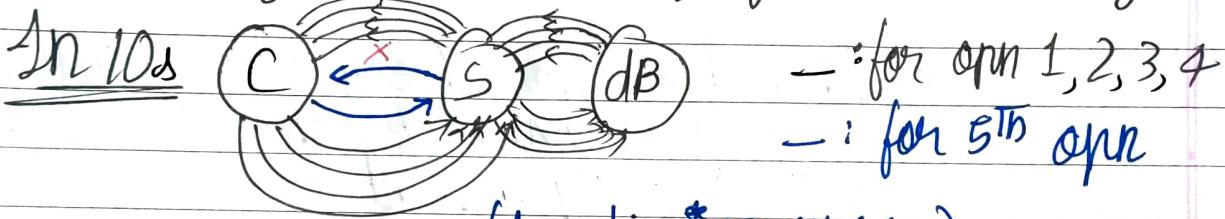


RATE LIMITING

Rate limiting :

- ① Setting a threshold of sorts on certain opns. past which these opns return errs. limiting the amt. of opns. that can be performed in given time

e.g.: - Only 5 HTTP req. from client every 10 secs.



→ (1 machine * many reqs) = many reqs

- ② Denial of Service (DoS) attack ⇒ Malicious client floods the sys with too many req and effectively clogs the sys with way more traffic than it can handle and thus, sys is brought down

- ③ Rate limiting prevents clogging of servers by DoS attacks. past a certain no. of reqs, it'll throw errs. and stop taking in reqs.

e.g.: - "code exec" engine on AlgoExp. Run code opn" is rate lim. to prevent someone from spamming the opn" i.e. clogging the DB with reqs.

- ④ Rate limiting can also be dependent on other factors:

- i) Based on user: Identify user issuing req.

by lookin at req. headers or authentica's credential
 and ~~user~~ once identified we can rate limit
 no of opns. by user eg:- 3 opns/min for
 a student to access his/her grades on insti. website

- i) Based on IP address
- ii) Based on region of the world
- iii) Based on whole sys. eg:- a sys as a whole
 should never allow servers to handle $> 10^4$
 reqs per min.
- v) Based on a certain org. fallin under same
 IP address

⑤

DDoS attack \rightarrow Distributed DoS attack

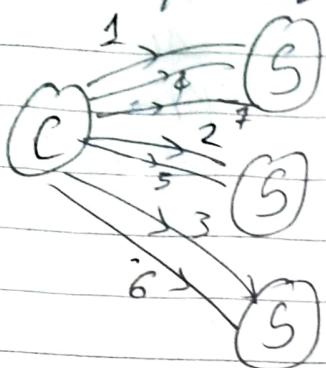
\hookrightarrow (many machines * some reqs by each machine) = (many reqs to servers)

might not be identifiable as part of same org/grp.

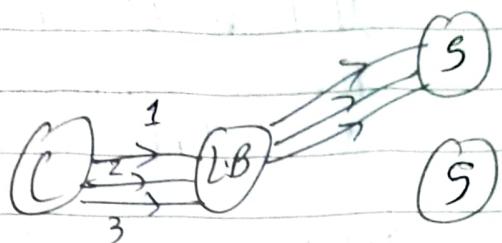
⑥

In large - scale distri sys \rightarrow we. gotta hr. v.
 powerful rate limiting ~~to prevent~~ \rightarrow i.e. we
 need to hr. really powerful load balancin
 to ensure that a client's req are always routed
 to the same server (amongst a set of servers
 cuz its a big sys) to ensure that we have ~~to~~
 some prev access records of this clients reqs. to
 the sys, stored in-memory in the ~~same~~ ^{some servers} sys
 we can compare and say whether amt. of
 reqs. this client is issuing is within a safe
 boundary or not

w/o LB in



with LB



Client can flood all servers as rate lim.
is met / server but not across servers (sys as a whole) so sys is flooded with traffic

Client always routed to same server → it has nr/r access records of this client so it'll rate lim. client if it's flooding with too many reqs.

∴ It is imp. to config load balancing keeping rate lim. in mind

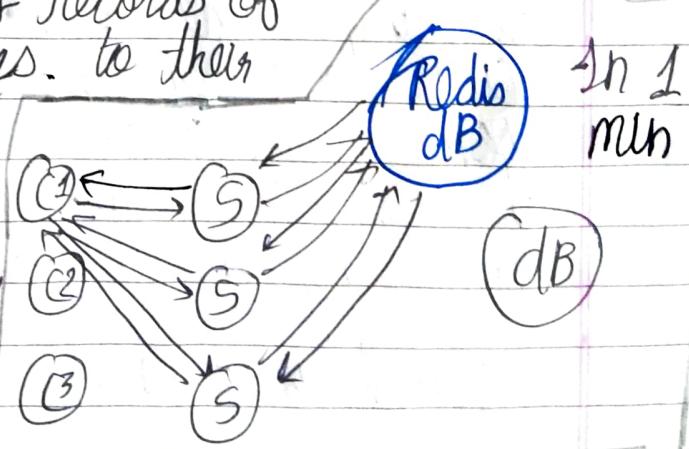
7 ∵ in most cases, large scale distri. sys's use handle rate lim. in a separated service or dB instead of in-mem. in servers. All ~~these~~ your servers speak to this ~~dB~~ and separate dB and realize if they gotta rat lim. client or not

which stores nr/r records of all clients' reqs. to their resp. servers

e.g:- Redis. → ^{eg 5 req/}
_{min allowed}

separate dB
that stores nr/r client access records and does rate lim logic and likewise,

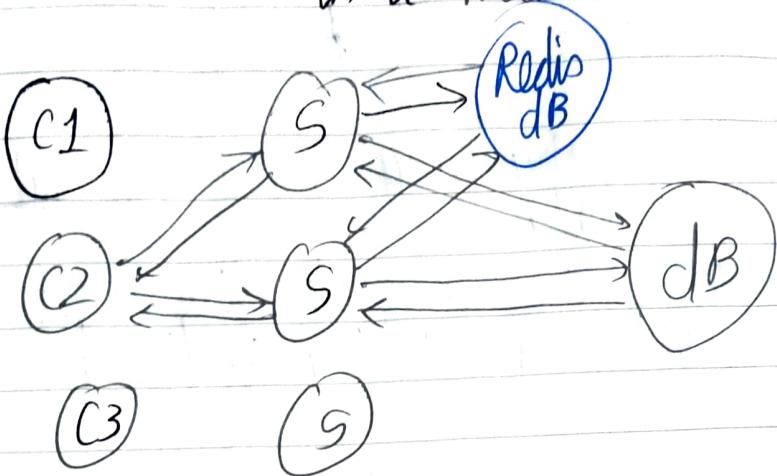
tells a server to rate lim a client or not



on its 3rd req.

Date: _____
M T W T F S S

In our ex:- C1 is blocked / rate lim from all other servers as it issued > 2 reqs. in a min.



C2 has ≤ 2 reqs per min so it gets back data from main dB both times.

⑧ tier based Rate Limiting

ex - HTTP reqs:

1 req./0.5s but only 3 reqs/10sec but only 10 reqs/min

Thus, we ~~still~~ give users bit more freedom whilst still maintaining a strong grasp over their power to keep req. in

Complex to code this out tho

server.js — rate_limiting

JS server.js X JS database.js

JS server.js > app.get('/index.html') callback > [0] previousAccessTime

```
1 const database = ...;
2 const express = require('express');
3 const app = express();
4
5 app.listen(3000, () => console.log('Listening on port 3000.'));
6
7 // Keep a hash table of the previous access time for each user.
8 const accesses = {};
9
10 app.get('/index.html', function(req, res) {
11   const {user} = req.headers;
12   if (user in accesses) {
13     const previousAccessTime = accesses[user];
14
15     // Limit to 1 request every 5 seconds.
16     if (Date.now() - previousAccessTime < 5000) {
17       res.status(429).send('Too many requests.\n');
18       return;
19     }
20   }
21
22   // Serve the page and store this access time.
23   database.get('index.html', page => {
24     accesses[user] = Date.now();
25     res.send(page + '\n');
26   });
27});
```

```
rate_limiting — bash — 93x25
...ents/Content/Design_Fundamentals/Examples/rate_limiting — node server.js | ...cuments/Content/Design_Fundamentals/Examples/rate_limiting — bash
[Clements-MBP:rate_limiting clementmihailescu$ curl -H 'user: clement' http://localhost:3000/index.html
<html>Hello World!</html>
[Clements-MBP:rate_limiting clementmihailescu$ curl -H 'user: clement' http://localhost:3000/index.html
<html>Hello World!</html>
[Clements-MBP:rate_limiting clementmihailescu$ curl -H 'user: clement' http://localhost:3000/index.html
Too many requests.
[Clements-MBP:rate_limiting clementmihailescu$ curl -H 'user: clement' http://localhost:3000/index.html
Too many requests.
[Clements-MBP:rate_limiting clementmihailescu$ curl -H 'user: clement' http://localhost:3000/index.html
Too many requests.
[Clements-MBP:rate_limiting clementmihailescu$ curl -H 'user: clement' http://localhost:3000/index.html
<html>Hello World!</html>
[Clements-MBP:rate_limiting clementmihailescu$ curl -H 'user: clement' http://localhost:3000/index.html
<html>Hello World!</html>
[Clements-MBP:rate_limiting clementmihailescu$ curl -H 'user: clement' http://localhost:3000/index.html
Too many requests.
Clements-MBP:rate_limiting clementmihailescu$ █
rate_limiting — curl -H user: antoine http://localhost:3000/index.html — 93x23
~/Documents/Content/Design_Fundamentals/Examples/rate_limiting — curl -H user: antoine http://localhost:3000/index.html
[Clements-MBP:rate_limiting clementmihailescu$ curl -H 'user: antoine' http://localhost:3000/index.html
<html>Hello World!</html>
[Clements-MBP:rate_limiting clementmihailescu$ curl -H 'user: antoine' http://localhost:3000/index.html
Too many requests.
[Clements-MBP:rate_limiting clementmihailescu$ curl -H 'user: antoine' http://localhost:3000/index.html
```