$= 0$ $\qquad$ $i > m$

$$\sum_{i=0}^{m} p_i = 1$$

Erlang B loss formula

$$p_0 = \left[\sum_{k=0}^{m} \left(\frac{\lambda}{\mu}\right)^k \cdot \frac{1}{k!} \right]^{-1}$$

$P[\text{blocking}] = p_m$ = Probability of being in state m

what should m be?

We set a acceptable value for $P[\text{blocking}]$

We also know λ calls request/sec

We also know $1/\mu$ call holding time

$$p_m = \frac{(\lambda/\mu)^m \cdot \frac{1}{m!}}{\sum_{k=0}^{m} \left(\frac{\lambda}{\mu}\right)^k \cdot \frac{1}{k!}}$$

Lecture 4 4/8/2021

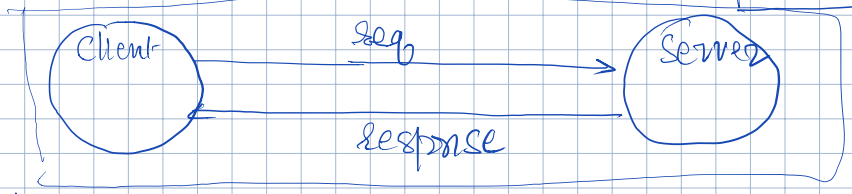
1. Go over the code
2. Move on to Application layer protocols

Announcements

1. Assignment
2. Write also what are the weakness of the paper (critical)
3. Set one more office on Monday

Application Layer Protocols

→ distributed application



Process that initiates the request
Client Server paradigm

Process that is always on and waiting for request from a client

Application Layer Protocol

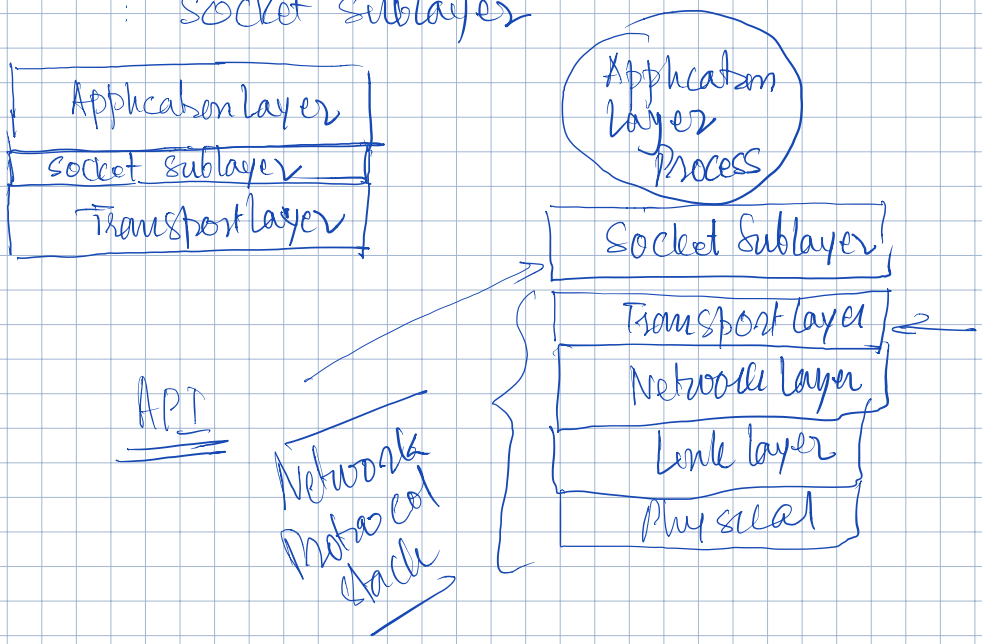
⇒ protocol which is used by the client and the server to implement some distributed application

⇒ messages are exchanged between the client and server

API : Application Programming Interface

[2] Application Layer Protocols
→ HTTP
→ Telnet

: socket sublayer

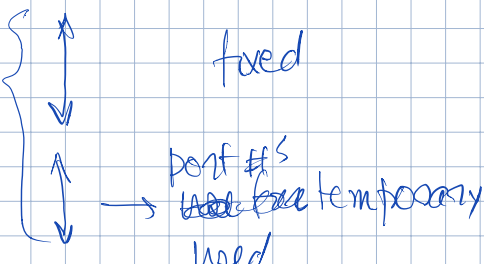


TCP based application (HTTP, FTP, SMTP, Telnet)

- Each application at the server side has a port number (well know, fixed identifier)
  e.g., HTTP server operates on port # 80

- Application at the client side has a ephimeral port number
  ↳ fixed for the session but can be reused once released

Port number are 16 bits

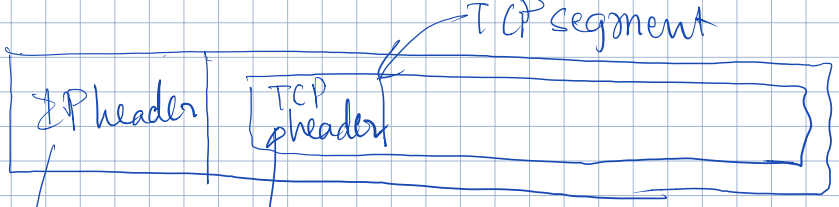


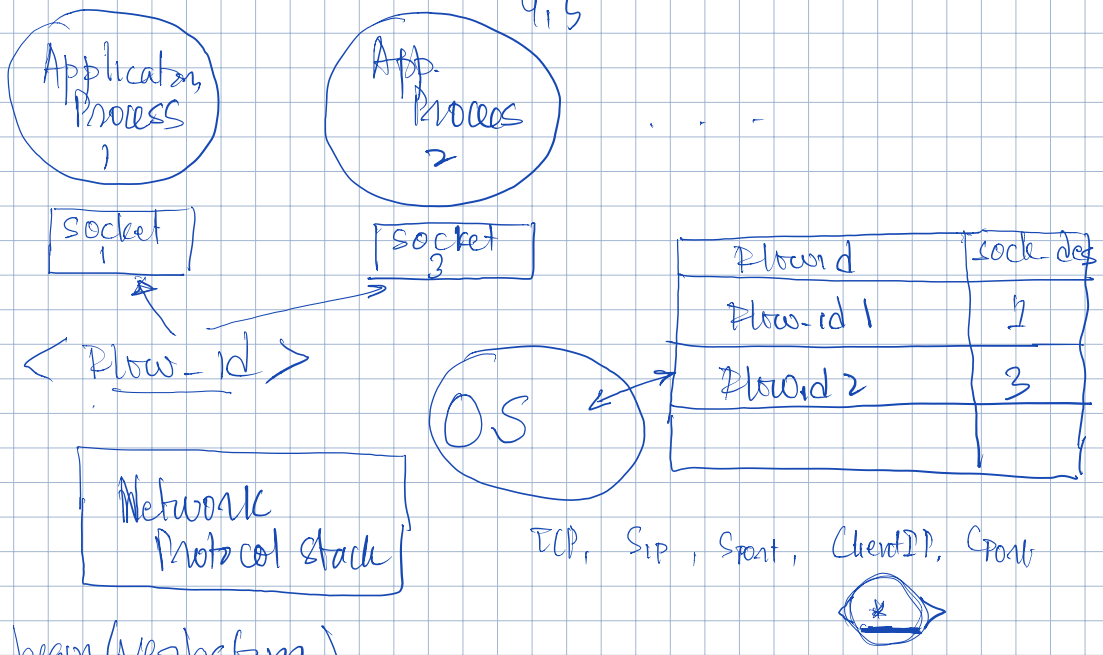
Both client and server have network addresses (IP addresses)

Flow-id : 5-tuple 1,2,3 → IP header; 4,5 → TCP header

<tcp, S_{IP}, $S_{port\#}$, C_{IP}, $C_{port\#}$>
(1, 2, 4, 3, 5)

all pkts exchange between the client & the server for a specific instance of the application can be identified by a unique flow-id

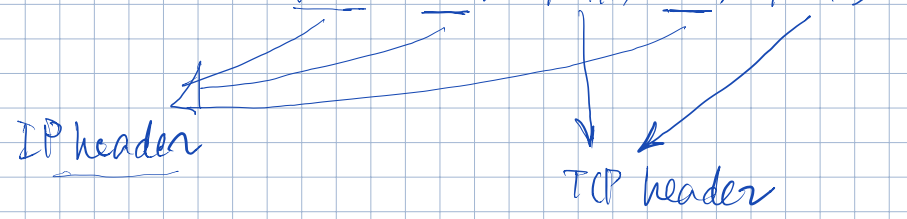




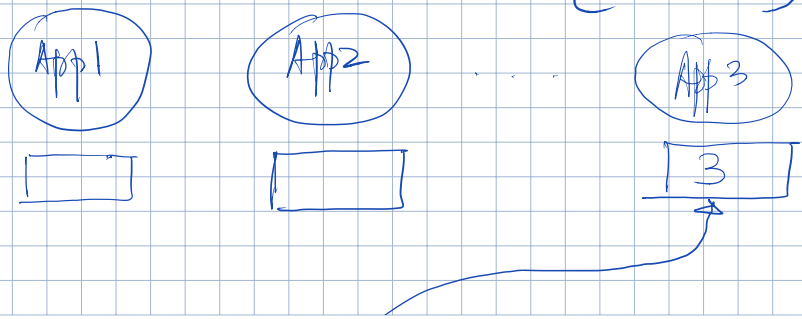
\begin{verbatim}
\end{verbatim}

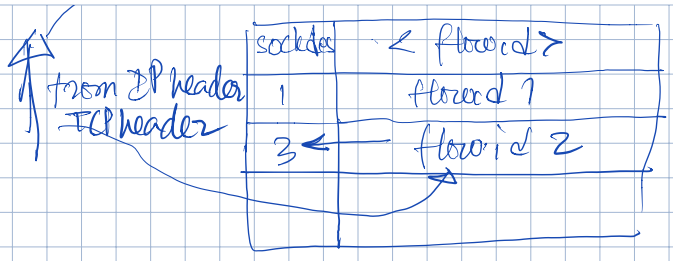
Lecture 5 4/13/2021

Flow-id : <proto, CIP, Cport#, SIP, Sport#>



- Each client server session will have a unique flow-id
- socket is a door to/from the application
- Each end system can be running multiple applications, infact, multiple instances of the same application
- Kernel/OS maintains a table of map between flow-id and socket descriptor
  - identifier of the socket (or door)

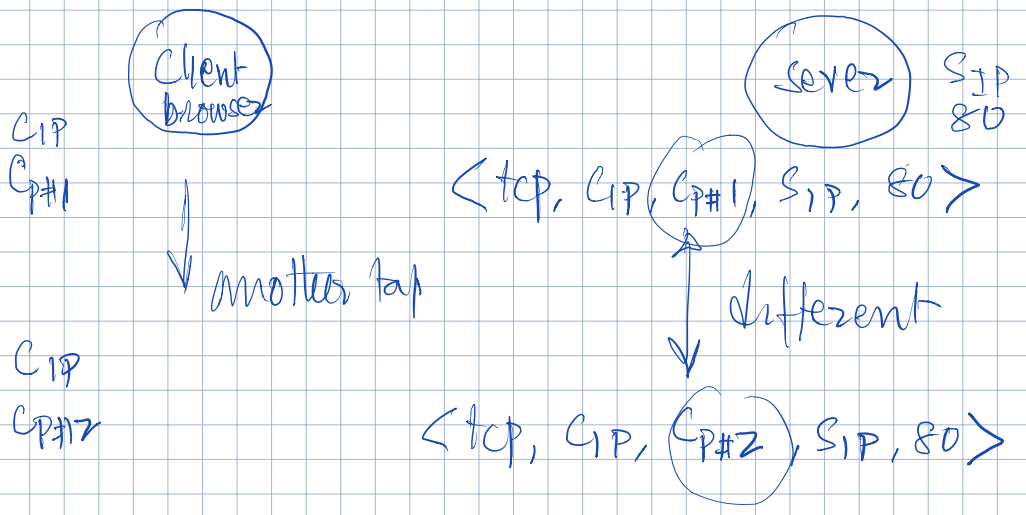




- Port# is a address of the application
  Server side port# is well known
  client side port# is dynamic

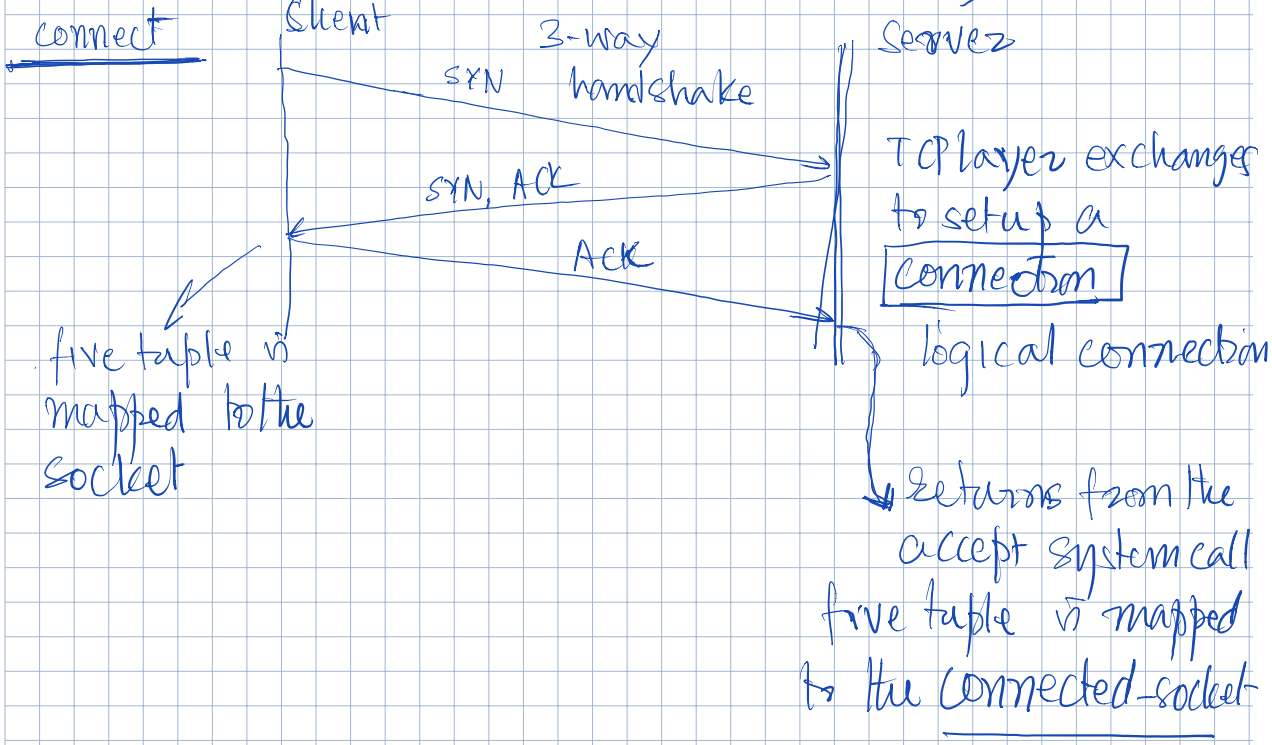
  HTTP server has port# 80
  HTTPS server has port# 8080

- A flow-id is unique even if one of the tuple values is different



- Jupyter notebook code

- connectedsocket.connect(S_IP, S_P#)



- Concurrent server

The server forks a child process to handle the request

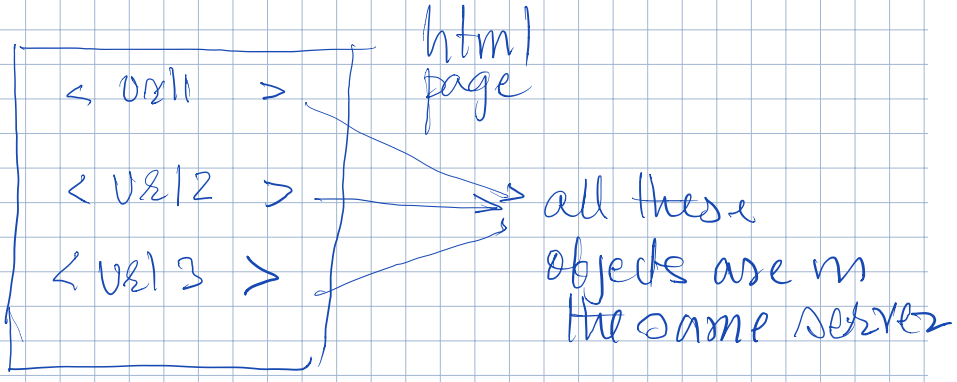
Child inherits all the socket descriptors

```
while(1)
    connectedSocket = listenSocket.accept()
    fork a child process
    Close(connected socket)

Child process
{   Close(listenSocket)
    Execute the code of the application
```

close (connected socket) ⟵

- Code that is in the Jupyter notebook is an iterative server
  → only 1 request will be service at a time and it is serviced by the parent process

- Please read about HTTP
- Persistent HTTP
  Keep the TCP connection open to service multiple Get request from the client to the same server
- 

- It is very wasteful in terms of time to setup different connections for each object
  - 3-way handshake takes time at least 2 RTT
- Set up a connection to the server
  - Send Get Request
  - Keep it alive →
  - Send Get Request
  - Send Get Request

  (pipelining req.)
- Not paying the cost of setting up TCP connection for every Get req.

Optimizing Performance

- Tuning and optimizing HTTP
- Network and the server can have bottlenecks that

impact the performance of distributed applications

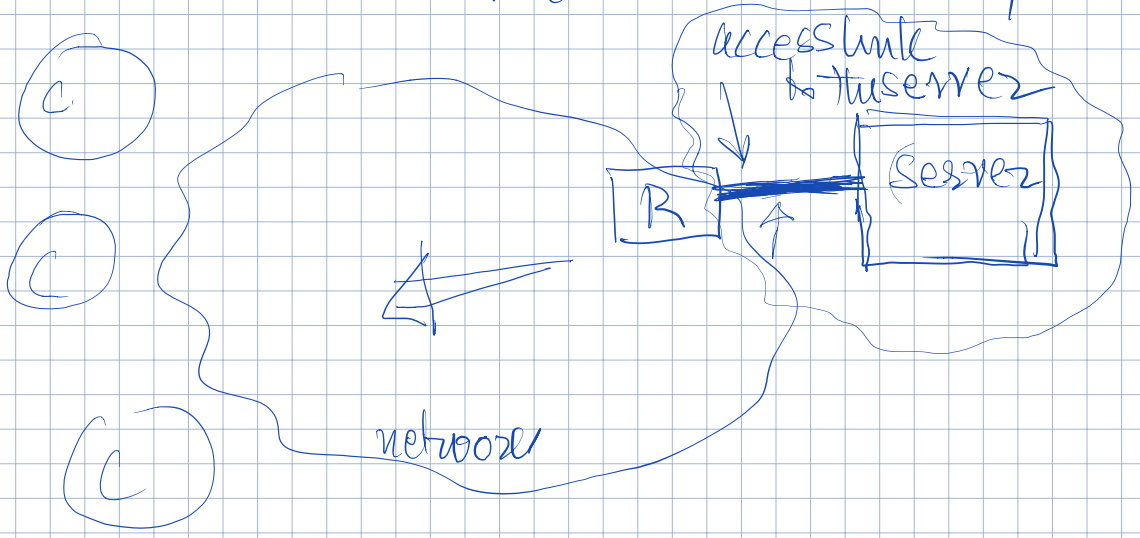
Side
Server Bottlenecks { server access link to the server

⇒ When request arrive faster than what the server can handle

⇒ Server farms to have more server capacity

⇒ { Peer-to-peer ⇒ technology related
⇒ { Web caching

⇒ If the access link is the bottleneck increase its capacity



Lecture 6

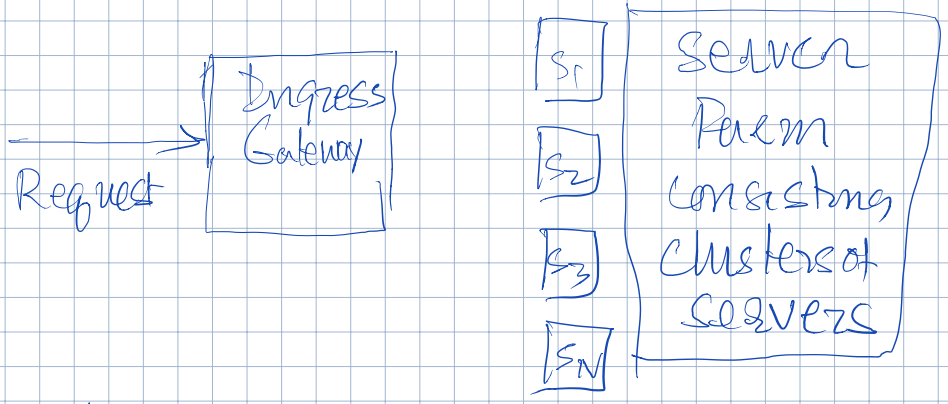
Announcements
1. Assignments in PDF in Canvas
2. Project today

Server Farms

Cloud Computing

Cloud Service Provider

Kubernetes : Cloud OS

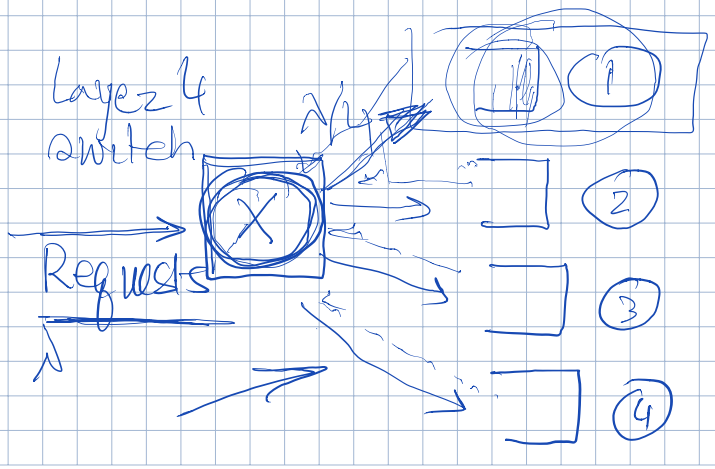


Load Balancing

3 different implementations

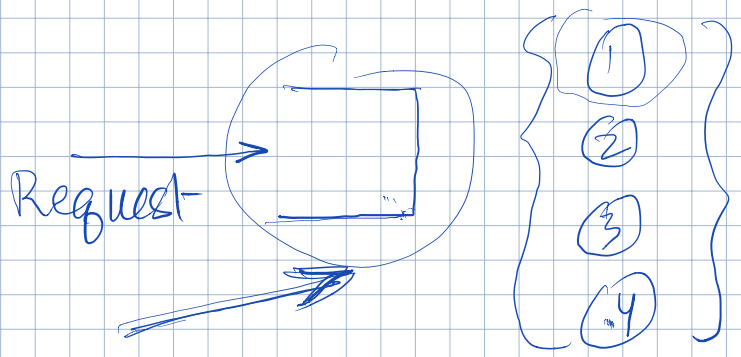
Implementation?

4 servers

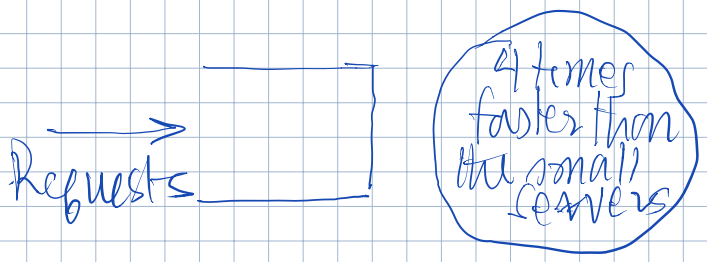


requests are either round robin or uniformly route the request to the servers

Implementation 2



Implementation 3



No load balancing is required

This is not scalable

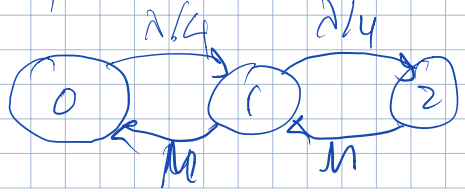
Compare the average latency: time to complete a request

function of λ: request rate

μ: service rate of the small processors

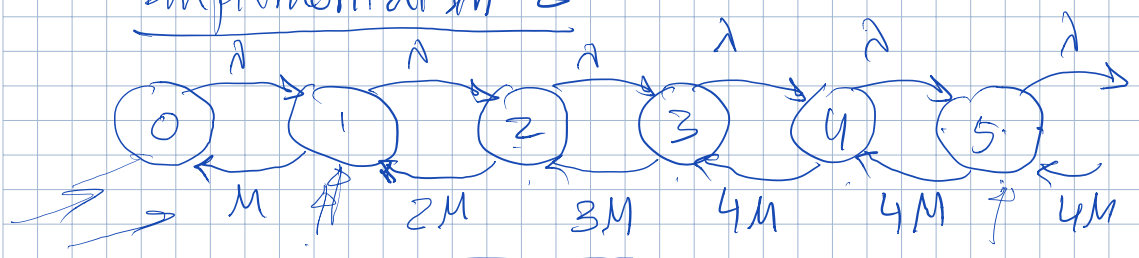
Buffer size is infinite

Implementation 1?



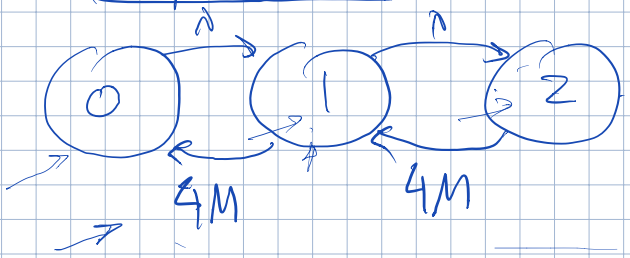
E(D) ✓

Implementation 2



λ E(D) = E(N) ✓

Implementation 3

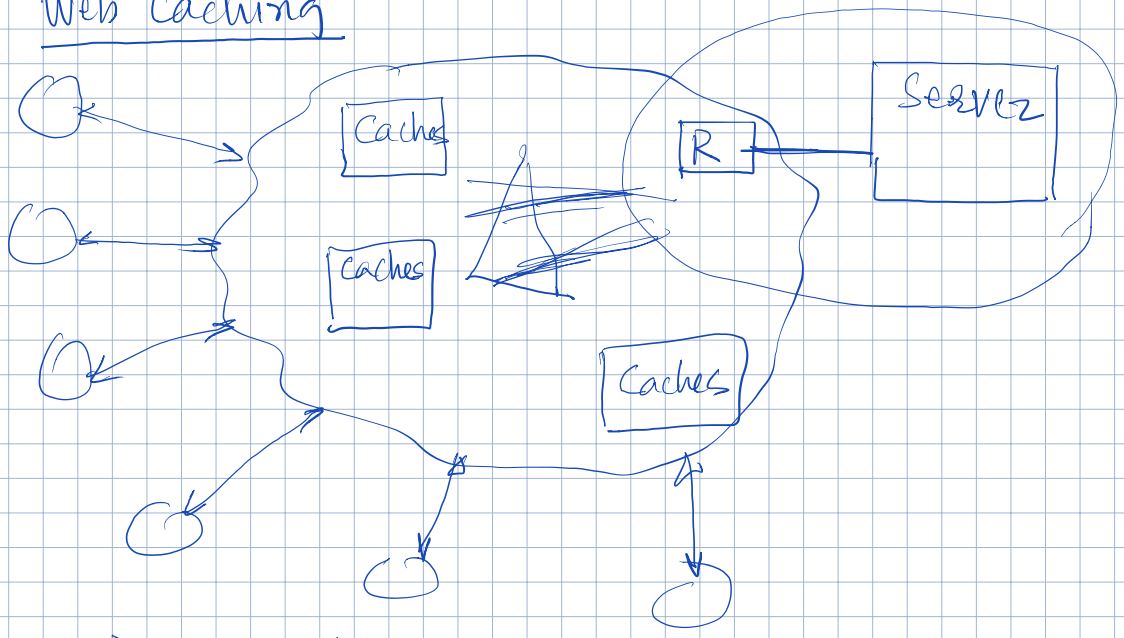


✓

Summary

- Implementation 3 is best but not scalable
- Implementation 2 is the perfect load balancing scheme
- In the cloud, Implementation 1 is used with different heuristics

Web Caching



- Populate the cache when client make requests
- Push the content to the cache
- Akamai CDN Content distribution network
  Interconnected distributed caching System

Processor - Memory

Caches leverage/exploit locality of reference

cache hit rate upwards of 90%

Web Caching

Cache leverage content popularity

popular contents

→ Cache hit rate

→ Lower latency of access

Reducing server capacity requirement

Zipf distribution

Special case in which $\alpha \geq 1$ and the sample space is finite. (Let us suppose that there N objects)

P(Request is for the i^{th} popular content)

$$= \frac{C}{i} \leftarrow \sum_{i=1}^{\infty} \frac{C}{i} = 1 \quad C.$$

$$\sum_{i=1}^{N} P(\text{Request is for } i^{th} \text{ popular content}) = 1$$

$$1/C = \sum_{i=1}^{N} \frac{1}{i} \sim \ln(N) \text{ for large } N$$

$$\boxed{P(i) = \frac{1}{i} \cdot \frac{1}{\ln(N)}} \checkmark$$

Lecture 7

Announcements

1. Midterm Exam
   May 12th Wednesday
   - 5pm — 10pm
   - Exam will < 2hrs
   - Exam uploaded canvas
   - Record yourself
2. Assignment 2 will uploaded
3. Project 1 after lecture

Study the Ethernet protocol Finals week

Recap

Methods to mitigate server-side bottlenecks

→ Server farms used in datacenters

→ Web caching

→ have the content available to the clients in caches that are closer to the clients

→ Cache in processor memory system

→ locality of reference

→ high hit rate with a relatively small cache

→ popularity of content

access to content follows a Zipf distribution ⟸ power law distribution

"90% of the access is for 10% of the content"

90-10 rule

- distribution of wealth

top 85 richest people ⇒ 50% of wealth

- Social n/w connection
- connection between pyramidal neurons in the brain has power law distribution

Suppose N objects — Objects are ranked ordered in popularity

$$P(i) = \frac{1}{\ln N} \cdot \frac{1}{i}$$

request is for the i^{th} popular content

Probability mass function

$$P(X \le i) = \sum_{j=1}^{i} \frac{1}{\ln N} \cdot \frac{1}{j}$$

CDF $= \frac{\ln(i)}{\ln(N)}$ for large i

Consider a simple scenario

Cache is of size 64 GB

Objects are of size 10 KB (average)

Number of objects that the cache can hold $= 6.4 \times 10^6$ $\frac{64\,Gb}{10\,KB}$

Suppose we consider 10×10^6 object in the Internet

Suppose we use Uniform access model each object is equally likely to be accessed

η_U^{10M} = hit rate of the cache

$= \frac{6.4 \times 10^6}{10 \times 10^6} = 0.64$

$\eta_U^{100M} = \frac{6.4 \times 10^6}{100 \times 10^6} = 0.064$

$\eta_Z^{10M} = P(X \leq 6.4 \times 10^6)$

$= \frac{\ln(6.4 \times 10^6)}{\ln(10 \times 10^6)} = 0.97$

$\eta_Z^{100M} = P(X \leq 6.4 \times 10^6)$

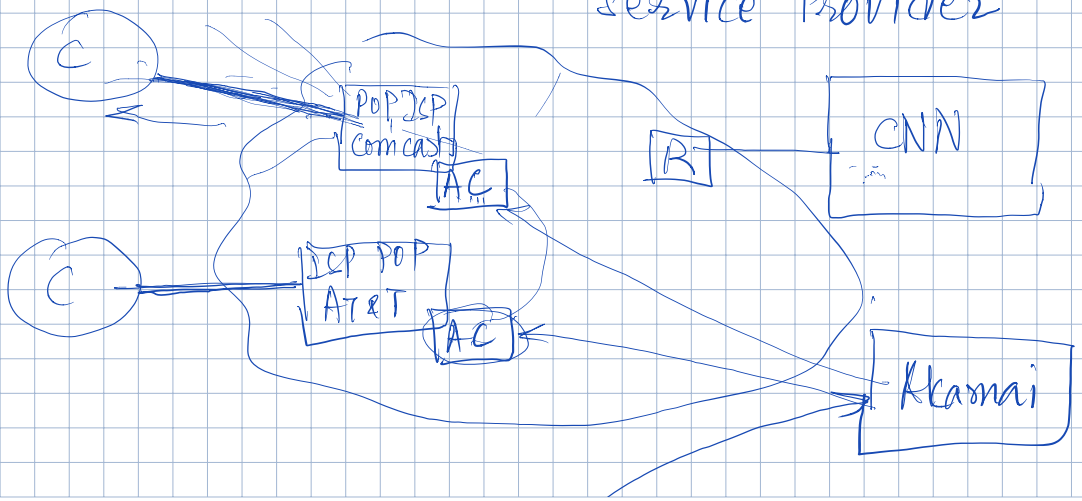
$$= \frac{\ln(6.4\times10^6)}{\ln(100\times10^6)} = 0.8507$$

Increasing the N by a factor of 10 reduces the hit rate much much less than in Uniform access model

Assumption

- The cache contains the top 6.4×10^6 most popular content
- Very intelligent cache replacement policy
  - → LRU, LFU
  - → ML based approach for the cache replacement

Akamai : CDSP Content Distribution Service Provider



CNN page ⇒ { Content that is changing
Content that relatively static { CNN logos

65% of the pages

Resolve : Akamai servers names ← DNS names are resolve

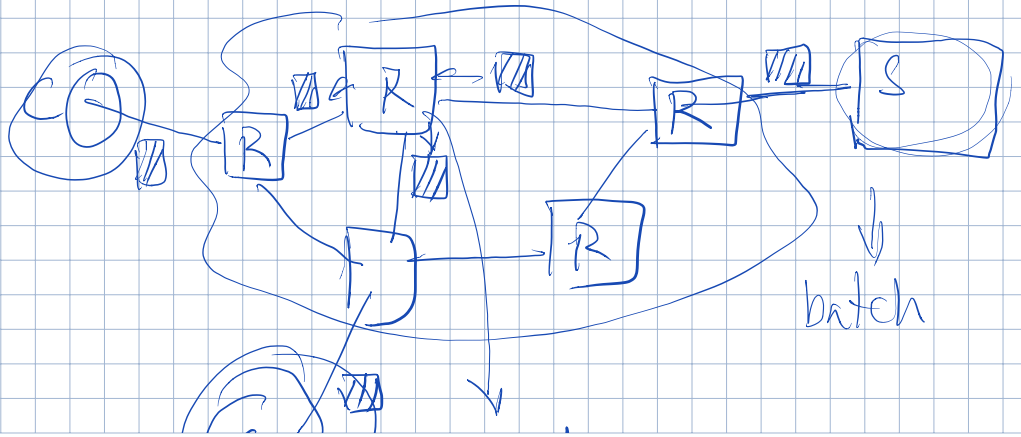
↓ logical name

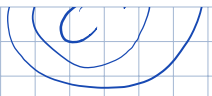
↓ IP address

- Mitigate server side bottleneck

⇒ multicasting

server sends one copy of the plot and intermediate routers make copies of it





multi cast enabled router

- Works for near simultaneous requests
  Synchronous-method
- Real-time streaming

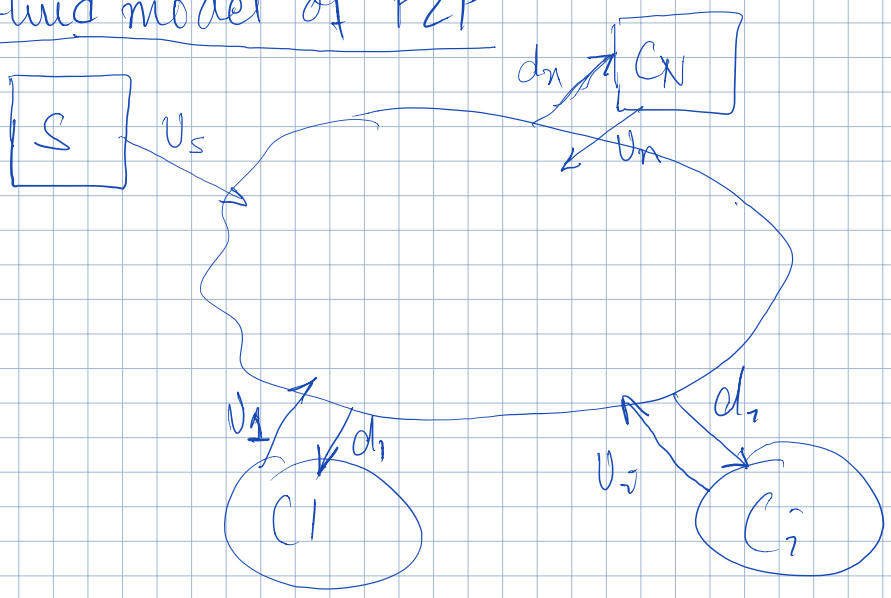
- Peer-to-peer Networking

  ⇒ extreme case of Web caching & multicasting

  ⇒ Using the client resources

  ⇒ clients are becoming servers once they have the content

Fluid model of P2P



A file of size F must be received by all the clients

$$D_{c-s} \geq \max\left(\frac{NF}{u_s}, \frac{F}{d_{min}}\right)$$

$$d_{min} = \min(d_1, d_2, \ldots, d_N)$$

client – server architecture

$$D_{P2P} \geq \max\left(\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right)$$

$d_{min} = \min(d_1, d_2 \ldots d_N)$

Server upload rate is the bottle neck

Client download rate is the bottle neck

$$\frac{NF}{u_s + \sum_{i=1}^{N} u_i} \text{ bits}$$

$$\frac{NF}{u_s + \sum_{i=1}^{N} u_i}$$

As N grows so does the total capacity to serve out the requests

⇒ self-scaling

⇒ Unique property of P2P networking

⇒ Free-riding: clients get the file but they do not serve

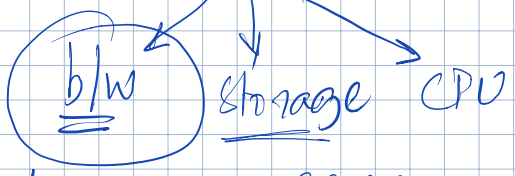
late 90s and early 2000s

many P2P ideas

→ Gnutella

→ BT - Torrent

- very clever algorithms to incentivize client to participate & contribute resources



- based on some game theoretic ideas

Keith Kong

Pseudo-serving ← Pseudo-serving ⇓ AEN Teams Netw

→ Pseudo-serving

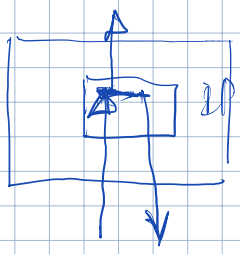
→ contract with the server

→ provide fast access to the file if the client agrees to serve the content to few other clients over a small interval of time (N, T)

→ CoopNet : Test-of time paper
↳ Video traffic

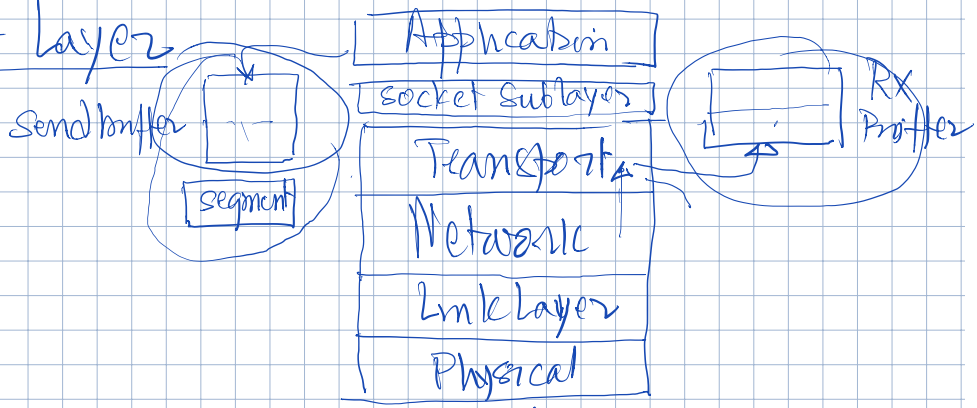
⇒ P2P as an extreme case of Web caching & Multicasting

⇒ Not putting caches in the n/w
Using client themselves as caches

⇒ Client  Client are operating as multicast routers

---

Transport Layer



- What are the functions of the transport layer?