iv) Persistent connection with pipe-lining

In Pipe-lining connection we can send all images in 1RTT. | Total time = 2 RTT + 1 RTT = **3RTT**

3. A Web client that has cached an object can use the HTTP **_conditional GET_request** to check with the Web server that the cached copy is still up-to-date.
4. True or false: In HTTP/1.1 (persistent HTTP), multiple HTTP GET requests sent in a pipelined fashion over a TCP connection must complete in the same order in which they were sent. **TRUE**