Google

Maglev

A Fast and Reliable Network Load Balancer

Danielle E. Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith,
Roman Kononov, Eric Mann-Hielscher, Ardas Cilingiroglu,
Bin Cheyney, Wentao Shang, Jinnah Dylan Hosein

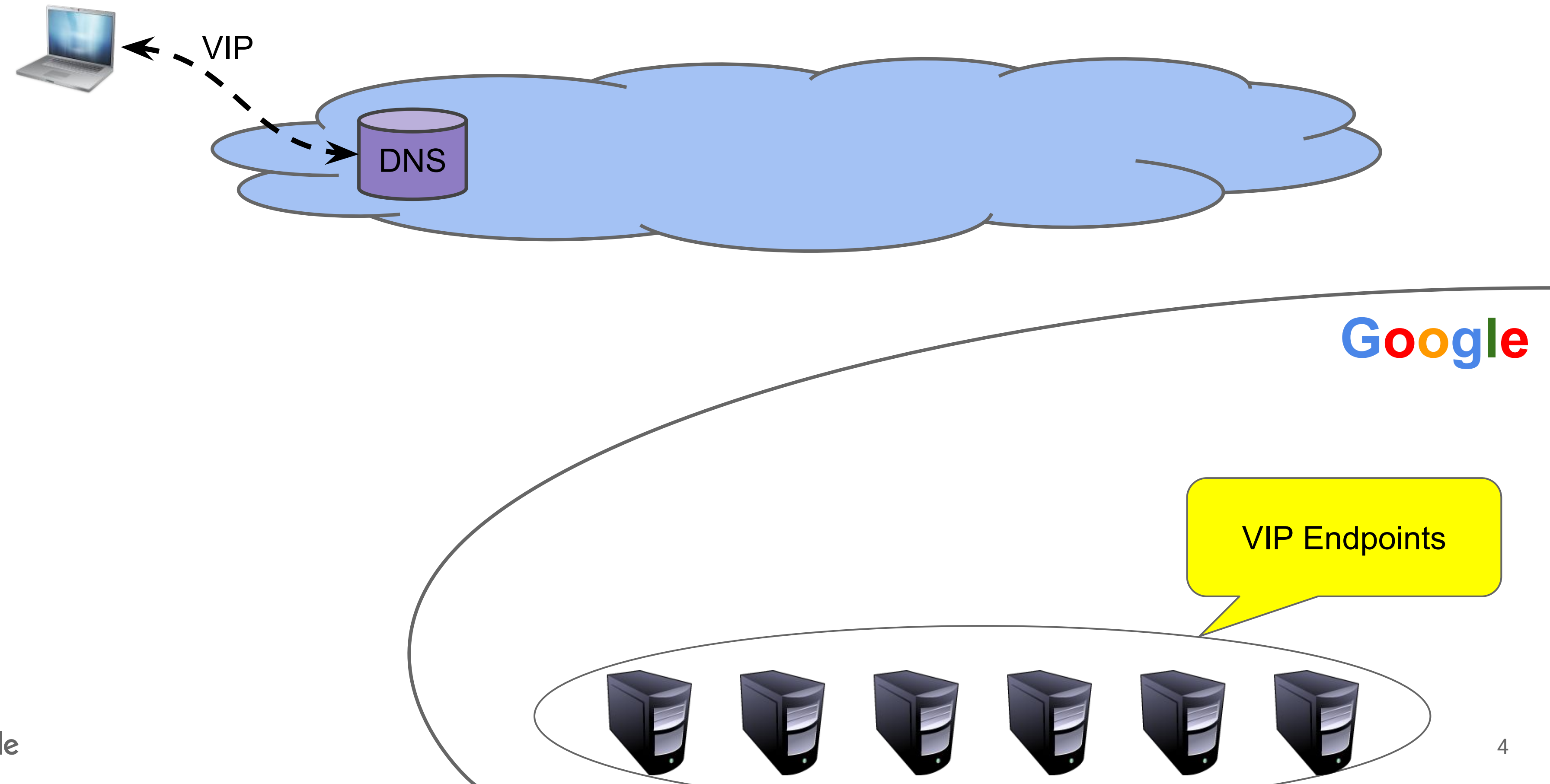
Maglev the Network Load Balancer

- What is a Network Load Balancer?
- Why Maglev?
- Maglev design
- Evaluation
- Conclusion

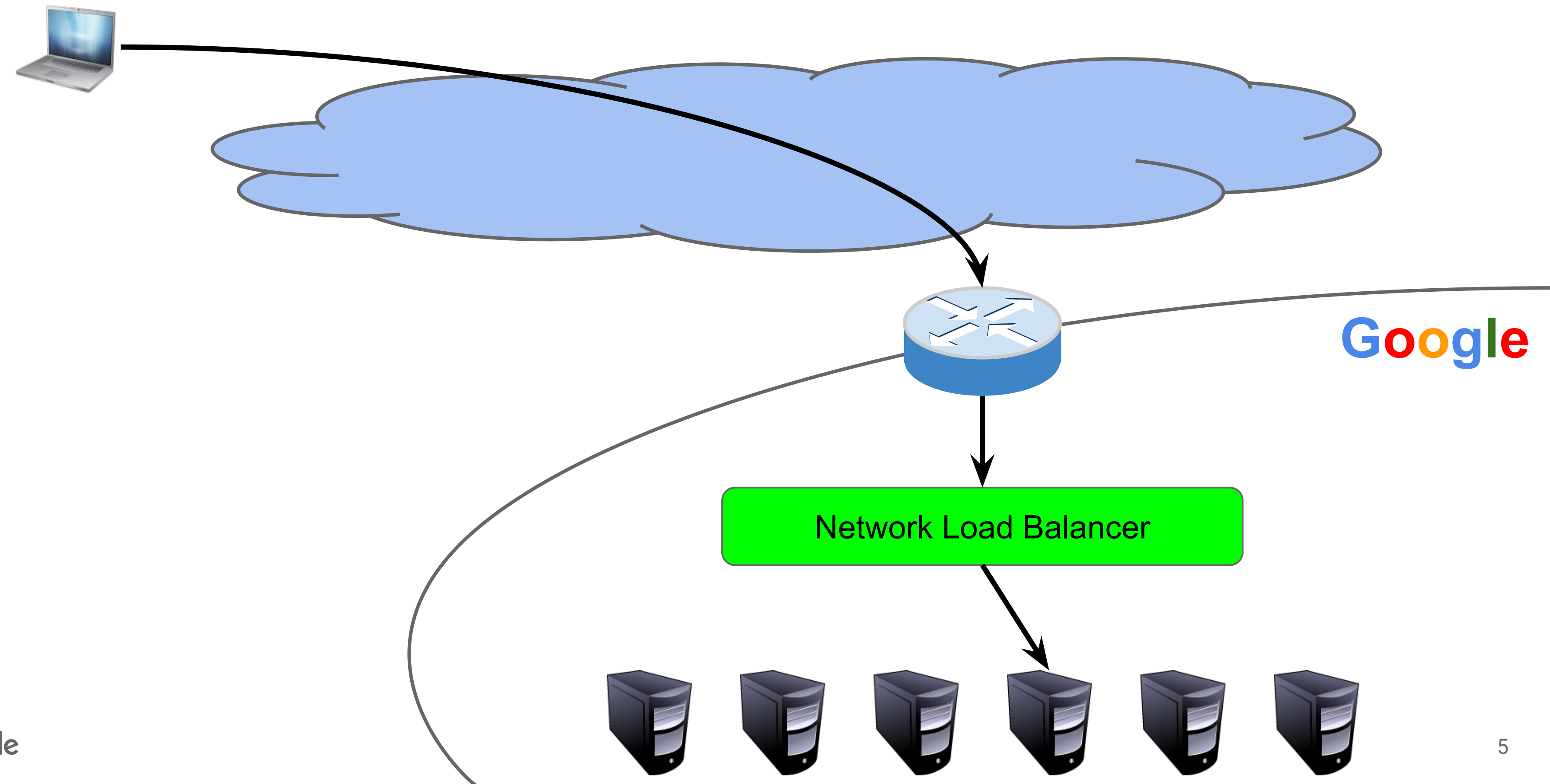
Maglev the Network Load Balancer

- What is a Network Load Balancer?
- Why Maglev?
- Maglev design
- Evaluation
- Conclusion

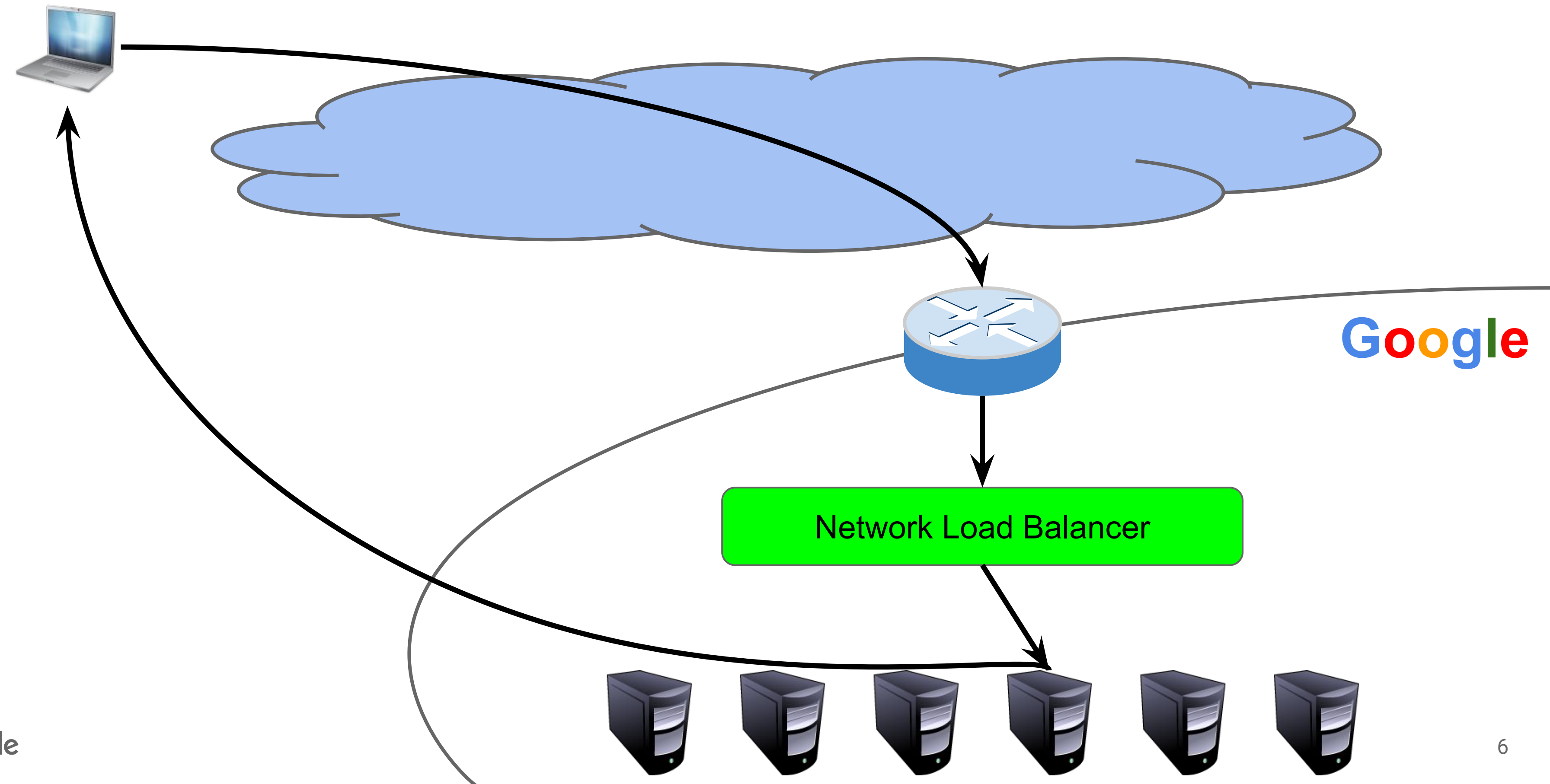
What is a network load balancer?



What is a network load balancer?



What is a network load balancer?



What do we need from a network LB?

- Balance load evenly
- Reliability: do not reset user connections
- Flexibility: iterate quickly
- Scalability: grow with cloud scale
- Efficiency: deliver high performance per dollar

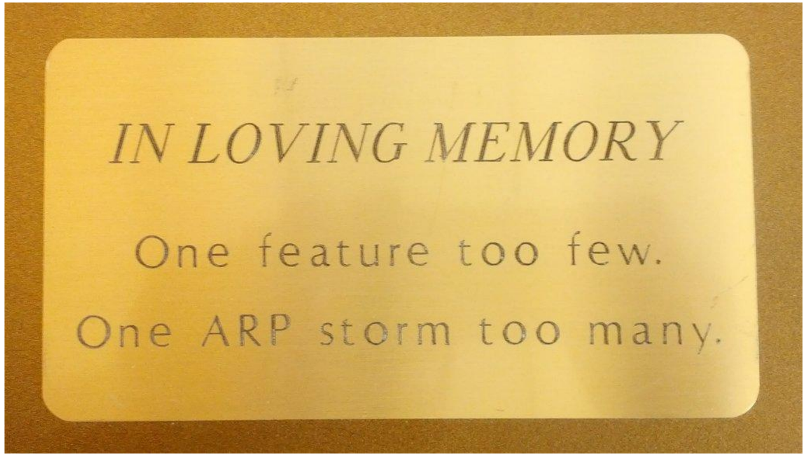
Maglev the Network Load Balancer

- What is a Network Load Balancer?
- Why Maglev?
- Maglev design
- Evaluation
- Conclusion

Google

Limitation of hardware appliances

- Poor flexibility
- Scaling is hard
- Active-passive failover
- Expensive at scale



IN LOVING MEMORY

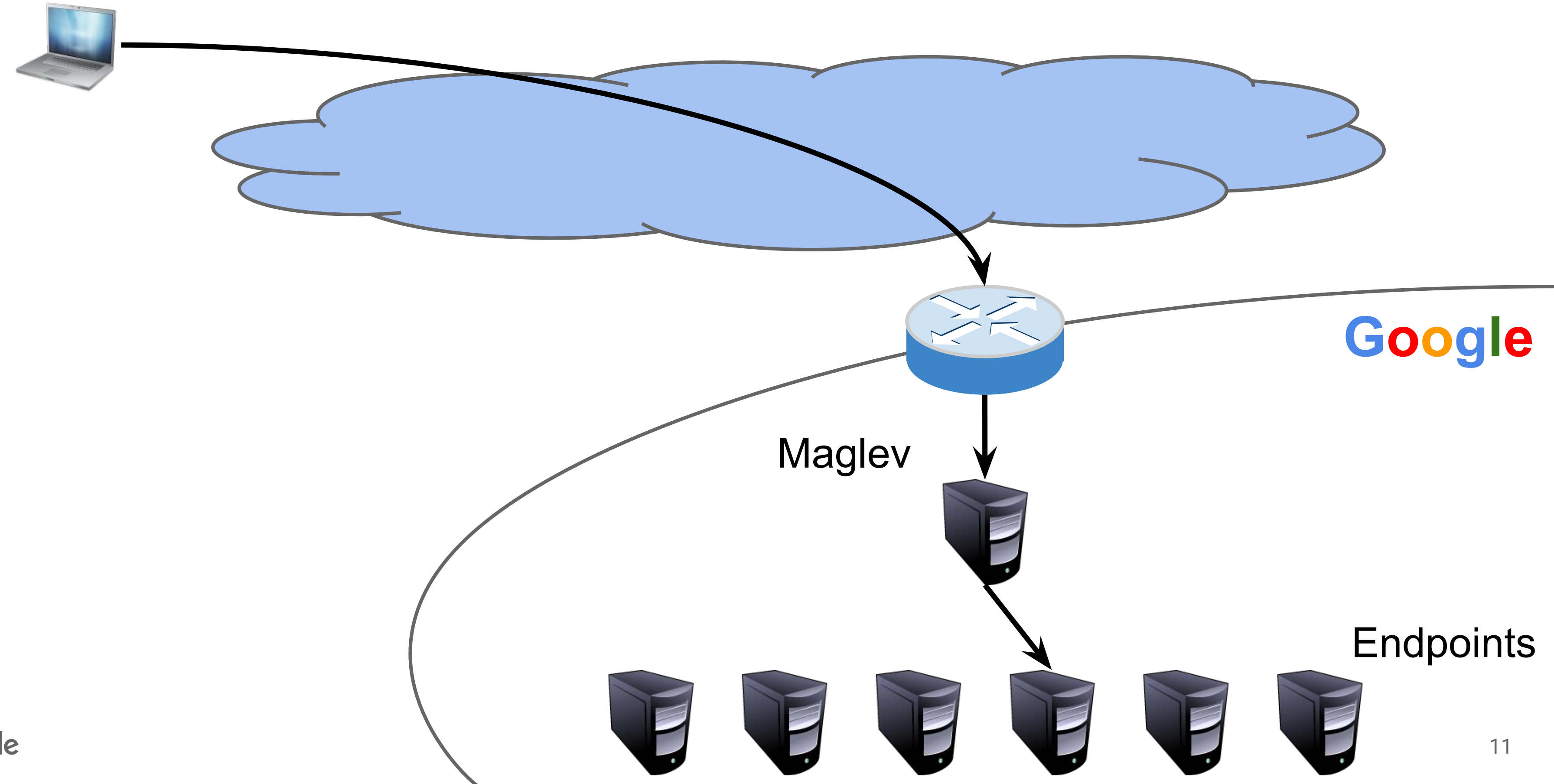
One feature too few.

One ARP storm too many.

Why Maglev?

- In 2008, hit wall with existing appliance solution
- Key insight: replace inflexible dedicated hardware
- With software running on existing servers
- Scalable deployment model
- Virtualize the network function
- Global control plane: SDN

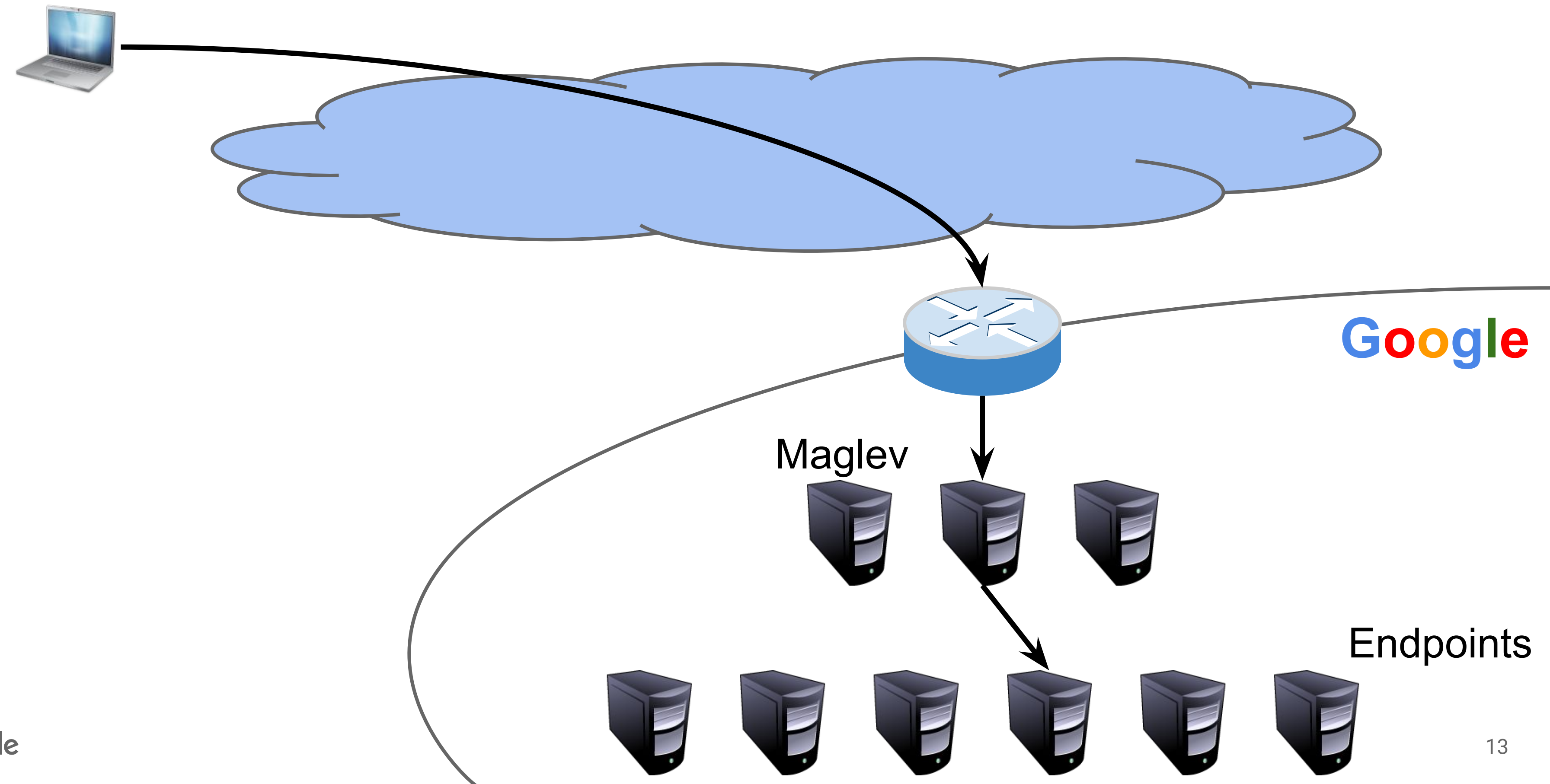
Runs on existing servers



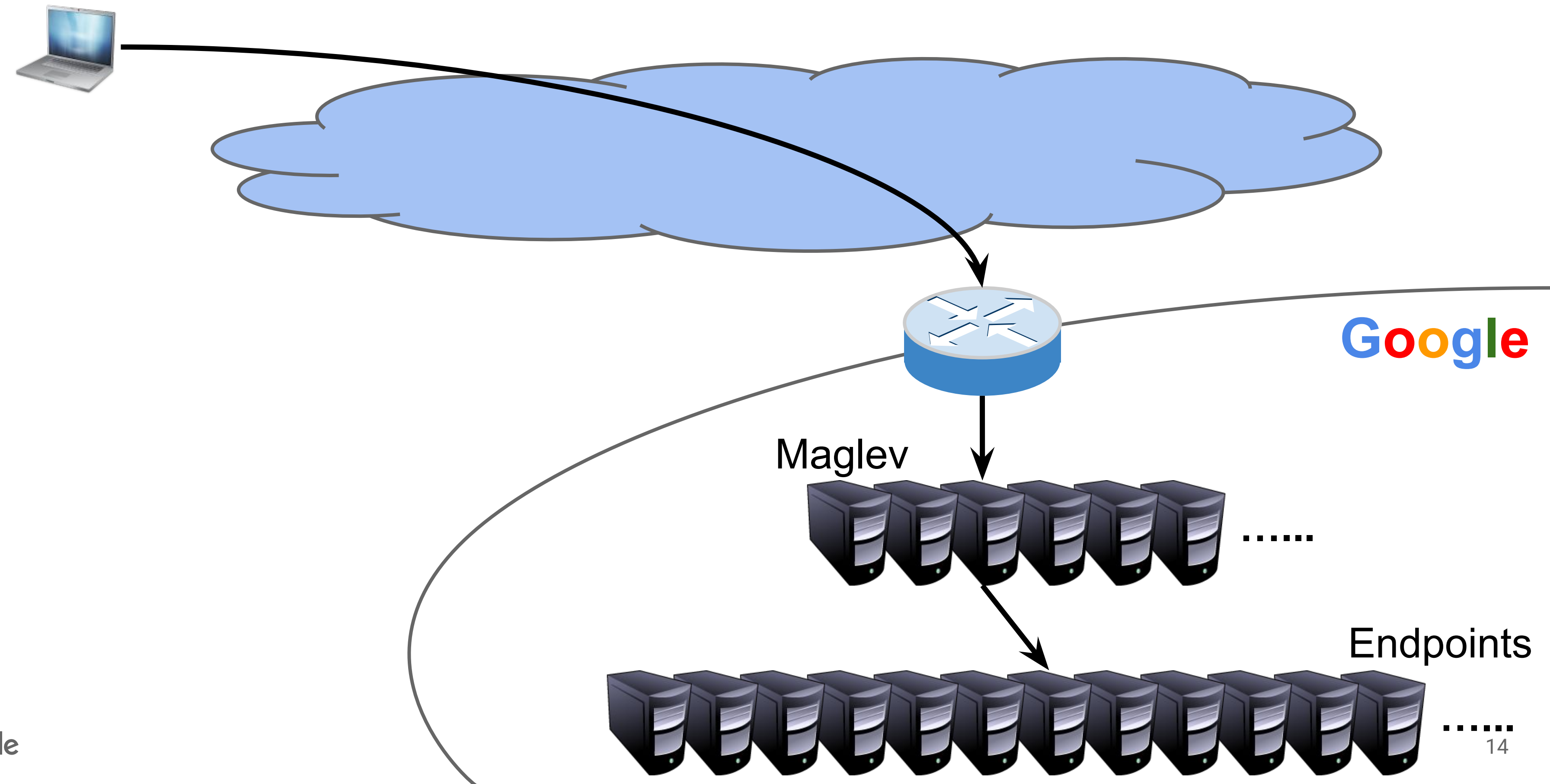
Scalability

- Huge scale in two dimensions:
- Scale out across many servers with ECMP
- Scale up to 10G line rate with kernel bypass
    - Even with very small packets; limited only by NIC
- Enables cloud-scale control plane

Scalability



Scalability



Maglev the Network Load Balancer

- What is a Network Load Balancer?
- Why Maglev?
- Maglev design
- Evaluation
- Conclusion

Maglev design challenges

- Reliability: keep connections alive
  - When set of Maglevs changes
  - When set of backends changes
  - Both at once with consistent hashing!
- Scaling
  - Scaling out with ECMP
  - Scaling up with kernel bypass

Google

Maglev design challenges

- Reliability: keep connections alive
    - When set of Maglevs changes
    - When set of backends changes
    - Both at once with consistent hashing!
- Scaling
    - Scaling out with ECMP
    - Scaling up with kernel bypass

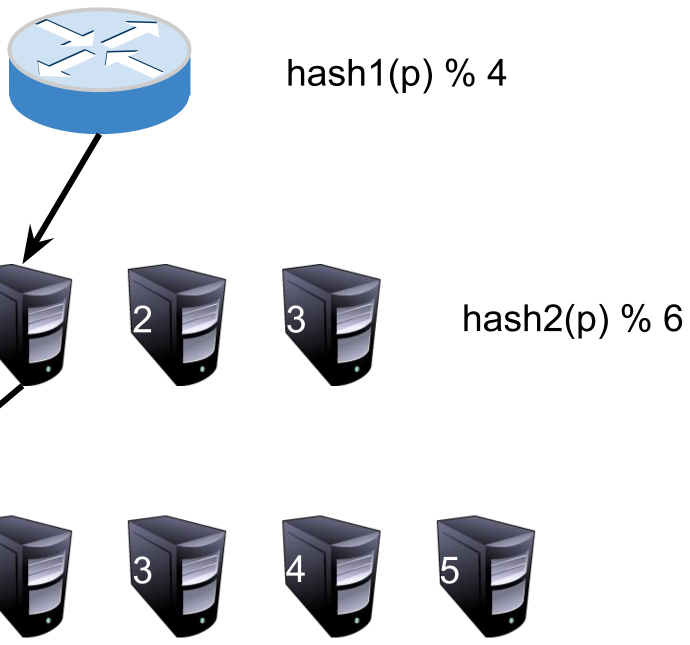
Reliability when set of Maglevs changes

- Reasons this happens
  - Health change of a Maglev
  - Adding or removing Maglev capacity
- ECMP change sends most connections to different Maglev
- Can't share connection state
- Can't do round-robin
- Hashing on 5-tuple solves the problem

Steady state

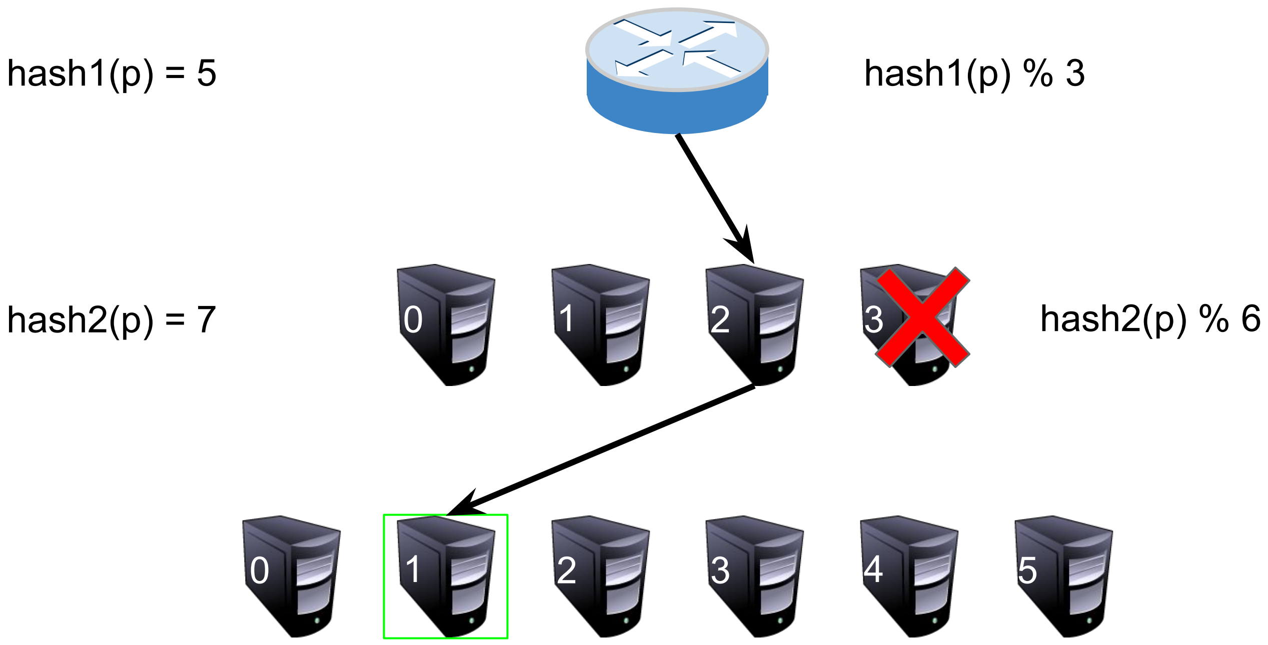
hash1(p) = 5

hash2(p) = 7



Google

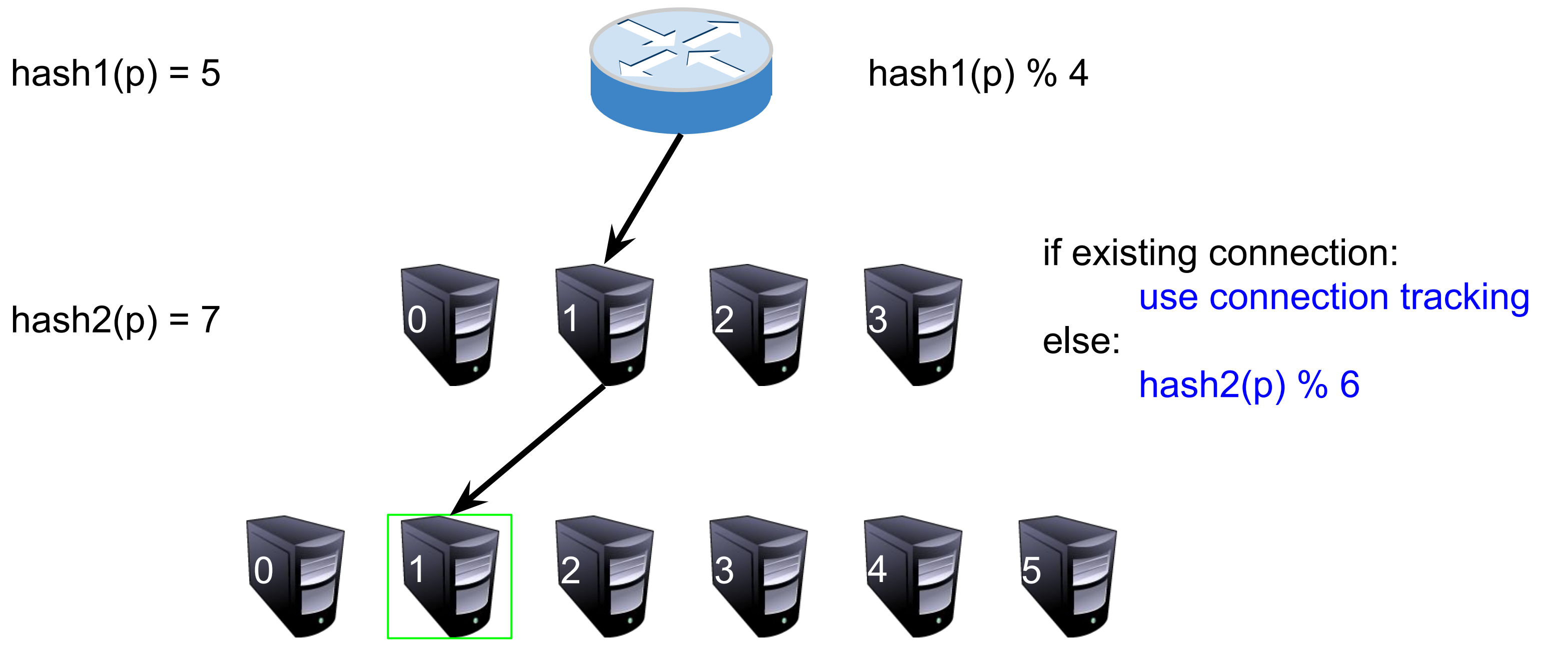
Maglev set changes



Reliability when set of backends changes

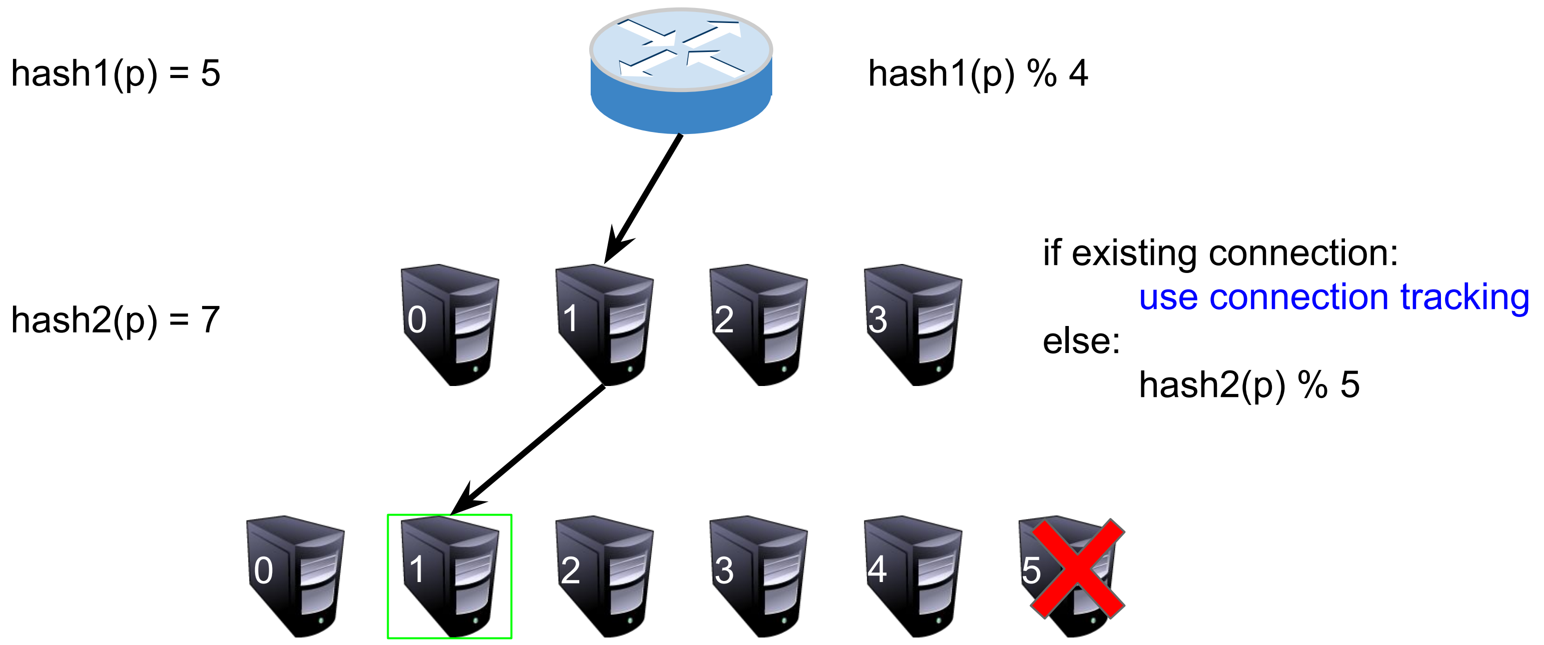
- Reasons this happens
  - Health change of a backend
  - Adding or removing backend capacity
- Hash space gets remapped
- Need to do connection tracking
  - Plenty of memory even in worst case

Steady state



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

Backend set changes



```
if existing connection:
        use connection tracking
else:
        hash2(p) % 5
```

Both at once!

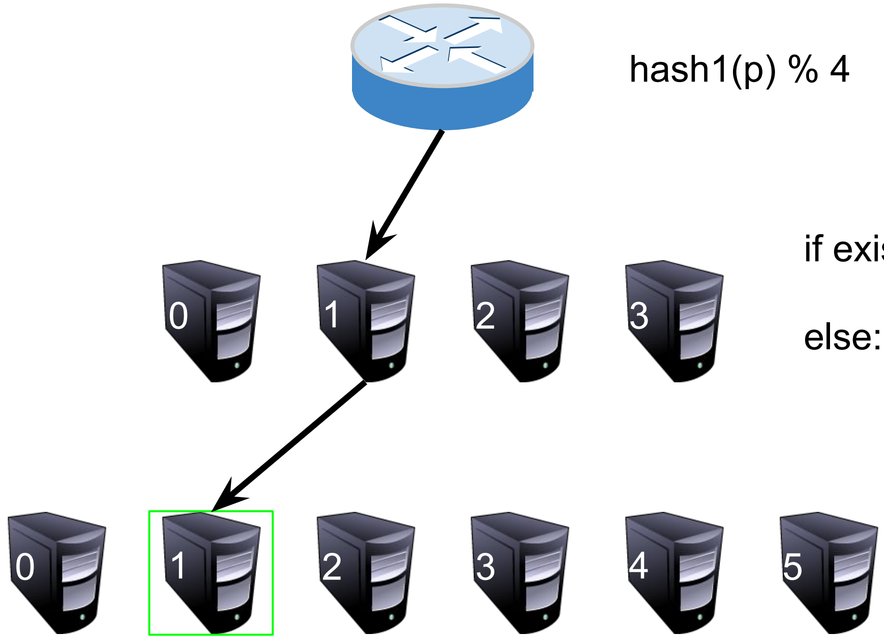
- ECMP change ruins Maglev affinity
- New Maglev does not have connection table entry
- Standard hashing: backend change ruins backend affinity
- Any backend change resets most connections

Steady state

hash1(p) = 5

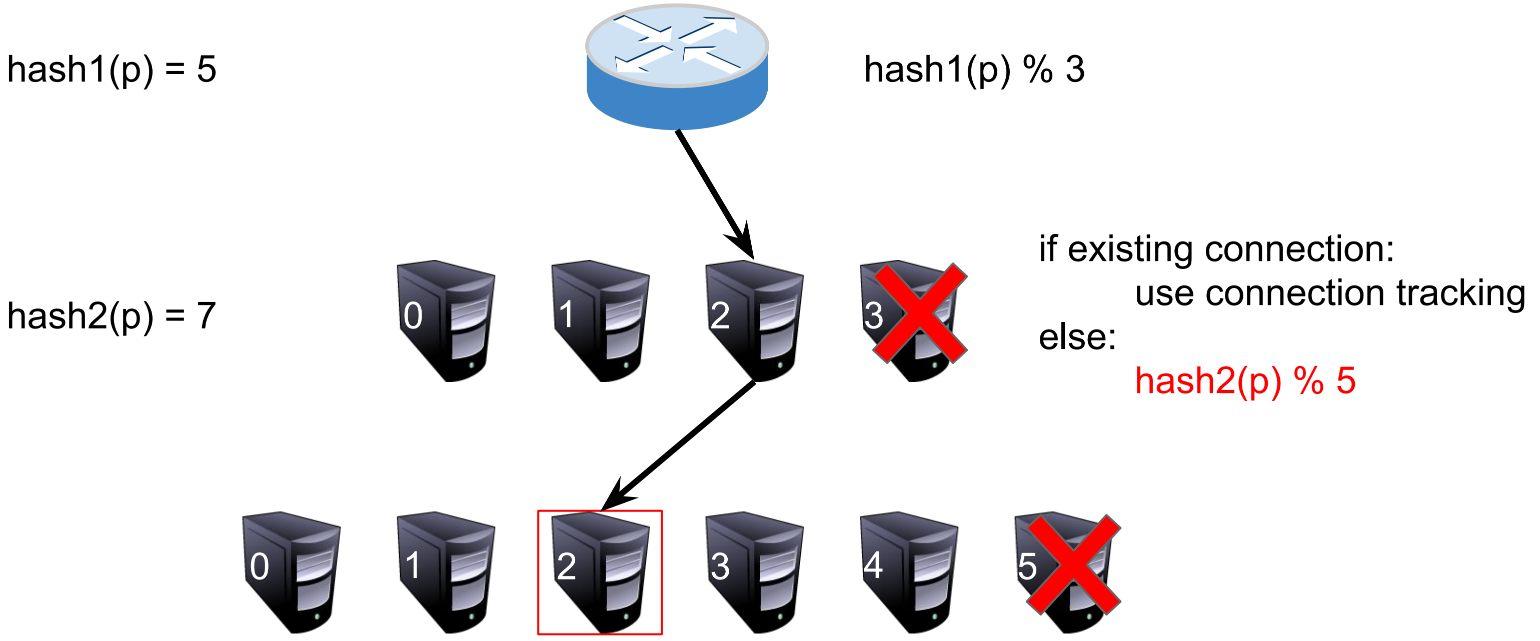
hash1(p) % 4

hash2(p) = 7



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

Everything changes

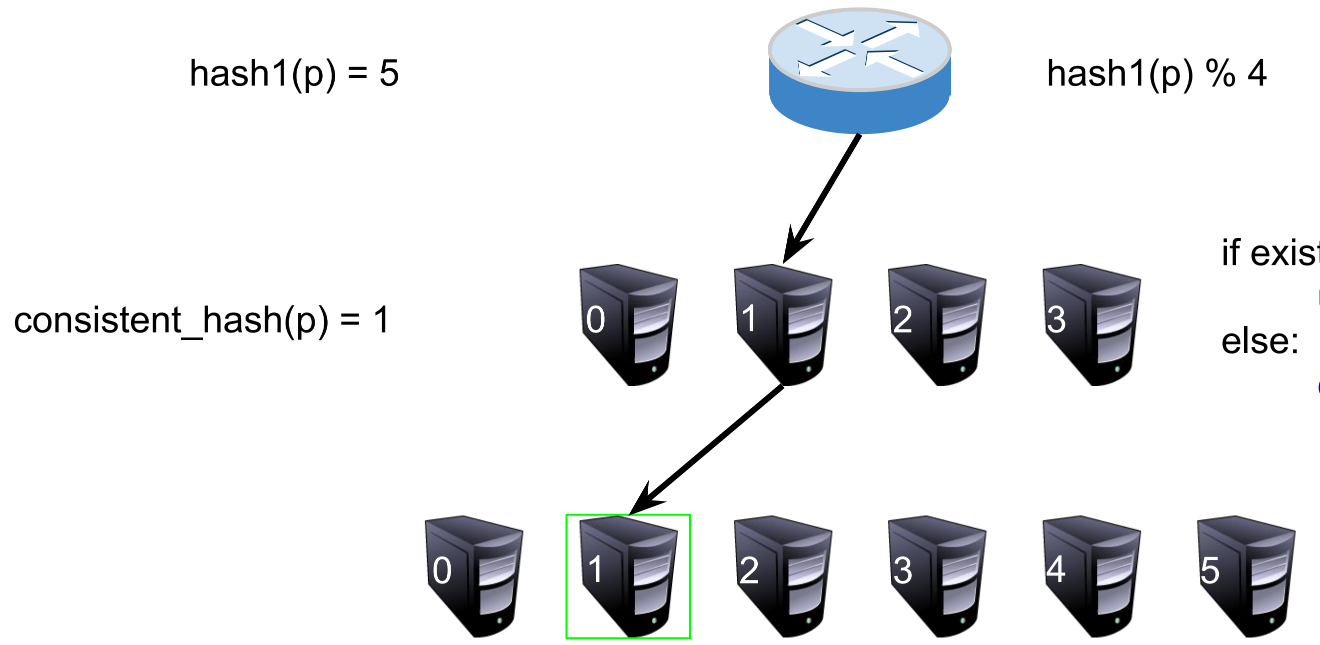


```
if existing connection:
        use connection tracking
else:
        hash2(p) % 5
```

Consistent hashing

- Consistent hashing is the answer
- Given similar inputs, will produce similar assignments
- Does not depend on backend history
- ECMP change will not cause many resets
    - Even with minor (routine) backend changes

Steady state

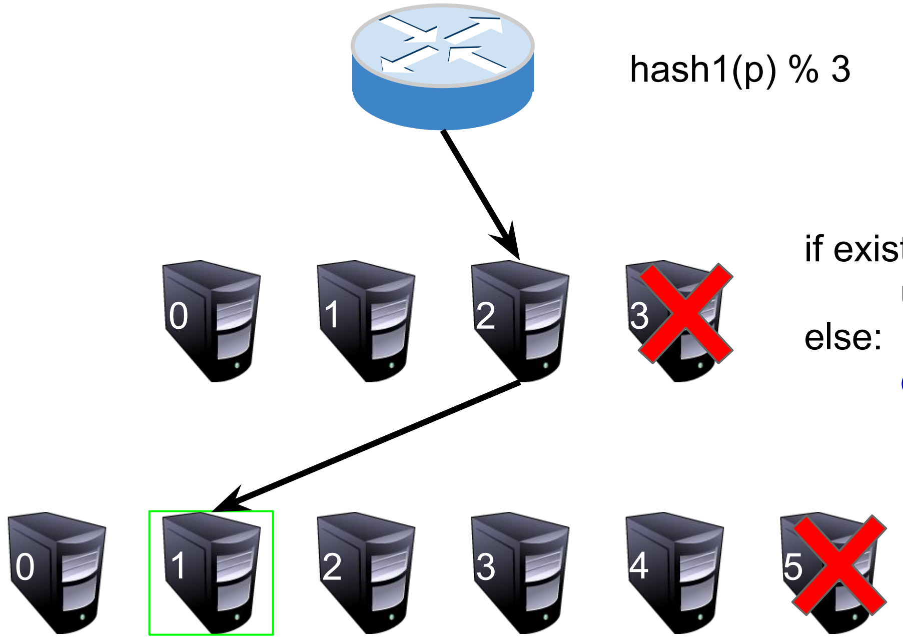


```
if existing connection:
        use connection tracking
else:
        consistent_hash(p)
```

Saved by consistent hashing

hash1(p) = 5

consistent_hash(p) = 1



hash1(p) % 3

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

Operational wins of consistent hashing

- Need to be able to upgrade Maglev binary
    - With consistent hashing, we can just do a rolling restart
    - No need to DNS drain traffic first
    - If a backend flaps during this, minimal impact

Consistent hashing algorithms

- Two good algorithms from '90s
- Work well with small backend sets
- With large backend sets (~1000), require huge tables
- So we invented our own
- Trades off a little consistency for very even balance

Maglev Consistent Hashing

- Hash every backend to preference list of table positions
- Prime table size P for easy computation
- Hash every backend to (offset, skip) $\in [0, P-1] \times [1, P-1]$
- Each backend's i'th preference is (offset + i × skip) mod P
- Backends take turns claiming most-preferred empty bucket

Consistent hashing example

	B0	B1	B2
Offset	3	0	3
Skip	4	2	1

permutation[i] = (offset + i * skip) % 7

Permutation Table

	B0	B1	B2
0			
1			
2			
3			
4			
5			
6			

Consistent hashing example

	B0	B1	B2
Offset	3	0	3
Skip	4	2	1

permutation[i] = (offset + i * skip) % 7

Permutation Table

	B0	B1	B2
0	3		
1			
2			
3			
4			
5			
6			

Consistent hashing example

	B0	B1	B2
Offset	3	0	3
Skip	4	2	1

permutation[i] = (offset + i * skip) % 7

Permutation Table

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

Consistent hashing example

	B0	B1	B2
Offset	3	0	3
Skip	4	2	1

permutation[i] = (offset + i * skip) % 7

Permutation Table

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

Consistent hashing example

	B0	B1	B2
Offset	3	0	3
Skip	4	2	1

permutation[i] = (offset + i * skip) % 7

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

Consistent hashing example

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	
1	
2	
3	
4	
5	
6	

Consistent hashing example

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	
1	
2	
3	B0
4	
5	
6	

Consistent hashing example

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	
2	
3	B0
4	
5	
6	

Consistent hashing example

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	
2	
3	B0
4	
5	
6	

Google

Consistent hashing example

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	
2	
3	B0
4	B2
5	
6	

Consistent hashing example

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	
2	
3	B0
4	B2
5	
6	

Consistent hashing example

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	
2	
3	B0
4	B2
5	
6	

Consistent hashing example

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	
3	B0
4	B2
5	
6	

Consistent hashing example

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

Consistent hashing example

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

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

Google

Maglev design challenges

- Reliability: keep connections alive
    - When set of Maglevs changes
    - When set of backends changes
    - Both at once with consistent hashing!
- Scaling
    - Scaling out with ECMP
    - Scaling up with kernel bypass

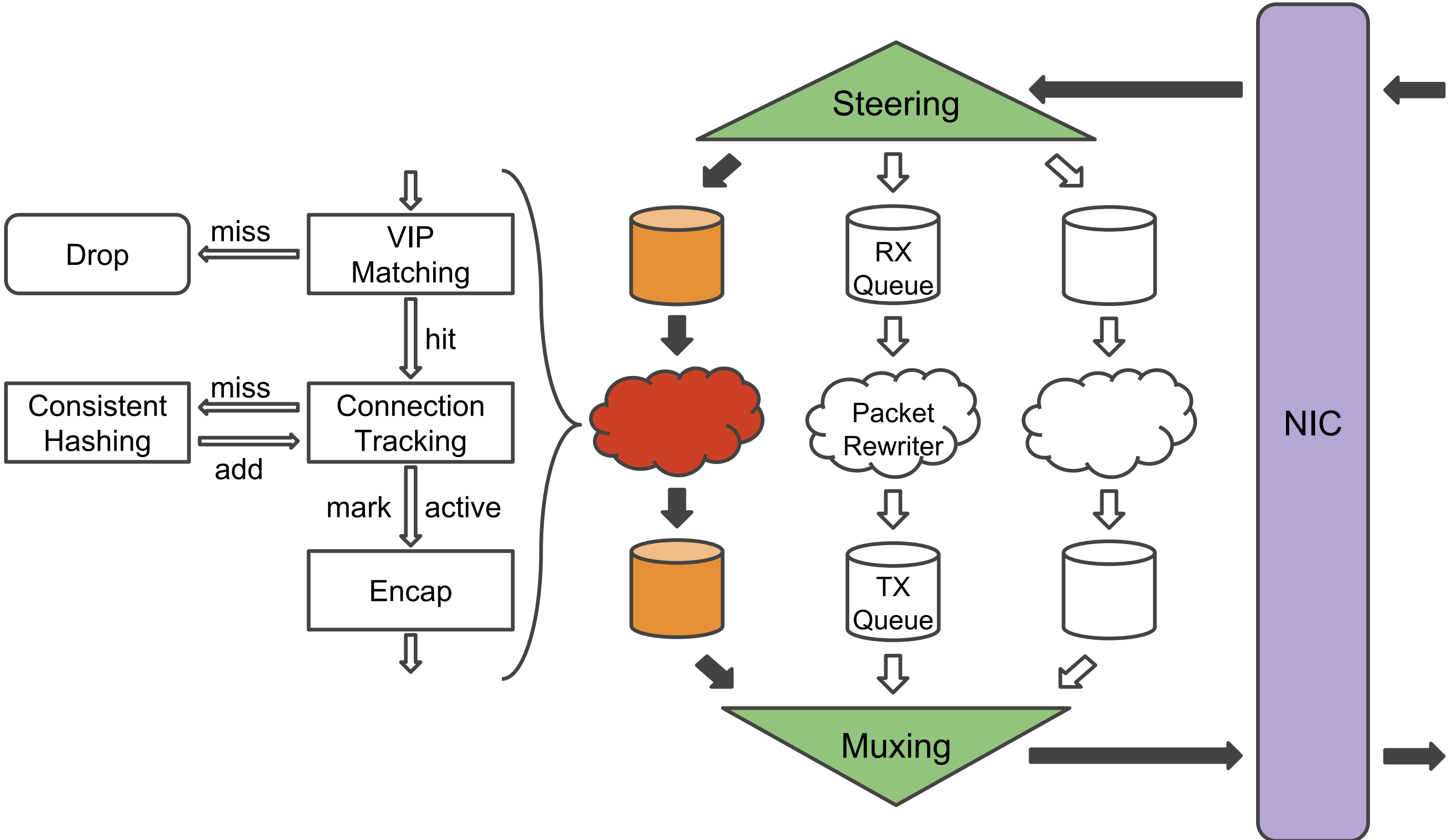
Scaling out with ECMP

- Use SDN switches with 256-way L3 ECMP
- Consistent hashing above makes for easy maintenance

Scale up with Kernel Bypass

- Linux kernel was a bottleneck
- Each machine needs to be fast for Maglev to be cheap
- Send/receive packets directly between user space and NIC
- Can go at 10G line rate
- Hashes packets across multiple queues
- Round robin overflow if queue fills up

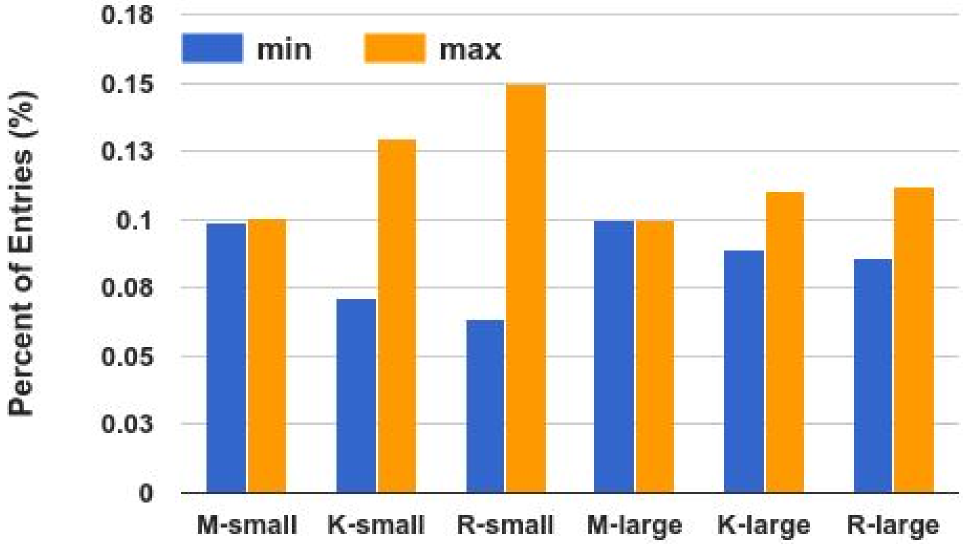
Bringing it all together



Maglev the Network Load Balancer

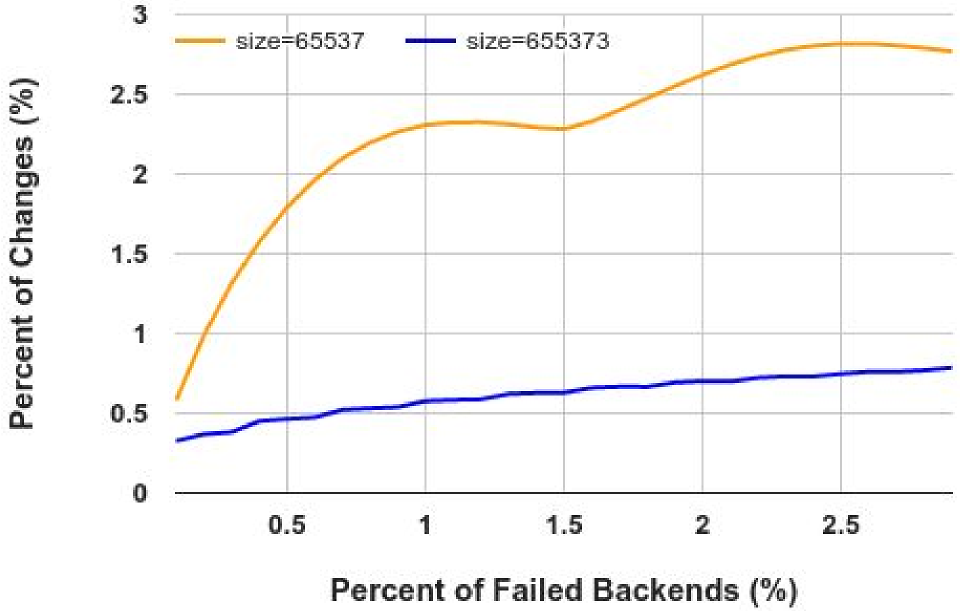
- What is a Network Load Balancer?
- Why Maglev?
- Maglev design
- Evaluation
- Conclusion

Consistent hashing evenness

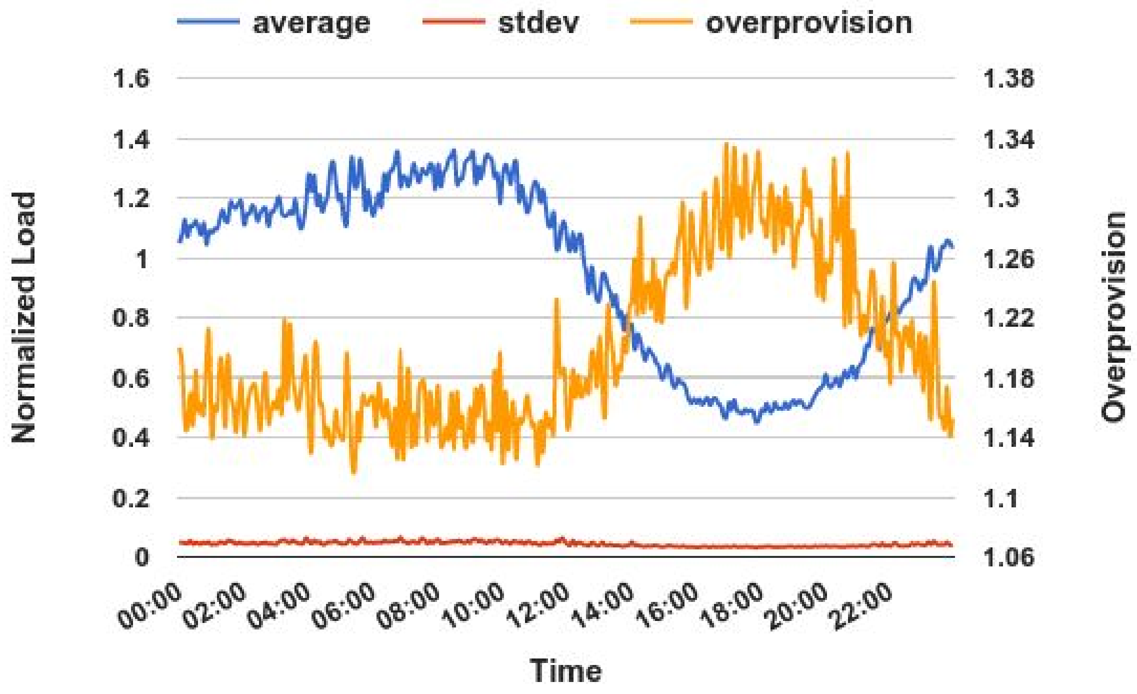


Google

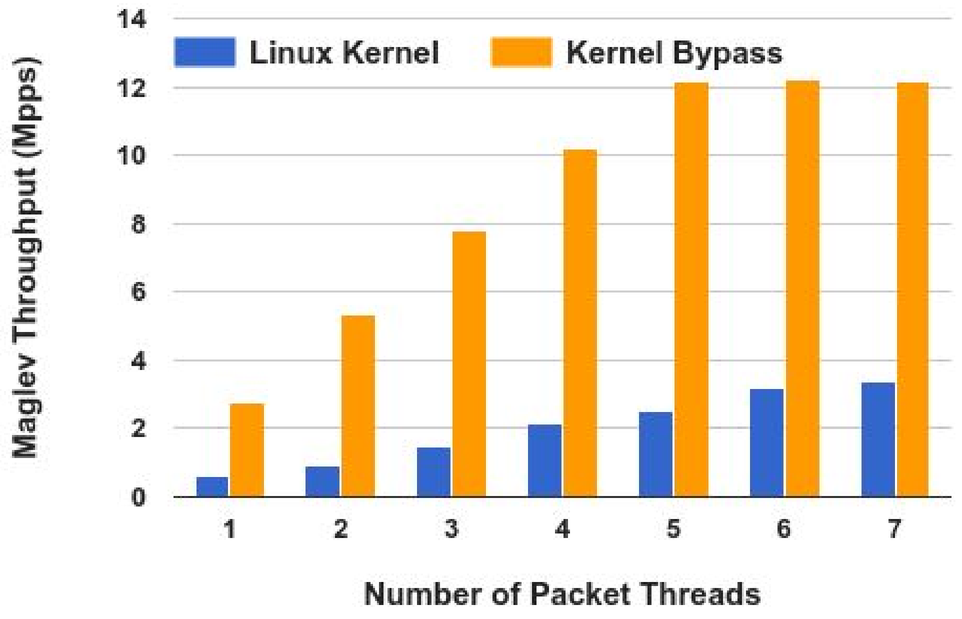
Consistent hashing consistency



Load balancing



Kernel bypass performance



Maglev the Network Load Balancer

- What is a Network Load Balancer?
- Why Maglev?
- Maglev design
- Evaluation
- Conclusion

Conclusion

- Maglev is a fast and reliable network load balancer
- ECMP, connection tracking, and consistent hashing combine to scale out reliably
- Kernel bypass gives performance needed to make software network LB economical
- Software is a good place for stateful network functions