# Scalable in-network caching for Kubernetes

STEFANOS SAGKRIOTIS

NETLAB
NETWORKED SYSTEMS RESEARCH LABORATORY
University of Glasgow | School of Computing Science

2-page extended abstract accepted @ SIGCOMM 2022, 22-26 August, Amsterdam, Netherlands.

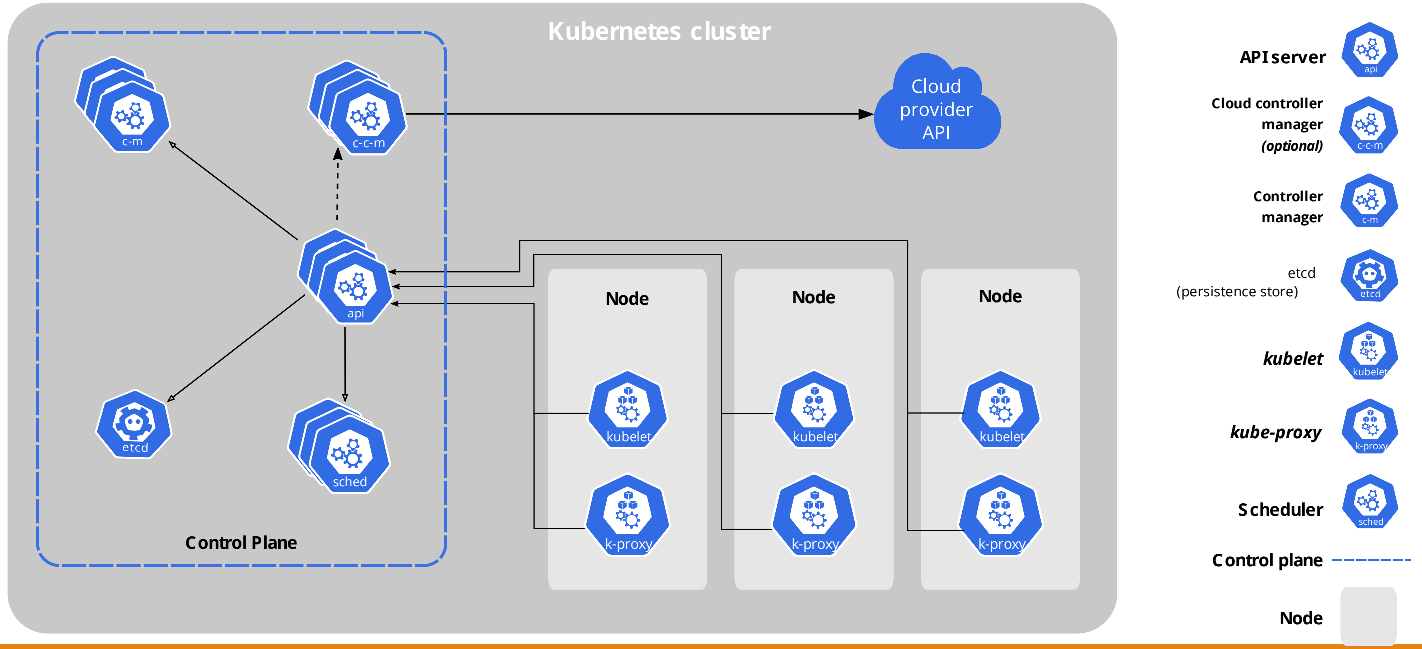
# kubernetes

Its architecture:

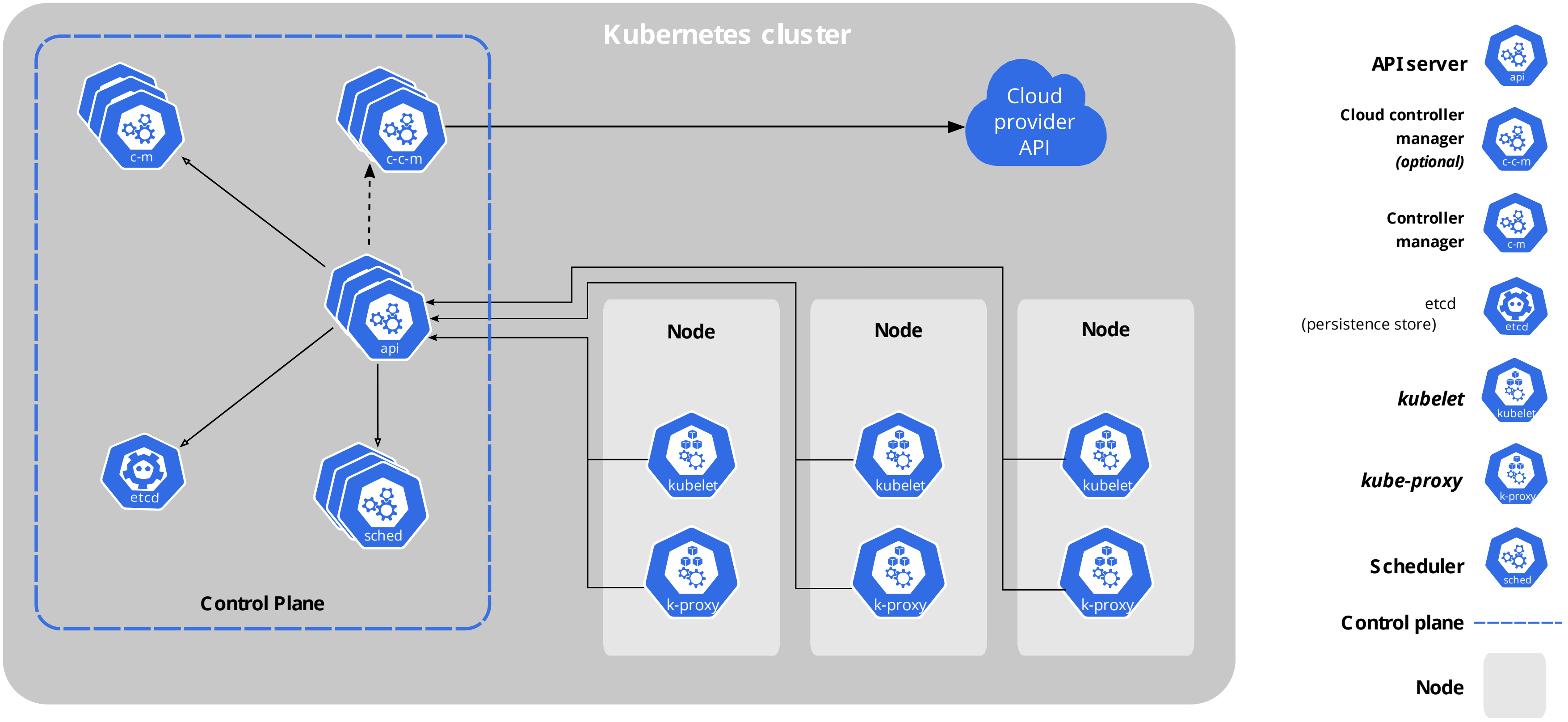
❖ Drove transition to era of microservices.

❖ Enabled scalable deployment in heterogeneous infrastructure.

❖ Allowed extensible components.



# Main components



# Zooming in

Etcd: coordination services for the control plane – through the API.

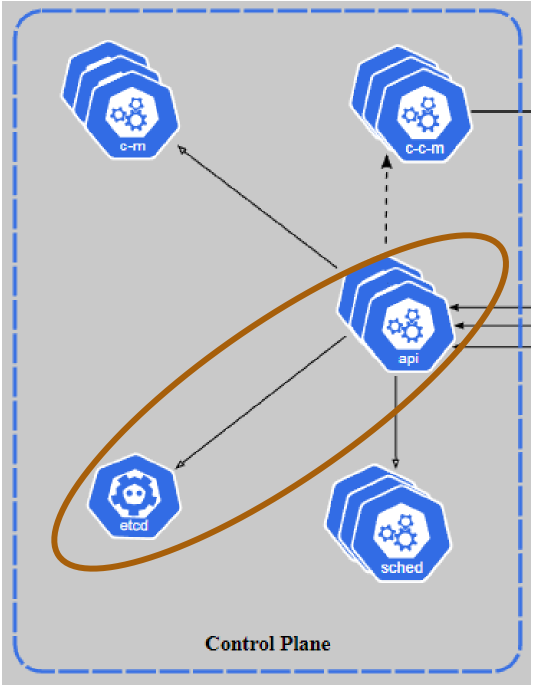
Etcd metrics:
nginx deployment scale: 2 – 10 – 15 – 25 pods

- ~15.3k reads
- 55% of all reads are directed to 25 KV pairs
- ~2.9k writes (16%)
- 90 watches
- 3.1k consensus proposals

Etcd benchmark:

- Average write query duration: 0.21s
- Average read query duration: 0.7ms | throughput: 1400QPS

Conclusion



# In-network caching (NetChain)

- Latency in current coordination services:

Client -> Server -> Client

Multiple RTTs required for consensus

| | Server | Switch |
|---|---|---|
| **Example** | NetBricks [12] | Tofino [13] |
| **Packets per sec.** | 30 million | a few billion |
| **Bandwidth** | 10-100 Gbps | 6.5 Tbps |
| **Processing delay** | 10-100 $\mu s$ | $< 1 \mu s$ |

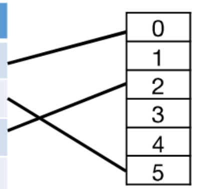
- Latency for in-switch coordination:

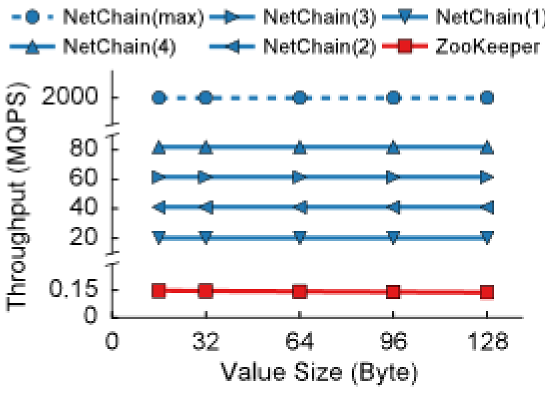
Client -> Network switch -> Client

Match-Action Table

| Match | Action |
|---|---|
| Key = X (0xfb2d7326dd4e36ac) | Read/Write RA[0] |
| Key = Y (0xfa21549a1e8926a7) | Read/Write RA[5] |
| Key = Z (0xfbeb2c7e4db86208) | Read/Write RA[2] |
| default | Drop() |

Register Array (RA)



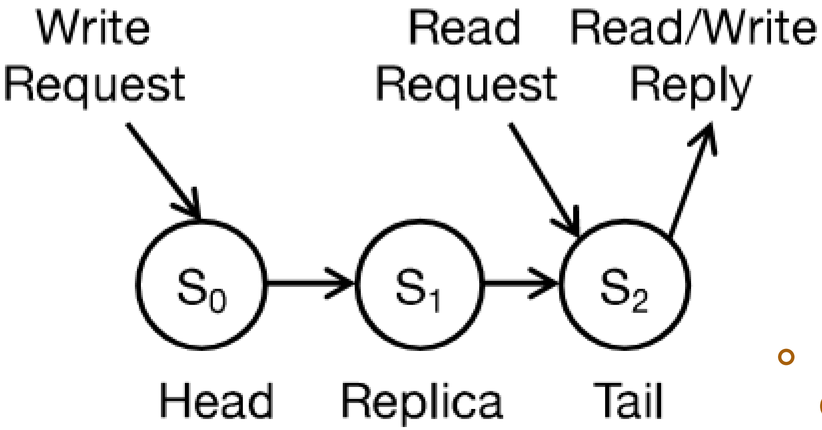




(a) Throughput vs. value size.

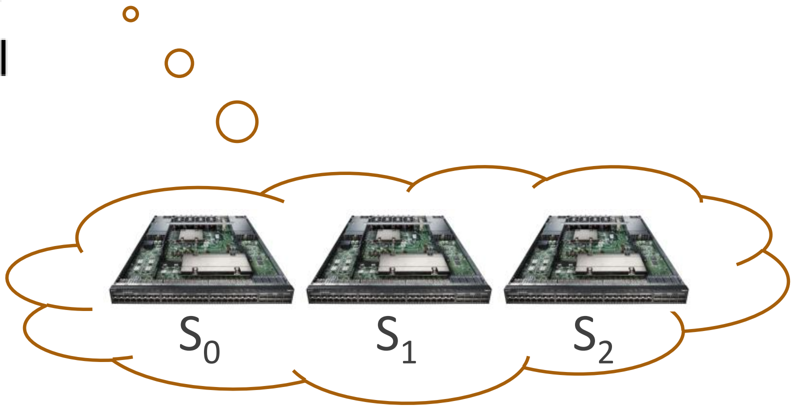
Jin, X., Li, X., Zhang, H., Foster, N., Lee, J., Soulé, R., Kim, C. and Stoica, I., 2018. Netchain: Scale-free sub-rtt coordination. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)* (pp. 35-49).

# Room for improvement?

How Chain works:

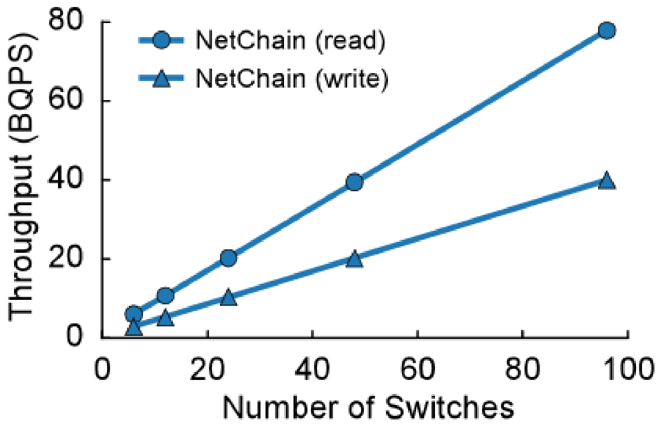



- Reads/Writes travel to Tail
  - Link saturation becomes possible.
  - Latency increases with distance from tail.
  - Chain size becomes a factor of performance.



# Room for improvement?

netChain on scalability:



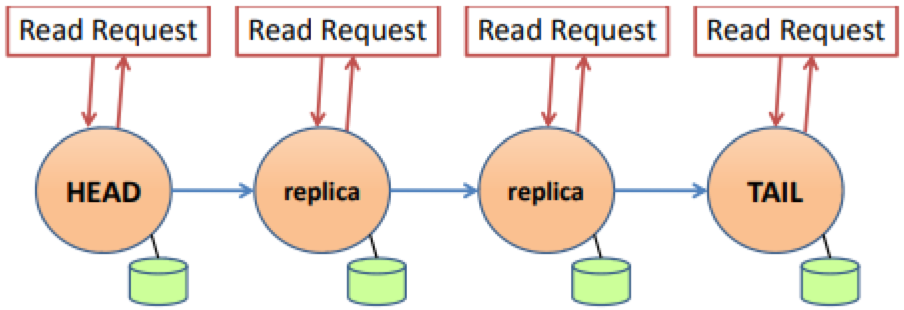

(f) Scalability (simulation).

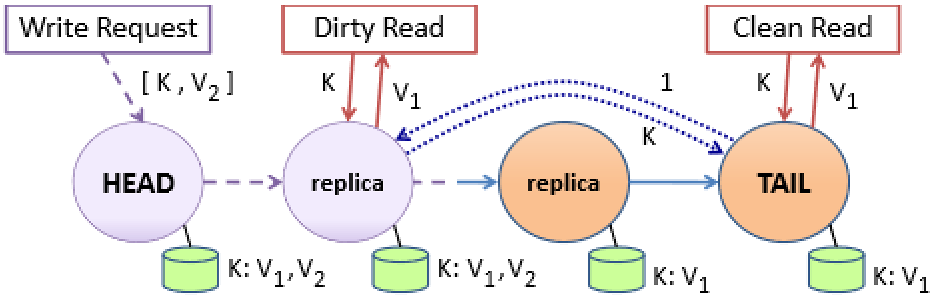
"loads. Both throughputs grow linearly, because in the two-layer network, the average number of hops for a query does not change under different network sizes."


Jin, Xin, Xiaozhou Li, Haoyu Zhang, Nate Foster, Jeongkeun Lee, Robert Soulé, Changhoon Kim, and Ion Stoica. "Netchain: Scale-free sub-rtt coordination." In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pp. 35-49. 2018.

# CRAQ

## Or Chain Replication with Apportioned Queries

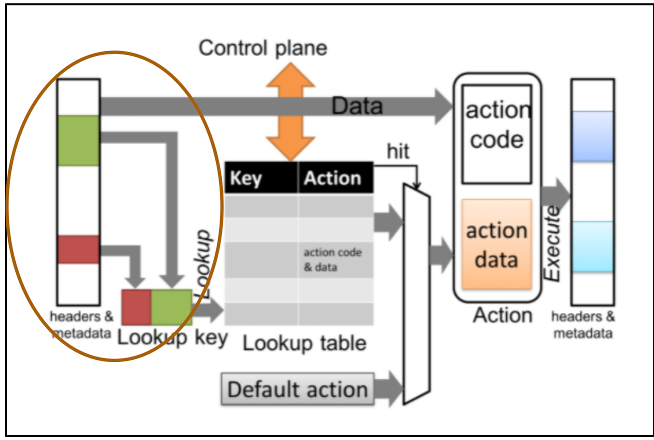



- Tail is not always the reference point:
  - Each node “knows” whether the version it holds is **clean** or **dirty**.
  - Clean: node responds directly.
  - Dirty: node fetches clean version from tail.
  - A write is marked clean when received by tail, which then sends acknowledgement to nodes.
- Consistency can be relaxed to favour performance.


Terrace, Jeff, and Michael J. Freedman. "Object Storage on CRAQ: High-Throughput Chain Replication for Read-Mostly Workloads." In *USENIX Annual Technical Conference*, no. June, pp. 1-16. 2009.

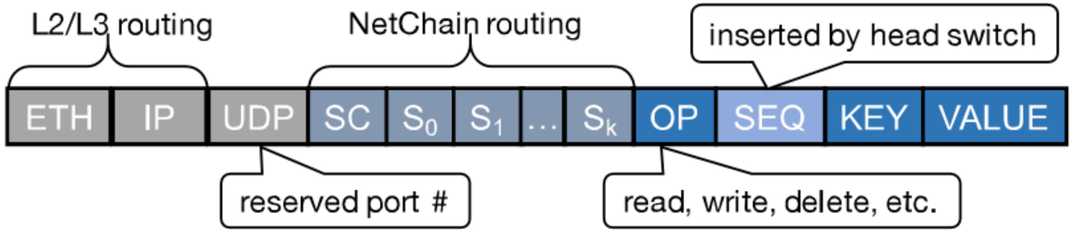
# Smaller packet format




Proposed packet format:

| | | | | | |
|---|---|---|---|---|---|
| ETH | IP | UDP | KV_OP | KEY_ID | VALUE |

NetChain's packet format:



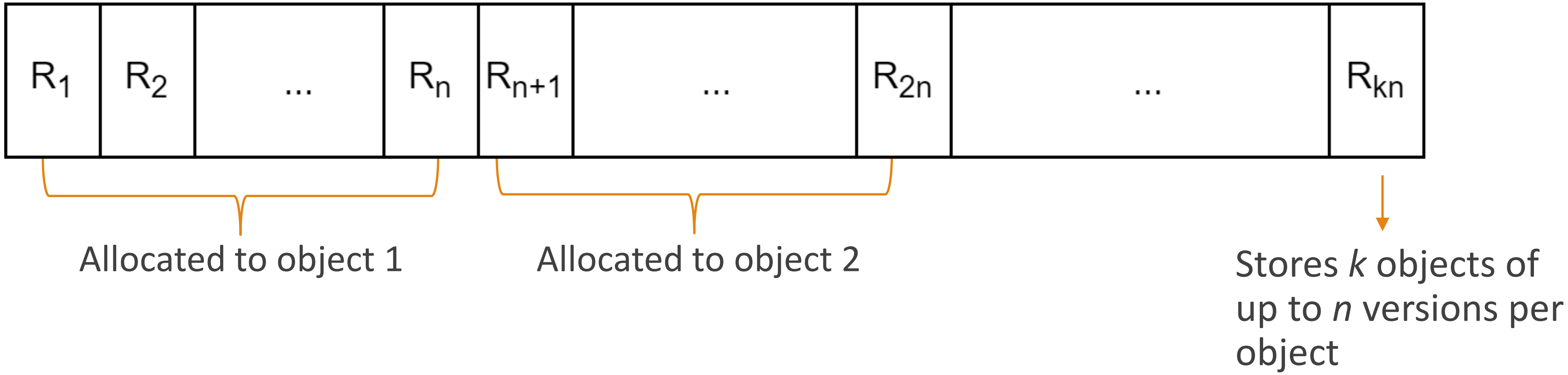

(b) NetChain packet format.

All chain IPs are on packet: chain size dictates packet's size. We propose storing this info in data plane.

# Used structures

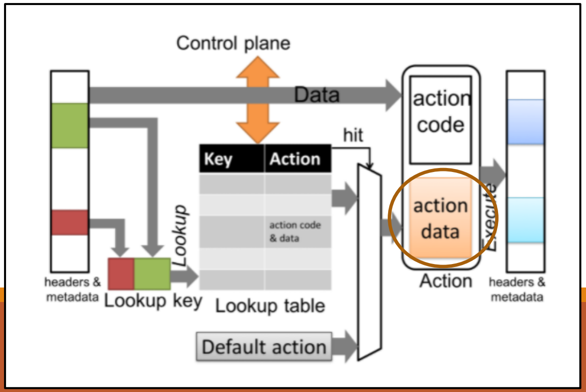
Using switch registers for storage.

`Objects store:`




i.e., The register can support up to *n* dirty versions per object.

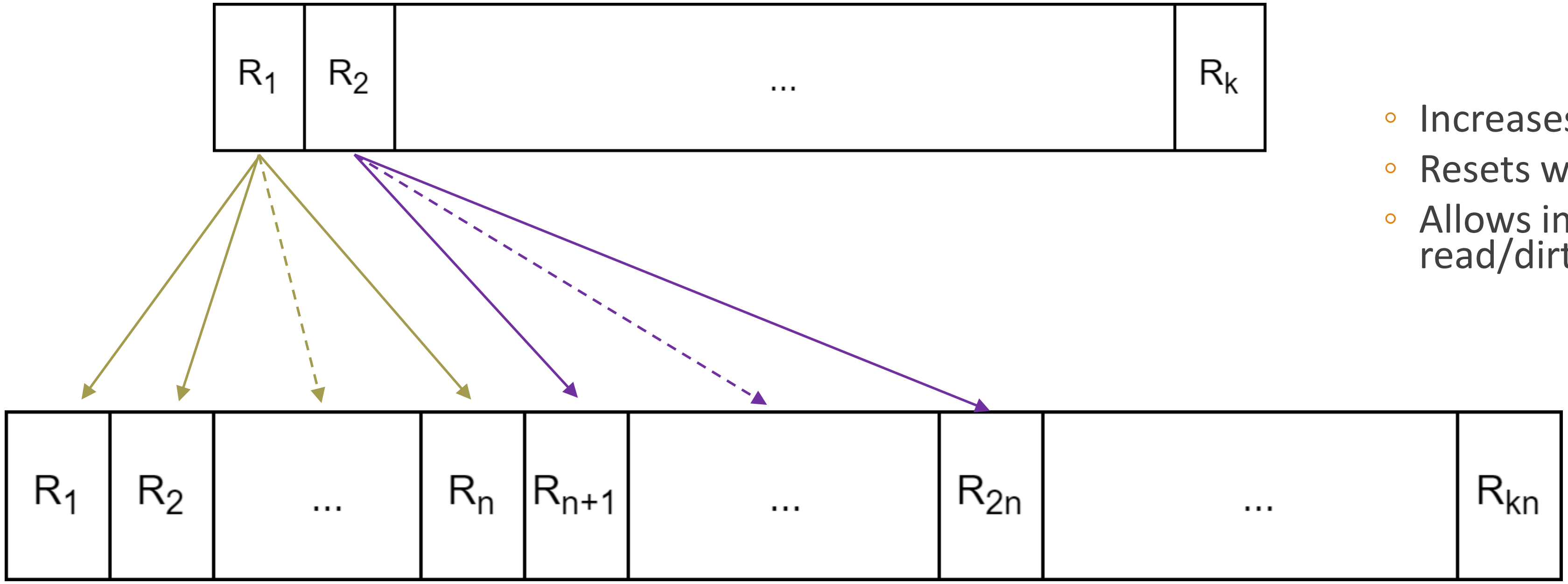
Fast access but limited resource. Both *k* and *n* need to be fixed.



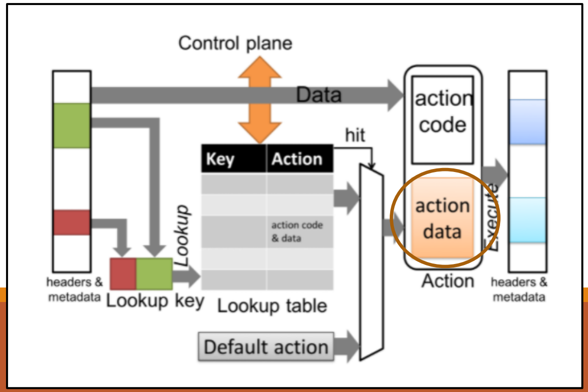
# Used structures

Auxiliary registers:

`Read index:`



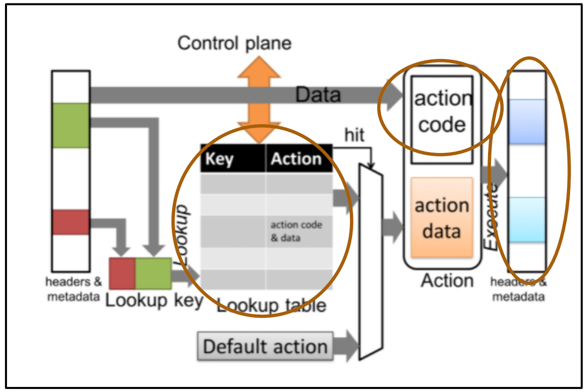

- Increases based on dirty commits.
- Resets when commit is clean.
- Allows implicit definition of read/dirty reads.



# Ingress control

```
- objects_store: stores all KV data
- read_index: points to latest read committed for each object in
  objects_store
- write_index: points to next write position for each object in
  objects_store
if kv_op == READ then
    get_read_index();
    if meta.read_index == 0 then
        read_operation();
    else if meta.my_role == TAIL then
        get_last_read_index();
        read_operation();
    else
        forward_to_tail();
else if kv_op == WRITE then
    get_write_index();
    if meta.write_index == 0 then
        clean_write();
        forward_to_tail();
    else
        if meta.write_index >= NUMBER_OF_VERSIONS then
            drop();
        else
            dirty_write();
            forward_to_tail();
    if meta.my_role == TAIL then
        generate_acknowledgement();
        clean_write();
        multicast();
else if kv_op == ACKNOWLEDGEMENT then
    clean_write();
```

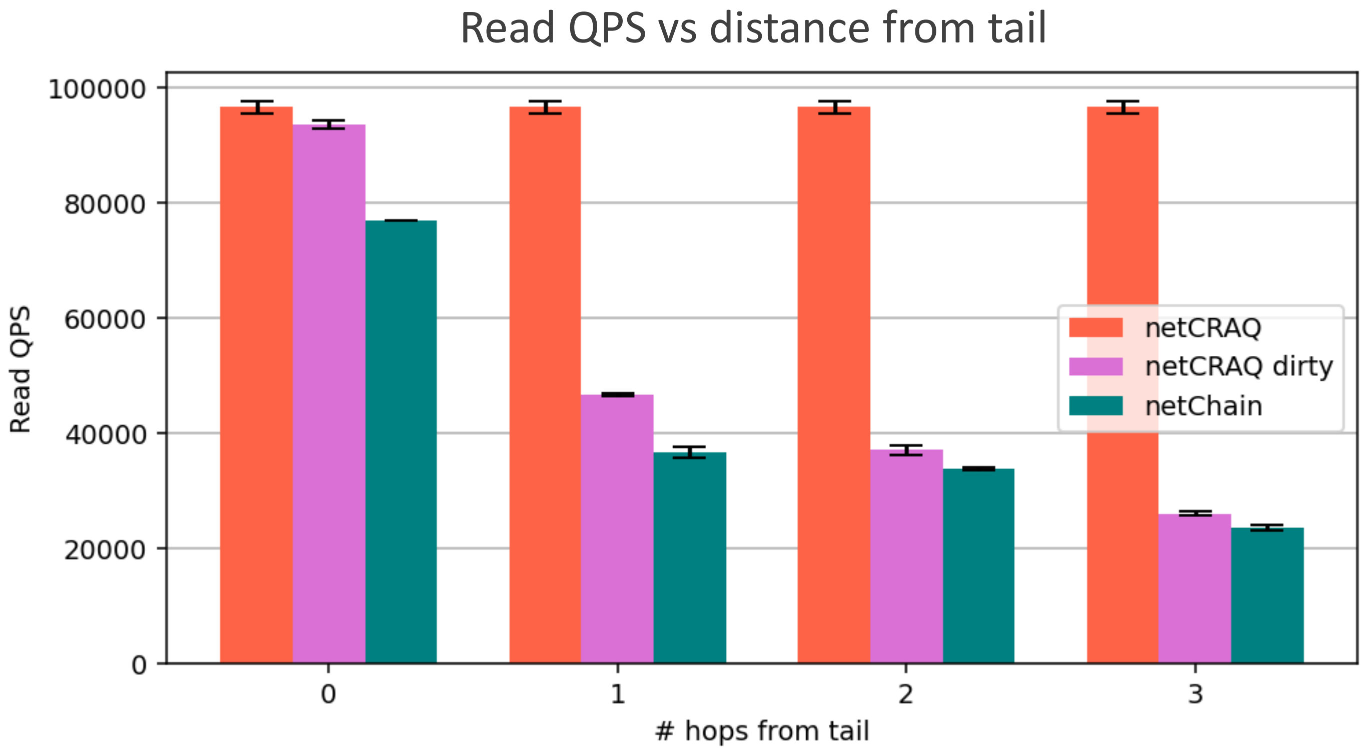
Stored in data plane




Acknowledgments always reset indexes

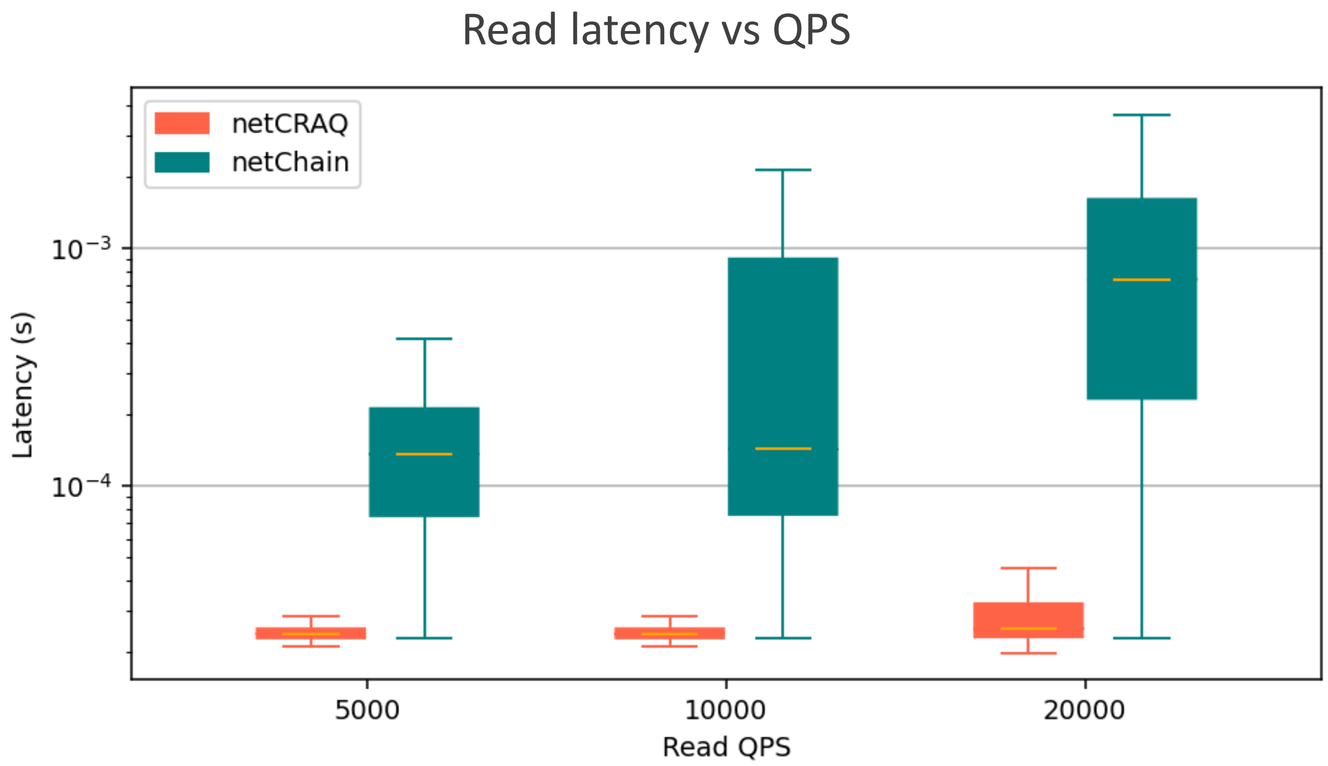
Improvement over sequential forwarding

# Perf evaluation



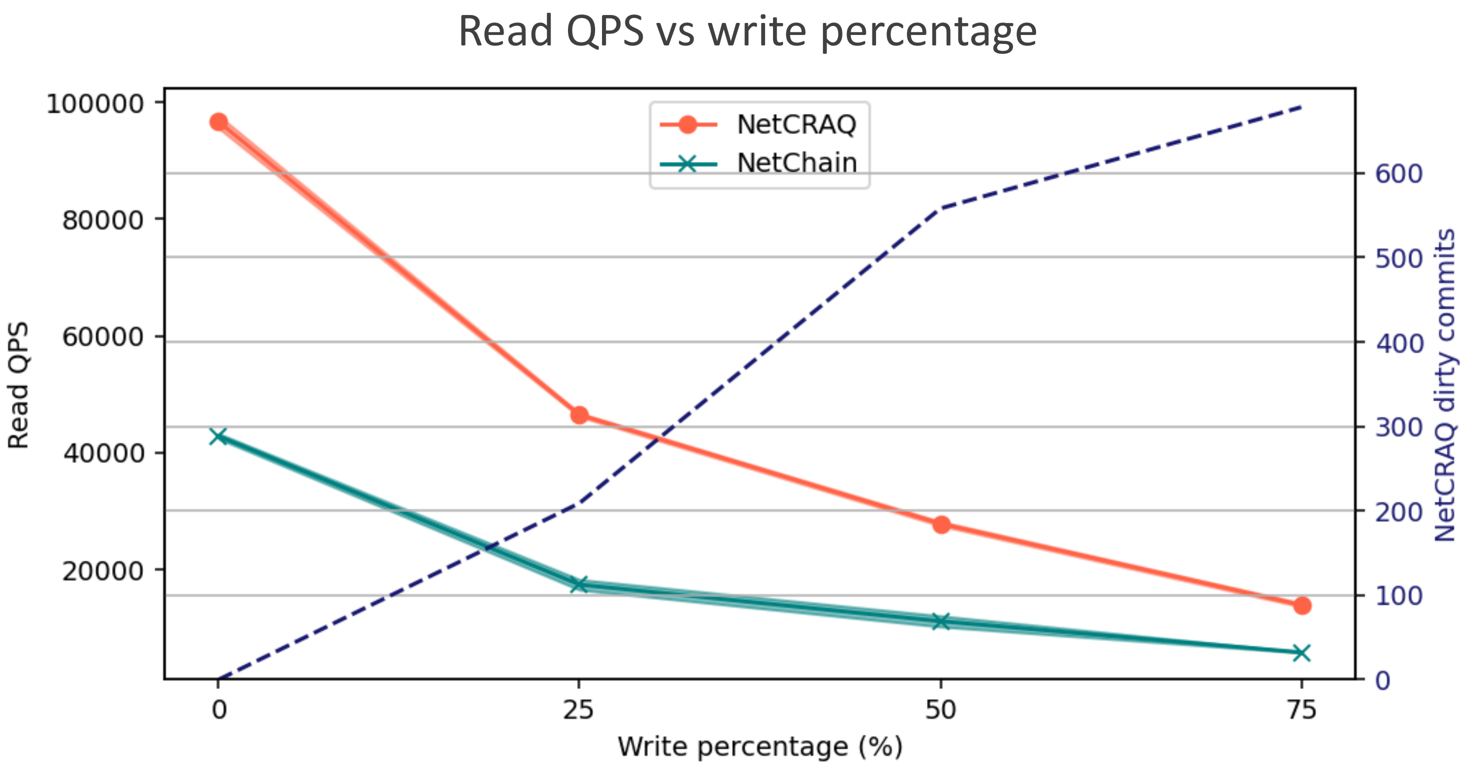

- NetCRAQ outperforms netChain, regardless of distance from tail.
- Dirty reads are affected, but perform better than netChain.
- Dirty reads should be a percentage of the workload.

# Perf evaluation



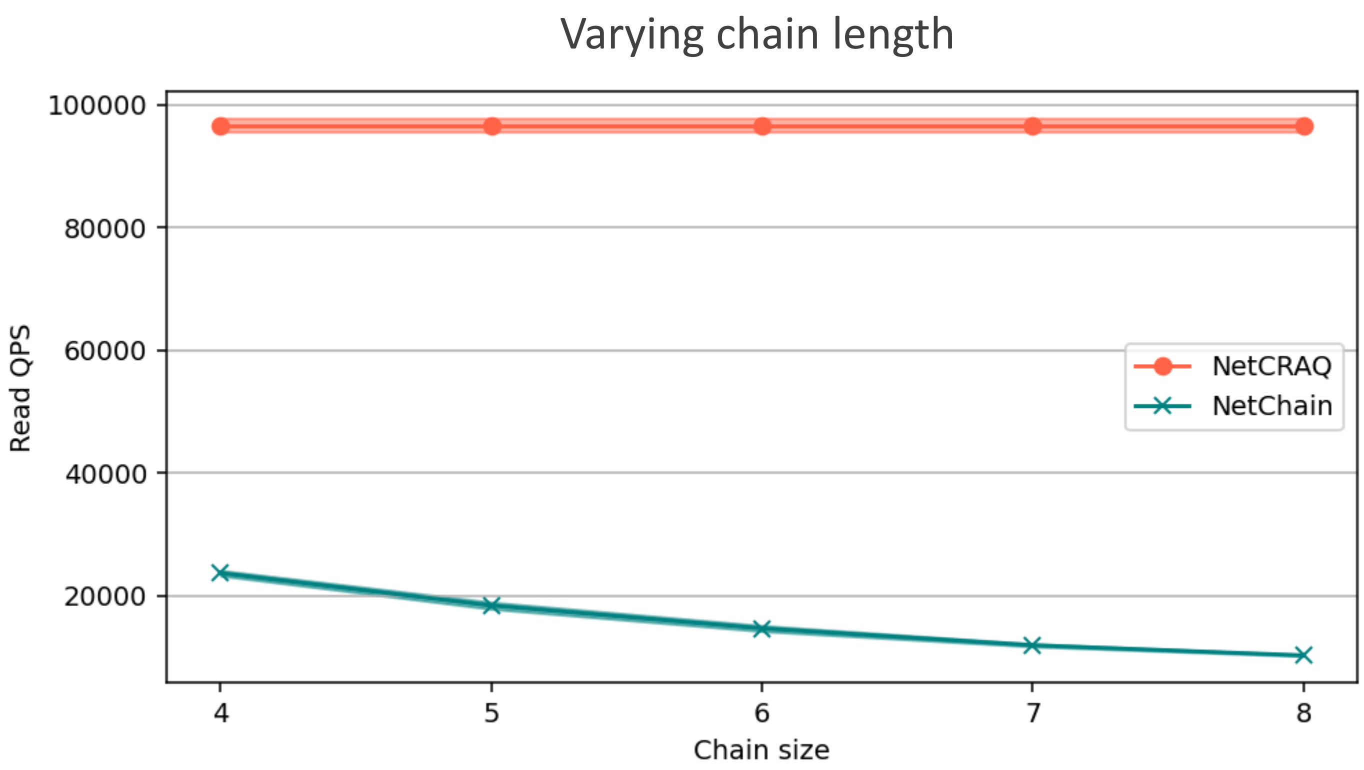

- NetCRAQ's fewer hops allow fast responses regardless of QPS or distance from tail.

# Perf evaluation



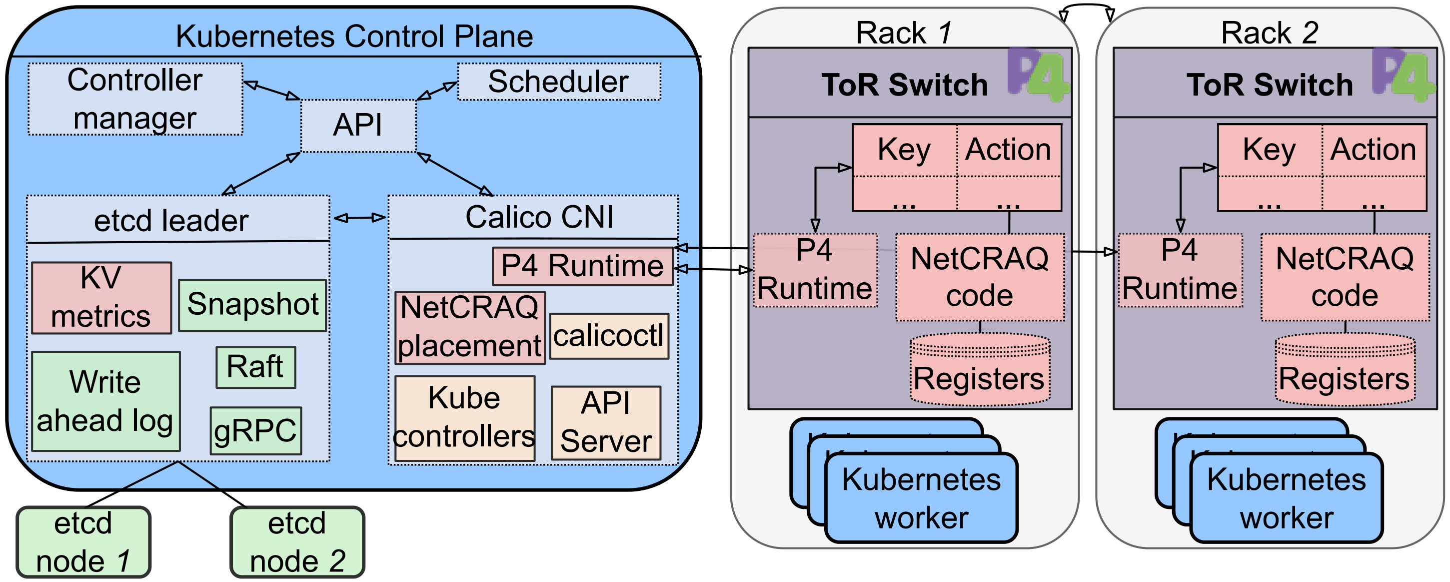

- Writes are more expensive transactions than reads.
- Another win for netCRAQ.
- At the cost of having enough registers to commit dirty writes.

# Perf evaluation




- Comparison in head nodes. (slightly unfair)
- Shows potential gap in performance

# Proposed Kubernetes architecture




Etcd leader:
- KV metrics

CNI:
- Runtime statistics
- Placement algorithm

ToR switch:
- Counters

# Future steps

1. Conclude Tofino implementation.
2. Integrate P4Runtime API to CNI
3. Evaluate end-to-end workloads

Thanks!