POSTED ON SEP 13, 2016 TO DATA CENTER ENGINEERING, DATA INFRASTRUCTURE, NETWORKING & TRAFFIC, PRODUCTION ENGINEERING

DHCPLB: An open source load balancer

By Angelo Failla

Facebook's DHCP infrastructure and traffic imbalance

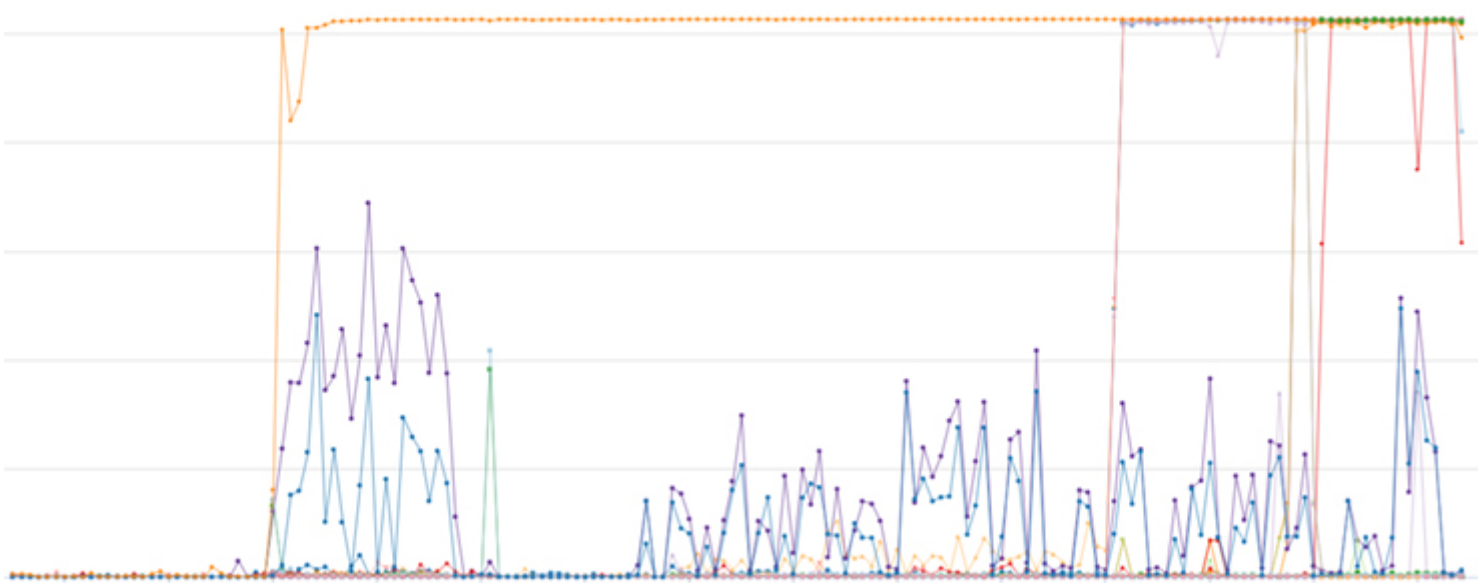
Last year we talked about our production DHCP infrastructure and how we use ISC KEA in production to deploy a stateless and dynamic DHCP server across the fleet. At the time, we used a hash-based ECMP selection algorithm for our BGP anycast IP distribution.

A question arose about whether the DHCP traffic was well-balanced across our KEA servers. We didn't know the answer, but upon follow-up we found it was not well-balanced. Our dashboards reflected this. The graph below shows requests per second among servers in the same data center, where a single server is taking most of the traffic.

We concluded that the imbalance was related to anycast and the network topology. We did not place our KEA instances symmetrically inside our fabric network to get more evenly distributed traffic. This was due to limitations of our container infrastructure, which does not have a view of our network topology down to the fabric pod. Load distribution when using anycast depends on where the clients are calling from and where the receivers are located.

BGP/ECMP 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. Balancing happens only if the receivers have the same distance from the senders.

Earlier this year we were faced with an incident in which racks in a data center became flooded with DHCP requests and only the closest KEA server was getting hit (the orange line in the graph below), while the others were basically idle. This led to high response latency initially and eventually to systems not being able to obtain their IPs.



We had to find a way to fix this imbalance. We decided to tackle this problem during an upcoming hackathon.

The Dublin Infra Hack: Building a proof of concept

A small team of engineers assembled for our Dublin Infra Hack this past April and spent two days hacking on a proof of concept for a DHCP load balancer that would sit behind our anycast VIP and source a list of DHCP servers from a file, trying to balance incoming requests. We deployed the dhcplb prototype in a cluster and tested a few provisioning jobs — and it worked! We landed the code in our repository but knew we still had more work to do. For one thing, we had only been able to tackle `DHCPv4` during the hackathon and wanted to build out additional features.

The internship project: Productionize the prototype

An intern joined the Cluster Infrastructure team in Dublin this summer and, working with two production engineers on the project, managed to ship the following features:

- **DHCPv6 support:** DHCPv6 is the protocol used in v6-only environments; DHCPv4 isn't supported. Because we've been turning on v6-only clusters over the past few years, building DHCPv6 support was the priority at the beginning of the project. Dhcplb implements DHCPv6 relaying as per RFC3315.
- **Better ways to test new versions of KEA:** Prior to having dhcplb, testing new versions of KEA was difficult, as there was no convenient way to redirect a set of machines to a test version of KEA to validate code changes and fixes. Dhcplb allows for the following:
    - **A/B testing support:** You can define two pools of DHCP servers in its configuration: a release candidate (RC) and a STABLE one. In the same configuration file there is a setting to change the proportion of traffic that hits RC versus STABLE. The decision is made by hashing the TransactionID field in the client's request message. A/B testing support is useful to validate new versions of our KEA DHCP servers. We build a new version of KEA twice a week. We always push it to RC first with a very low ratio (typically around 5 percent), and we have dashboards and monitoring to pick up on regressions before we increase the ratio. When we see that no regressions are there, we promote RC to STABLE.
    - **MAC address-based overrides:** Another useful feature of dhcplb is the possibility to override its balancing decision. This is done via a file in which you can specify how to forward requests coming from a given MAC or list of MACs to a DHCP server of your choice. This is useful for the debugging/development phase, as it allows us to reproduce problems locally on a development machine and fix them in isolation.

- **Integration with our monitoring infrastructure:** dhcplb logs every request, every exception, and performance data to Scuba. Scuba is a system we developed for doing real-time, ad-hoc analysis of data sets. Scuba allows us to get a sense of how well (or poorly) our DHCP infrastructure is performing, to determine where the bottlenecks are, and more.
- **All configuration files are hot-reloaded:** dhcplb makes use of the Go fsnotify library so that a restart is not required when a change is made in its configuration.

- **BGP advertisement**: dhcplb advertises its VIP talking to ExaBGP via a background goroutine.
- **Two balancing algorithms:** dhcplb implements two balancing algorithms, a RoundRobin one and a ModuloN one. The latter is what we currently use in production, and it's defined in lib/modulo.go. The decision is based on the DHCP Transaction ID for a session (the Transaction ID is a randomly generated number the client generates when it initiates a connection). The TransactionID is used to compute a hash, which is then involved in a modulo operation against the number of servers to decide which server to pick:

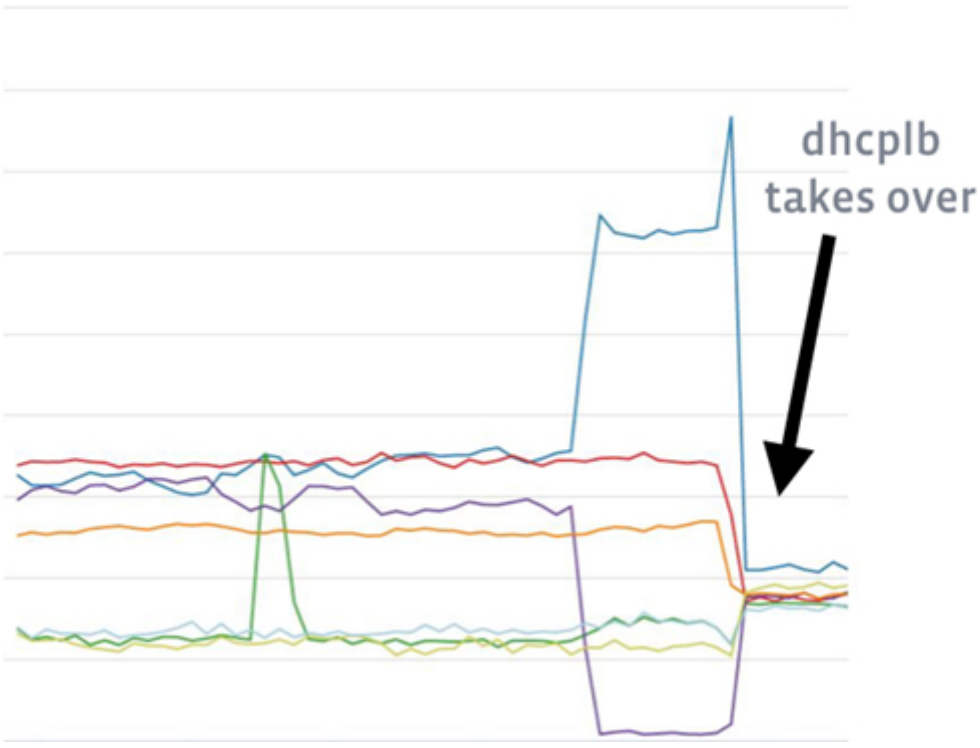
```
import (
    "errors"
    "hash/fnv"
    "sync"
)

func (m *modulo) selectServerFromList(list []*DHCPServer, message *DHCPMessage) (*DHCPServer, error) {
    m.lock.RLock()
    defer m.lock.RUnlock()
    if len(list) == 0 {
        return nil, errors.New("Server list is empty")
    }
    // New32a returns a new 32-bit FNV-1a hash.Hash
    hasher := fnv.New32a()
    // Write adds more data to the running hash
    hasher.Write(message.ClientID)
    // Sum32 returns the resulting hash
    hash := hasher.Sum32()
    // Pick one server in list[] *DHCPServer
    return list[hash%uint32(len(list))], nil
}
```

Rolling out to production

We rolled out dhcplb four weeks before the end of the internship and immediately saw improvements.

The graph below shows how dhcplb is doing a better job at balancing DHCP requests more evenly across our KEA dhcp servers:



We are now able to apply an A/B testing model to our KEA servers as well as redirect given MAC addresses to the KEA servers we want, which has made testing KEA changes simpler. If we want to, we can black-hole bad actors or throttle them appropriately.

Using dhcplb at Facebook

TORs (top-of-rack switches) at Facebook run DHCP relayers, which are responsible for relaying broadcast DHCP traffic (DISCOVERY and SOLICIT messages) originating within their racks to anycast VIPs, one for DHCPv4 and one for DHCPv6.

At Facebook we use Open Compute rack switches running FBOSS. However, if you are using, for example, a Cisco switch, the configuration would look like this:

```
ip helper-address 10.10.10.67
ipv6 dhcp relay destination 2001:db8:85a3::8a2e:370:67
```

Because the VIP is unique, it makes the TOR configuration simpler, and this part of the configuration doesn't need to be templatized.

We have `dhcplb` Tupperware instances in every region listening on those VIPs (using the ExaBGP advertiser). They are responsible for receiving traffic relayed by TOR agents and load-balancing them among the actual KEA DHCP servers, distributed across clusters in that same region.

The configuration for `dhcplb` consists of three files:

- **JSON config file:** Contains the main configuration for the server, as explained in the Getting Started section on GitHub. It's hot-reloaded when it changes.

```
{
  "v4": {
    "version": 4, // DHCP operation mode
    "listen_addr": "0.0.0.0", // address to bind the receiving socket to
    "port": 67, // port to listen on
    "packet_buf_size": 1024, // size of buffer to allocate for incoming
packet
    "update_server_interval": 30, // how often to refresh server list (in
seconds)
    "free_conn_timeout": 30, // how long to wait after removal before
closing a connection to a server (in seconds)
    "algorithm": "xid", // balancing algorithm, supported are xid and rr
(client hash and roundrobin)
    "host_sourcer": "file:hosts-v4.txt", // load DHCP server list from
hosts-v4.txt
    "rc_ratio": 0 // what percentage of requests should go to RC servers
  },
  "v6": { // same options for v6, only the ones that are different are
showed
    "version": 6, // DHCP operation mode
    "listen_addr": "::", // address to bind the receiving socket to
    "port": 547, // port to listen on
    "host_sourcer": "file:hosts-v6.txt", // load DHCP server list from
hosts-v6.txt
  }
}
```

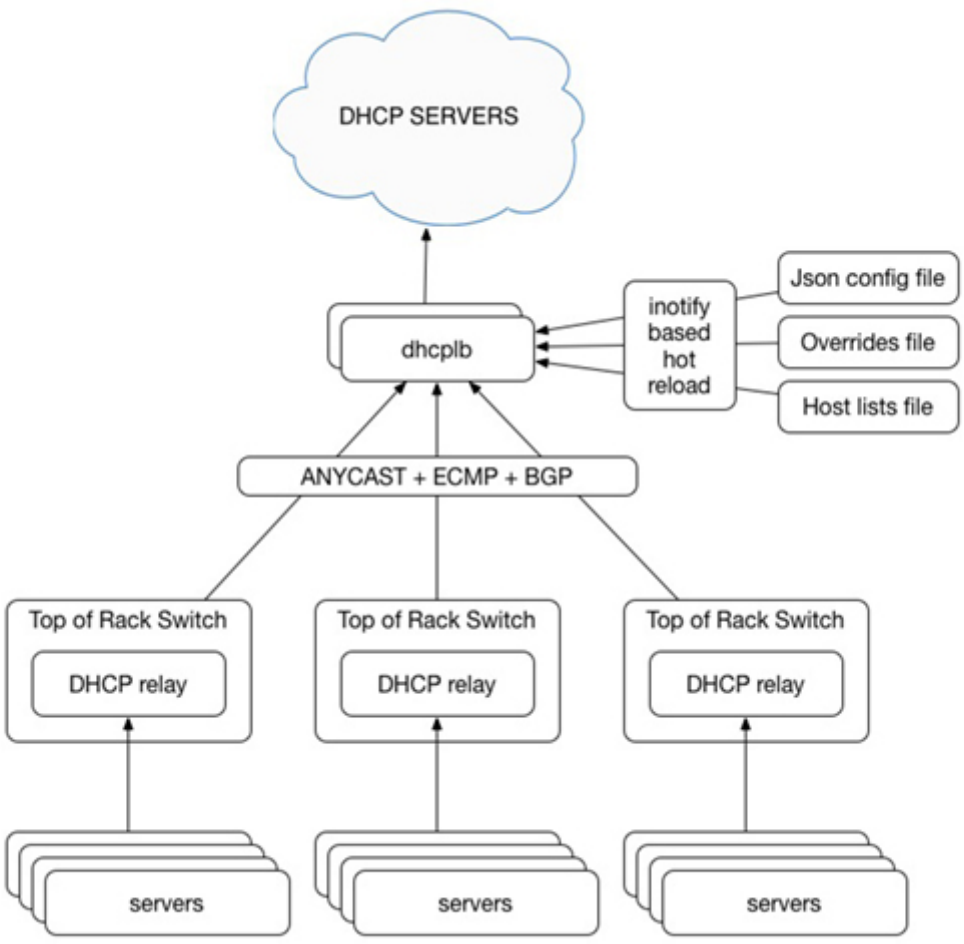
- **Host lists file:** Contains a list of DHCP servers, one per line, that `dhcplb` will try to balance on. The open source version hot-reloads the file when the file changes, but it is trivial to extend dhcplb and adapt to your infrastructure (for example by having it source the list of DHCP servers from your internal systems).

- **Overrides file**: This file contains per MAC overrides and is hot-reloaded and managed by Configerator, our application management system (you can read the paper here). See the Getting Started section on GitHub for details.

```
{
  "v4": {
    "12:34:56:78:90:ab": {
        "host": "1.2.3.4"
    }
  },
  "v6": {
    "12:34:56:78:90:ab": {
        "host": "2001:0db8:85a3:0000:0000:8a2e:0370:7334"
    }
  }
}
```

With this overrides file, DHCPv4 requests coming from MAC `12:34:56:78:90:ab` will be sent to the DHCP server at `1.2.3.4`, and a DHCPv6 request from the same MAC will go to server `2001:0db8:85a3:0000:0000:8a2e:0370:7334`.

The architecture is represented in the figure below:



Try it out!

Dhcplb is written in Go and can be fetched from GitHub here.

To install `dhcplb` in your `$GOPATH`, simply run:

```
$ go get github.com/facebookincubator/dhcplb
```

This will fetch the source code and write it into `$GOPATH/src/github.com/facebookincubator/dhcplb`, compile the binary, and put it in `$GOPATH/bin/dhcplb`.

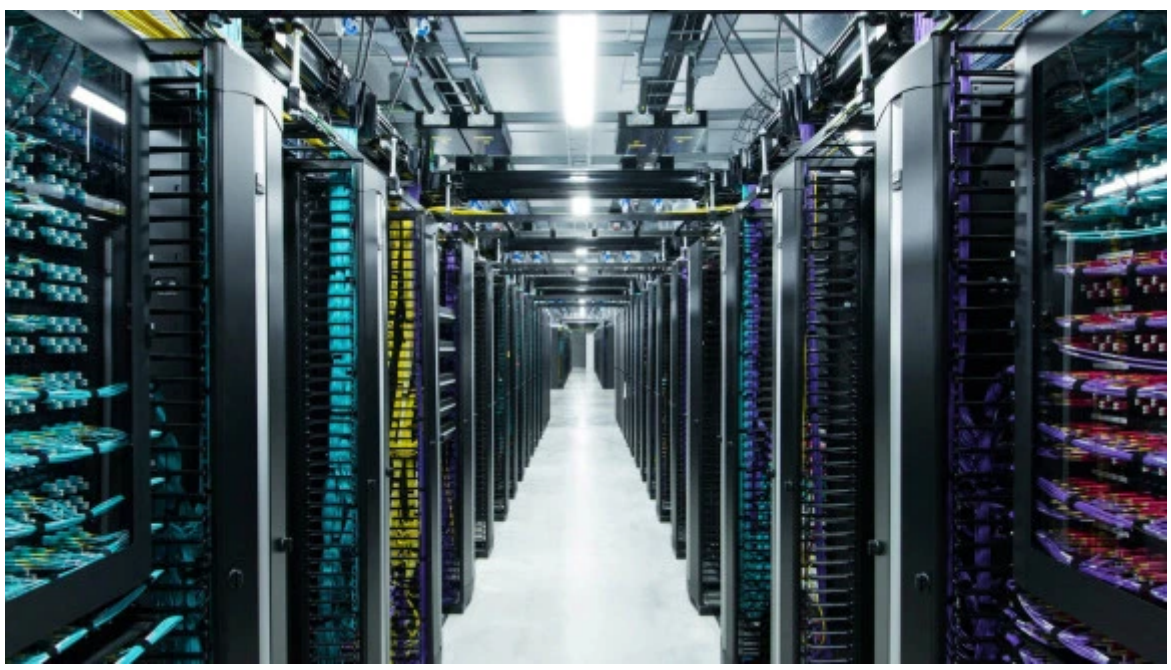
If you want to extend dhcplb to satisfy the needs of your infrastructure, you can read the Getting Started guide.

TAGS: OPEN SOURCE

Related Posts

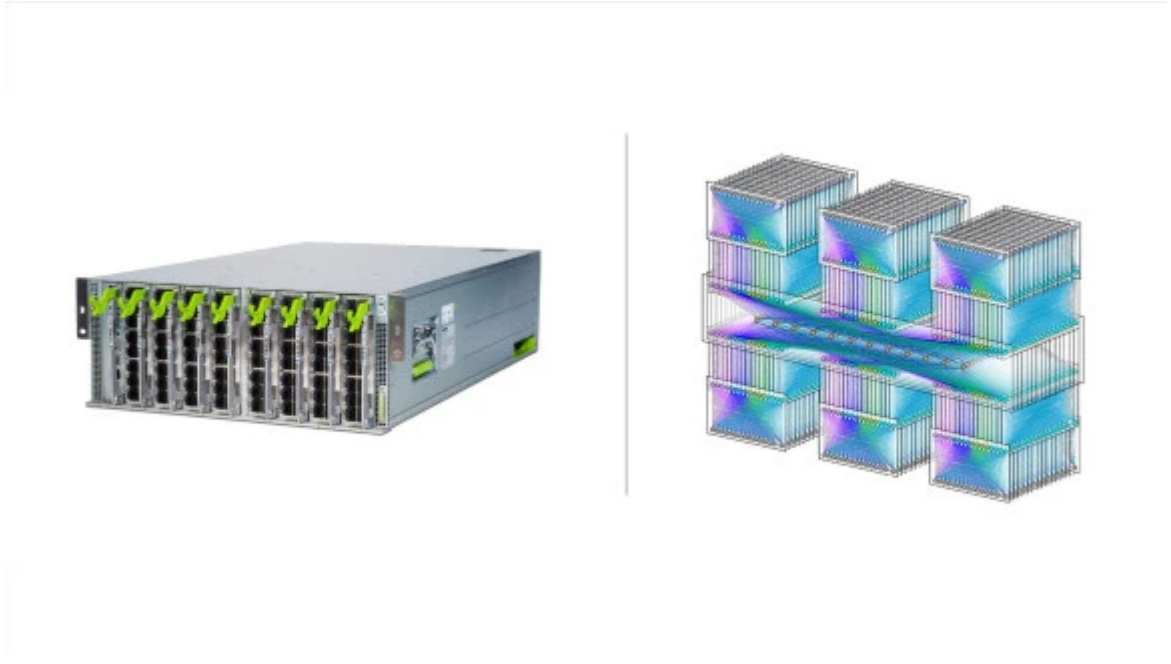
May 28, 2019

Extending DHCPLB: The path from load balancer to server



May 21, 2018

Scaling the Facebook backbone through Zero Touch Provisioning



Mar 14, 2019

Reinventing Facebook's data center network

Related Positions

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
DUBLIN, IRELAND
Data Center Facilities Engineering, Electrical Engineer
WASHINGTON, US

See All Jobs

Stay Connected

Facebook Engineering

Like

@fbOpenSource

Follow

facebook research

Facebook Research

Like

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

iOS

WEB

BACKEND

HARDWARE

Learn More

Facebook Developers

Like

RSS

Subscribe