Domain Name System (DNS)

the DNS response for connect the domain to the IP address like bilibili.com → 8.134.50.24

feature

- server ⟷ client
- UDP, port 53
- distributed database implemented in hierarchy of many name server

Services

- Services
  - hostname to IP address translation
  - Host aliasing (cname)
  - Mail server aliasing (mx)
  - Load distribution like many web server ip has the same name

distributed:

- single point of failed will be destructive if all the clients use the single DNS
- Traffic volume: large query, low write in ops, and the latency request is critical
- Distant centrized database: same as point 2
- Maintenance: distributed make DNS can be maintained easily

Hierarchy:

Root DNS Server

1. com DNS Servers
   - Yahoo.com DNS servers
   - amazon DNS servers
2. org DNS Servers
   - pbs.org NDS servers
   - …
3. edu DNS server …

Overview: Root → Top Domain → Authoritive

Root Name Servers

- Contacted by local name server that can not resolve name
- Root name serves:
  - Contact authoritative name server if name mapping is unknown
  - Gets mapping
  - Returns mapping to local name server

1. Top-level Domain (TLD) Servers:

   - responsible for com, org, net, edu, aero, jobs, museums, and all Top-level country domains, like cn, uk, fr, ca, jp

2. Authoritative DNS Servers:

   - Organization's own DNS servers providing authoritative hostname to IP mappings for organization's name hosts
   - can be maintained by organization or service provider

3. Local DNS Servers:

   does not strictly belong to hierarchy each ISP has one when host makes DNS query, query is sent to local DNS server, and resent the query to the hierarchy structure.

4. **Query Flow**:

   (a) Iterated Query

   - Client wants IP for www.amazon.com:
   - Client contact the local DNS

   • Local DNS queries root server to get ip, but not found, and redirect to com DNS Server by root DNS • Local DNS querues .com (TLD) DNS server to get amazon.com DNS server, but not found, and asked to go to Authoritative DNS • Client queries amazon.com (Authoritative) DNS server to get IP address for www.amazon.com

   - Done

   (b) Recursive Query

   - dig + trace
   - put burden of name resolution on contacted name server, like guest ask: the restraunt manager go ask the waiter, and watier find the dish, then the dish pass to the waiter → manager → guest
   - Heavy load at upper levels of hierarchy

Caching, Updating records

- Once Any name server learns mapping, it caches mapping and cache entries **timeout** after some time (TTL) TLD servers typically cached in local name servers (thus, the root name servers nt often visited)
- cache entries might have been out-of-date: if name host exchanges IP address, may not be known internet-wide untail all TTLs expired

DNS records:

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

type name:

- type=A name is hostname value is IP address
- type=NS name is domain value is hostname of authoritative name server for this domain
- type=CNAME name is alias name for some "canonical (the real)" name
- type=MX value is name of mailserver associated with name

Dns MSG

Query and reply uses the same messages format

========== = identification | flags =| = question | answer RRs =| = authoriry | additional RRs =| = questions (variable # of RRs) =| = authority (variable # of RRs) =| = additional info (variable # of RRs) | ==========

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

    - recursion available
    - reply is authoritative
- **question**: name, type fields for a query
- **ansers**: RRs in response to query
- **authority**: Records for authoritative servers
- **Additional** info: Additional ”helpful” info that may be used

P2P Application

- No always-on server
- Arbitary end systems directly communicate
- Peers change IP addresses

Time Cost Comparison with S/C

assume that the whole network contains N clients, and 1 server (for p2p, the server can be peer itself and / or Tracker Server) the network has enough bandwidth, we do not consider the bandwidth limit here and client has upload speed u_i, and has min download speed d_{min} the server has upload speed u_s we got a file sized F, in the C/S architecture, the server need to send the file to all the N clients sequentlly

then

the C/S architecture time cost:

- the servre upload time: $N(F/u_s)$
- the client download max time: (F/d_{min})

→ D_c-s = $max\{N(F/u_s), F/d_{min}\}$

- Cost of time is linear with N

the P2P architecture time cost:

- the server upload time: (F/u_s)
- the slowest client download time: (F/d_{min})
- the whole system upload time: $(F / u_s + sum(u_i))$

→ D_{p2p} = $max\{F/u_s, F/d_{min}, NF / (u_s + sum(u_i))\}$

- Cost of time is not linear with N, when N is large enough, the time cost will be stable
- fianlly, the p2p is faster than C/S when N is large enough

Architecture of P2P file sharing

Network Components:

- Peers: end systems that run P2P applications
- Tracker server: keeps track of all peers participating in file sharing
- Peers communicate directly with each other to share files

BitTorrent

to share a file or group of files, the initiator first creates a .torrent, a small file that contains:

- Metadata about the files to be shared
- information about the tracker, the computer that coordinates the file distribution

1. BitTorrent Metadata:

   Explanation: A 20MB file is divided into 20 pieces, each 1MB in size. Each piece is associated with a **SHA-1 hash value** to verify its integrity. Additionally, a map specifies which pieces belong to which part of the file(s).

   Expanded Details:

   - **Length of the File**:
     - Specifies the exact size of the file (in bytes).
     - Helps peers track progress and identify how much data remains.
   - **Piece Size**:
     - Pieces are uniform in size (e.g., 1MB or 256KB).
     - Smaller sizes improve parallelism but increase overhead; larger sizes decrease overhead but may limit swarm efficiency.
   - **Mapping of Pieces to Files**:
     - Provides a correspondence between byte ranges or offsets in the file(s) and specific pieces.
     - Useful for multi-file torrents, enabling accurate reassembly of the original files.
   - **SHA-1 Hashes of All Pieces**:
     - Ensures integrity for each piece when it is downloaded.
     - Peers calculate the hash of received pieces and compare it to this precomputed value to discard corrupted data.
   - **Tracker Reference**:
     - Indicates the tracker server that coordinates the swarm.
     - The tracker maintains peers’ IPs, available pieces, and acts as a directory for the torrent.

   Overall, metadata ensures smooth, secure, and efficient sharing within the BitTorrent network.

2. DataFlow:

   (a) Peer joining torrent:
     - has no pieces, but will accumulate pieces over time from other peers
     - register with **tracker** to get list of peers, connects to subset of peers (”neighbors”)

   (b) while downloading: peer uploads pieces to other peer

   (c) peer may change peers with whom it exchanges pieces

   (d) Perrs may come and go

   (e) Once peer has entire file, it may (selfishly) leave or remain in torrent

   Leecher Converting to Seeder:

   - As soon as a leecher has a complete piece, it can potentially share it with other downloaders
   - Eventually each leecher becomes a seeder by obtains all the pieces, and assemble the file,. Verifies the ”checksum” of the file
   - at the initial stage of torrent, there are few seeders, so the download speed is slow
   - but as more leechers convert to seeders, the download speed increases for all peers
   - the teacher’s ppt display that the seeder will finally more than leecher.

3. Piece selection

peers may have different pieces, and selection algorithm will affect performance

(a) Rarest First

- Determine themost rare among you peers, and download them first
- this ensures that the most commonly available pieces are left for later, when there are more peers to get them from
- and it reduce the risk that the rarest piece distincted when downloading not so rare pieces. (seeders may leave, including those hold the rarest piece)

(b) Endgame Mode

- Near the end, missing pieces are requested from every peer contains
- this ensures taht a download is not prevented from completion due to a single peer with a slow
- some bandwidth is wasted, but in practice, this is not too much

(c) Random First Piece

- initially, a peer has nothing to trade
- Important to get a complete piece As Soon As Posible
- Randomly select a piece of file and download it

4. BitTorrent Built-in incentive mechanism

(a) Choking Algorithm

- chocking is temporarily refusal to upload (I don't want to share my file with you!). It is one of BT's most powerful idea to deal with free riders
- the choked peer can still upload to me, but I won't upload to it
- thus, this peer have to sacrefice its upload capacity to get download capacity from me
- and I can same my bandwidth for those who are willing to share with each other
- for Avoiding fre rider and avoiding network congestion: Tit-for-tat is the best
- (btw, those who chocked might not be unwilling to share with you, in the tit-for-tat algo, those who have more upload bandwidth will group and be centered by the other peers to establish stable connection)

(b) Optimistic unchoking:

- A peer sends pieces to those four peers currently sending here chunks at highest rate
- for an interval, your client will sort out the fastest peers who upload to me, and your client will unchoking them (happy to upload to them)
- that is, as you upload getting faster, youre more possible to sort to unchocked list by other, which establishing virtuous cycle

Issues

- the newer peer has no chance: it has not been upload to anyone
- might missing the best peer: the best 4 might be not the globally best

Solution the Optimisticc unchocking give the random peer to upload to.

- if this peer has fast upload speed with me, then it is highly possible to sort to top 4 next time
- if it sucks, then it will be chocked by your client, and the other will give it a chance

thus: top4 → stable coworker optimistic unchoking → find new coworker

Socket Programming:

Socket: door between application proess and end-end-transport protocol

UDP-

unreliable, datagram no need to establish connection: no handshake, the msg will contains the ip and port **Issue** the msg might get lost or the sequence might be wrong

Flow ==========

Server client

1. create socket port = x create socket socket(AF_{INET}, $SOCK_{DGRAM}$)
2. read datagram from serverSocket «< Create datagram with server IP and port via ClientSocket
3. write reply to ServerSocket specifying client address port »> read datagram from clientSocket
4. close clientSocket

Server

```
from socket import *

SERVER_PORT = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(("", serverPort))
while True:
     message, clientAddress = serverSocket.recvfrom(2048)
     modifiedMessage = message.decode().upper()
     serverSocket.sendto(modifiedMessage.encode())
```

Client

```
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_DGRAM)

message = input("input lowercase sentence")
clientSocket.sendto(message.encode(), serverName, serverPort)
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print(modifiedMessage.decode())
clientSocket.close()
```

TCP

- reliable, byte stream-oriented
- the server process must run and create a socket first that welcom the clients
- clients contact the server through following ways:
  - IP, port
  - client establish connection to server TCP when client creates socket
- when the client contact hte server, server TCP will create a new socket for server process to communicate
- allow multiple clients communicate with single server

Server

```
from socket import *

SERVER_PORT = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(("", serverPort))
while True:
     message, clientAddress = serverSocket.recvfrom(2048)
     modifiedMessage = message.decode().upper()
     serverSocket.sendto(modifiedMessage.encode())
```

Client

```
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_DGRAM)

message = input("input lowercase sentence")
clientSocket.sendto(message.encode(), serverName, serverPort)
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print(modifiedMessage.decode())
clientSocket.close()
```