POSTED ON MAY 28, 2019 TO DATA CENTER ENGINEERING, DATA INFRASTRUCTURE, NETWORKING & TRAFFIC, PRODUCTION ENGINEERING

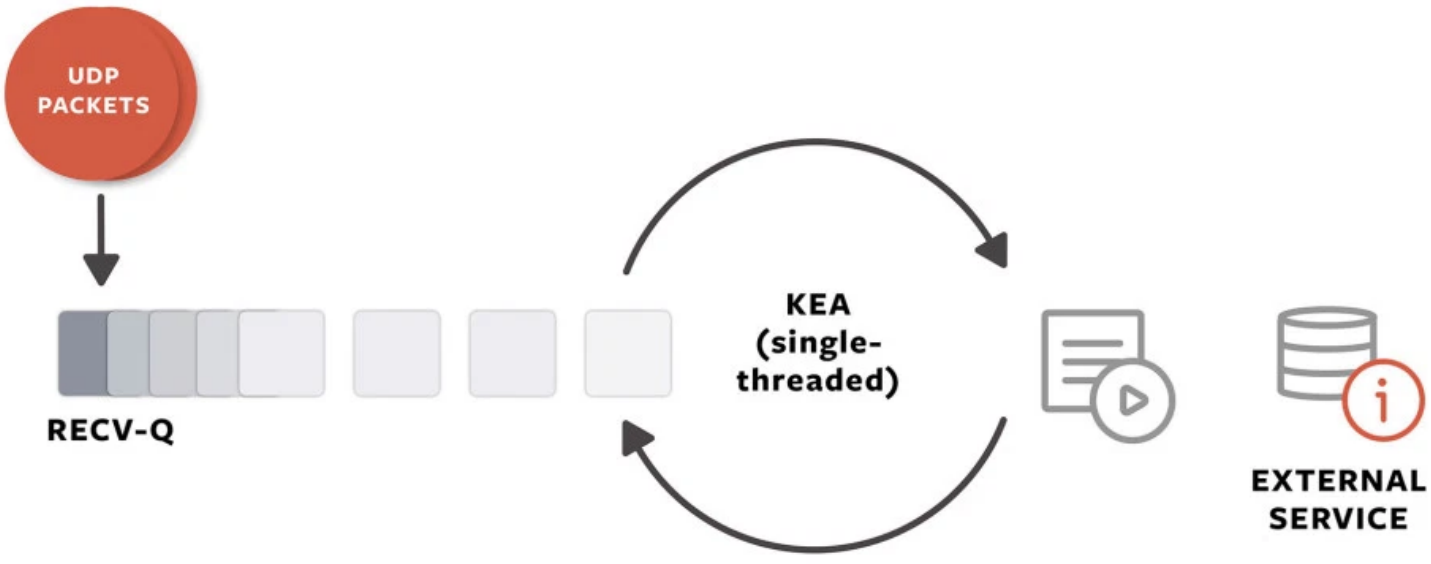
Extending DHCPLB: The path from load balancer to server

By Pablo Mazzini

DHCP is the network management protocol we use for provisioning servers in our production data centers. We use it for bare metal provisioning (to install the operating system) and to assign IP addresses to out-of-band management interfaces. As traffic demands have grown, production engineers have evolved our DHCP infrastructure to make it more scalable. For many years, we ran the Kea DHCP software, then implemented an open source load balancer to more evenly distribute traffic. Our latest incarnation replaces Kea entirely and extends DHCPLB to operate as a server. With this version, we've seen better throughput and are able to iterate faster than we could with our previous solution. In fact, we are now handling the same volume of traffic with 10 times fewer servers.

The Recv-Q buffer

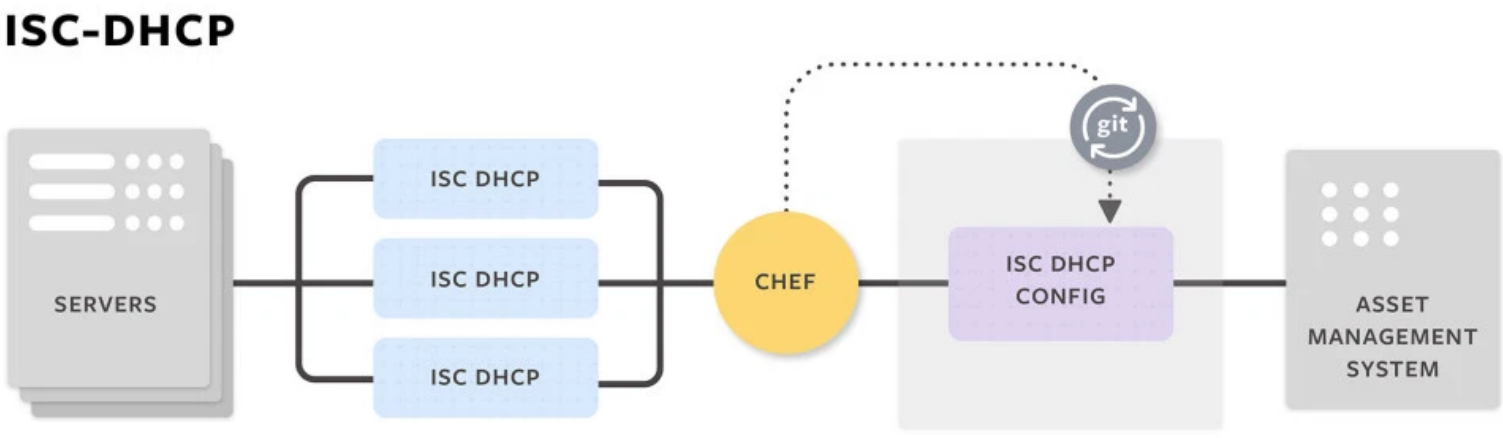
One of the main issues we ran into with Kea is that it's a single-threaded server. When a request is served, the process blocks while incoming requests queue up in the kernel Recv-Q buffer. Once the buffer gets full, incoming requests start to get dropped. This backup is especially problematic when Kea is doing backend calls to another service. The single-threaded nature of the software means that only a single transaction may be processed at a time; thus, if each backend call takes 100ms, then a Kea instance will be capable of doing, at maximum, 10 queries per second (QPS).



If the number of packets in the Recv-Q increases, it implies the application running on the system is not picking up the data being signaled by the kernel fast enough. To avoid maxing out the system's Recv-Q buffer and having packets drop, we were constantly adding more Kea servers. It was not scaling well for us, so we had to step back and think about an alternate solution.

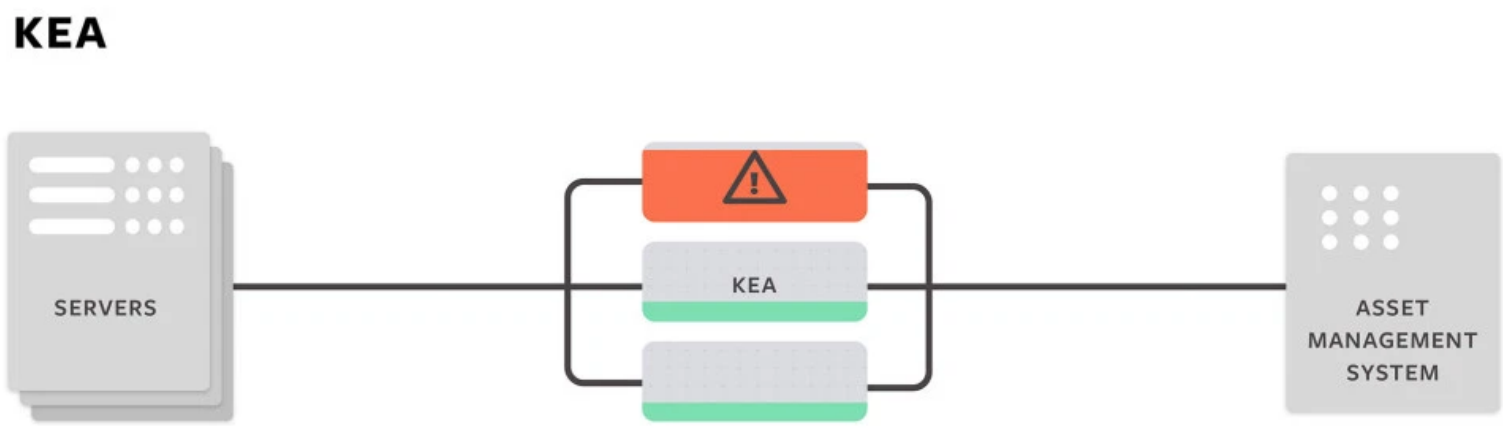
Around that time, one of our engineers attended a RIPE NCC hackathon alongside other companies with a large infrastructure. While there, he saw a lack of DHCP server-side alternatives. So he decided to create a DHCP library written in Go. When he shared his library, we saw the potential for using it in production. Because DHCPLB is also written in Go, with a multithread design, the library was one we could leverage to overcome the performance problem we were facing.

How the DHCP infrastructure evolved



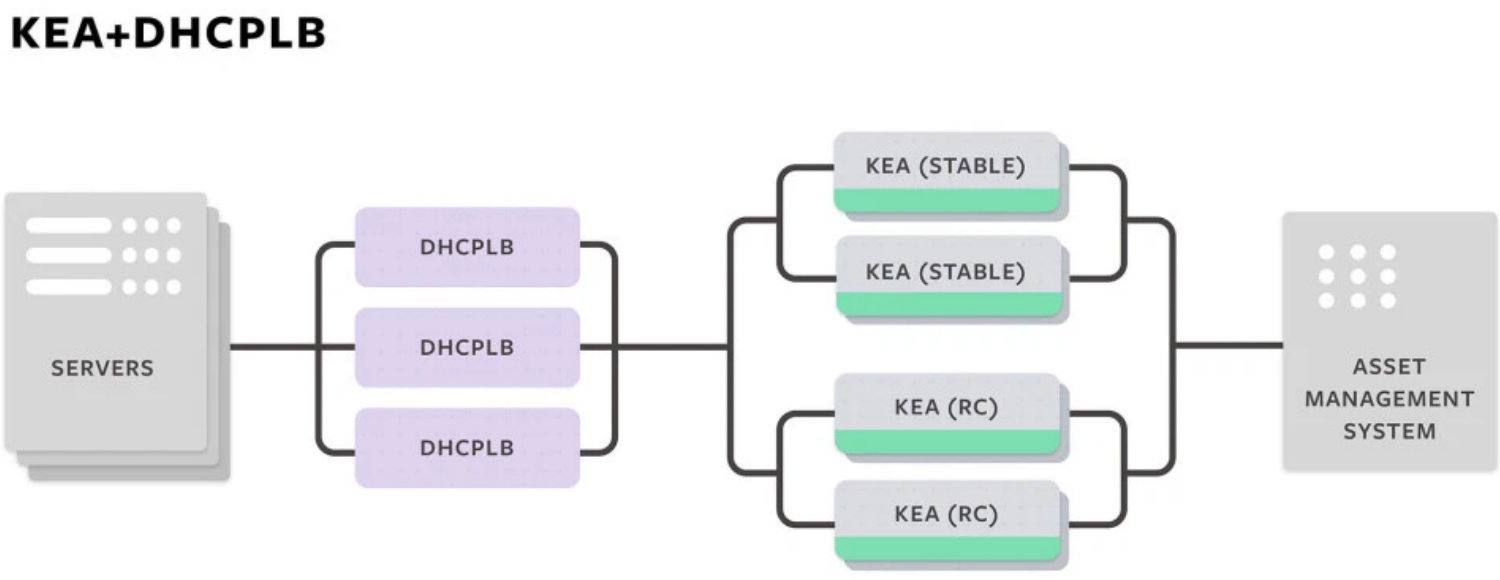
Our first system was based on ISC-DHCP and static configuration files periodically generated by a pipeline. The pipeline fetched assets from our inventory system and generated ISC-DHCP configuration files, then propagated them to each DHCP server. We loaded them into our DHCP servers using a complex git/rsync-based periodic pipeline, restarting the DHCP servers to pick up the changes.

Given the nature of the production environment, the addition, repair, and decommissioning of servers happens frequently, which translates into configuration changes. The DHCP servers were spending more time restarting, so they could pick up the changes, than they were serving actual traffic.



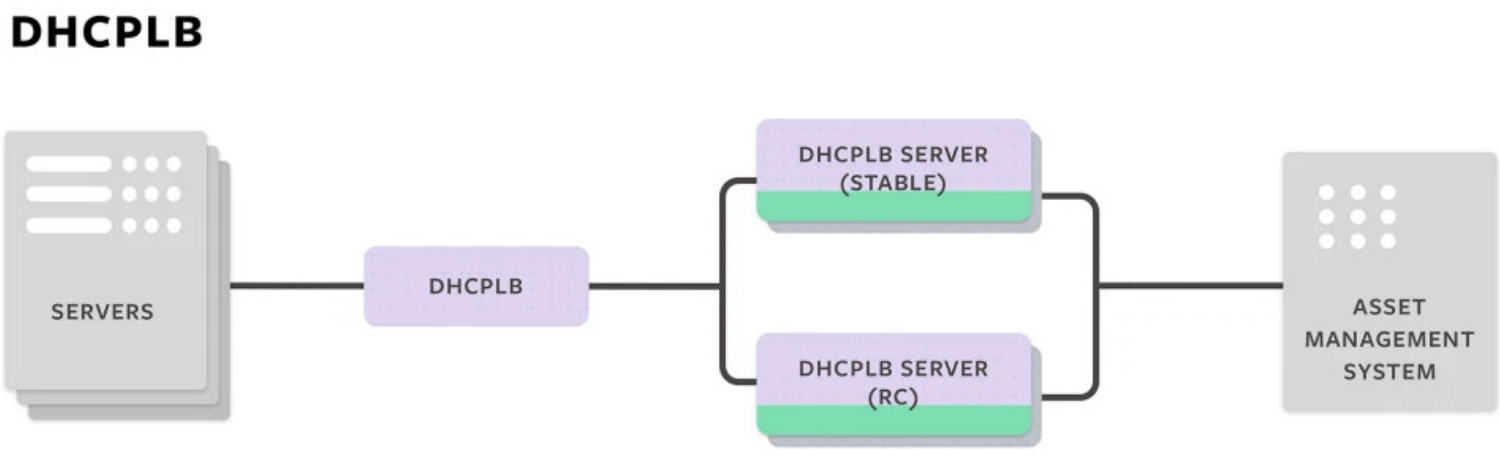
In 2014, we migrated our DHCP infrastructure onto ISC Kea to centralize configuration data and avoid long config files. We liked that ISC Kea was designed to be extensible: It has hook points where you can register callbacks to implement your own logic. We leveraged the hooks feature to fetch the information dynamically from our inventory system. We made each Kea server advertise a global Anycast IP using ExaBGP. Every DHCP relay in our infrastructure (typically running on our rack switches) is configured to relay all DHCP traffic to the above Anycast IP.

After a while, however, we discovered that traffic was not evenly balanced across all Kea instances. The imbalance was related to the Anycast IP distribution. We used BGP/ECMP, which cares only about the topology of the network it is operating on and the number of hops between the source and destination; it does not take into account the number of requests hitting each path. We didn't have our Kea instances evenly distributed in our network topology.



DHCPLB is a relay that balances incoming requests across a list of DHCP servers. It also allows us to define two pools of DHCP servers: stable and release candidate (RC); and set the proportion of traffic each one receives. This allowed us to A/B test any changes made to the Kea server. This setup worked well, but we still had Kea's single-threaded application doing back-end calls. As mentioned, we were experiencing frequent bottlenecks and needed a better solution.

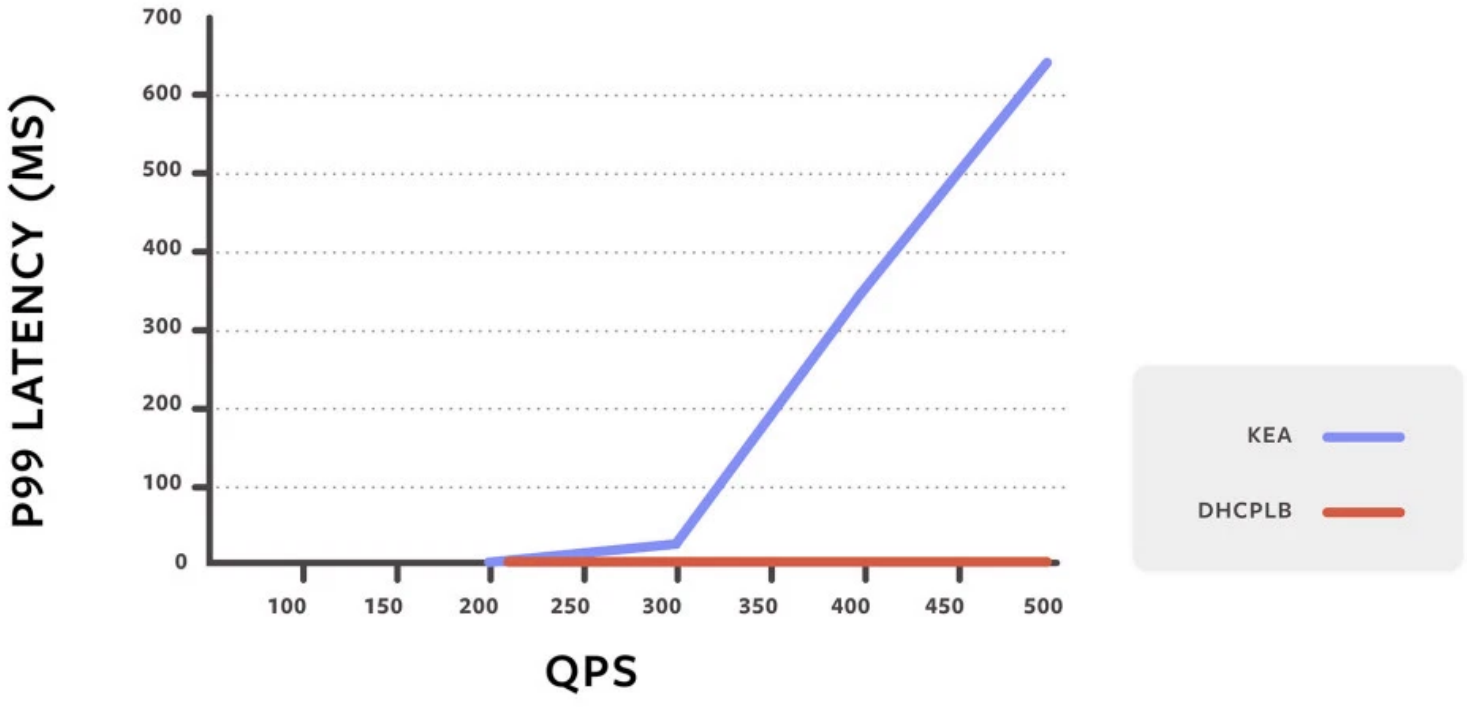
Building the DHCPLB server



Once we had the new library, we set out to extend DHCPLB to make it responsible for serving requests. The new setup allowed us to take advantage of the multithread design to prevent the new server from blocking and queuing up packets when doing back-end calls. We first had to make changes to DHCPLB to add a new mode that doesn't forward packets but instead generates responses. The open source DHCPLB project is the framework we use for our own implementation. It now provides an interface for writing your own handler to craft DHCP responses and make calls to a back-end system.

Having the load balancer in front of the servers helped with the development, testing, and rollout of the new servers. DHCPLB gives us the ability to A/B test changes on the server implementation. Even after we rolled out DHCPLB, we continued to run the Kea servers in parallel so we could monitor error logs until we were confident that the replacement would be at least as reliable as Kea had been. We have since deprecated the Kea DHCPv6 server.

The graph below shows that, for Kea, latency increases as QPS increases, until it starts timing out and dropping requests. For DHCPLB, it stays constant, showing that our move to DHCPLB allows us to handle a significantly higher QPS load without timeouts or failures. Performance was validated using our open source load-testing tool FBender.



In addition, this testing helped us improve not only DHCPLB but also the new DHCP library. DHCPLB was one of the first large-scale projects using the library. It helped us detect and fix bugs, and add quite a few improvements to make it more comprehensive. Since then, the library has gained traction, and its use has expanded to other projects.

We have seen great success with our internal implementation. We are handling more QPS and no longer have problems with the Recv-Q buffer. We are now sharing our new DHCPLB server framework, which you can download here. We invite the community to contribute to this project via the GitHub issue tracker.

Related Posts

Mar 14, 2019

Reinventing Facebook's data center network

Mar 02, 2019

Building backbone network infrastructure

Jul 19, 2018

Location-Aware Distribution: Configuring servers at scale

Related Positions

Data Center Systems Thermal Engineer

MENLO PARK, US

Workforce Planning Program Manager, Mission Control

MENLO PARK, US

Architect, Data Center Design

DUBLIN, IRELAND

Enterprise Support Tech

DUBLIN, IRELAND

Data Center Facilities Engineering, Electrical Engineer

WASHINGTON, US

See All Jobs

Join Our Engineering Community

Available Positions

Data Center Systems Thermal Engineer
MENLO PARK, US
Workforce Planning Program Manager, Mission Control
MENLO PARK, US
Architect, Data Center Design
DUBLIN, IRELAND
Enterprise Support Tech

Stay Connected

Facebook Engineering

Like

@fbOpenSource

Open Source

Facebook believes in building community through open source technology. Explore our latest projects in Artificial Intelligence, Data Infrastructure, Development Tools, Front End, Languages, Platforms, Security, Virtual Reality, and more.

ANDROID

DUBLIN, IRELAND

Data Center Facilities Engineering, Electrical Engineer

WASHINGTON, US

See All Jobs

Follow

facebook research

Facebook Research

Like

Facebook Developers

Like

RSS

Subscribe

iOS

WEB

BACKEND

HARDWARE

Learn More