Lecture 4 4/8/2021

3. Reading assignment for Chapter 2

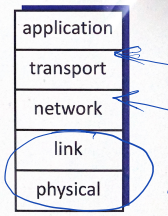
4. Feedback & comments

Announcements

1. Correction in the simulation code
2. Regarding the deep space problem

Internet protocol stack

- *application:* supporting network applications
  - IMAP, 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, 802.11 (WiFi), PPP
- *physical:* bits "on the wire"

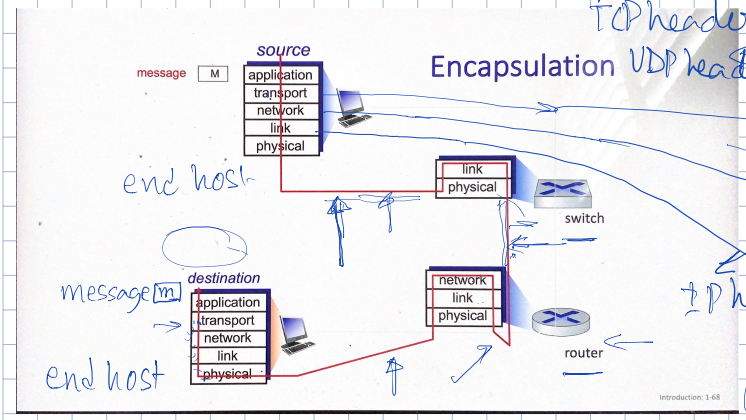


Link/Phy

4 layer protocol stack

Object oriented programming

} interface between the layers



TCP header

UDP header

Application layer exchanging messages

H_T m_p segment

H_N H_T m_p IP packet datagram

IP header

H_L H_N H_T m_p Frame

Link layer header

Frame is transmitted into the link

Switch is a link layer device

→ decapsulation only upto the link layer

Routers is a network layer device

⇒ Routers & Switches only operate upto the n/w layer

⇒ Transport layer & application layer only runs on the end hosts

⇒ end-to-end principle of modern Internet ⇒ dumb

⇒ Very fast / best effort network

Highly adaptable & intelligent end systems

⇒ Very different than the design of the original telephone n/w

end-devices were very simple

core network was very intelligent

⇒ Intelligent n/w (circuit switching)

⇒ Signalling n/w which packet based

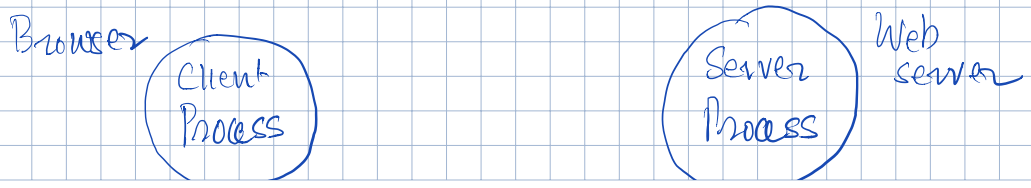
⇒ How the call was set up

⇒ How the circuit was routed

⇒ In case of overloads

SS7

Client-Server Paradigm



- Initiates and send the request (message)
- Active entity

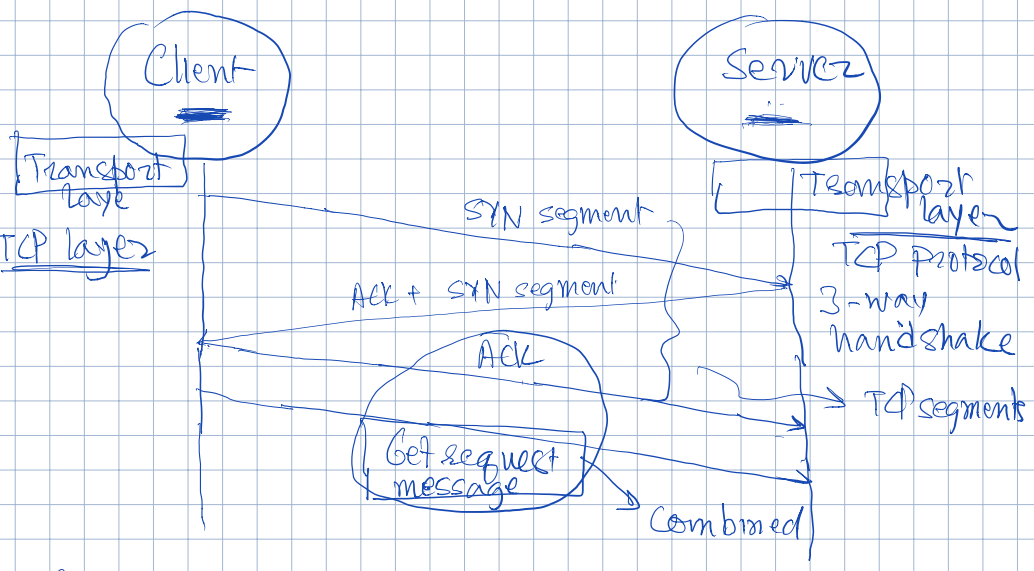
- Server is always on
- Server is waiting for requests from clients
- Passive entity

Most applications are implement over TCP

Client can only send a message (request) once it has established a connection with the server

Connection is called TCP connection

Established using a 3-way handshake

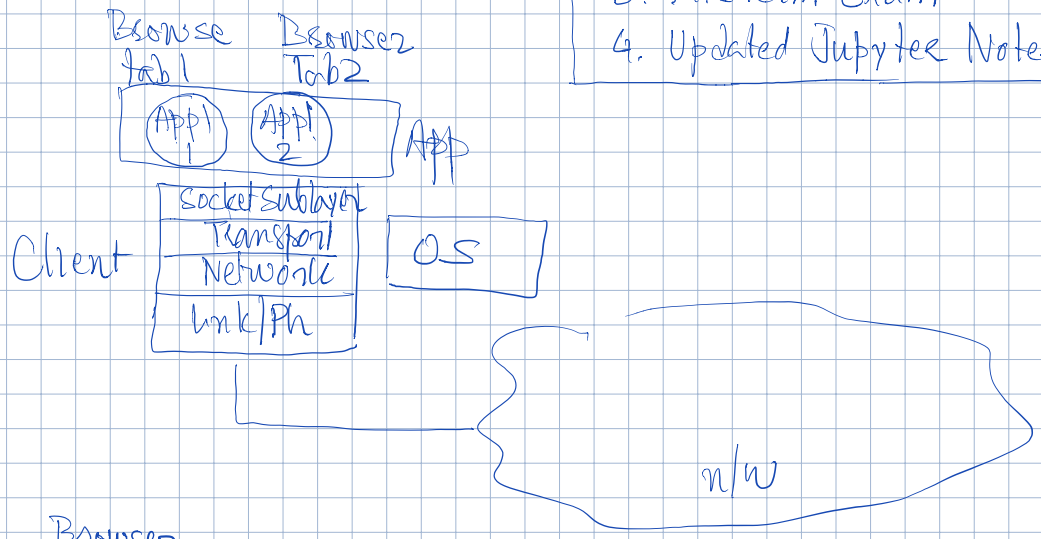


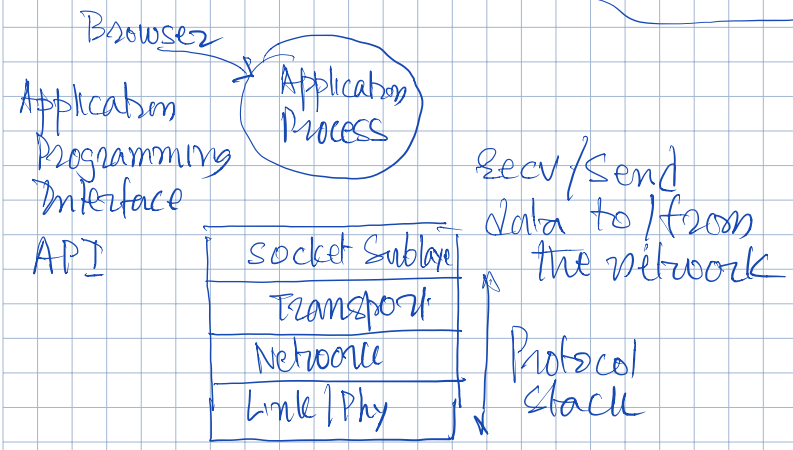
- Establishing a logical connection
- Both the client and the server needs to allocate resources (memory) to process the request
- Establish a unique identifier for all data exchanged between the client & server

Lecture 5 : 4/13/2021

Announcements

1. Assignment 1 with rubric
2. Assignment 2
3. Midterm Exam
4. Updated Jupyter Notebook





- Socket is like a door between the protocol stack and the application
- For any data transfer over the network

Flow-id : < 5-tuple >

< protocol, C_{IP}, $C_{port\#}$, S_{IP}, $S_{port\#}$ >

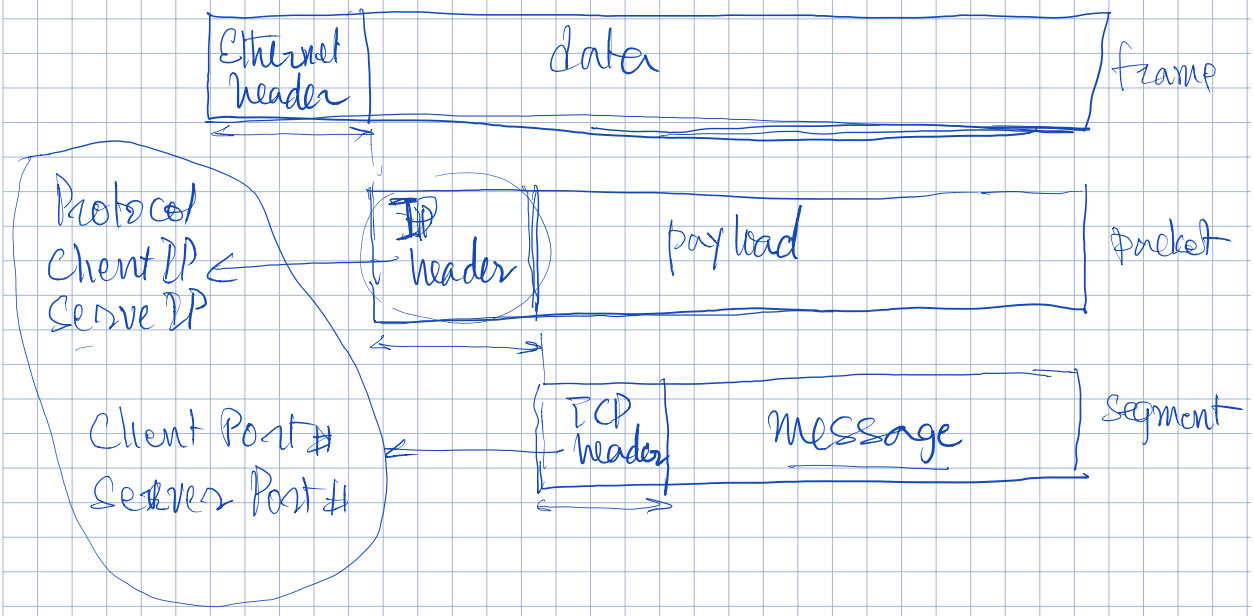
protocol = {TCP, UDP}

are in the IP header (network layer header)

are in the TCP header Transport layer header

- Sniff a (packet) in the wire

frame

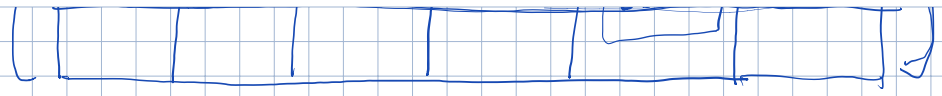


- Each socket has a socket descriptor no different than file descriptor

- Kernel/OS maintains a mapping of the flow-id to the socket descriptor

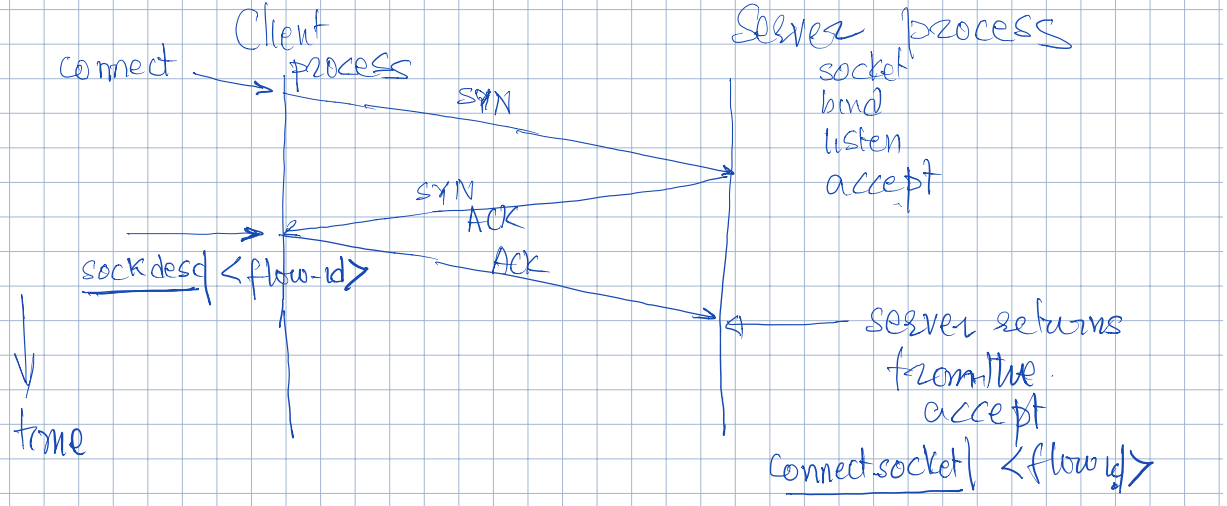
Servers

socket descriptor	Proto	Client IP	Client Port	Server IP	Server Port
1	tcp	128.120.50.10	12000	192.132.65.11	80
2	tcp	128.121.56.11	12000	192.132.65.11	80



- An instance of this table is in each end system (server and client)
- Flow-id
  1) Protocol can either be TCP or UDP
  2) IP addresses: network layer addresses
  3) Port numbers are address of applications
     - Port #'s of applications in the server are well known & fixed
     - Port #'s of application in the client are dynamic/ephemeral

Port # of http server is 80
Port # of secure HTTP server is 8080



- Current version of the server
  - ↳ code that I have given you

⇒ Iterative server

Only 1 request can be service at a time

⇒ Concurrent server

Each request is serviced by a different child server process

⇒ → connectionsocket = listensocket.accept()

→ fork a child process

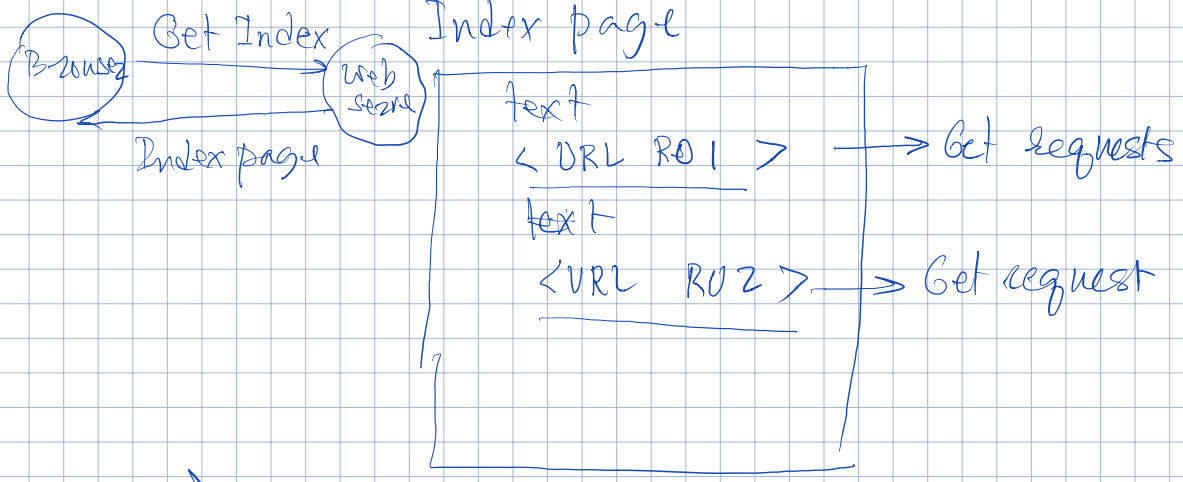
connectionsocket.close()

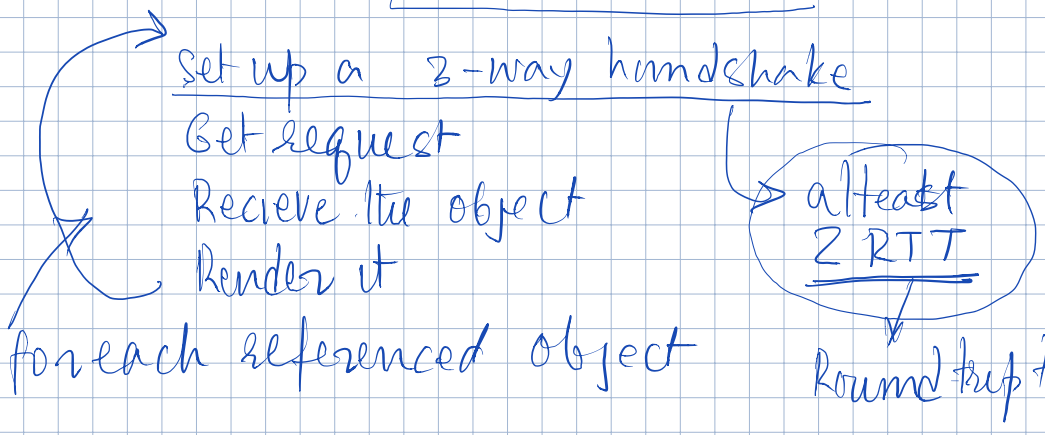
listensocket.close()
do the application on the connectionsocket
connectionsocket.close()

Lecture 6 (4/15/2021)

- HTTP protocol
- Persistent HTTP

Content NY times page

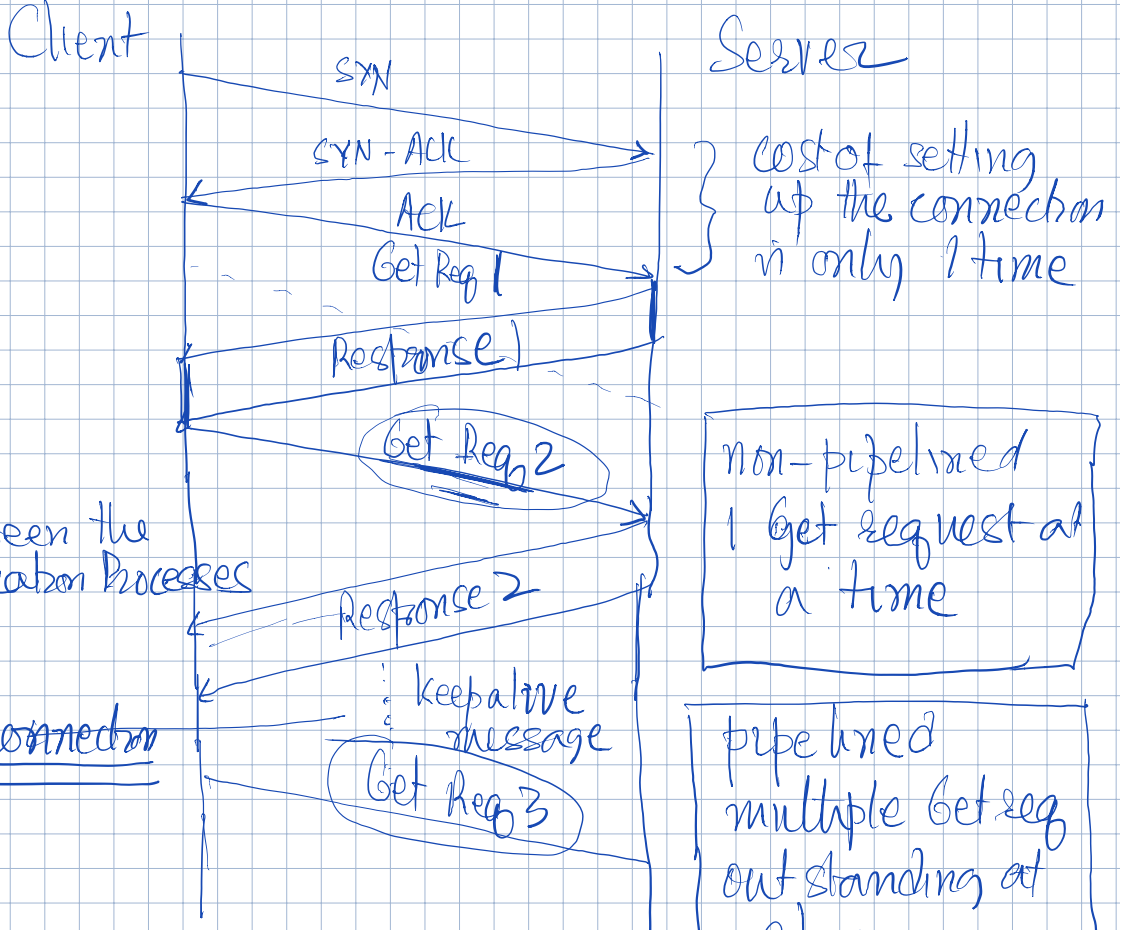




- If the URLs reference different servers then you have to pay the cost of setting up the connection
- If the URLs are in the same server then keep the TCP connection alive

and send multiple request on the same connection

→ Persistent HTTP



→ Between the application Processes

TCP connection

non-pipelined
1 Get request at a time

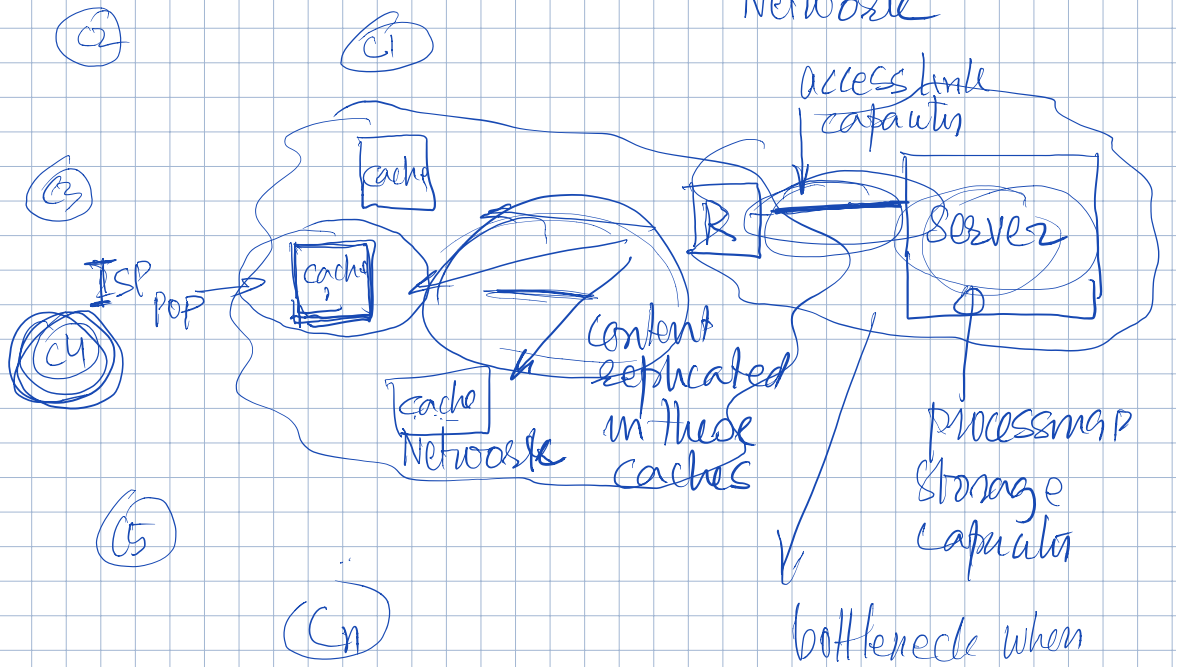
pipelined
multiple Get req out standing at a time

HTTP protocol

→ cookies

→ Browser caching & Condition Get Request

Web Caching : Akamai Content Distribution Network



Cache : proxy server

bottleneck when many clients request content simultaneously

Server-side bottleneck

Web Caches

→ Reduces server side bottlenecks

→ Reduce down load time

→ Reduce core network bottlenecks

We should keep the popular content in the cache

⇒ have a high hit rate

Cache in the Processor-memory system

→ Instruction cache

→ Data cache

cache → a small but very fast access time memory

→ expensive

Locality of reference

→ temporal locality

→ Spatial locality

- high hit rate even with a small cache size upwards of 90%

Web Caches:

- By keeping the popular content in the cache, we get a high hit rate

Zipf* distribution

N objects

Ranked orders of these objects 1, ..., N

Object with rank order i is most popular

Relative probability of a request for the ith most popular object is proportional to $1/i$

$$P(X=i) = P(i) = C/i$$

$$\sum_{i=1}^{N} P(i) = 1 \Rightarrow \sum_{i=1}^{N} C/i = 1$$

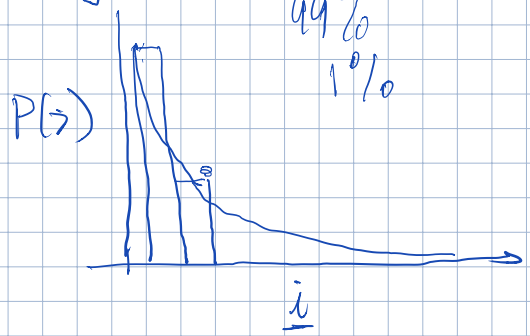
$$\Rightarrow C \sum_{i=1}^{N} 1/i = 1 \Rightarrow C = \frac{1}{\sum_{i=1}^{N} 1/i}$$

$\sum_{i=1}^{N} 1/i$ → Harmonic series

$$\Rightarrow C \cong \frac{1}{\ln N}$$ when N is large

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

$i^{-\alpha} \quad \alpha > 1$



probability mass function

$P(X \leq i)$ = Probability distribution function

CDF

$$= \sum_{j=1}^{i} P(j)$$

$$= \sum_{j=1}^{i} \frac{1}{j} \cdot \frac{1}{\ln N}$$

$$= \frac{1}{\ln N} \sum_{j=1}^{i} \frac{1}{j} \approx \frac{\ln(i)}{\ln(N)}$$

Consider a cache of size 64 GB

Consider the Internet contains 10×10^6 objects

Object size is 10 KB (average)

Cache can hold $\frac{64\ GB}{10\ KB} \approx 6.4 \times 10^6$ objects

Case 1

- Consider a uniform access model

  $\Rightarrow$ each object in the Internet is equally likely to be accessed

- What is the hit rate in the cache?

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

- If we increase the number of object by a factor of 10

$$\eta_u = \frac{6.4 \times 10^6}{100 \times 10^6} = \underline{0.064}$$

- Consider the request to follow the Zipf* distribution

$c \cdot i^{-\alpha}$, $\alpha > 1$

$$\eta_z = P(X \leq 6.4 \times 10^6)$$

$$= \frac{\ln(6.4 \times 10^6)}{\ln(10 \times 10^6)} \quad 10\text{M objects}$$

$$= \underline{0.9723}$$

$$\eta_z = \frac{\ln(6.4 \times 10^6)}{\ln(100 \times 10^6)} \quad \text{@ } \underline{100\text{M objects}}$$

$$= \underline{0.8507}$$

logarithmic function is really compressing function (grows slowly)

- Super big assumption
  the cache contains the top
  6.4×10^6 popular objects
- Very intelligent cache replacement policy

Lecture #7 4/20/2021

Announcements

- close book / close notes (A)
- Zoom recording

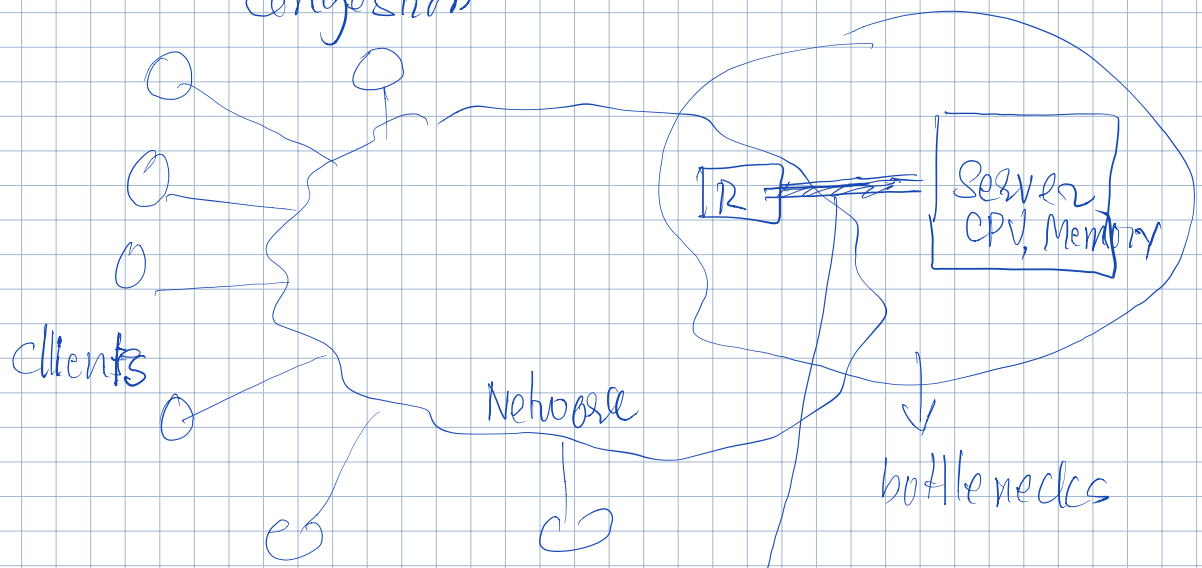
- Midterm Exam
  May 11th (Tuesday)
- Exam duration < 2hrs
- Time window ~ 3pm - 10pm

(B) Assignment 3
to be released later today
or tomorrow

(C) Homework self-grading: details added in the document

Recap

Methods to mitigate server-side congestion



(A) access link is bottleneck
add more capacity
10 Mbps → 100 Mbps

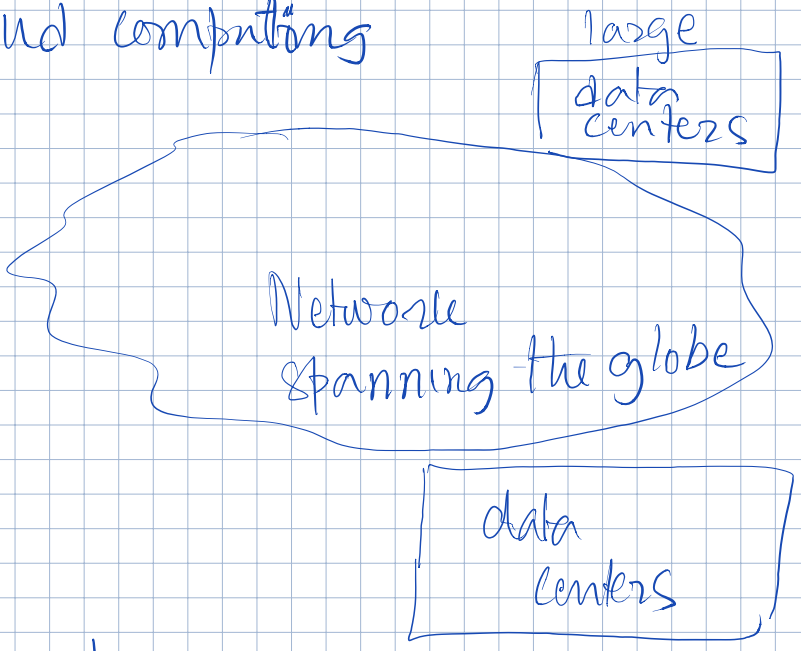
10 Mbps → 100 Mbps

② Server capacity is the bottleneck

⇒ add servers

Rent servers from AWS
Azure
Google Cloud

Cloud computing



Kubernetes

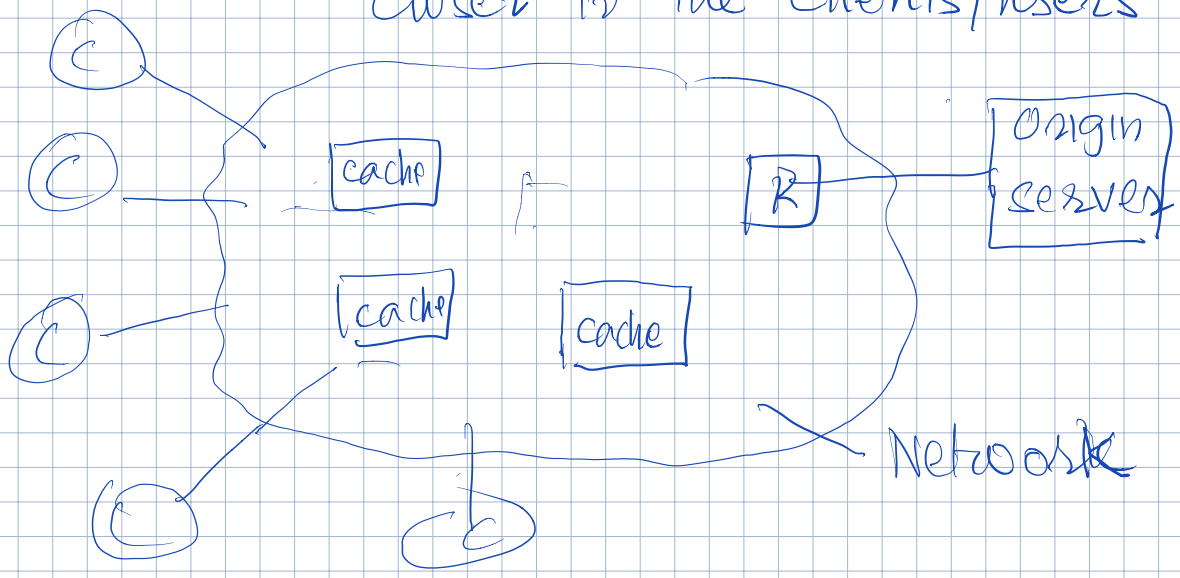
↳ Cloud OS

↳ Scaling functions
add capacity (server)
when the demand increases

Web Caching

⇒ deploy proxy server / web caches

closer to the clients/users



If request are served by the caches then

→ No more server side bottleneck
less capacity required at the server

→ lower download time

→ lower demand on the capacity of the core network

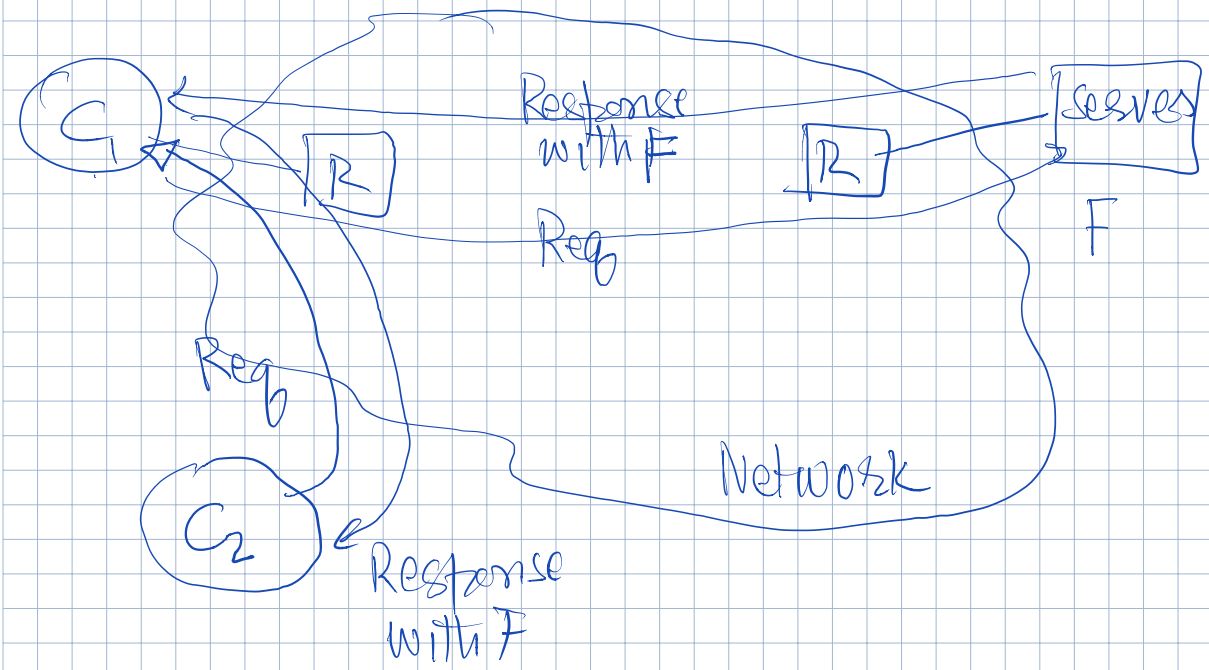
Request for content follows a Zipf like distribution

⇒ high cache hit rate

Peer-to-Peer Networking

- P2P Networking
- P2P Computing

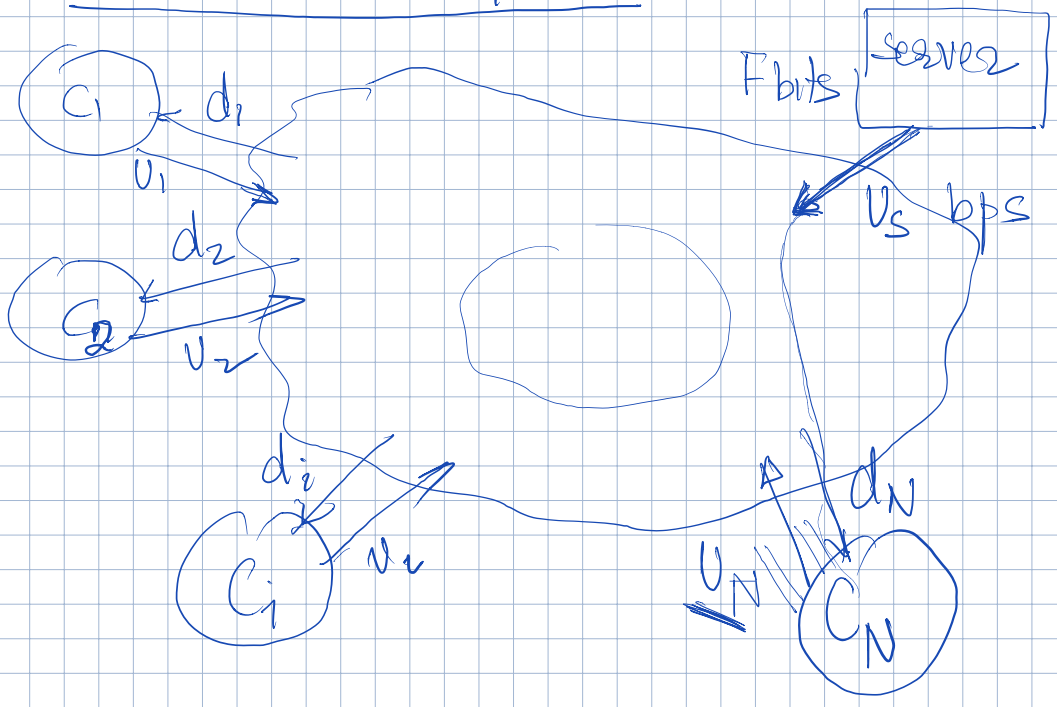
- Client side resources are used to serve out content



- Taking the web caching idea to the clients

- Fluid model

Client-Server System



All clients must receive the file of size F bits

$$D_{C\text{-}S} \geq \max\left(\frac{NF}{u_s}, \frac{F}{d_{min}}\right)$$

↓ time for all the clients to receive the file

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

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