Tomek Mrugalski, Director of DHCP Engineering

Optimizing Kea DHCP Performance

Photo by Karl Anderson on Unsplash

Optimizing DHCP Performance

1. Obvious stuff
2. Multi-threading
3. Lease storage
4. Host reservation
5. Client classification
6. Hardware platform
7. Minimize latency between components
8. HA choices
9. Reduce iterations to allocate lease
10. Hooks and latency
11. Don’t do these things
12. Fine tuning

Planning for capacity and performance

Server load factors:

- Clients without a lease (new client or expired)
- Clients renewing an existing lease
- API requests
- Bad clients (chatty)
- Lease expiration times
- Lease renewal times
- HA

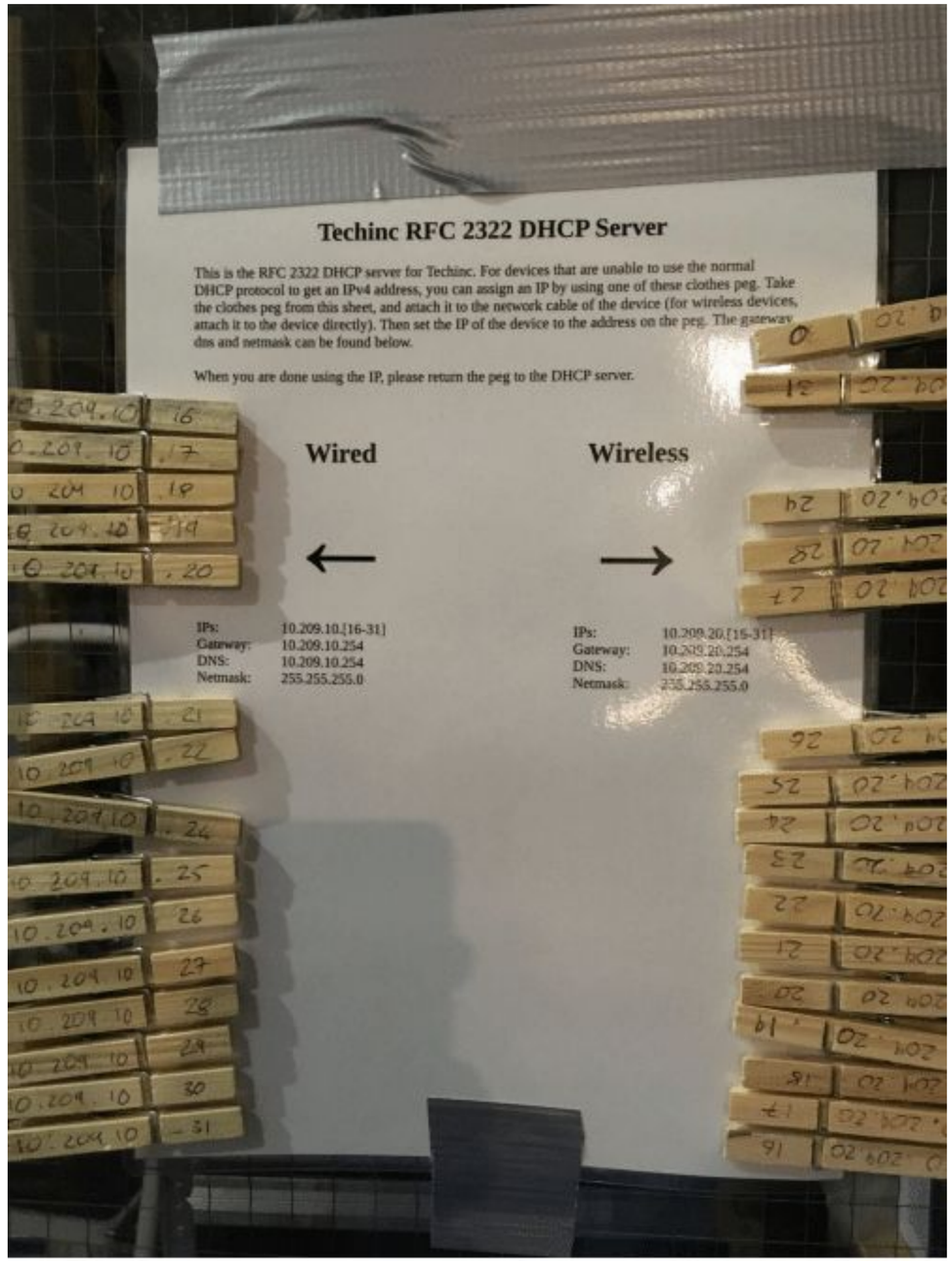
Operations per Second						
	Client Lease Times					
Active Leases	30 min	1 hr	1 day	1 week	2 weeks	30 days
1,000	1	1	-	-	-	-
10,000	11	6	-	-	-	-
100,000	111	56	2	-	-	-
500, 000	556	278	12	2	1	-
1,000,000	1,111	556	23	4	2	1
1,500,000	1,667	833	35	5	2	1
2,000,000	2,222	1,111	46	7	3	2
4,000,000	4,444	2,222	93	13	7	3
6,000,000	6,667	3,333	139	20	10	5

Source: Cisco

Obvious stuff

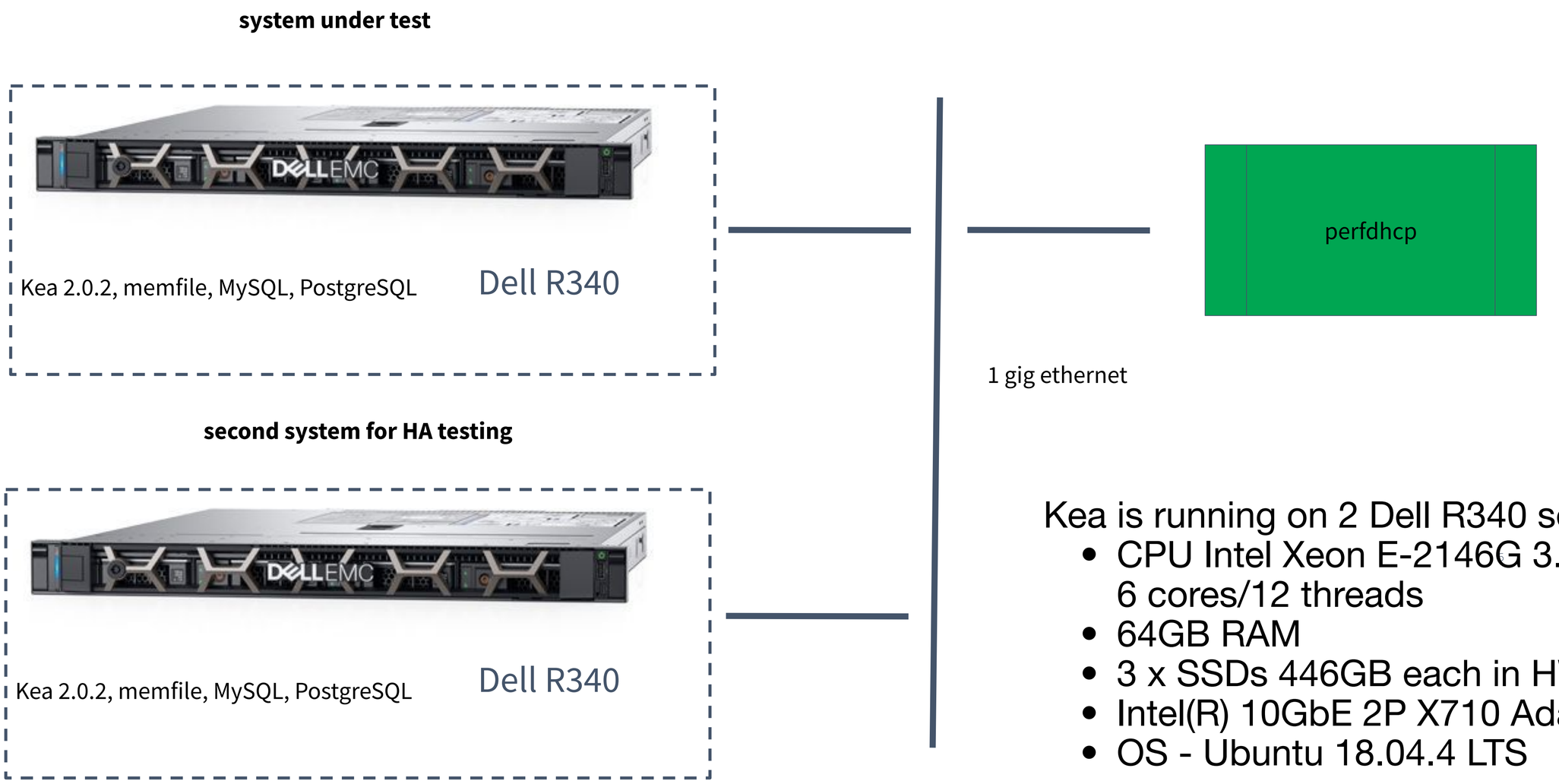
- **Upgrade**, it really makes sense
  - still on 1.4? ugh
  - 1.6 single-threaded
  - 1.8 added MT (for packet processing, doesn’t cover HA)
  - 2.0 added HA+MT
  - 2.1.x added cache threshold, subnet selection speed-up, early global HR lookup, ...

- **Don’t run stuff you don’t need**
  - HR
  - HA
  - shared networks
  - logging
  - extra hooks



The utimate DHCP server

Test setup



Kea is running on 2 Dell R340 servers:

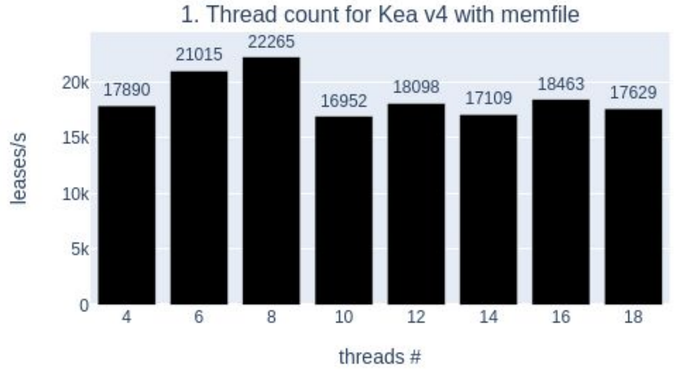
- CPU Intel Xeon E-2146G 3.5GHz
  6 cores/12 threads
- 64GB RAM
- 3 x SSDs 446GB each in HW RAID-0
- Intel(R) 10GbE 2P X710 Adapter (2 ports)
- OS - Ubuntu 18.04.4 LTS

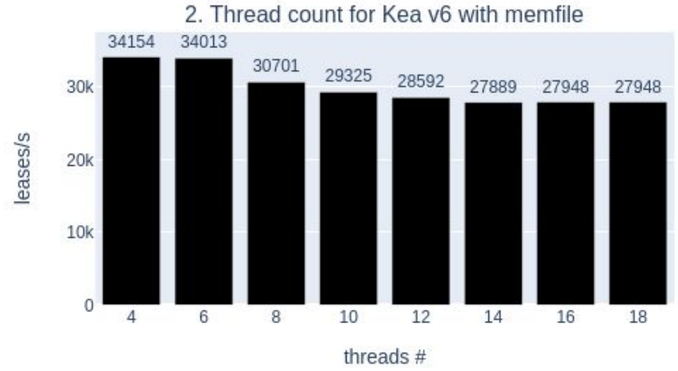
Performance test details:https://kb.isc.org/docs/kea-20-performance-tests
MySQL DB tweak: https://kea.readthedocs.io/en/latest/arm/admin.html#improved-performance-with-mysql

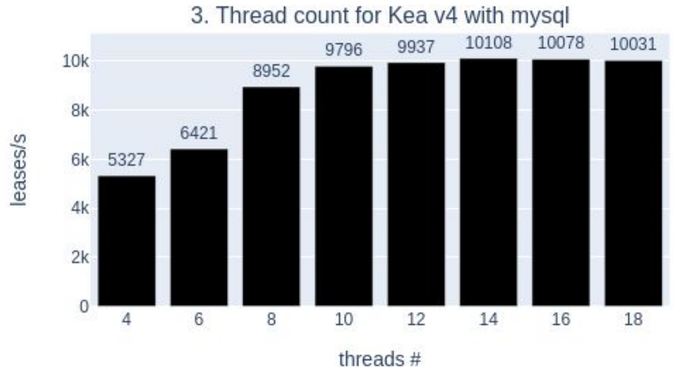
MT :: # of threads

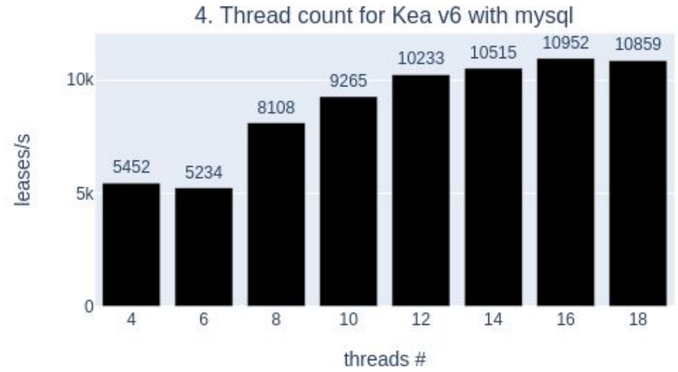
- More is not always better
- Heavily dependent on the backend and HW

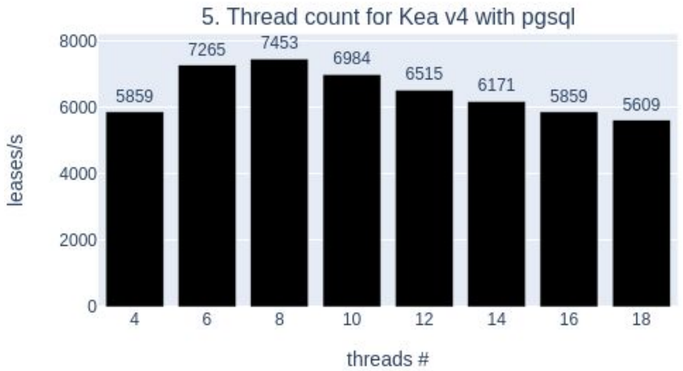
```
{
  "Dhcp4": {
    "multi-threading": {
      "enable-multi-threading": true,
      "packet-queue-size": 16,
      "thread-pool-size": 8
    },
  ...
}
```

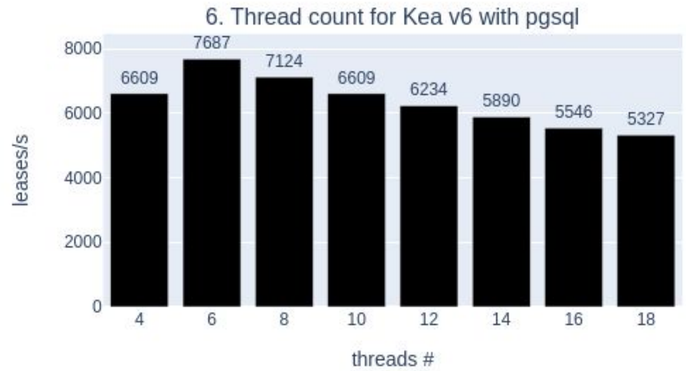








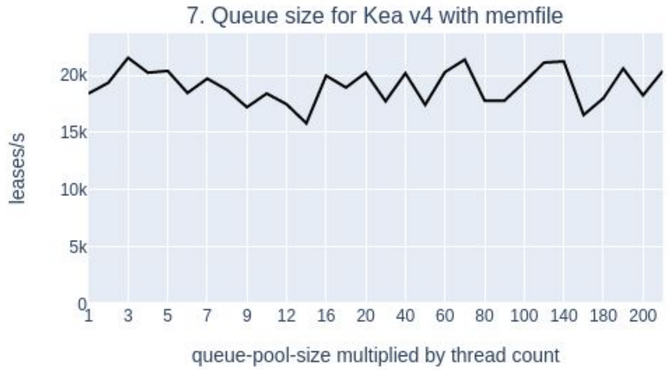


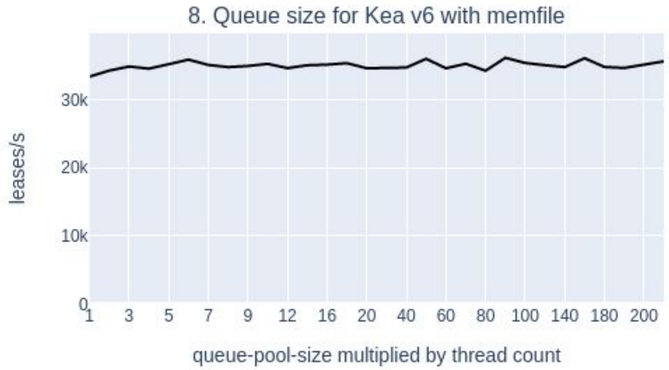


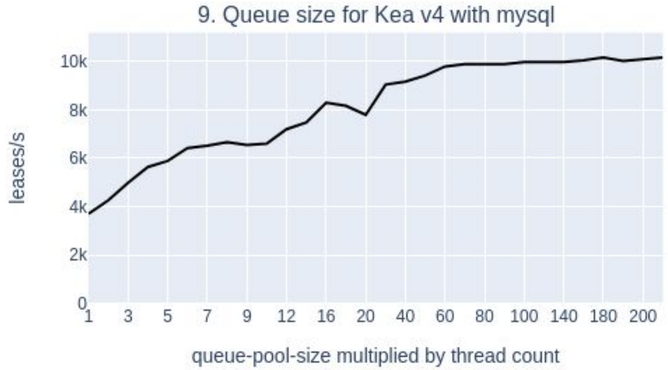
MT :: queue size

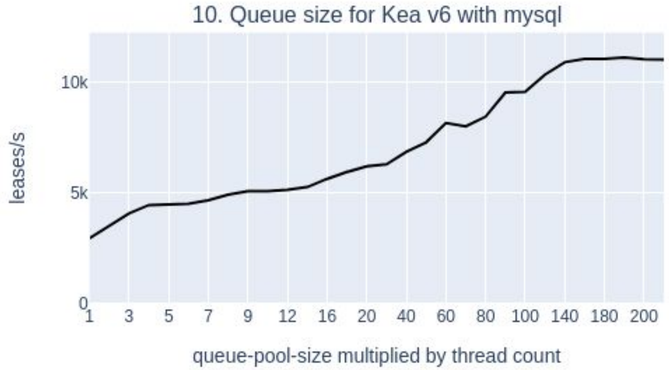
- Defines queue size ***globally***
- Longer queue:
  - often does not give you more performance
  - usually increases response time

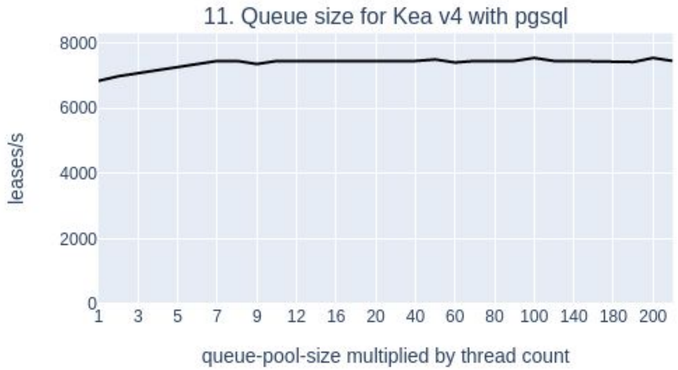
```
{
  "Dhcp4": {
    "multi-threading": {
      "enable-multi-threading": true,
      "packet-queue-size": 80,
      "thread-pool-size": 8
    },
  ...
}
```

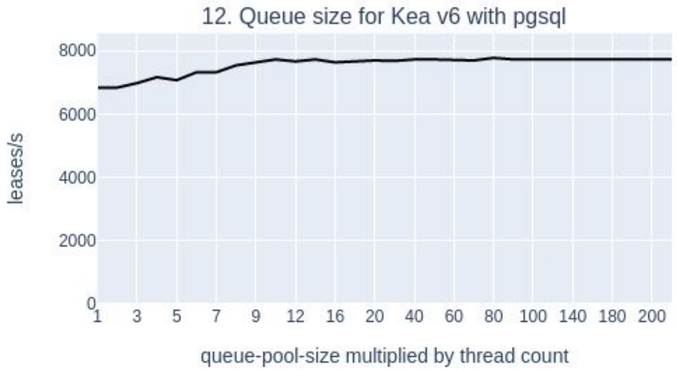




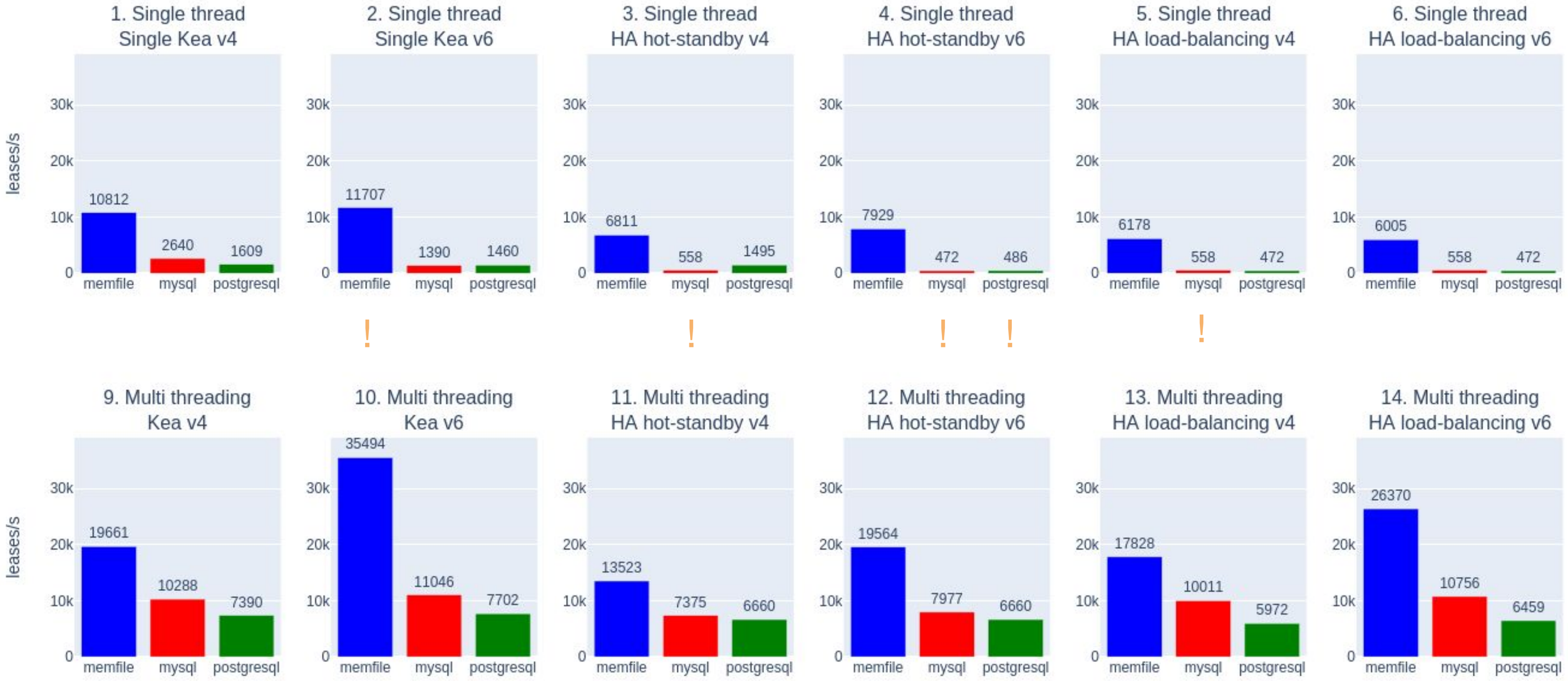








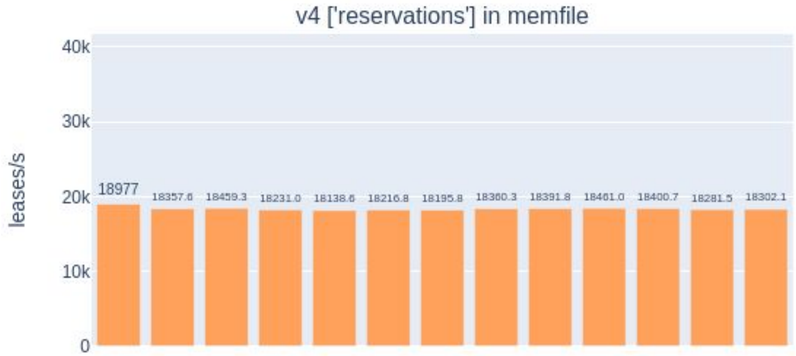
MT :: Comparative results

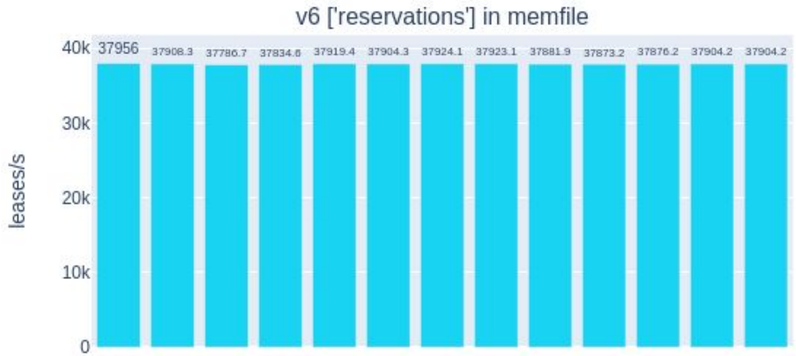


Interactive version: https://reports.kea.isc.org/performance/stable/2.0.2/report.html

Lease Storage

memfile > mysql > pgsql





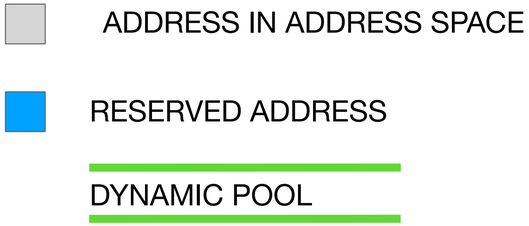
Interactive version: https://reports.kea.isc.org/performance/stable/2.0.2/report.html

Host Reservations :: In-pool, out-of-pool

- disabled - fastest (pure dynamic)

SUBNET, e.g. 192.168.1.0/24

POOL, e.g. 192.168.1.3-192.168.1.100



- out of pool - faster (no HR lookup for dynamic)

SUBNET, e.g. 192.168.1.0/24

POOL, e.g. 192.168.1.3-192.168.1.100

- all - safer (you can put HR anywhere, but it's slower)

SUBNET, e.g. 192.168.1.0/24

POOL, e.g. 192.168.1.3-192.168.1.100

Recommended reading: https://kea.readthedocs.io/en/kea-2.0.2-doc/arm/dhcp4-srv.html#fine-tuning-dhcpv4-host-reservation

Host Reservations :: Subnet, global, early

Processing order:

- early global HR lookup (disabled by default), introduced in 2.1.4
- subnet selection
- global HR lookup (disabled by default)
- subnet HR lookup

Subnet HR: normal networks have reservations specific to subnet

device X connected in this subnet should get address Y and option Z

Global HR: roaming users

this device, regardless where it's connected, should get option Z (and maybe address W)

Don't use address reservation in global HR, unless you know exactly what you're doing.

Host Reservations :: Identifiers

- The DHCPv4 default (safe, but slow)

```
"Dhcp4": {
  "host-reservation-identifiers": [ "hw-address", "duid", "circuit-id", "client-id" ],
  ...
}
```

- Pick only reservation type you're actually using. In most cases, it will be:

```
"Dhcp4": {
  "host-reservation-identifiers": [ "hw-address" ],
  ...
}
```

- Recommendation: use only one reservation identifier
- supported options: hw-address, duid, circuit-id, duid, client-id, flex-id

Recommended reading: https://kea.readthedocs.io/en/kea-2.0.2-doc/arm/dhcp4-srv.html#fine-tuning-dhcpv4-host-reservation

Classification (expressions in general)

Cool way to classify when used in moderation.

Each packet is evaluated against **each class**. For each class, **each token** needs to be evaluated.

Each token (a primitive part of the expression) marked with different colors.

Good example

```
{
  "name": "DROP",
  "test": "pkt4.msgtype == 5"
},
```

Bad example

```
{
  "name": "DROP",
  "test": "pkt4.mac == 0x0102030405 or pkt4.mac == 001122334455 or ..."
},
```

Affects (if used): classification, custom logging in forensic logging, flex-id, ...

Classification impact

```
"Dhcp6": {
  "client-classes": [
  {
    "name": "CLASS0"
  },
  {
    "name": "CLASS1"
  },
  {
    "name": "CLASS2"
  },
  {
    "name": "CLASS3"
  },
  // 100 classes total
```

- **Scenario 1: memfile v6, no classes**
- **Scenario 2: memfile v6, 100 empty classes**


```
"Dhcp6": {
  "client-classes": [
  {
    "name": "CLASS0",
    "test": "'a'=='a'"
  },
  {
    "name": "CLASS1",
    "test": "'a'=='a'"
  },
  {
    "name": "CLASS2",
    "test": "'a'=='a'"
  },
  {
    "name": "CLASS3",
    "test": "'a'=='a'"
  },
  // 100 classes total
```

- **Scenario 3: memfile v6, 100 classes with simple expression**

Scenario 1	35038 leases/s
Scenario 2	28546 leases/s
Scenario 3	12732 leases/s

Considerations when choosing hardware

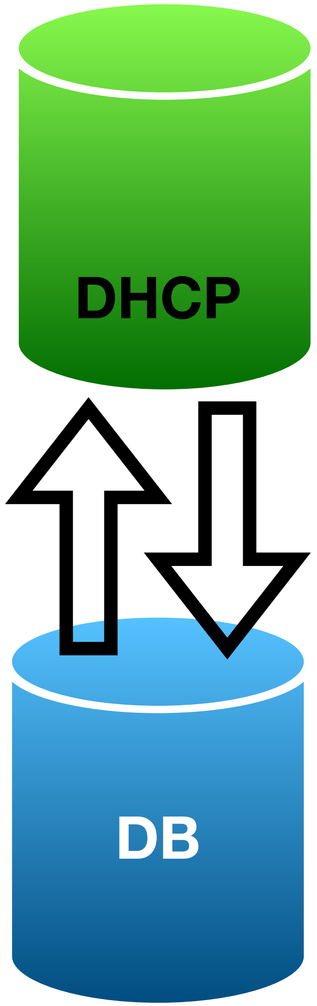
Application	Limitation	Optimize
DHCPv4, DHCPv6 servers	CPU	Fast clock speed, # of threads
Lease backend	lease writes to disk, round trip time	disk access speed, network latency
Host backend	reads, round trip time	disk read speed, memory, network latency
Configuration backend	(same as lease), pooling time	?

Network latency

- Planning your DB
  - Fastest: don’t use DB at all
  - If you can’t, keep your DB local
  - If you can’t, keep your DB close
  - If you can’t, ...

- Clustering solutions are hairy beasts.
  ISC has ***limited*** experience. For MySQL, Galera and Percona seems to work well.
  More work planned in 2.3.x series (kicking off in July’22)

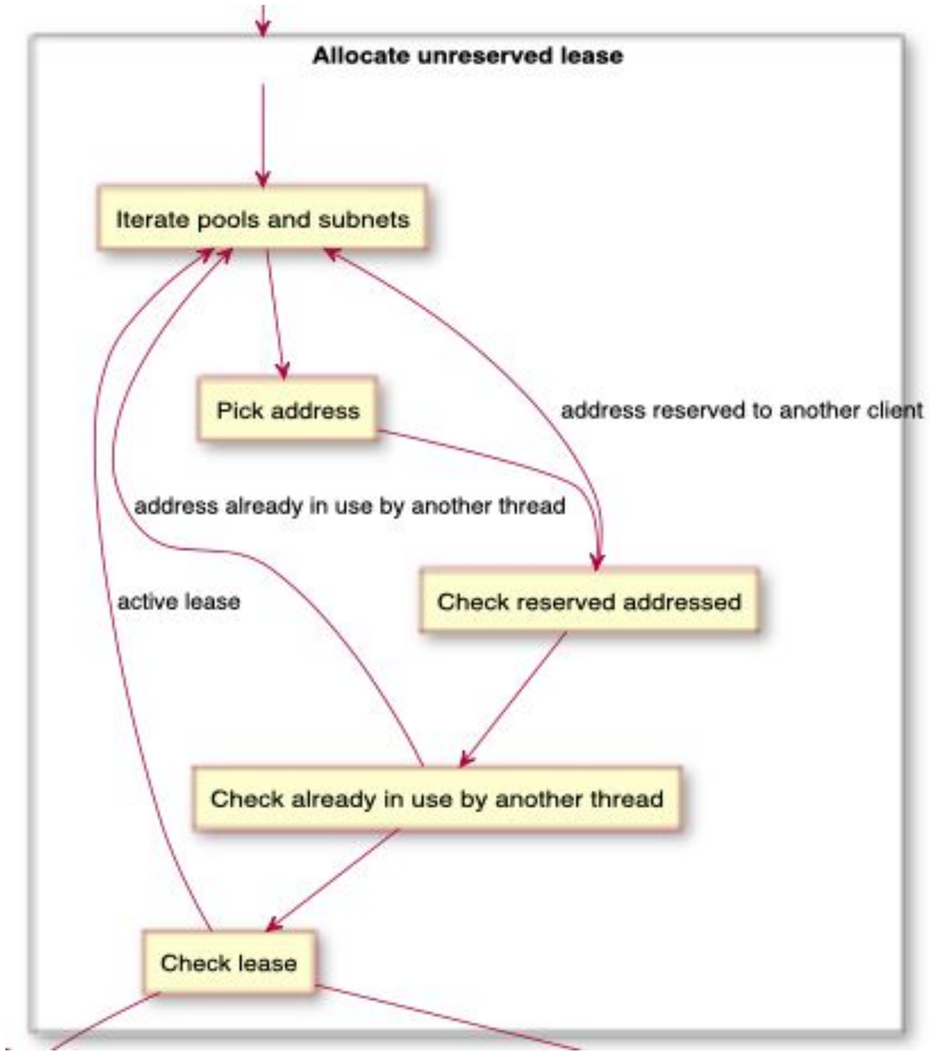
- Can point different backends to different DBs.
  - Lease backend most sensitive, then hosts backend, then config backend



Recommended reading: https://kb.isc.org/docs/getting-started-with-galera-or-percona-for-kea

Reduce iterations

Better	Worse
Large subnets (low utilization)	Small subnets (higher utilization)
Fewer subnets, fewer pools	Many subnets
No shared networks	Shared networks (just another subnet to check)
just use chaddr	look up by client ID look up again by chaddr



High Availability and Multi-threading

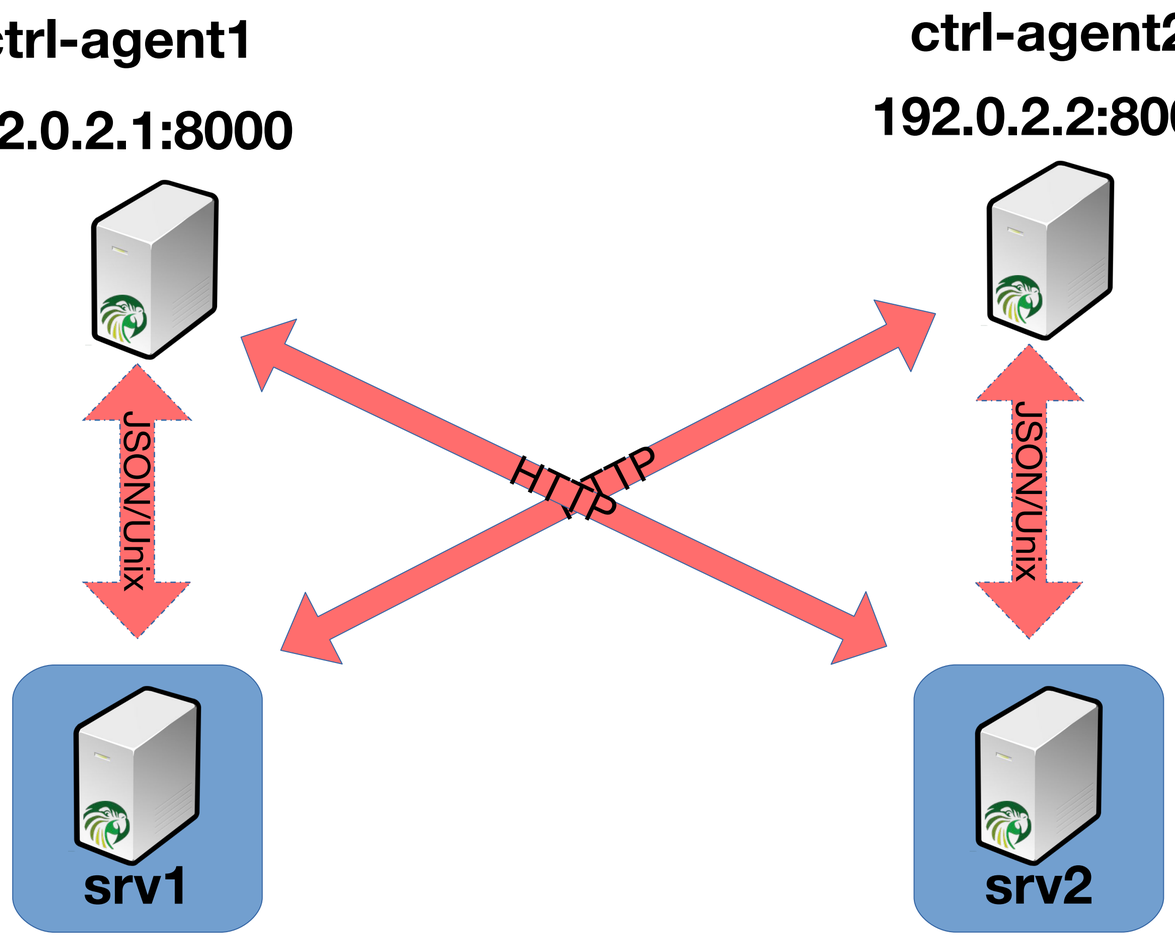
- Enable **MT** and **direct connection** between HA partners.
- Hot-standby has a more lightweight setup than load balancing - it avoids splitting the pools between the servers using client classification. As a result, the servers can find suitable pools faster.
- Specify a reasonable parked-packet-limit to avoid congestion when the HA-enabled server becomes swamped with a stream of packets exceeding its capacity to respond.
- Reduce the network latency between the HA partners (network configuration, geo-location, etc.).

High Availability and Multi-threading

- Use as few backup servers as possible - avoid lease updates overhead.
- Avoid low heartbeat delay values to reduce the heartbeat processing overhead.
- Disable lease updates and leases synchronization when HA-enabled servers use a common database for leases.

Multi-threading (Kea 1.8)

Sequential | Multi-threaded




```
"Control-agent":
{
   "http-host": "192.0.2.1",
   "http-port": 8000,
   ...
}
```

```
"Dhcp4": {
   ...
"this-server-name": "srv1",
"peers": [{
  "name": "srv1",
  "url": "http://192.0.2.1:8000/",
  "role": "primary",
  "auto-failover": true
},
{
  "name": "srv2",
  "url": "http://192.0.2.2:8000/",
  "role": "secondary",
  "auto-failover": true
},
```

```
"Control-agent":
{
   "http-host": "192.0.2.2",
   "http-port": 8000,
   ...
}
```

```
"Dhcp4": {
   ...
"this-server-name": "srv1",
"peers": [{
  "name": "srv1",
  "url": "http://192.0.2.1:8000/",
  "role": "primary",
  "auto-failover": true
},
{
  "name": "srv2",
  "url": "http://192.0.2.2:8000/",
  "role": "secondary",
  "auto-failover": true
},
```

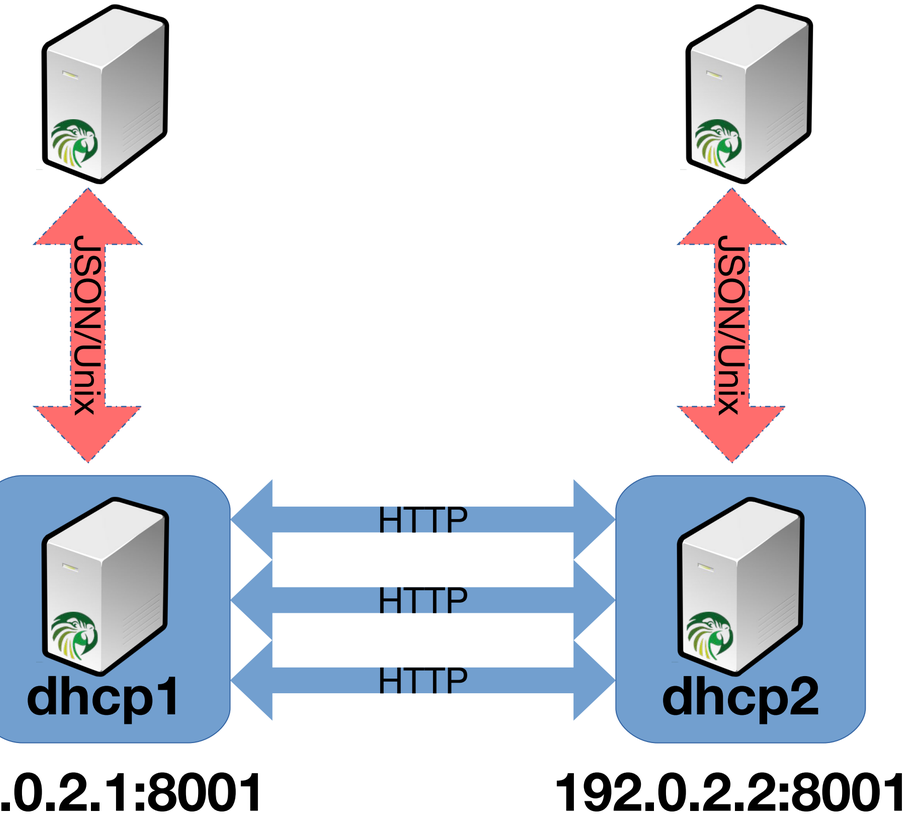
High Availability with Multi-threading (Kea 2.0)

Sequential

Multi-threaded

```
"Control-agent":
{
   "http-host": "192.0.2.1",
   "http-port": 8000,
   ...
}
```

```
"high-availability": [ {

  "multi-threading": {
    "enable-multi-threading": true,
    "http-dedicated-listener": true,
    "http-listener-threads": 4,
    "http-client-threads": 4
  },

  "this-server-name": "srv1",
  "peers": [{
    "name": "srv1",
    "url": "http://192.0.2.1:8001/",
    "role": "primary",
    "auto-failover": true
  },
  {
    "name": "srv2",
    "url": "http://192.0.2.2:8001/",
    "role": "secondary",
    "auto-failover": true
  },
}
```




```
"Control-agent":
{
   "http-host": "192.0.2.2",
   "http-port": 8000,
   ...
}
```

```
"high-availability": [ {

  "multi-threading": {
    "enable-multi-threading": true,
    "http-dedicated-listener": true,
    "http-listener-threads": 4,
    "http-client-threads": 4
  },

  "this-server-name": "srv2",
  "peers": [{
    "name": "srv1",
    "url": "http://192.0.2.1:8001/",
    "role": "primary",
    "auto-failover": true
  },
  {
    "name": "srv2",
    "url": "http://192.0.2.2:8001/",
    "role": "secondary",
    "auto-failover": true
  },
}
```

Hooks add latency

- RADIUS hook - still not MT capable
- run script hook
- any hook with a DB lookup
- be cautious with Forensic logging
  - flexible, but at a cost (custom, DB logging)

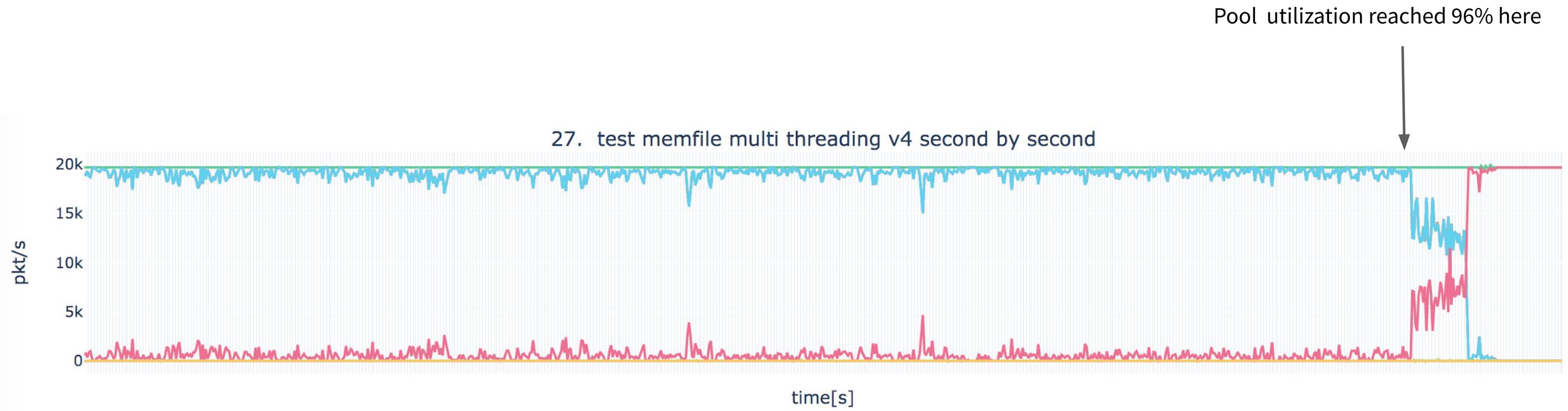
Chatty Clients (2.0)

- Problem: Buggy clients renewing early
- Each renewal:
  - Host reservation lookup
  - Lease lookup
  - Logging*
  - HA: partner update*
  - DNS Update*
- Solution: cache replies
- IPv4 and IPv6

Mitigation, not 100% solution

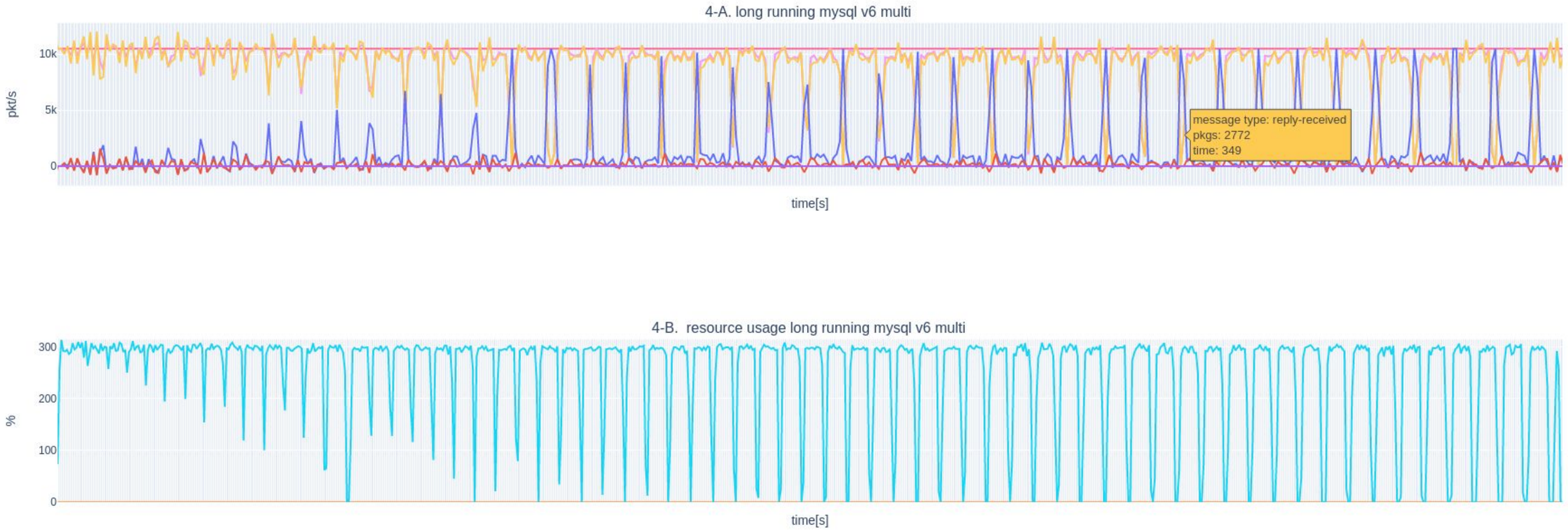
```
"subnet6": [
    {
        "subnet": "2001:db8::/64",
        "pools": [ { "pool": "2001:db8::/64" } ],
        "renew-timer": :1800,
        "valid-lifetime": 3600,

        "cache-threshold": .25,
        "cache-max-age": 900,
        ...
    }
],
```

The danger of high pool utilization



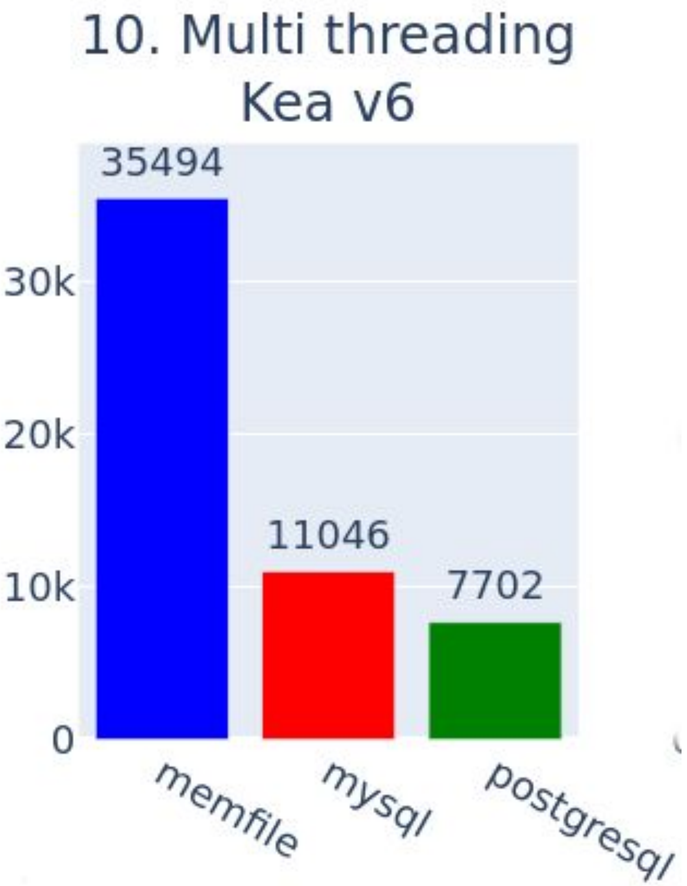
ISC Performance Reports

https://reports.kea.isc.org



Going over the top

- 36k leases/s x 4 packets x ~300 bytes = 430.2MB/s
  that’s ~540 Mbps
- traffic control starts to play a role
  - socket buffers
  - queue disciplines
  - NIC drivers tweaking
  - ...



Trust, but verify

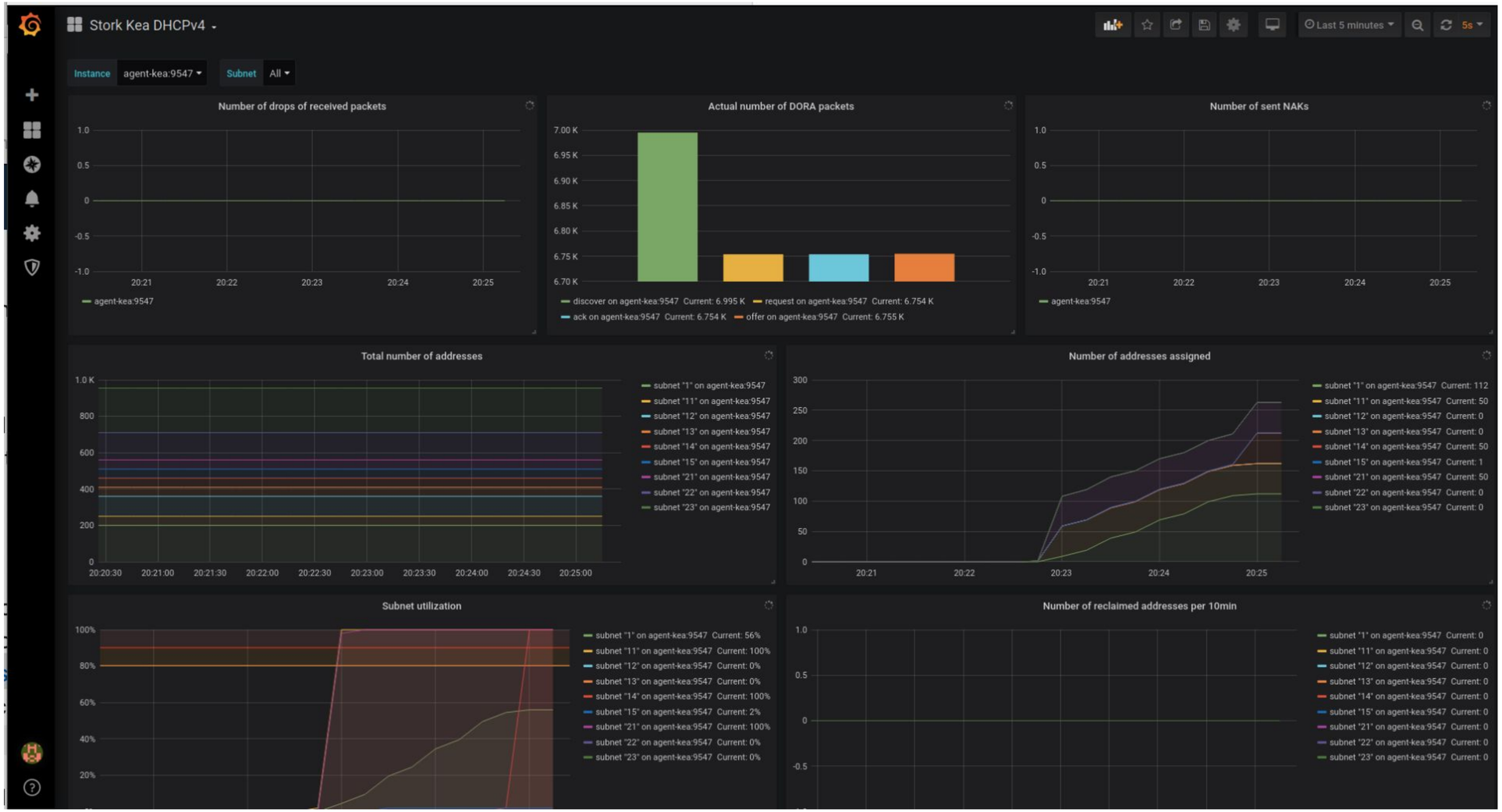
- Run your own experiments, it's easy!

```
$ ./perfdhcp --help
perfdhcp [-1] [-4 | -6] [-A encapsulation-level] [-b base] [-B] [-c]
       [-C separator] [-d drop-time] [-D max-drop] [-e lease-type]
       [-E time-offset] [-f renew-rate] [-F release-rate] [-g thread-mode]
       [-h] [-i] [-I ip-offset] [-J remote-address-list-file]
       [-l local-address|interface] [-L local-port] [-M mac-list-file]
       [-n num-request] [-N remote-port] [-O random-offset]
       [-o code,hexstring] [-p test-period] [-P preload] [-r rate]
       [-R num-clients] [-s seed] [-S srvid-offset] [--scenario name]
       [-t report] [-T template-file] [-u] [-v] [-W exit-wait-time]
       [-w script_name] [-x diagnostic-selector] [-X xid-offset] [server]
```

- Simulate 1k clients at 500 pkts/s: **perfdhcp -R 1000 -r 500 -i eth0 192.0.2.1**
- Run for certain time: **-p <seconds>**
- Read MAC addresses from file: **-M <mac-list-file>**
- Add custom option: **-o code,hexstring** (wanna simulate cable modems? voip? ...)
- Able to simulate multiple relays: **-J <remote-address-list-file>**
- Can do traffic engineering to some degree: **-f renew-rate, -F release-rate**
- Can do various scenarios:
  - How long it takes to configure X clients?
  - How many leases/s can my system assign/renew?
  - Avalanche mode (impatient clients start renewing)

Stork Grafana charts

Many metrics on DORAs, with time-based buckets



I’d Really Rather You Didn’ts

- Run at pool utilization close to 100%
- Log at debug level in production
- Use client classification expressions to enumerate a long list of specific clients
- Employ lengthy regular expressions
- Poll the rest api constantly (e.g. collecting statistics from thousands of subnets)
- Load hooks you don’t really need
- Run old versions - 2.0 got its number for a reason
- Run HA without enabling HA+MT processing

References

- Performance test results: https://reports.kea.isc.org
- Kea documentation: https://kea.readthedocs.io/en/latest/
- Kea performance webinar (Apr 2020):
  - Slides: https://www.isc.org/docs/KeaPerformance042220.pdf
  - Recording: https://youtu.be/ipUjlqg5pMY
- Description of the performance test design (from an earlier run of this test): https://kb.isc.org/docs/kea-20-performance-tests
- Knowledgebase article on Kea Performance Optimization: https://kb.isc.org/docs/kea-performance-optimization
- Kea-users mailing list: https://lists.isc.org/mailman/listinfo/kea-users

Questions?