Jak działa internet?
Jak obsłużyć O(Google) zapytań na sekundę?

Maciej Dębski
md319428@students.mimuw.edu.pl

Jak działa internet?

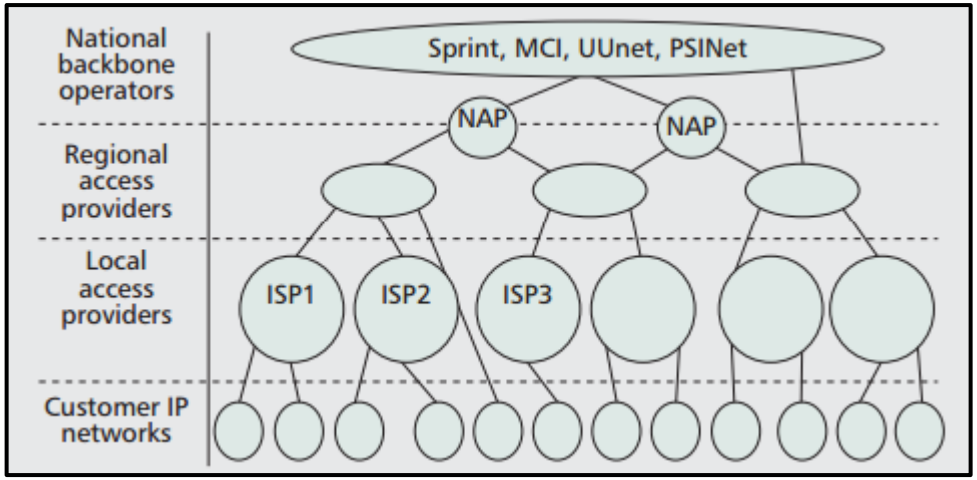
- duże organizacje utrzymują swoje sieci - systemy autonomiczne (AS)
- organizacje wymieniają między sobą ruch - “peering”
- za pomocą protokołu BGP wymieniają informacje o routingu między sobą
- umowy i prawnicy służą do zapewnienia, że peering opłaca się obu stronom
- często fizycznie peering zrealizowany jest w IXP - globalnych punktach wymiany ruchu

Demo

```
md@eduroam-mimuw:~$ traceroute -A -n 8.8.8.8
 1  10.20.0.1 [AS65534]  8.140 ms
 2  193.0.96.31 [AS8890]  10.875 ms
 3  212.191.224.33 [AS8501]  19.353 ms
 4  62.40.125.245 [AS20965]  13.584 ms
 6  62.40.125.202 [AS20965]  58.267 ms
 7  209.85.241.110 [AS15169]  29.628 ms
 8  209.85.251.246 [AS15169]  32.828 ms
    72.14.234.231 [AS15169]  35.118 ms
11  216.239.49.28 [AS15169]  41.367 ms
12  * * *
13  8.8.8.8 [AS15169]  34.678 ms
```

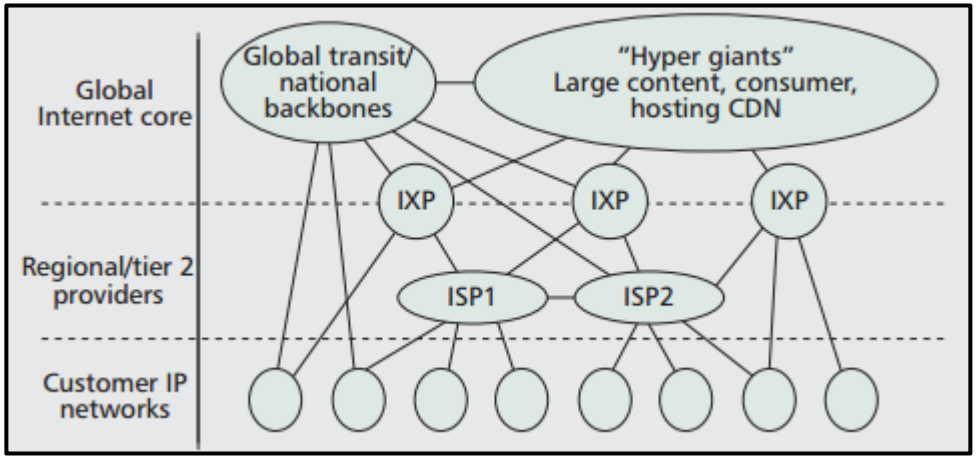
https://stat.ripe.net/widget/bgplay#w.resource=193.0.96.218

Ewolucja internetu



1995-2007 “podręcznikowy” internet

2009+ rzeczywisty internet



Ewolucja internetu

(a) Top ten, 2007			(b) Top ten, 2009		
Rank	**Provider**	**%**	**Rank**	**Provider**	**%**
1	Level(3)	5.77	1	Level(3)	9.41
2	Global Crossing	4.55	2	Global Crossing	5.7
3	ATT	3.35	3	**Google**	5.2
4	Sprint	3.2	4		
5	NTT	2.6	5		
6	Cogent	2.77	6	**Comcast**	3.12
7	Verizon	2.24	7		
8	TeliaSonera	1.82	8		
9	Savvis	1.35	9		
10	AboveNet	1.23	10		

największe organizacje pod względem generowanego ruchu

IXP na świecie

BGP

- zdecentralizowany dynamiczny protokół routingu
- trwałe sesje zestawiane pomiędzy routerami
- każdy router rozgłasza swoje ścieżki do wszystkich prefiksów, wraz z informacją o wszystkich AS na tej ścieżce
- BGP zbiera wszystie otrzymane ścieżki i wybiera najlepszą
- “lepsze” ścieżki do przede wszystkim te z mniejszą liczbą AS, ale można też ustawiać preferencje

Bezpieczeństwo

- BGP w sporym stopniu opiera się na zaufaniu*
- nie bardzo da się wprowadzić bezpieczeństwo - ustawienia sprowadzałyby się w zasadzie do statycznej konfiguracji światowego routingu
- wszyskie większe AS bez problemu mogą przeprowadzić BGP hijacking - czasem przez przypadek
- stosuje się filtrowanie prefiksów w ograniczonym zakresie.
- główne rozwiązanie to monitoring i reakcja na incydenty
- wiadomości BGP są szyfrowane i podpisywane

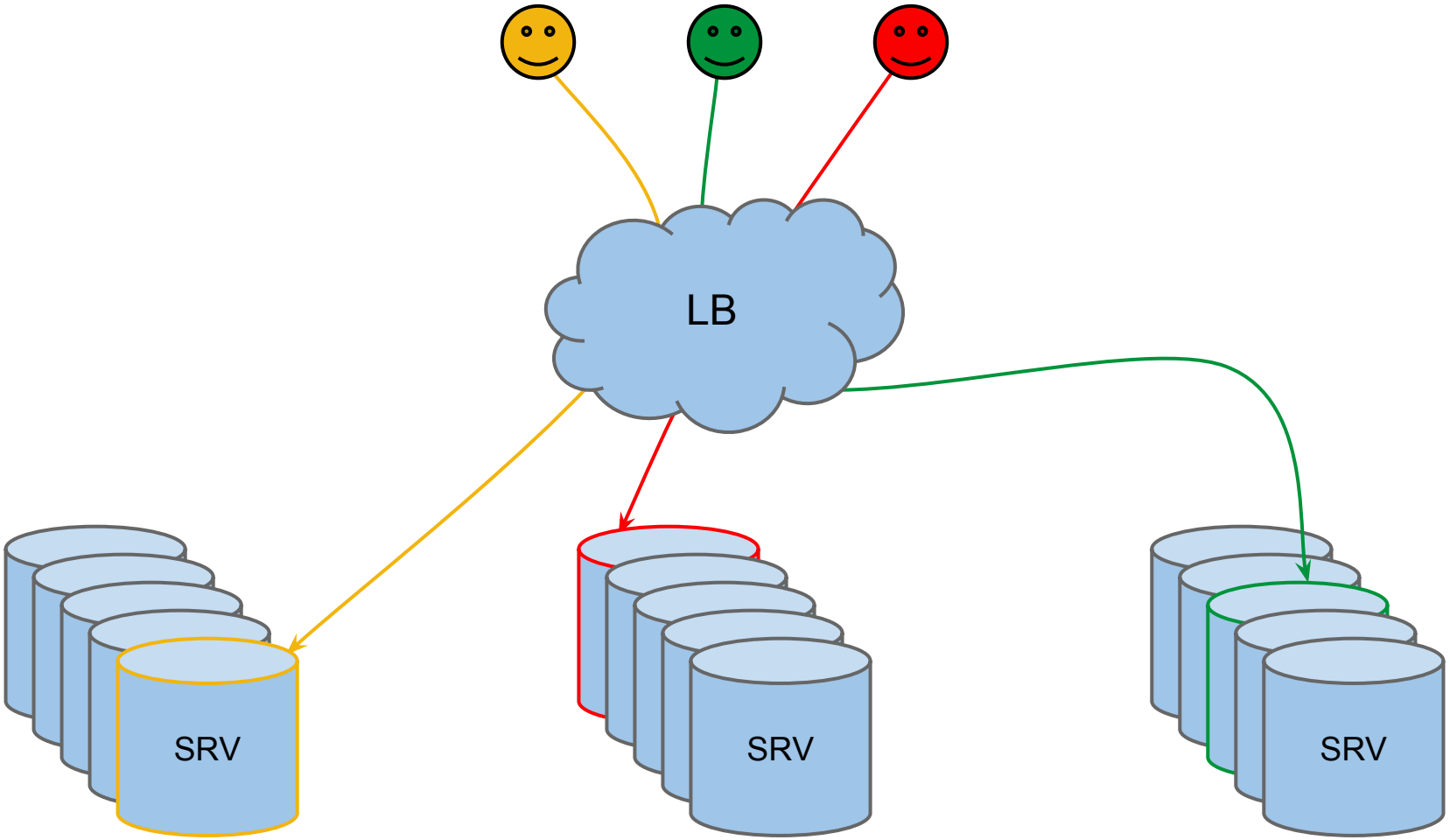
Bezpieczeństwo - przykłady

- “Pakistan causes YouTube outage for two-thirds of world”
- Meksyk -> Waszyngton DC przez Białoruś
- Kradzież bitcoinów z minerów
- “Chinese Routing Errors Redirect Russian Traffic”

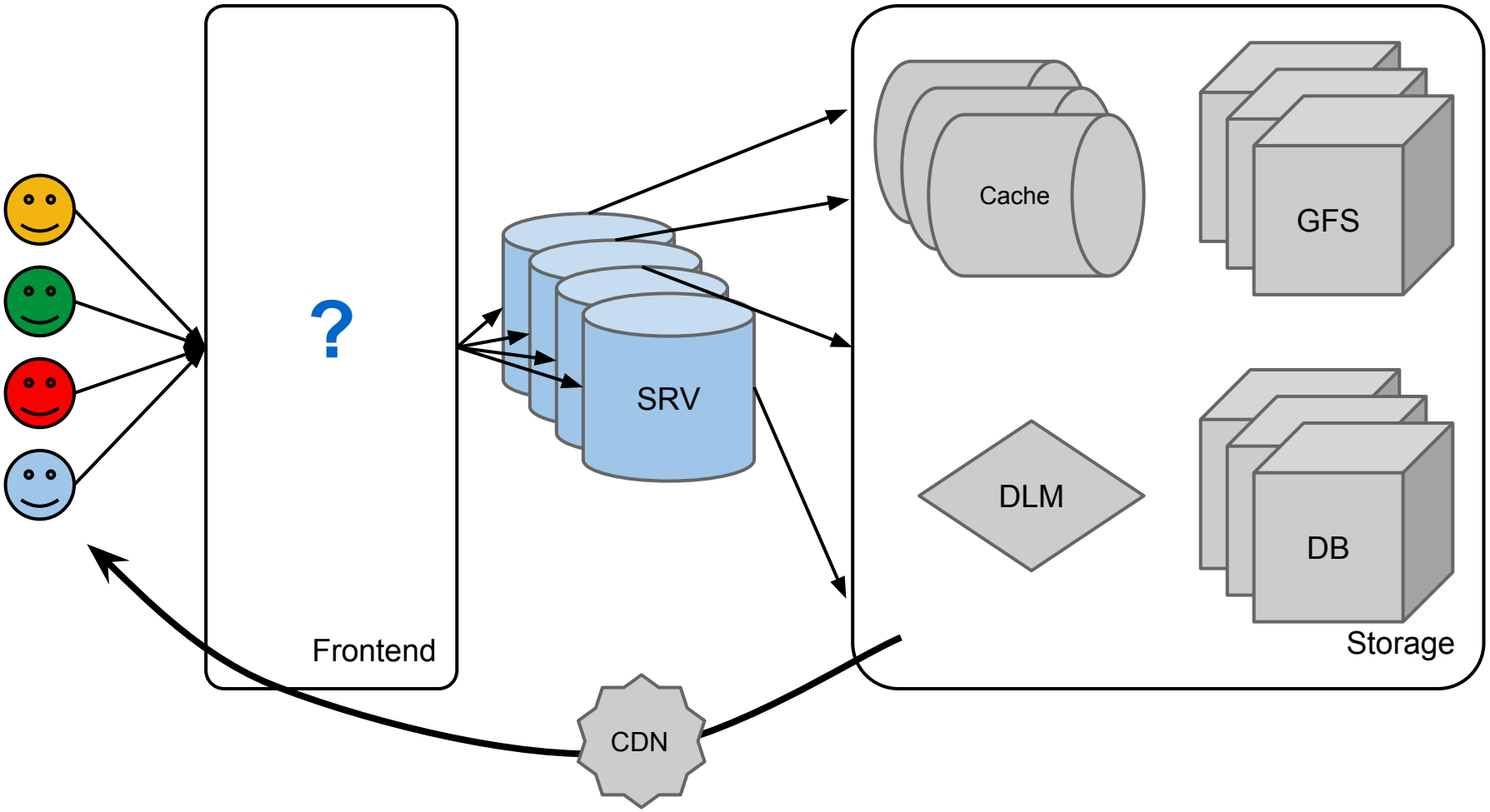
Równoważenie obciążenia - problem

- bardzo szybko usługi przestają się mieścić na jednym serwerze
- wysoka dostępność wymaga posiadania więcej niż jednej maszyny /pomieszczenia, budynku, kontynentu…/
- geograficzne rozmieszczenie użytkowników a opóźnienie
- ochrona serwerów przed nadmiernym obciążeniem

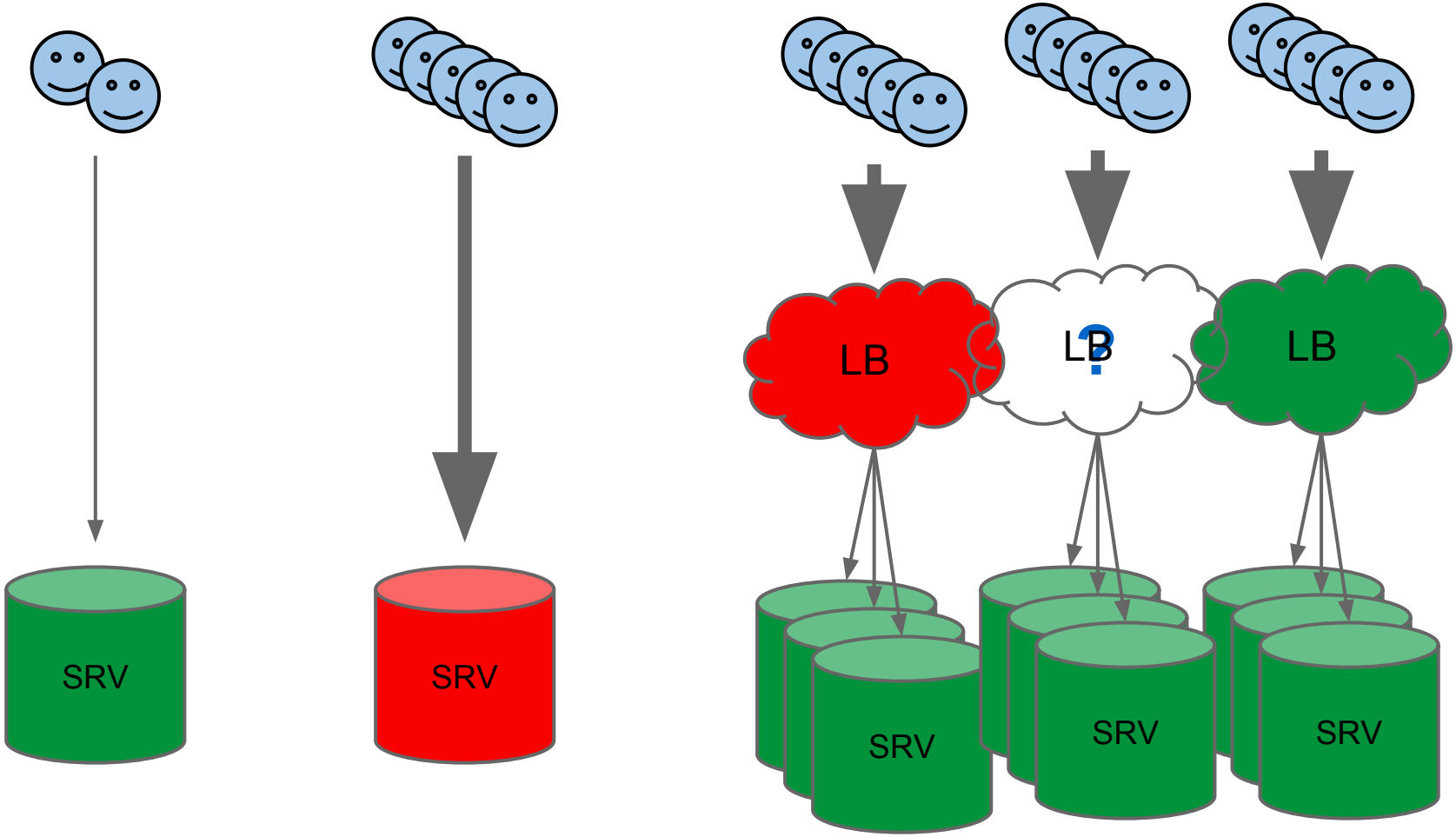
Jak to działa?



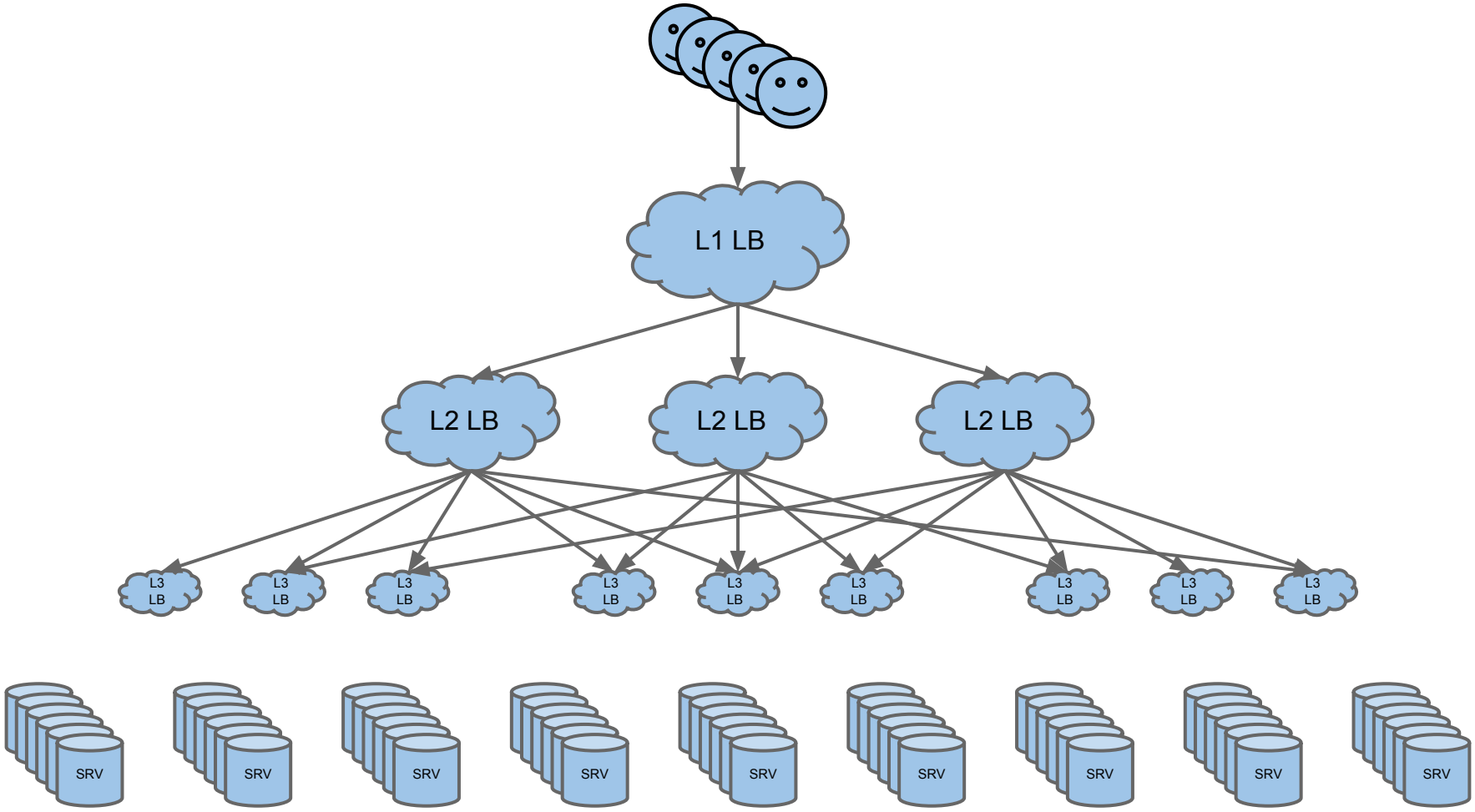
Architektura aplikacji www



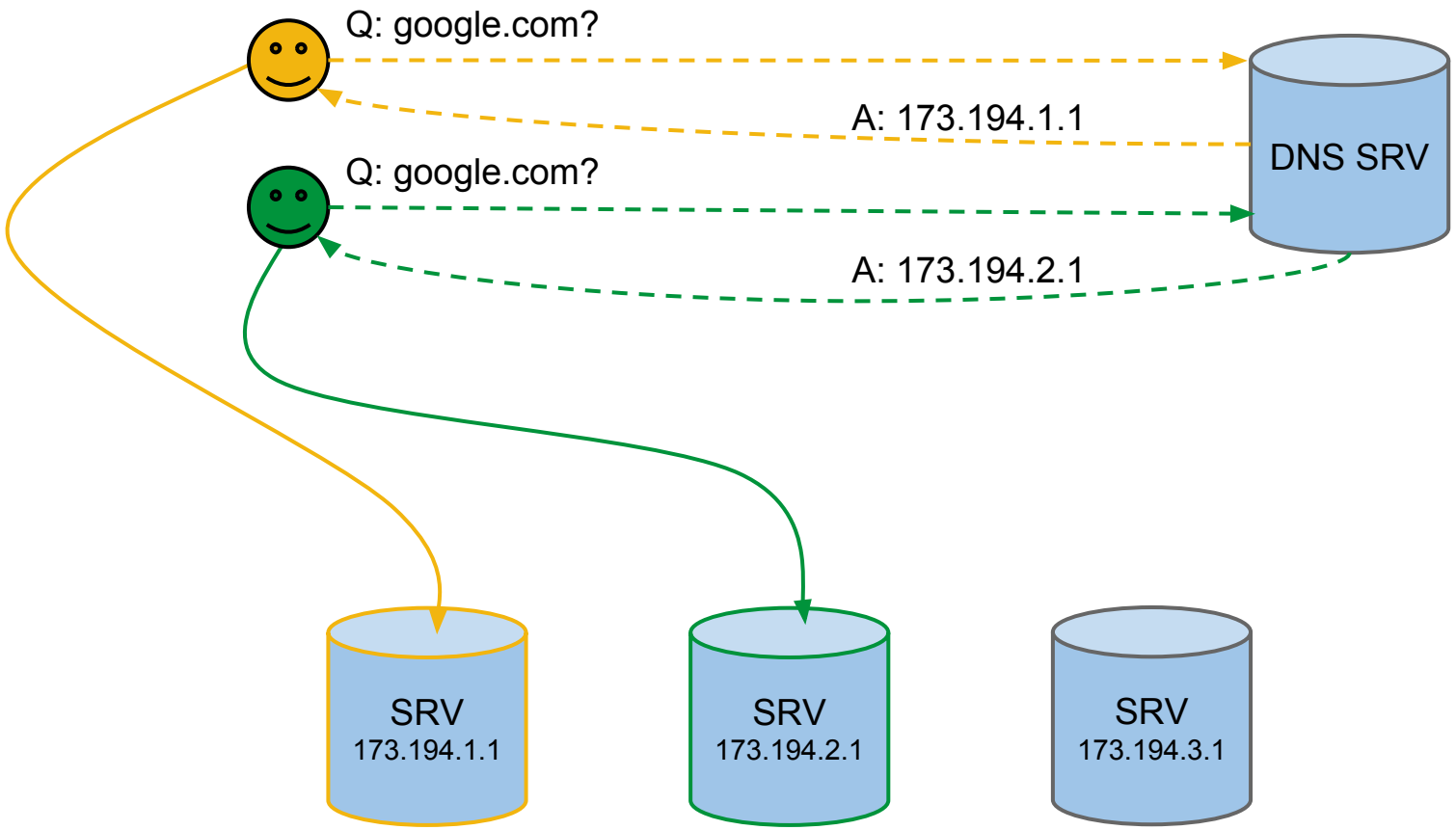
Czy to ma szanse działać?



A tak naprawdę...



DNS


Q: google.com?
A: 173.194.1.1
DNS SRV
Q: google.com?
A: 173.194.2.1
SRV
173.194.1.1
SRV
173.194.2.1
SRV
173.194.3.1

DNS - cechy

- Proste, powszechnie używane
- Mało elastyczne - DNS jest cache'owany na całym świecie, trzeba ustawić małe TTL (=> dużo zapytań).
- Delikatne dla użytkownika - zmiana nie zrywa TCP, zmiany następują dopiero przy następnym zapytaniu
- Ruch nie przechodzi przez LB - w zasadzie to komputer użytkownika służy za LB, my tylko dostarczamy metadane
- Słaba korelacja między liczbą zapytań DNS a HTTP

Zapytania DNS a HTTP

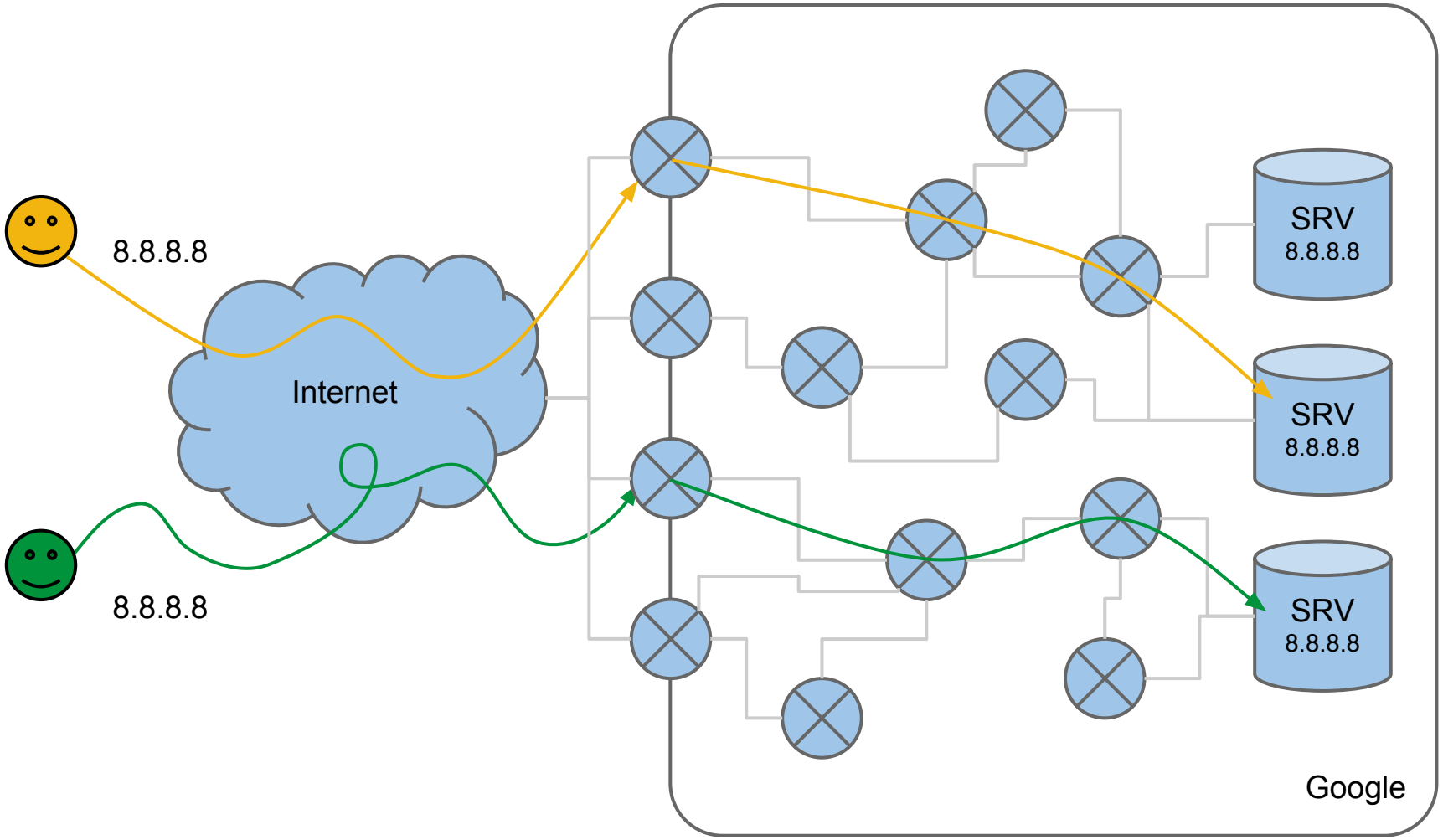
- Niektórzy tylko wyszukają coś z komórki: HTTP/DNS ~ 1, inni właśnie piszą prezentację i szukają dużo informacji: HTTP/DNS ~ 50
- Tak naprawdę to dopiero nasz serwer DNS pyta googlowego (autorytatywnego). Czyli odpowiedź dotyczy nie jednego użytkownika a wszystkich za tym serwerem.
- Google próbuje wprowadzić rozszerzenie do DNS, żeby serwer przekazywał “kto pyta”

Równoważenie obciążenia - DNS

Demo

```
md@home$ ./lookup.py www.google.com
 173.194.112.176  (fra07s32-in-f16.1e100.net.)
 173.194.112.177  (fra07s32-in-f17.1e100.net.)
 173.194.112.178  (fra07s32-in-f18.1e100.net.)
…
md@xivlo$ ./lookup.py www.google.com
 173.194.39.17  (ham02s13-in-f17.1e100.net.)
 173.194.39.18  (ham02s13-in-f18.1e100.net.)
 173.194.39.19  (ham02s13-in-f19.1e100.net.)
…
md@ec2$ ./lookup.py www.google.com
 74.125.228.82  (iad23s07-in-f18.1e100.net.)
 74.125.228.83  (iad23s07-in-f19.1e100.net.)
…
```

Anycast (BGP)



Anycast - cechy

- równoważenie obciążenia sieci
- bardzo szybkie, mało inteligentne
- w ten sposób w ogóle sieć wytrzymuje taki ruch
- niestabilne - słabe gwarancje dokąd pakiet dotrze, jeśli więcej niż jedno miejsce z danym IP
- bezpośrednio może obsługiwać tylko usługi bezstanowe.
- słaba metryka - można łatwo przeciążyć jeden - najbliższy - serwer

Demo

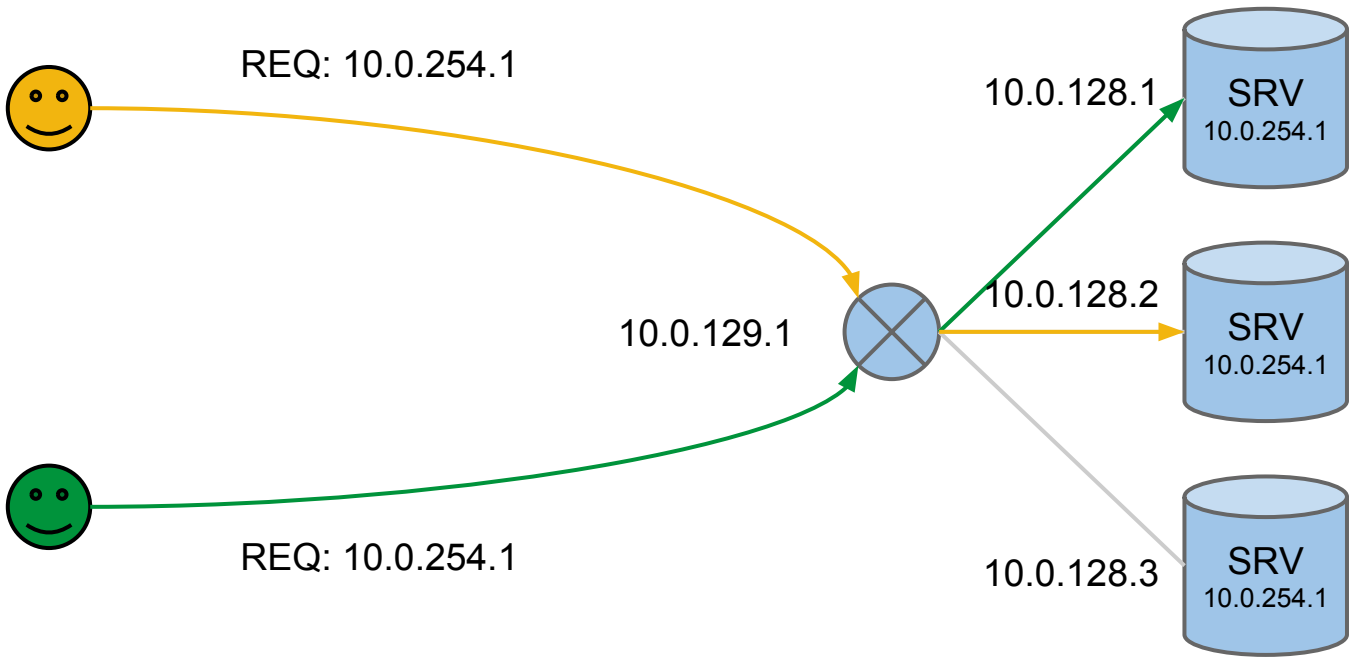
```
md@ec2$ traceroute 8.8.8.8
 5  72.21.222.34
 6  72.21.220.68
 7  72.14.215.157
 8  209.85.252.46
 9  72.14.236.98
10  66.249.95.231
11  72.14.234.55
12  * * *
13  8.8.8.8 (~10 ms)
```

```
md@home$ traceroute 8.8.8.8

 7  72.14.221.94
 8  209.85.240.64
 9  72.14.234.231
10  209.85.241.226
11  209.85.254.227
12  209.85.255.51
13  * * *
14  8.8.8.8 (~40 ms)
```

Równoważenie obciążenia

ECMP



ECMP - cechy

- LB w warstwie sieci, na standardowych routerach
- szybkie, mało inteligentne
- hash po 5-tuple gwarantuje że to samo połączenie trafia do tego samego serwera
- Ale jak zmienimy pulę serwerów, to wszystko się miesza i zrywamy większość połączeń TCP.

Równoważenie obciążenia - ECMP

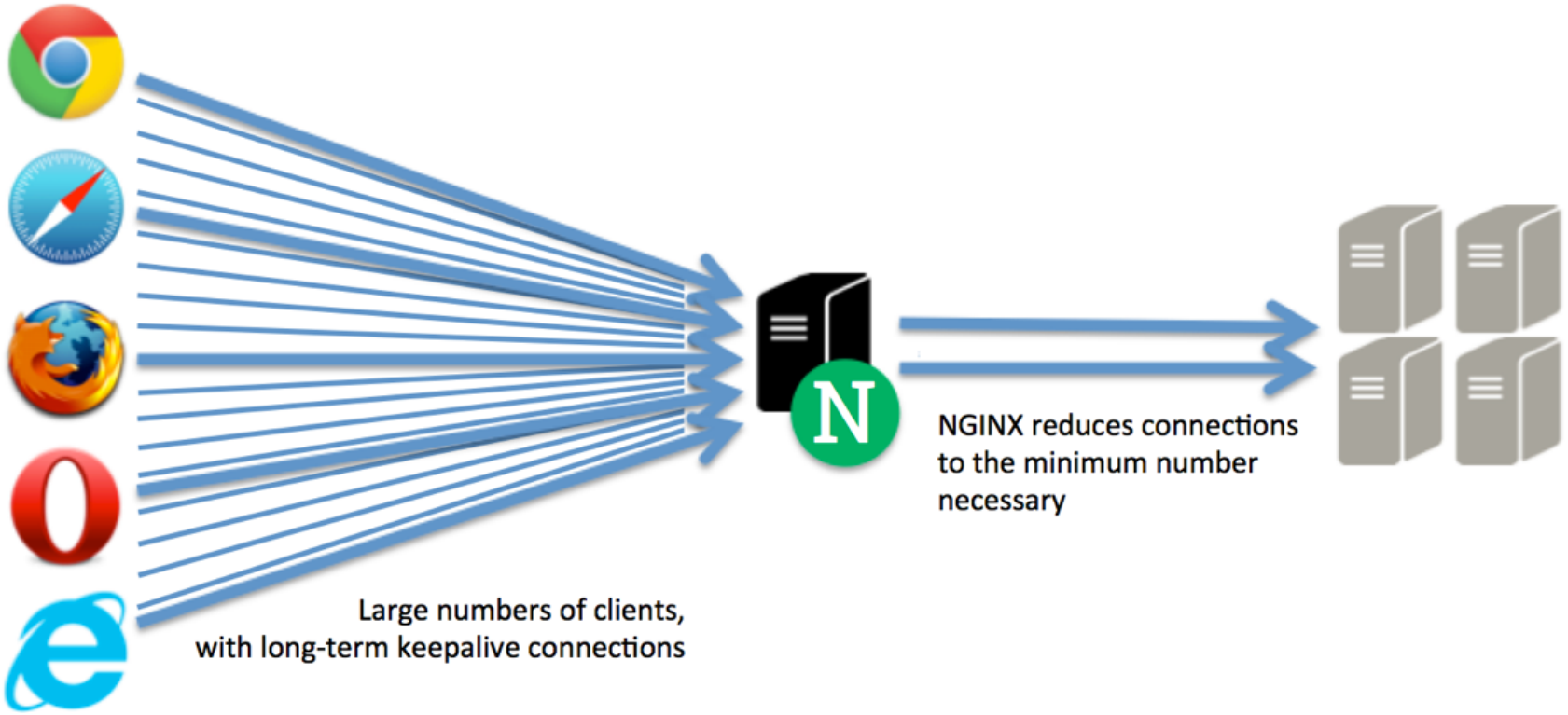
Demo

```
root@debian1:~# ip a sh
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 10.0.254.1/32 brd 10.0.254.1 scope global lo
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
    link/ether 52:54:00:92:aa:01 brd ff:ff:ff:ff:ff:ff
    inet 10.0.128.1/24 brd 10.0.128.255 scope global eth0
```

```
root@debian2:~# ip a sh
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 10.0.254.1/32 brd 10.0.254.1 scope global lo
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
    link/ether 52:54:00:92:aa:02 brd ff:ff:ff:ff:ff:ff
    inet 10.0.128.2/24 brd 10.0.128.255 scope global eth0
```

```
[admin@MikroTik] /ip route> print
Flags: X - disabled, A - active, D - dynamic, C - connect, S - static
 #      DST-ADDRESS        PREF-SRC        GATEWAY            DISTANCE
 0 ADC  10.0.128.0/24      10.0.128.10     ether1                    0
 1 A S  10.0.254.1/32                      10.0.128.2                1
                                           10.0.128.1
```

HTTP Reverse Proxy



HTTP Reverse Proxy - cechy

- L7 - wie wszystko o aplikacji, np. interpretuje nagłówki http
- sporo wolniejsze
- dużo większe możliwości
- tu można użyć bardziej wyszukanych algorytmów - bo i tak jest dosyć wolne
- można zterminować SSL, HTTP - ograniczyć liczbę połączeń do serwerów, przejąć część ich pracy

Równoważenie obciążenia - ECMP

Demo

```
upstream backend {
   ip_hash;
   keepalive 20;
   sticky cookie srv_id expires=1h domain=.example.com path=/servlet;
   server webserver1 weight=1;
   server webserver2 weight=4;
}

server {
   listen 80;
   limit_conn perserver 1000;
   location / {
      proxy_pass http://backend;
   }
   location /internal-health-check1 {
      proxy_pass http://backend;
      health_check interval=2s fails=1 passes=5 uri=/test.php match=statusok;
   }
}

match statusok {
   status 200;
   body ~ "Server[0-9]+ is alive";
}
```

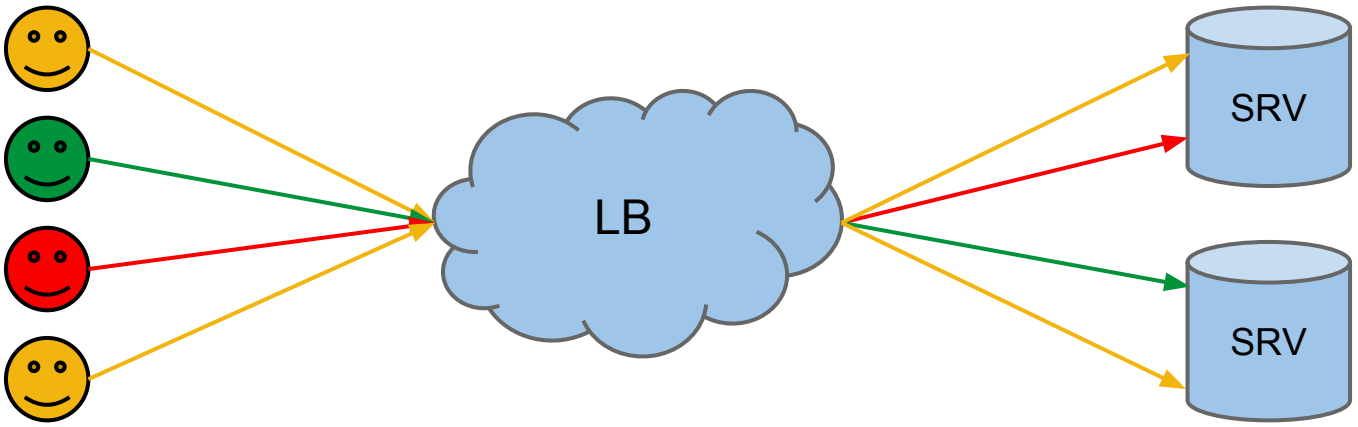
http://nginx.com/blog/load-balancing-with-nginx-plus-part2/

Algorytmy LB

- wybór serwera docelowego musi być bardzo szybki
- sytuacja zmienia się dynamicznie - serwery się restartują, psują, czasem regulujemy ich liczbę
- na zmiany powinniśmy reagować natychmiast
- chcemy żeby wybór był *stabilny* - nie zmieniał się za często dla danego użytkownika
- podział obciążenia pomiędzy serwery powinien być równomierny

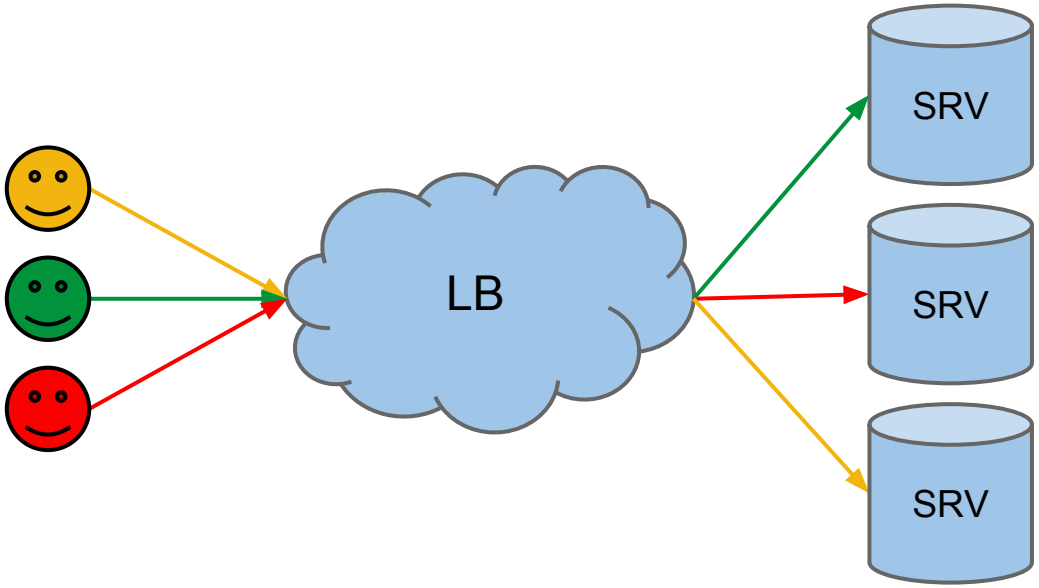
Round robin

- +bardzo szybkie
- +dokładnie równy podział
- -niestabilne



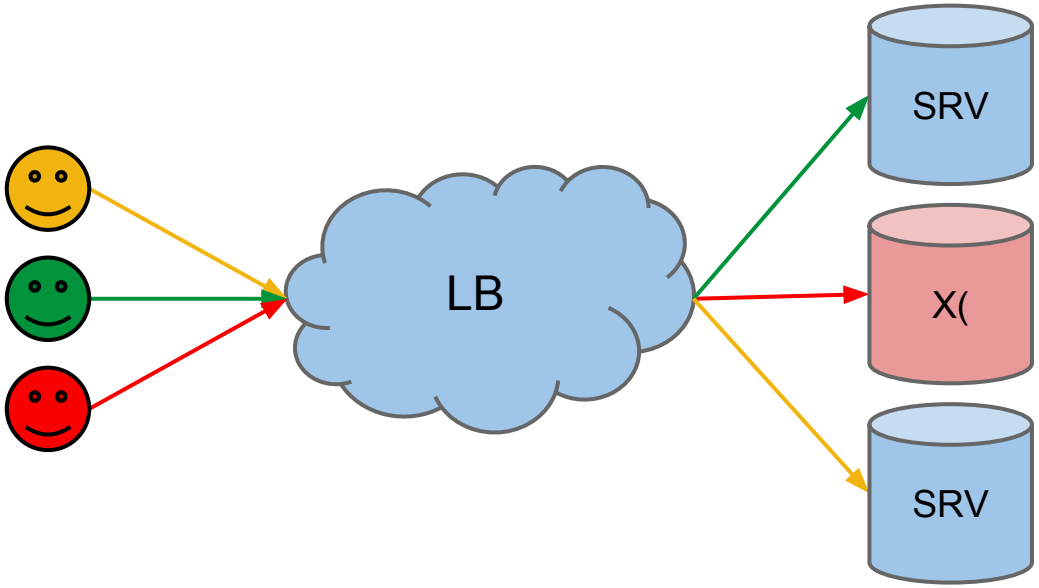
Haszowanie

- +szybkie
- +wystarczająco równy podział
- +stabilne
- -traci stabilność przy zmianach



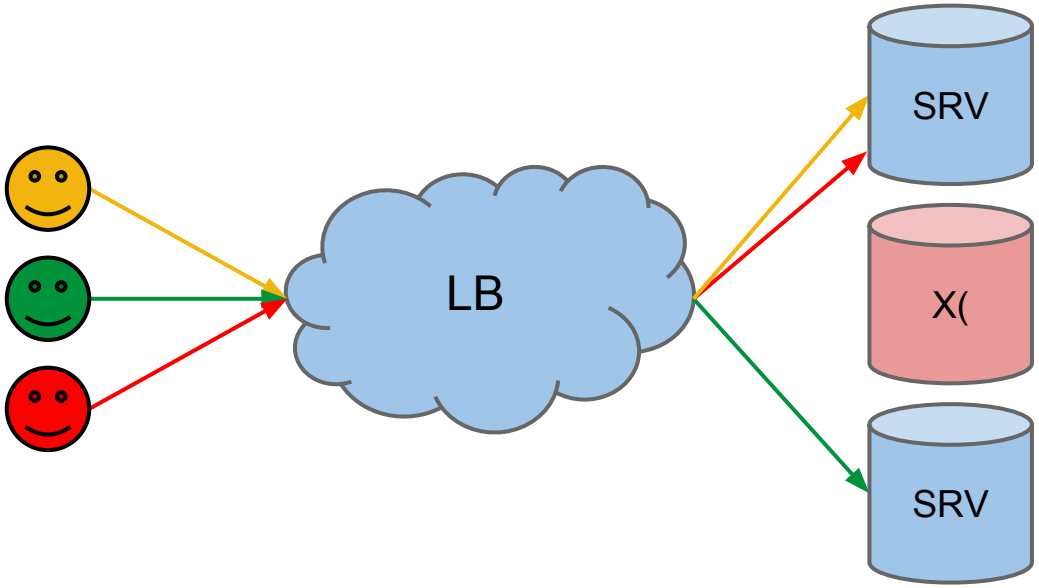
Haszowanie

- +szybkie
- +wystarczająco równy podział
- +stabilne
- -traci stabilność przy zmianach



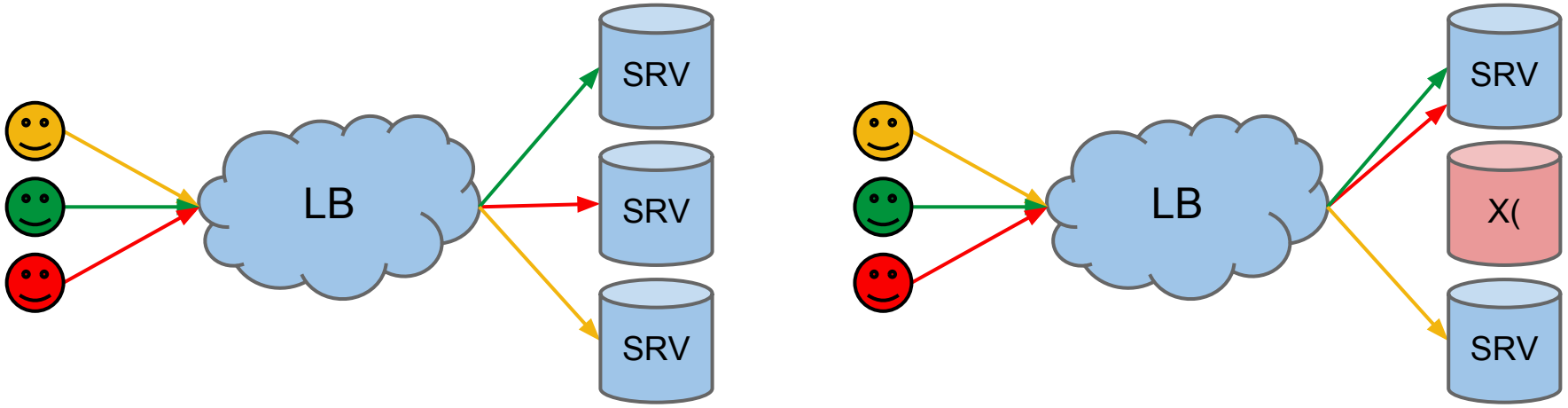
Haszowanie

- +szybkie
- +wystarczająco równy podział
- +stabilne
- -traci stabilność przy zmianach

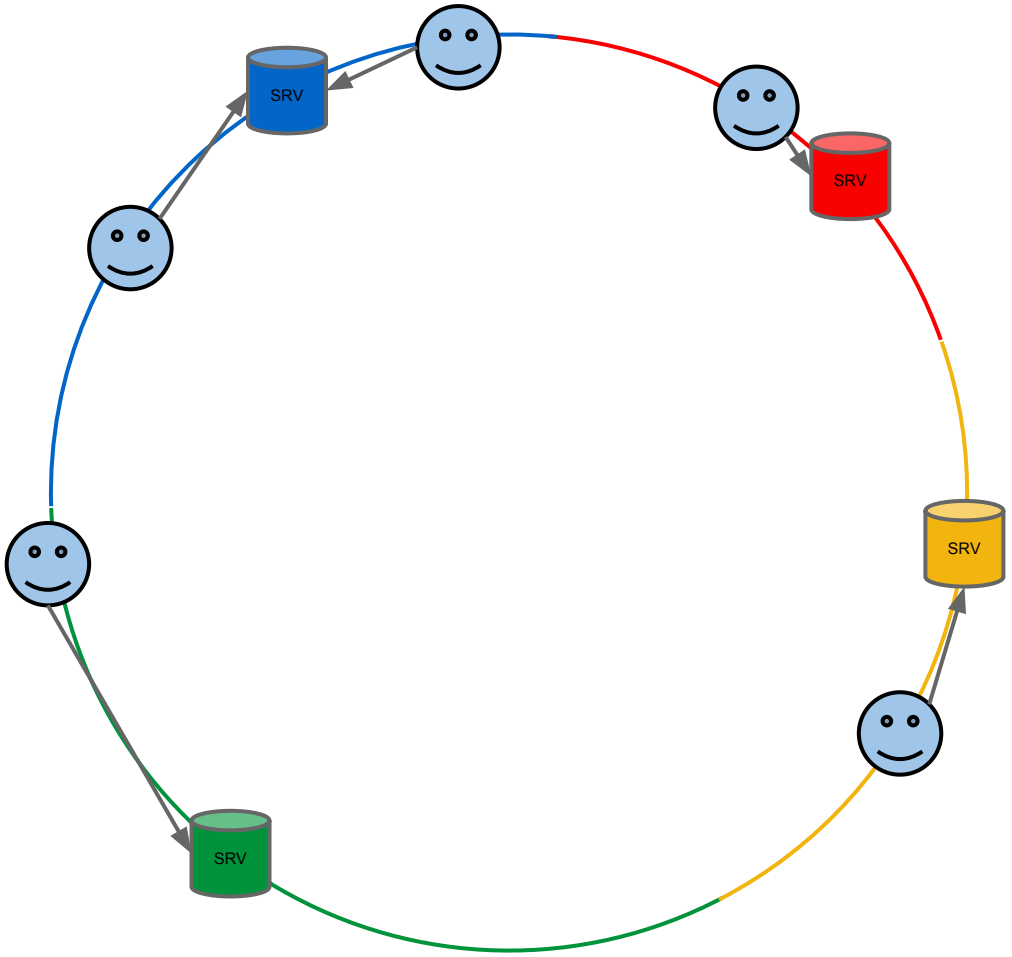


Spójne haszowanie

- a może haszowanie mogłoby nie tracić stabilności?



Standardowy algorytm

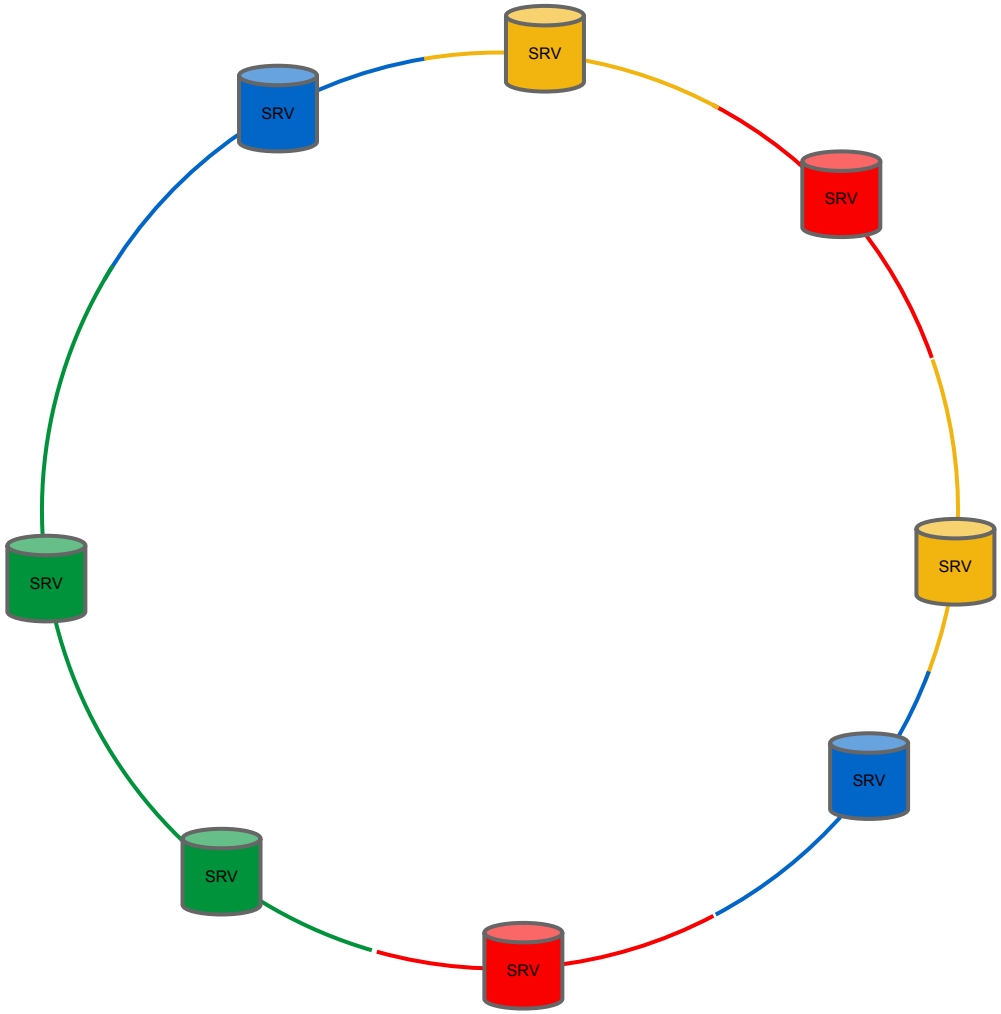


Standardowy algorytm - cechy

- słabo z równomiernością
- w miarę szybki -> O(log n) na wyszukanie najbliższego serwera
- O(n) pamięci

Spójne haszowanie

Poprawka



Poprawka - cechy

- nieźle z równomiernością, można regulować
- w miarę szybki -> O(log (n*k)) na wyszukanie najbliższego serwera
- O(n*k) pamięci

Points per Bucket	Standard Error	Bucket Size 99% Confidence Interval
1	0.9979060	(0.005, 5.25)
10	0.3151810	(0.37, 1.98)
100	0.0996996	(0.76, 1.28)
1000	0.0315723	(0.92, 1.09)

Haszowanie rendezvous

- policzyć hash(klucz, serwer) dla każdego serwera, i wybrać ten z maksymalnym
- O(n)
- pamięć O(1)
- dobra równomierność

Pre-Jump consistent hash

```
int ch(int key, int num_buckets) {
    random.seed(key);
    int b = 0; // This will track ch(key, j+1).
    for (int j = 1; j < num_buckets; j++) {
        if (random.next() < 1.0 / (j + 1)) b = j;
    }
    return b;
}
```

Jump consistent hash

```
int jch(int key, int num_buckets) {
    random.seed(key);
    int b = -1; // bucket number before the previous jump
    int j = 0; // bucket number before the current jump
    while (j < num_buckets) {
        b = j;
        r = random.next();
        j = floor((b + 1) / r);
    }
    return b;
}
```

Jump consistent hash - cechy

- nie spełnia warunków zadania - przestrzeń kubełków (=serwerów) postaci {0, 1, …, n} - nie można zepsuć serwera ze środka
- O(log(n)), pamięć O(1)
- dobra równomierność
- wg. autorów dobre do storage, gdzie i tak nie chcemy żeby dane znikały
- wydaje mi się że powinno się dać dostosować do potrzeb LB

Co ciekawego robi Google?

- tworzy LB - bardzo wydajne oprogramowanie (a czasem sprzęt…), ogromna przepustowość, minimalne opóźnienie, straszna odpowiedzialność
- rozwija algorytmy LB
- rozwija SDN (software defined networking)
- pilnuje globalnego systemu równoważenia obciążenia
- pilnuje ogromnej sieci (netops)

Dzięki za uwagę!

Pytania?

Maciej Dębski
md319428@students.mimuw.edu.pl

Backup slides

Maciej Dębski
md319428@students.mimuw.edu.pl

Skąd google obsługuje zapytania

- Data center - ogromne budynki z dala od ludzi, większość mocy obliczeniowej, pamięci, dysków
- Edge - trochę maszyn blisko klientów - tylko load balancing i cache, bardzo lekkie usługi
- Google Global Cache - LB i cache, w sieciach ISP, mniej zaufane

Skąd google obsługuje zapytania

Data centers

Skąd google obsługuje zapytania

Edge

Google global cache (GGC)

Wszędzie…

Nie ma oficjalnej listy, ale można łatwo zaobserwować że większość polskich operatorów ma u siebie GGC.

Czemu tak?

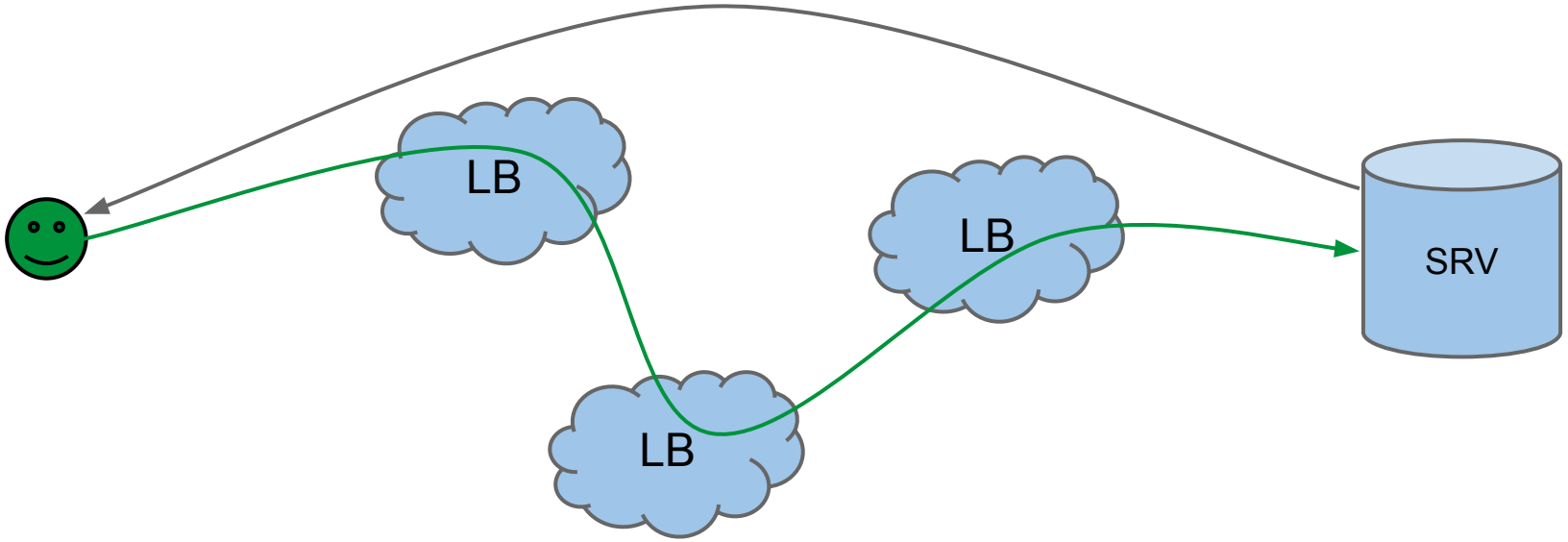
chcemy żeby użytkownik jak najszybciej dotarł do infrastruktury googlowej

- lepsza kontrola
- terminacja TCP, SSL
- agregacja połączeń
- cache

Jak wyłączyć serwer/klaster?

- wycofać z DNS
- wyłączyć w konfiguracji reverse proxy
- przestać rozgłaszać BGP
- route injection

DSR



Sprites