2. P2P Architecture, Socket Programming

04 February 2025 09:21

Peer-to-Peer Architecture

- No always on server
- Arbitrary end systems directly communicate

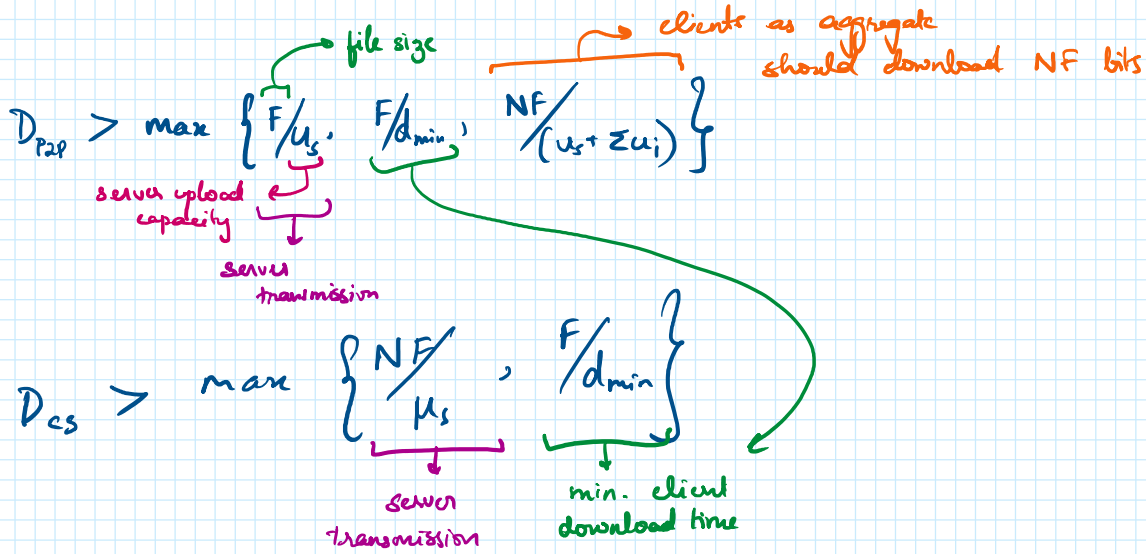
Spotify? Skype?

Self scalability

New peers → More service capacity
New peers → More service demands

Distribution time: Time it takes to get a copy of the file to all N peers

- IP addresses change; devices intermittently connected



BitTorrent

- File divided into 256 Kb chunks; exchanged by group of peers → torrent
- Tracker: keeps list of peers in torrent, gives this list to new peers
- New peers connect to "neighbours": subset of list of all peers
- Churn: peers come & go. Once peer has file → selfishly leave / altruistically remain
- Periodically ask neighbours for list of chunks, then request rarest missing first
- Tit for tat: send chunks to current top 4 w/ highest rate; other users choked
- Reevaluate top 4 every 10 seconds
- Every 30 seconds, optimistically unchoke

Socket Programming

SOCKET PROGRAMMING

socket → door b/w application process and end-to-end transport protocol

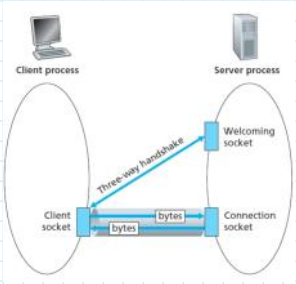
types

UDP

- connectionless
  - ↓
  - no handshake
- explicitly attached IP destination address, port #
- data may be lost/arrive out of order
- groups of bytes (datagrams) transferred

TCP

- connections established
- reliable, in order byte stream (pipe)
- three way handshake
  - ↳ welcoming socket
  - ↳ connection socket
  - ↳ client socket
- when contacted by client, server TCP creates new socket to communicate
  - ↳ allows it to service multiple clients



how many client requests can big servers accomodate?

HTTP, DNS → application layer

SSH

memorize port nos, protocol and description

UDP Socket Programming

```
from socket import *
serverName = 'hostname'
serverPort = 12000

clientSocket = socket(AF_INET, SOCK_DGRAM)

message = input("Enter a message: ")

clientSocket.sendto(message.encode(),
(serverName,serverPort))
modifiedMessage, serverAddress =
clientSocket.recvfrom(2048)
print(modifiedMessage.decode())

clientSocket.close()
```

```
from socket import *
serverPort = 12000

serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(("", serverPort))

while True:
   clientMessage, clientAddress = serverSocket.recvfrom(2048)
   modifiedMessage = clientMessage.decode().upper()
   serverSocket.sendto(modifiedMessage.encode(), clientAddress)
```

TCP Socket Programming

```
from socket import *
serverName = 'hostname'
```

```
serverPort = 12000

clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))

message = input("Enter a message: ")

clientSocket.send(message.encode())
modifiedMessage = clientSocket.recv(2048)
print(modifiedMessage.decode())

clientSocket.close()
```

```
from socket import *
serverPort = 12000

serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(("", serverPort))
serverSocket.listen(1)

while True:
   connectionSocket, clientAddress = serverSocket.accept()
   modifiedMessage = connectionSocket.recv(2048).decode().upper()
   connectionSocket.send(modifiedMessage.encode())
   connectionSocket.close()
```