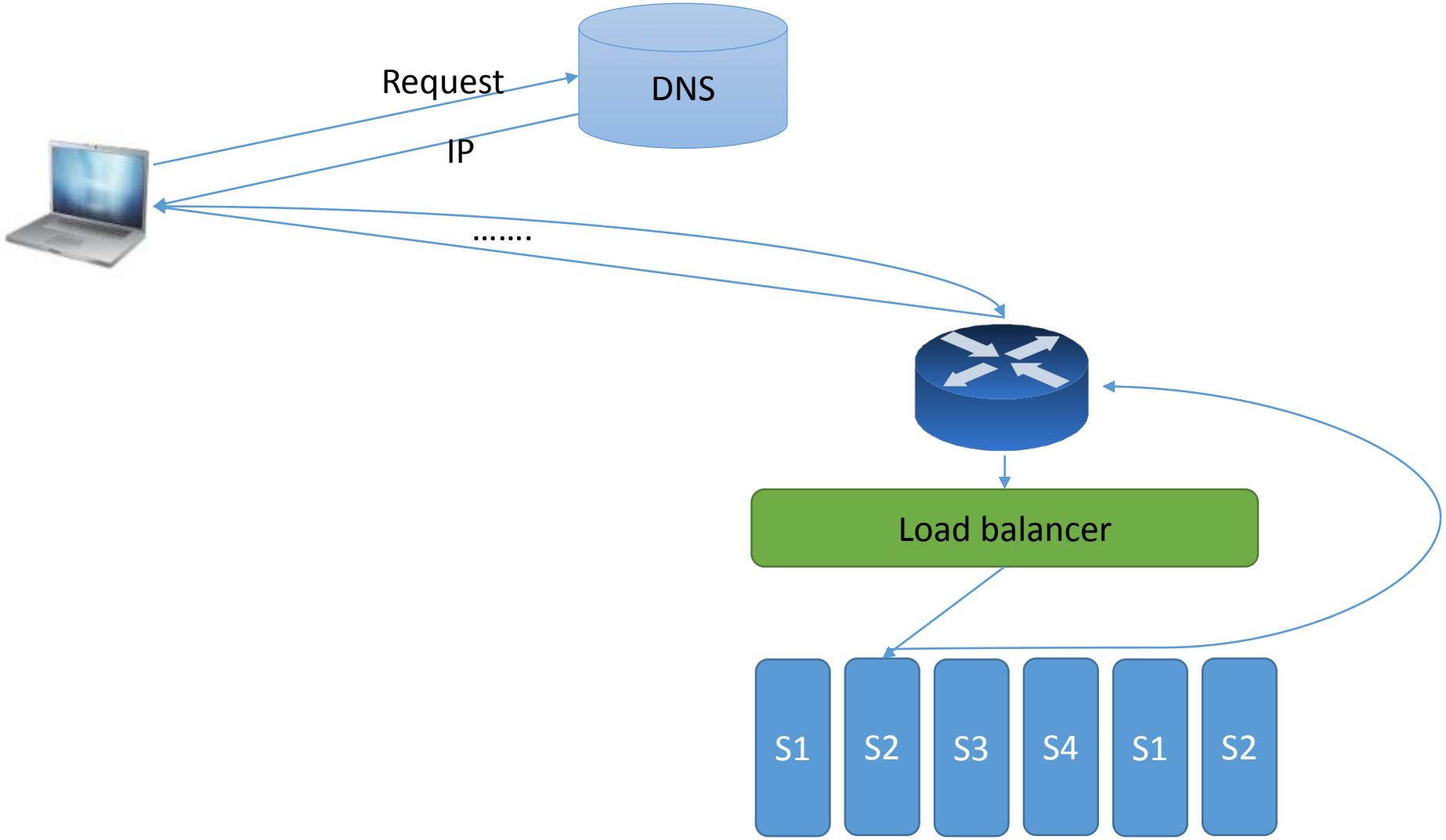
Maglev: A Fast and Reliable Software Network Load Balancer

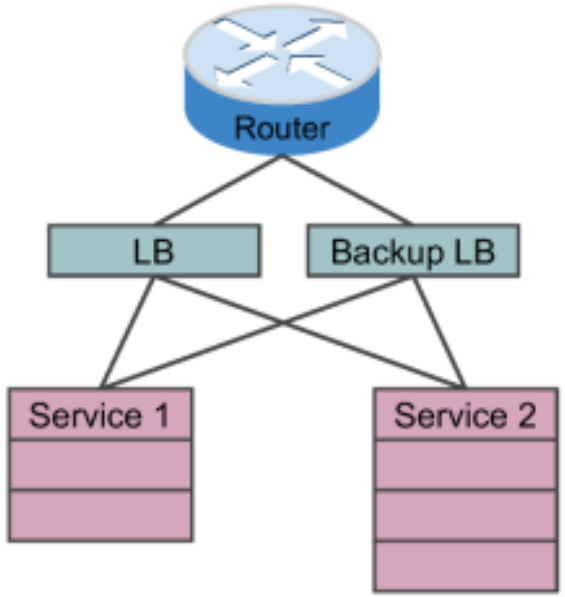
Daniel E. Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith, Roman Kononov, Eric Mann-Hielscher, Ardas Cilingiroglu, and Bin Cheyney, *Google Inc.;* Wentao Shang, *University of California, Los Angeles;* Jinnah Dylan Hosein, *SpaceX*

Introduction to load balancer



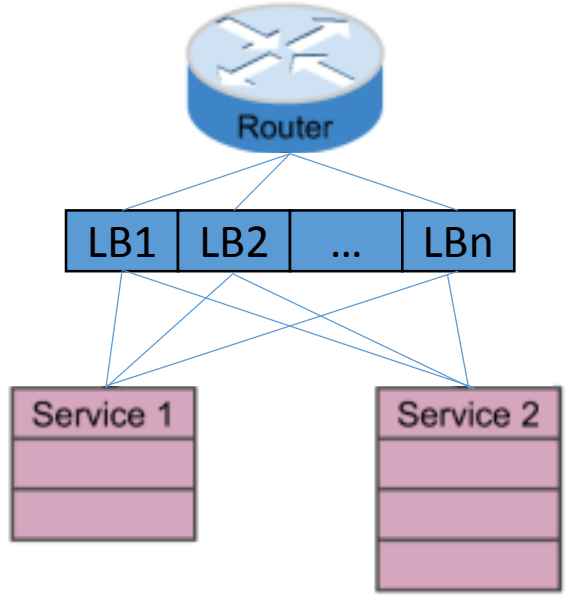
Limitation of dedicated hardware based Load balancer

- Scalability is constrained by hardware devices.
- Only providing 1+1 redundancy: poor reliability.
- Lacking flexibility and programmability for quick iteration.
- Costly to upgrade.



Advantages of software load balancer

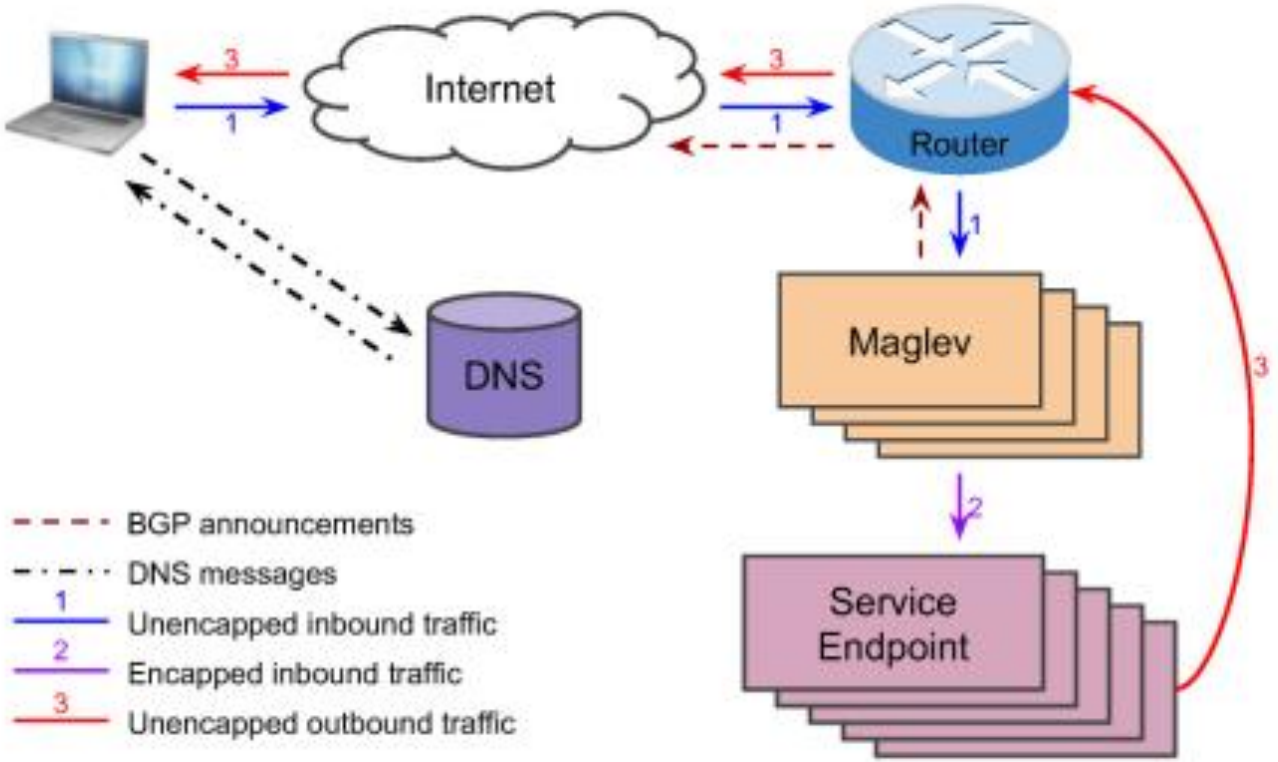
- Achieving Scalability by scaling out.
- Providing N+1 redundancy: high reliability.
- High flexibility and programmability for quick iteration.
- Easy to upgrade.



Challenges for software load balancer

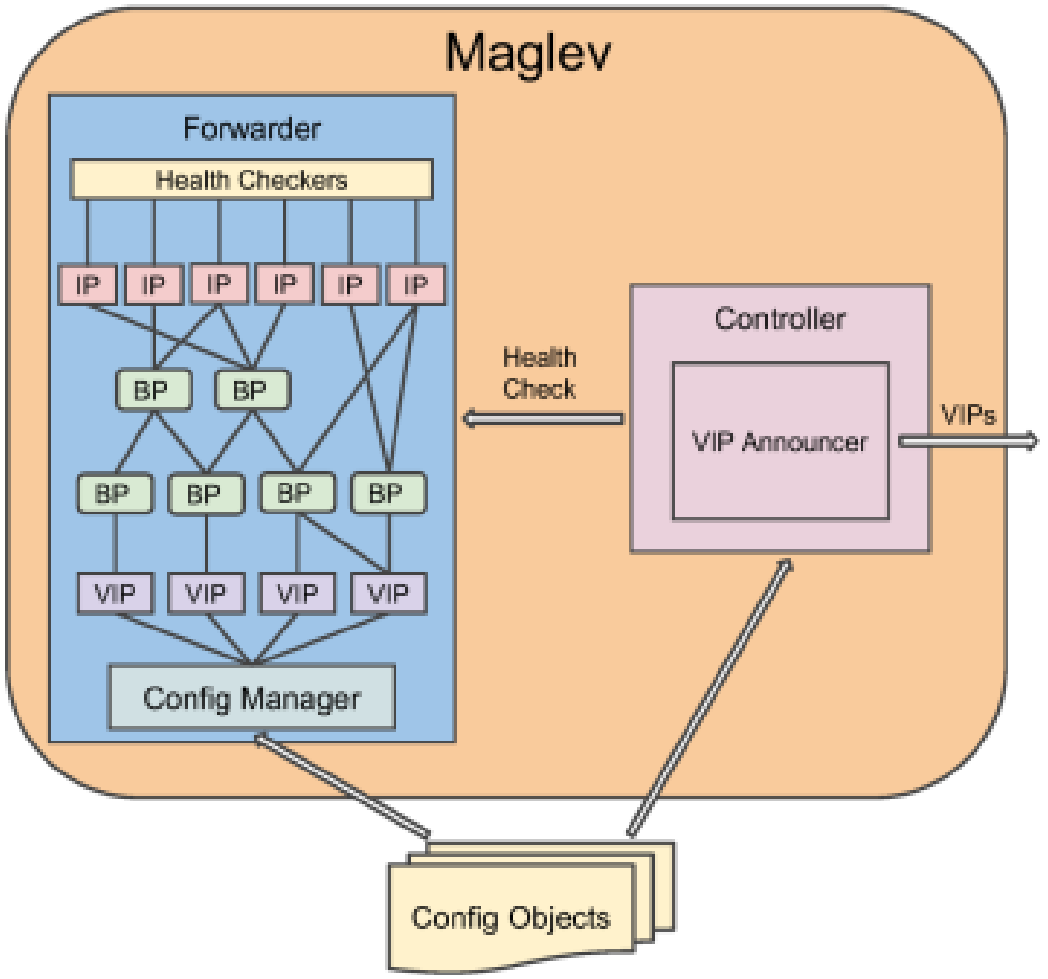
- Individual machine must provide high throughput.
- Connect persistence: packets belonging to the same connection should always be directed to the same service endpoint when the system dynamically changes.
- Achieving both load balancing and keeping most of connection alive when the number of maglev or service instance changes.

Maglev System Overview



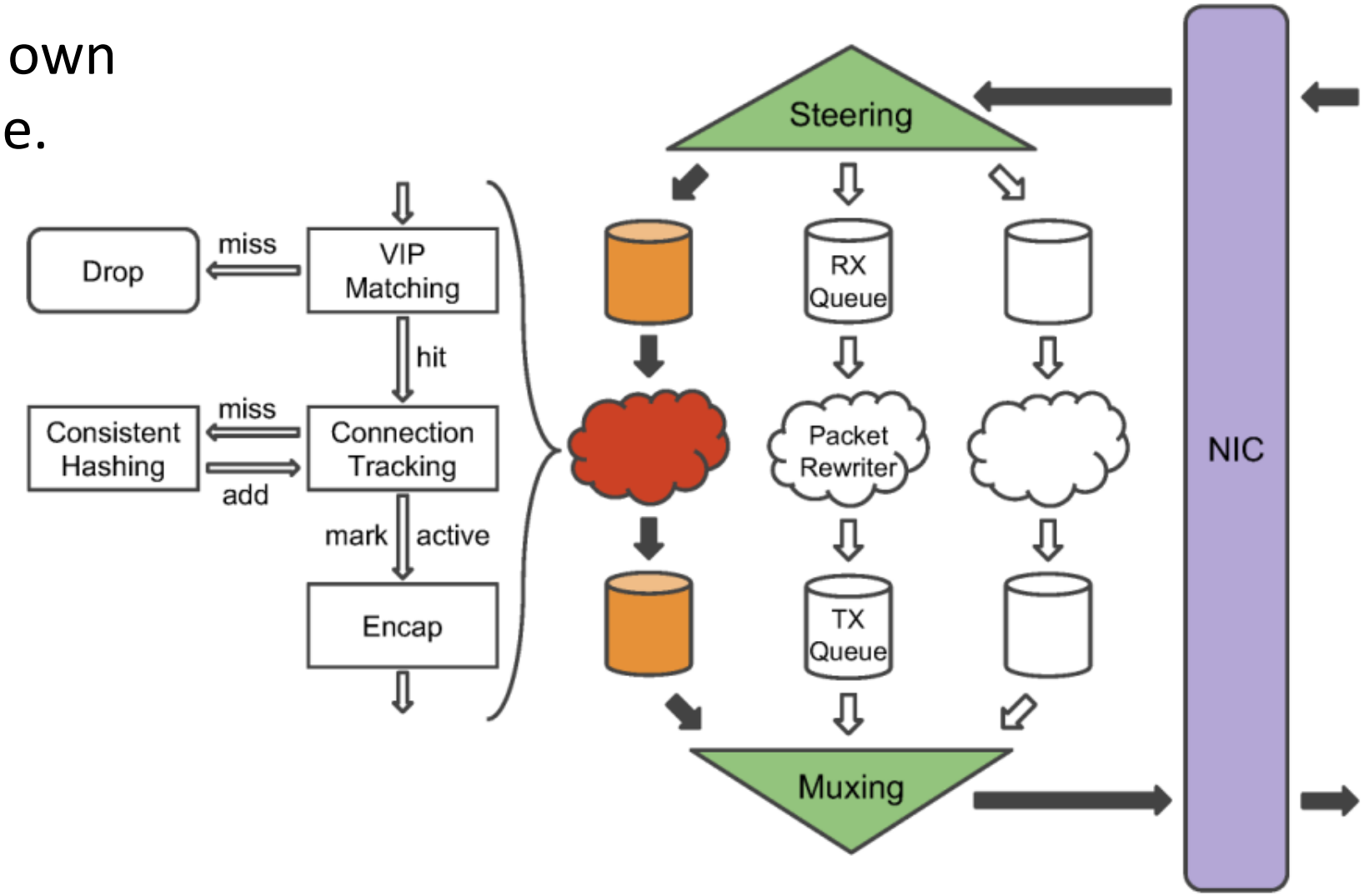
Maglev config

- Controller
  - VIP Announcer
- Forwarder
  - Config Manager
  - BP(Backend Pools)
  - Health Checkers



Maglev forwarder structure

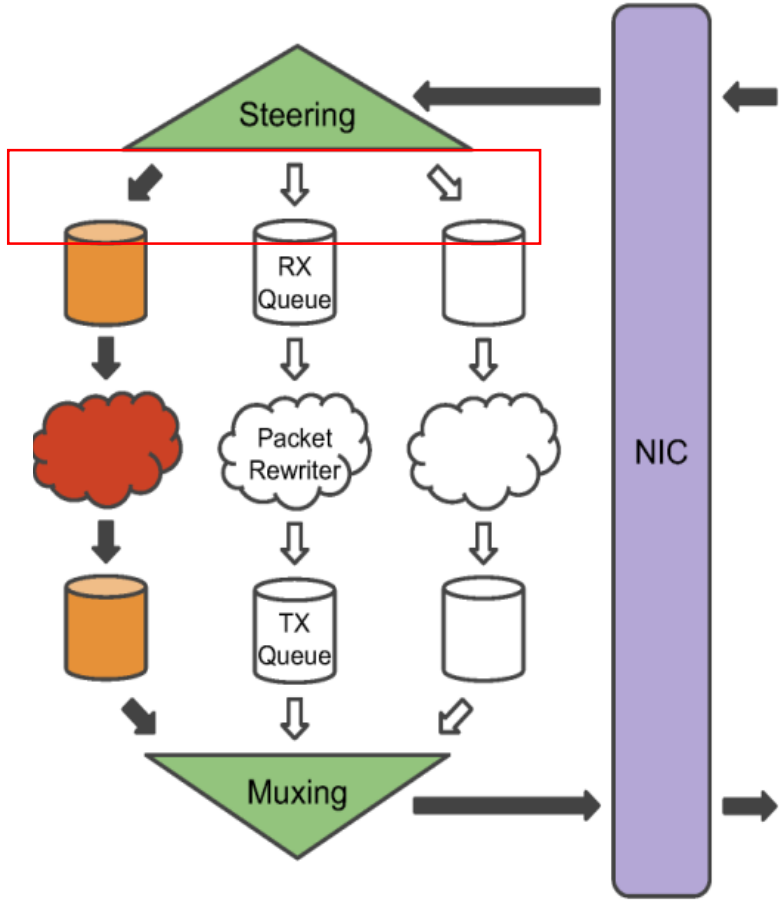
Each thread contains its own connection tracking table.



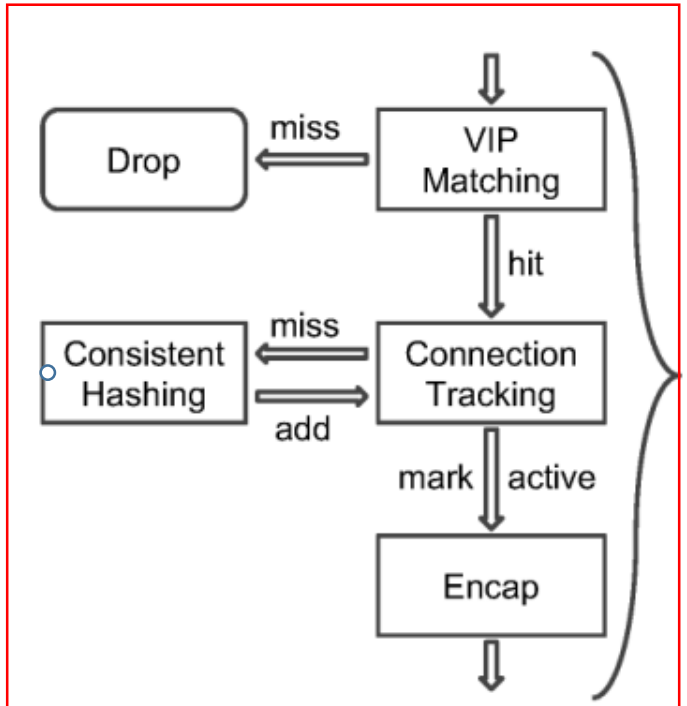
Steering module calculates the 5-tuple hash of the packets.

Why not use round-robin?

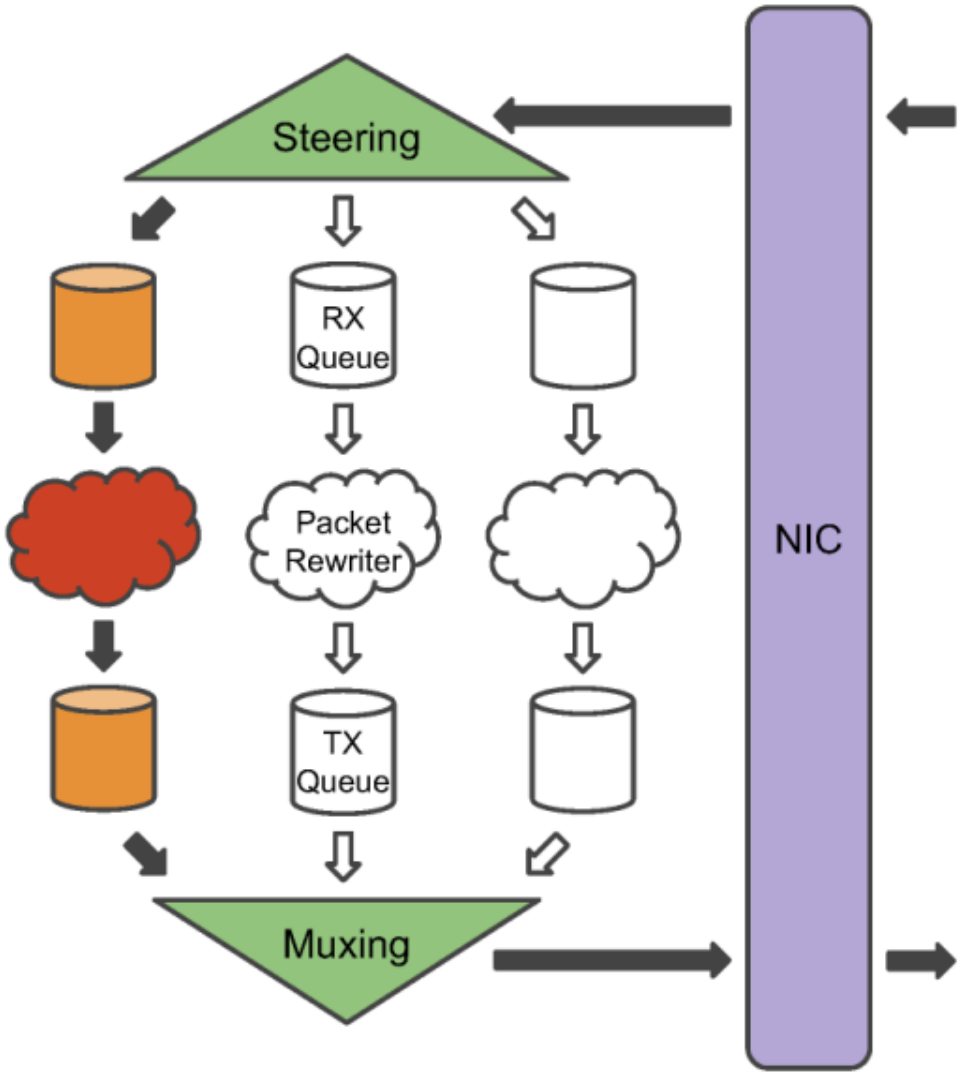
- Packet reordering
- Recalculate the hash of packets belonging to the same connection.



Why not just use hash?



Most important part of Maglev



Steady state for only hash

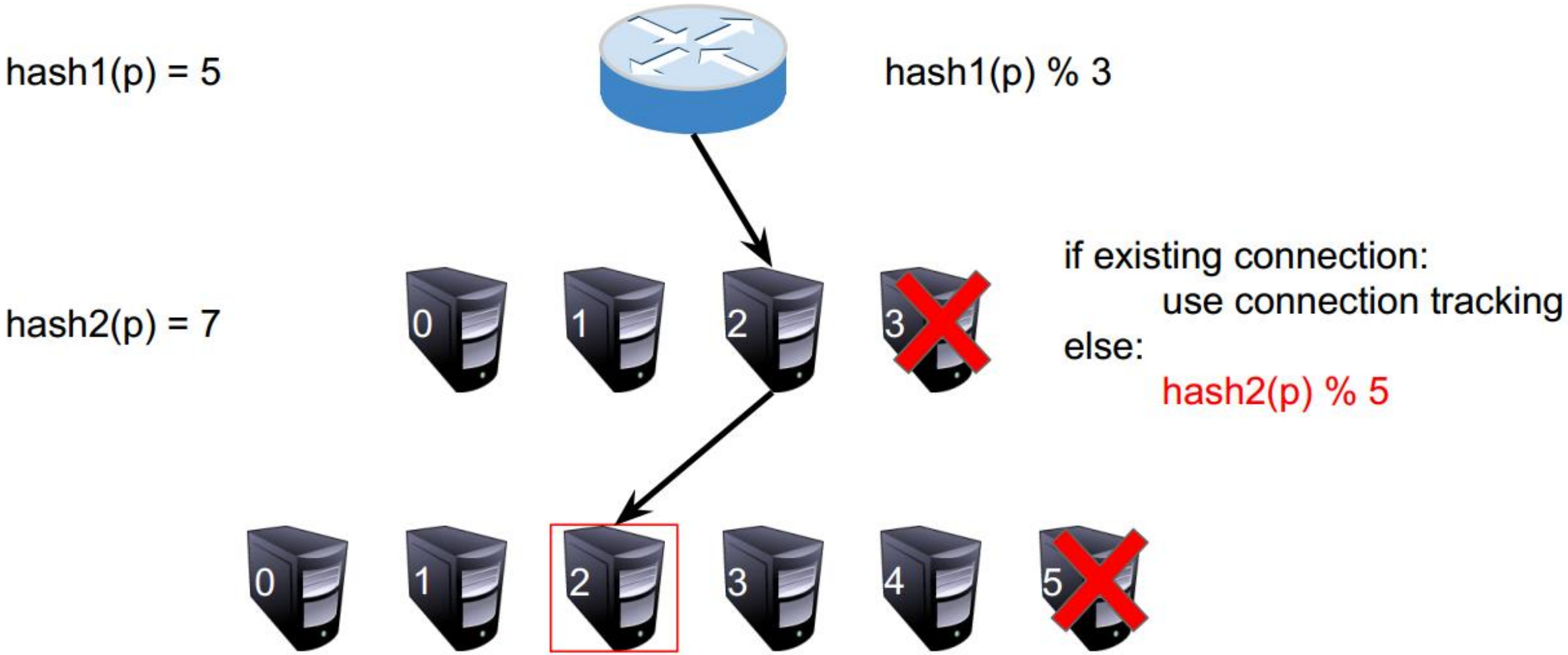
hash1(p) = 5

hash1(p) % 4

hash2(p) = 7

if existing connection:
use connection tracking
else:
hash2(p) % 6

Both of maglev and backend change for only hash



Maglev consistence hash

Assign unique names for each backend in backend pool.
Let *M* be the size of the lookup table.
Let *N* be the size of a VIP's backend pool.

offset ← *h*1(*name*[*i*]) mod *M*
skip ← *h*2(*name*[*i*]) mod (*M* - 1) + 1
permutation[*i*][*j*] ← (*offset*+ *j* × *skip*) mod *M*

Pseudocode 1 Populate Maglev hashing lookup table.

```
1:  function POPULATE
2:      for each i < N do next[i] ← 0 end for
3:      for each j < M do entry[j] ← −1 end for
4:      n ← 0
5:      while true do
6:          for each i < N do
7:              c ← permutation[i][next[i]]
8:              while entry[c] ≥ 0 do
9:                  next[i] ← next[i] + 1
10:                 c ← permutation[i][next[i]]
11:             end while
12:             entry[c] ← i
13:             next[i] ← next[i] + 1
14:             n ← n + 1
15:             if n = M then return end if
16:         end for
17:     end while
18: end function
```

M=7 N=3

	B0	B1	B2
Offset	3	0	3
Skip	4	2	1

permutation[i][j] ← (offset+ j × skip) mod M

Permutation Table

	B0	B1	B2
0	3	0	3
1	0	2	4
2	4	4	5
3	1	6	6
4	5	1	0
5	2	3	1
6	6	5	2

Permutation Table

	B0	B1	B2
0	3	0	3
1	0	2	4
2	4	4	5
3	1	6	6
4	5	1	0
5	2	3	1
6	6	5	2

Lookup Table

0	B1
1	B0
2	B1
3	B0
4	B2
5	B2
6	B0

Permutation Table

	B0	B1
0	3	0
1	0	2
2	4	4
3	1	6
4	5	1
5	2	3
6	6	5

Lookup Table

0	B1
1	B0
2	B1
3	B0
4	B0
5	B0
6	B1

Lookup Table

0	B1
1	B0
2	B1
3	B0
4	B2
5	B2
6	B0

Steady state for consistent hash

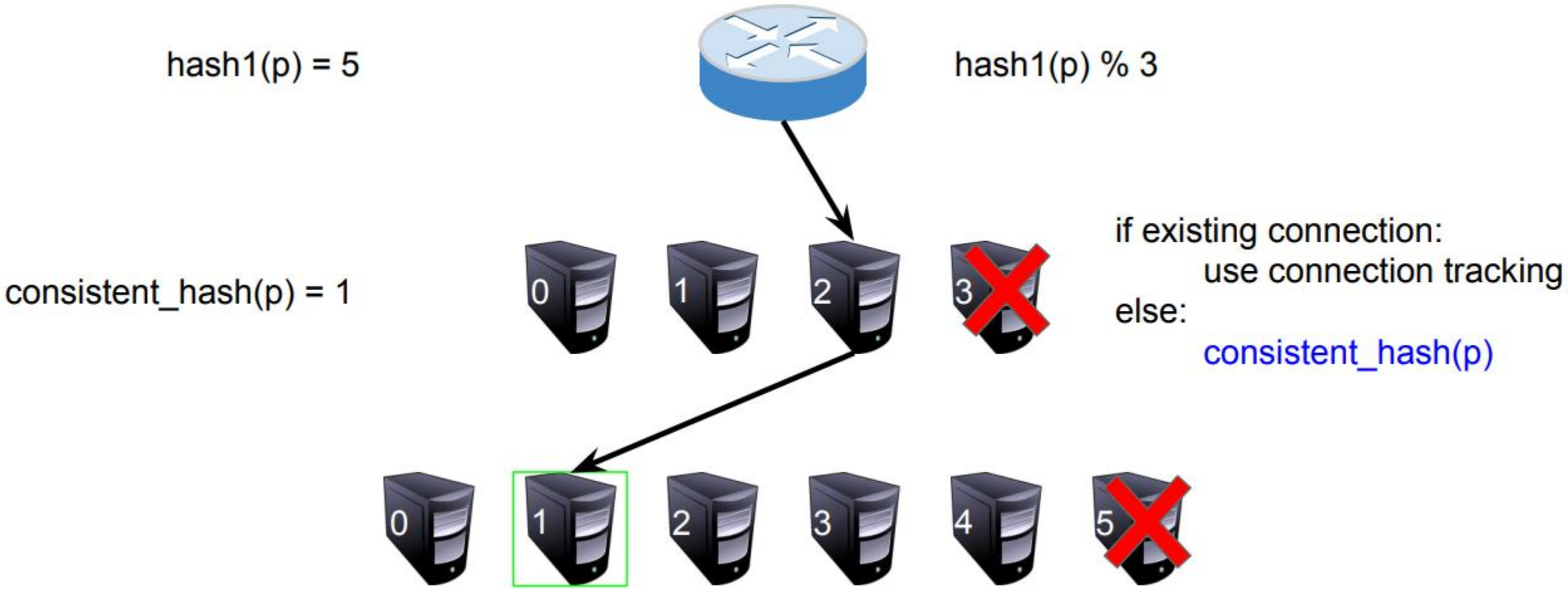
hash1(p) = 5

hash1(p) % 4

consistent_hash(p) = 1

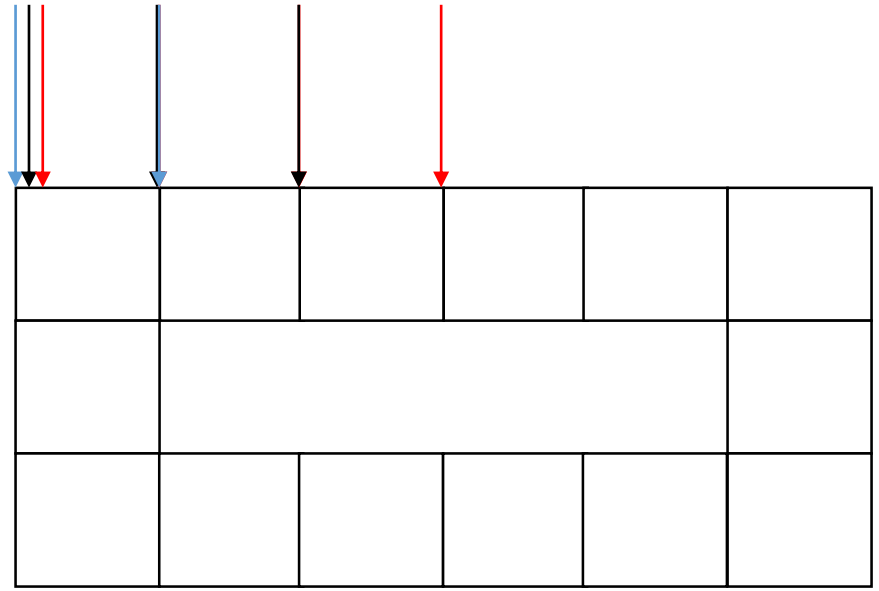
if existing connection:
    use connection tracking
else:
    consistent_hash(p)

Both of maglev and backend change for consistent hash

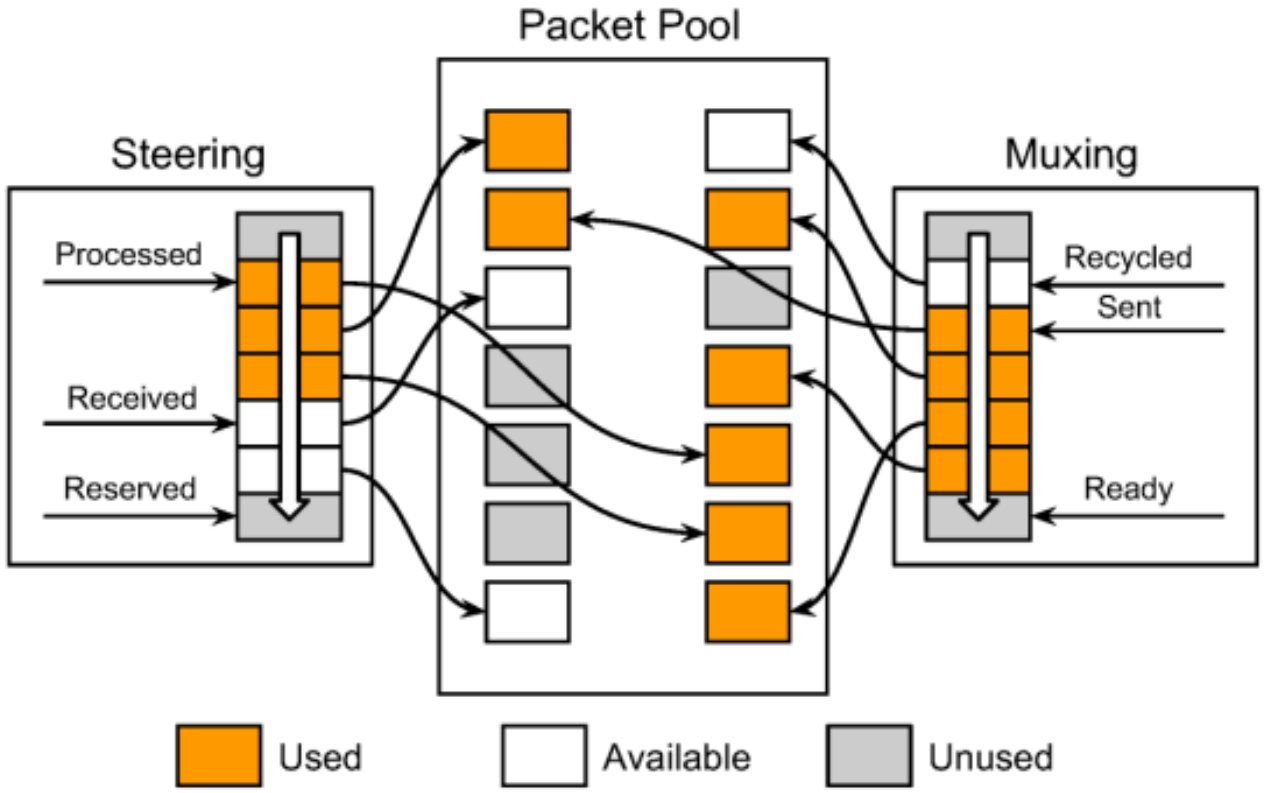


Packet fetching bypass Kernel

Both the steering and muxing modules maintain a *ring queue* of pointers pointing to packets in the packet pool.

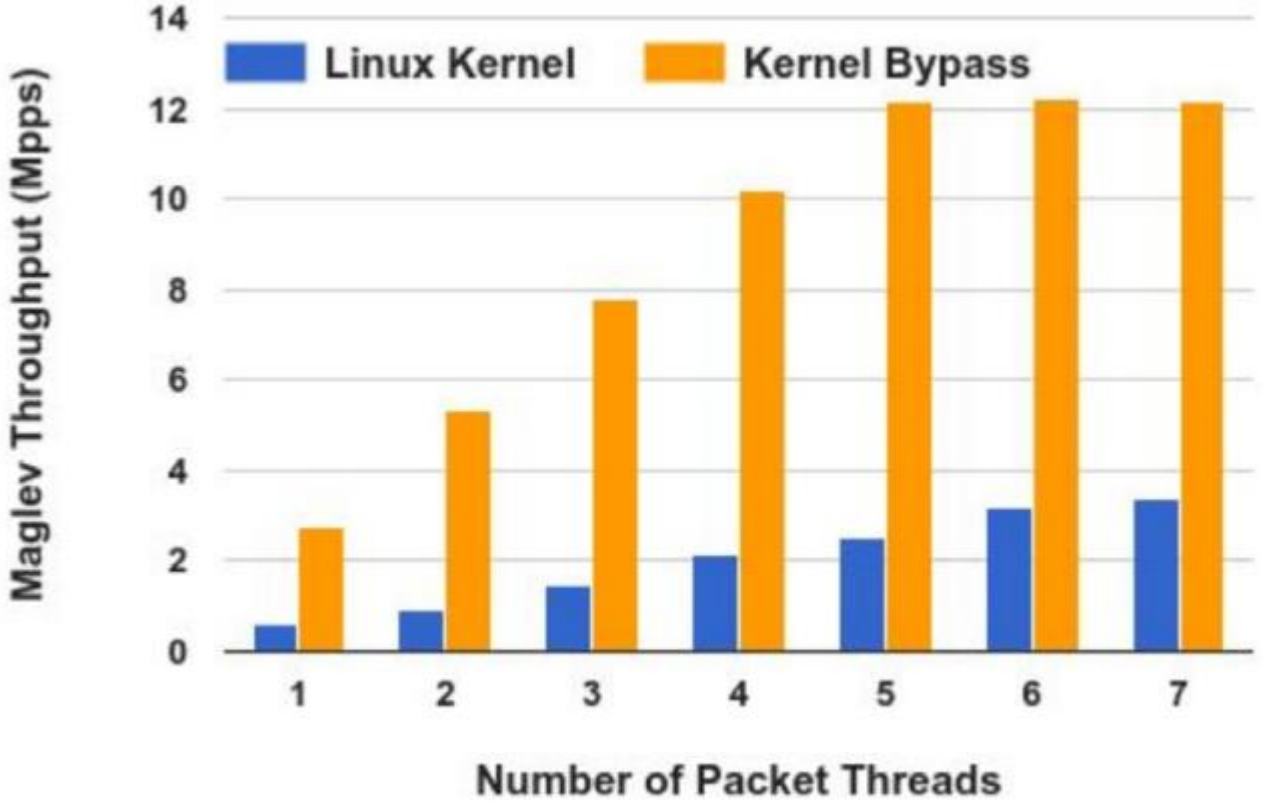


Processed
Received
Reserved



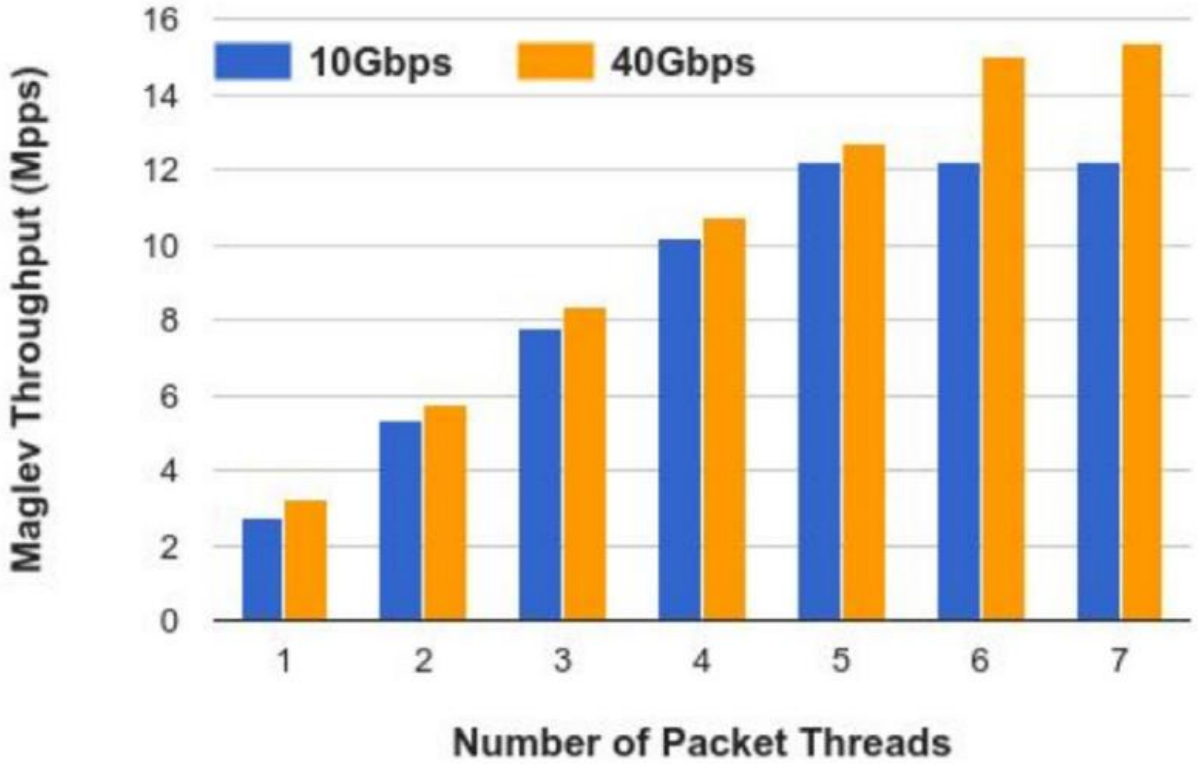
Evaluation

Throughput with and without kernel bypass



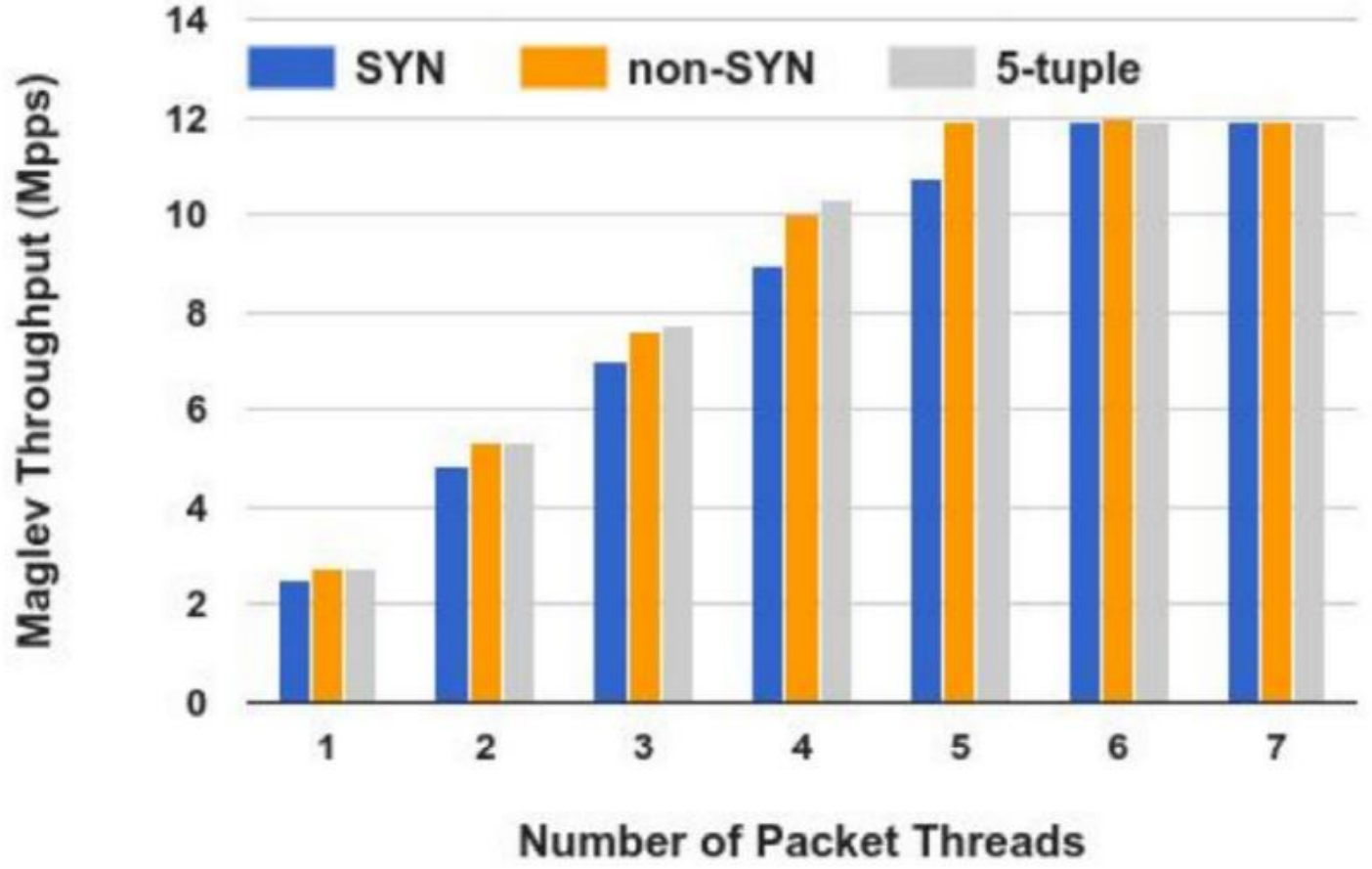
Evaluation

Throughput with different NIC speeds



Evaluation

Throughput with different TCP packet type

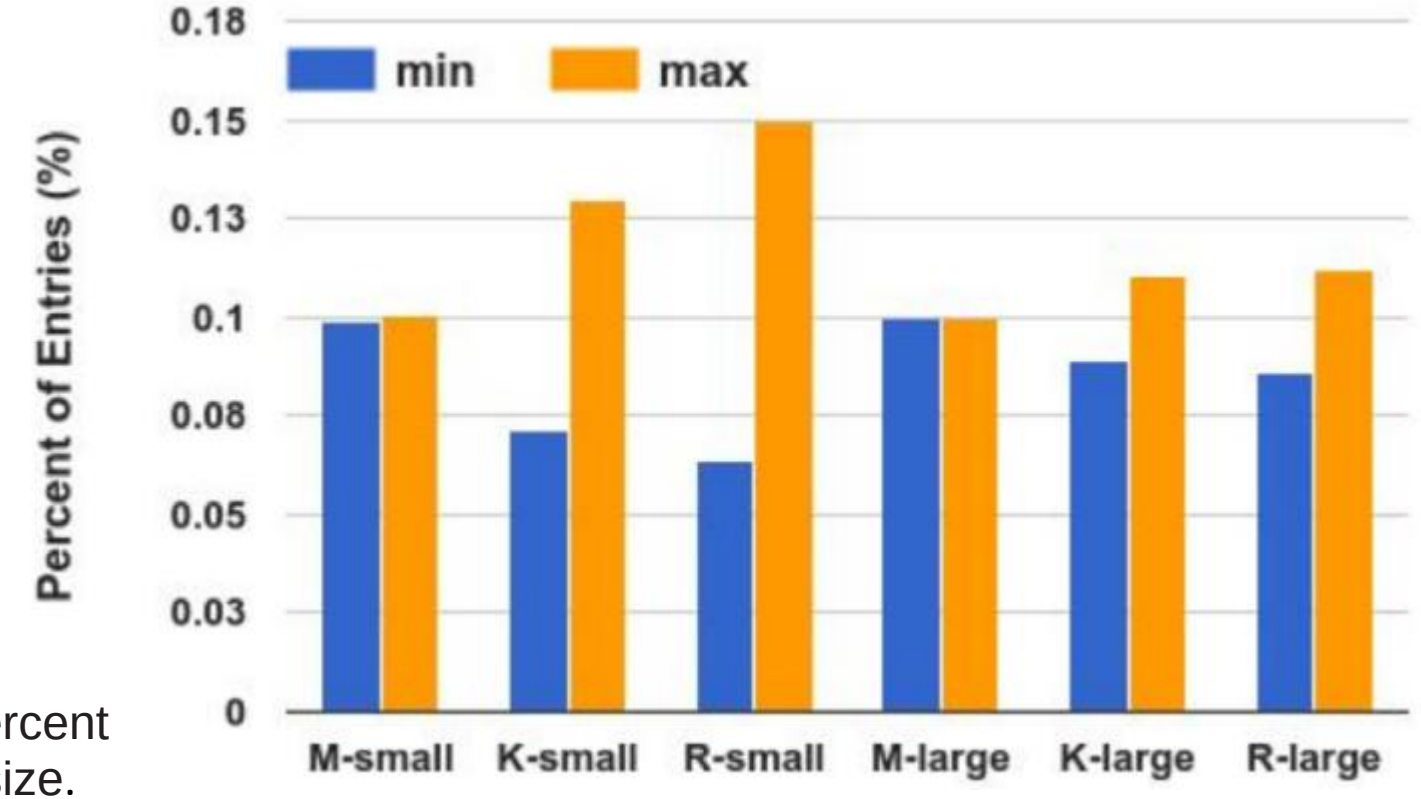


Evaluation

Load balancing efficiency of different hashing methods. M, K and R stand for Maglev, Karger and Rendezvous, respectively. Lookup table size is 65537 for *small* and 655373 for *large*.

The total number of backends to be tested is1000 and the lookup table size to be 65537 and 655373.

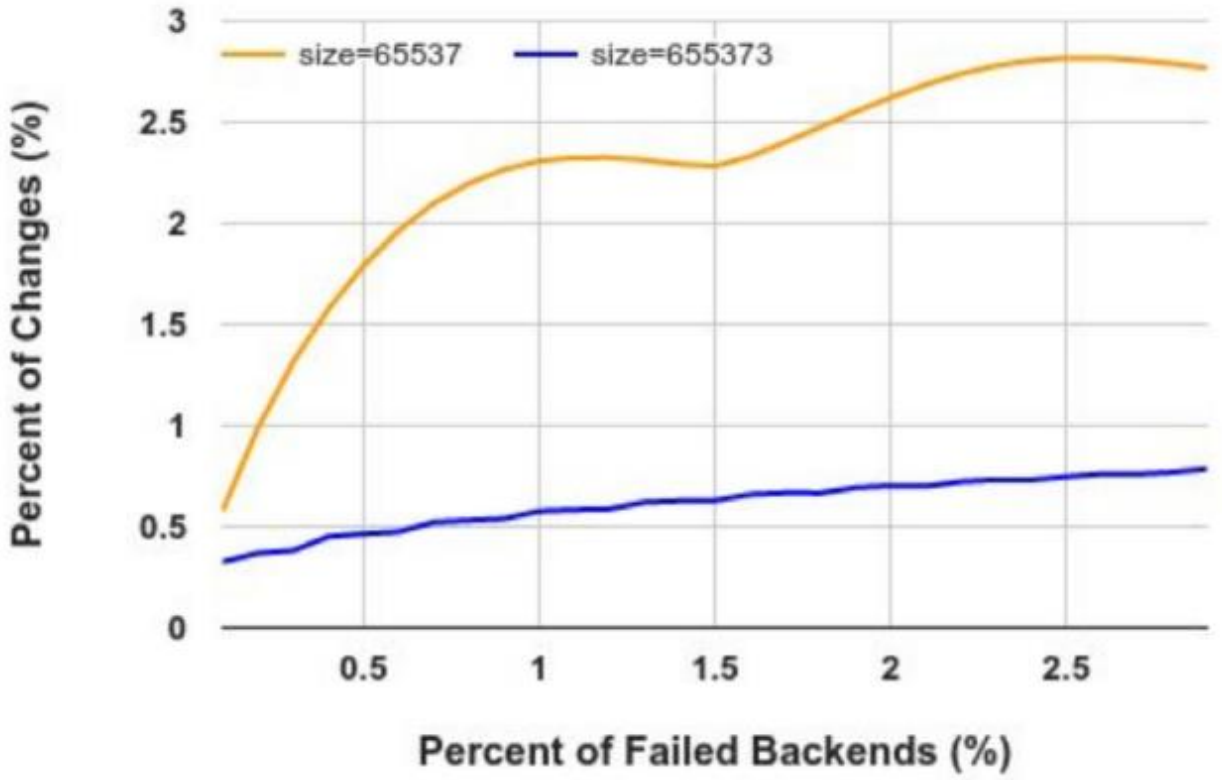
The figure presents the maximum and minimum percent of entries per backend for each method and table size.



Evaluation

Maglev hashing is more resilient to backend changes when the table size is larger.

Microbenchmarks show that the lookup table generation time increases from 1.8ms to 22.9ms as the table size grows from 65537 to 655373, which prevents Maglev from increasing the table size indefinitely.



Conclusion

Summary

- Using software load balancer to replace dedicated hardware.
- Using consistence hash to keep most connections alive when the number of maglev or service instance changes.
- Using kernel bypass technique to accelerate the speed to process packets.

What I learn

- Industrial products concern much more about reliability and stability rather than only functionality and speed.
- When doing evaluation, we can not only compare with other’s projects but also our own project with/without some specific strategies or algorithms.