IPVS for **Docker** Containers

Production-level load balancing and request routing without spending a single penny.

UBER

What is IPVS

And why didn't I hear about it before?

- Stands for **IP Virtual Server**. Built on top of Netfilter and works in kernel space. *No userland copying of network packets involved.*
- It's in the mainline Linux Kernel since 2.4.x. Surprisingly, it's one of the technologies which successfully pumps your stuff through world's networks for **more than 15 years** and almost nobody ever heard of it.
- Used in many world-scale companies such as Google, Facebook, LinkedIn, Dropbox, GitHub, Alibaba, Yandex and so on. During my time in Yandex, we used to route **millions of requests per second** through a few IPVS boxes without a sweat.
- Tested with fire, water and brass trombones (untranslatable Russian proverb).
- Provides flexible, configurable load-balancing and request routing done in kernel space – it plugs in even before PCAP layer – making it **bloody fucking fast**.
- As all kernel tech, it looks like a incomprehensible magical artifact from outer space, but bear with me – it's actually very simple to use.

What is IPVS

And what can it do if you say it’s so awesome?

- Supports TCP, SCTP and UDP (both v4 and v6), works on L4 in kernel space.
- Load balancing using weighted RR, LC, replicated locality-based LC (LBLCR), destination and source-based hashing, shortest expected delay (SED) and more via plugins:
  - **LBLCR**: «sticky» least-connections, picks a backend and sends all jobs to it until it’s full, then picks the next one. If all backends are full, replicates one with the least jobs and adds it to the service backend set.
- Supports persistent connections for SSL, FTP and One Packet Mode for connectionless protocols such as DNS – will trigger scheduling for each packet.
- Supports request forwarding with standard Masquerading (DNAT), Tunneling (IPIP) or Direct Routing (DR). Supports Netfilter Marks (FWMarks) for **virtual service aggregation**, e.g. http and https.
- Built-in cluster **synchronization daemon** – only need to configure one router box, other boxes will keep their configurations up-to-date automatically.

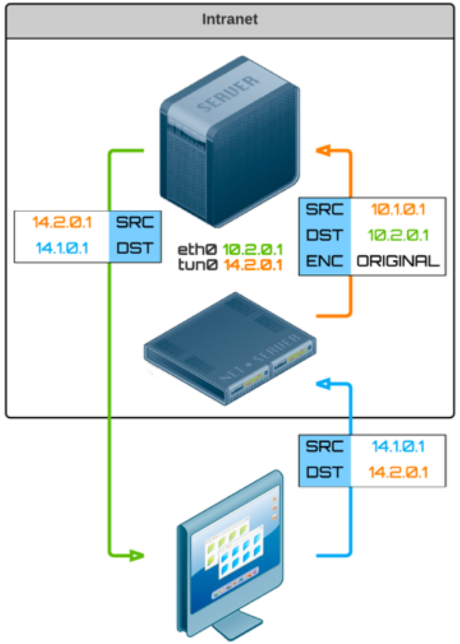
What is IPVS

And a little bit more about all these weird acronyms.

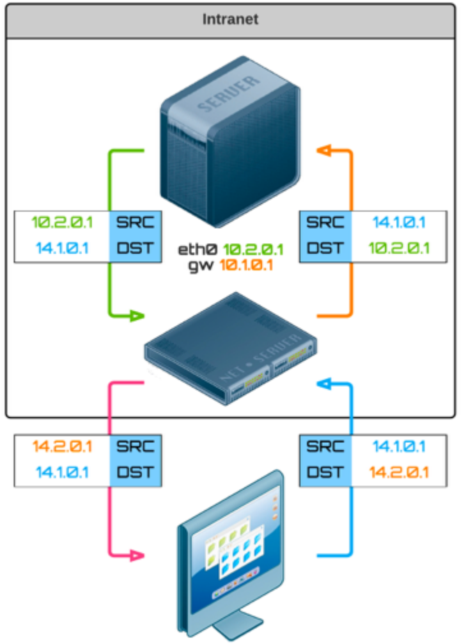
- **DNAT** is your ordinary NAT. It will replace each packet's destination IP with the chosen backend IP and put the packet back into the networking stack.
    - *Pros*: you don't need to do anything to configure it.
    - *Cons*: responses will be a **martian packets** in your network. But you can use a local reverse proxy to workaround this or point the default gateway on backends to load balancer's IP address for IPVS to take care of it.
- **DR** (or DSR: direct server response) is the fastest forwarding method in the world.
    - *Pros*: speed. It will be as fast as possible given your network conditions. In fact, it will be so fast that you will be able to **pump more traffic through your balancers than the interface capacity they have**.
    - *Cons*: all backends and routers should be in the same L2 segment, need to configure NoARP devices.
- **IPIP** is a **`tun/tap`** alternative to DR. Like IPSEC, but without encryption.
    - *Pros*: it **can be routed anywhere** and won't trigger **martian packets.**
    - *Cons*: lower MSS. Also, you need to configure NoARP **`tun`** devices on all backend boxes, like in DR.

What is IPVS

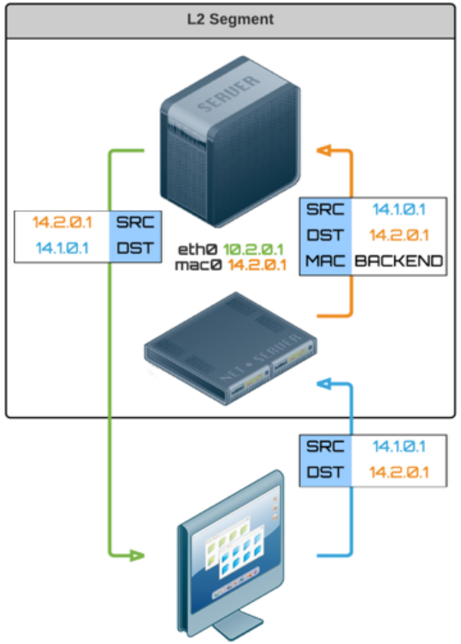
And a little bit more about all these weird acronyms.



IPIP
Encapsulates IP
Routable anywhere



DNAT
Rewrites DST IP
Same L4



DSR
Rewrites DST MAC
Same L2

What is IPVS

And a little bit more about DR since it’s awesome!

- Works by replacing the job’s destination MAC with the chosen backend MAC and then putting the packet back into the INPUT queue. Once again: the requirement for this mode is that all backends should be in the same L2 network segment.
- **The response will travel directly to the requesting client**, bypassing the router box. This essentially allows to cut router box traffic by more than 50% since responses are usually heavier than requests.
- This also allows you to **load balance more traffic than the load balancer interface capacity** since with a proper setup it will only load balance TCP SYNs.
- It’s a little bit tricky to set up, but if done well is indistinguishable from magic (as well as any sufficiently advanced technology).
- **Fun fact**: this is how video streaming services survive and avoid bankruptcy!

I don't need this

And why would I load balance and route anything at all?

- If you have more than one instance of your application, you need load balancing, otherwise you won't be able to distribute requests among those instances.
- If you have more than one application or more than one version deployed at the same time in production, you need request routing. Typically, only one version is deployed to production at the same time, and one testing version is waiting in line.
- Imagine what you can do if you could **deploy *142* different versions of your application** to production:
    - **A/B testing**: route 50% to version A, route 50% to version B.
    - **Intelligent routing**: if requester's cookie is set to *«NSA»*, route to version *«HoneyPot»*.
    - **Experiments**: randomly route 1% of users to experimental version *«EvenMorePointlessAds»* and stick them to that version.

I don't need this

Also, my **nginx** (**haproxy**, third option) setup works fine, get off the stage please!

- In modern world, instances come and go as they may, sometimes multiple times per second. This rate will only go up in the future.
- Instances migrate across physical nodes. Instances might be stopped when no load or low load is in effect. New instances might appear to accommodate the growing request rate and be torn down later.
- Neither **nginx** (for free) nor **haproxy** allows for online configuration. Many hacks exist which essentially regenerate the configuration file on the routing box and then cycle the reverse proxy, but it's neither sustainable nor mainstream.
- Neither **hipache** (written in node.js) nor **vulcand** (under development) allows for non-RR balancing methods.
- The fastest userland proxy to date – HAProxy – is ~40% slower than IPVS in DR mode according to many publicly available benchmark results.

I don’t need this

And I run my stuff in the cloud, it takes care of everything – my work is perpetual siesta.

- Lucky you!
- But running stuff in the cloud is not an option in many circumstances:
    - Performance and/or reliability critical applications.
    - Exotic hardware or software requirements. CG or CUDA farms, BSD environments and so on.
    - Clouds are a commodity in US and Europe, but in some countries with a thriving IT culture AWS is more of a fairy tale (e.g. Vietnam).
    - Security considerations and vendor lock-in.
- **Cloud-based load balancers are surprisingly dumb**, for example AWS ELB doesn’t really allow you to configure anything except real servers and some basic stuff like SSL and Health Checks.
- After a certain point, AWS/GCE becomes quite expensive, unless you’re Netflix.

What is IPVS, again

And how do I use it now since it sounds amazing!

- Make sure you have a properly configured kernel (usually you do):
  - `cat /boot/config-$(uname -r) | grep IP_VS`
- Install IPVS CLI to verify that it's healthy and up:
  - `sudo apt-get install ipvsadm`
  - `sudo ipvsadm -l`
- Command-line tool allows you to do everything out there with IPVS, but it's the 21st century and CLIs are only good to show off your keyboard-fu ;)
- That's why I've coded a **dumb but loyal REST API daemon** that talks to kernel and configures IPVS for you. In Go, because, you know.
- You can use it to add and remove virtual services and backends in runtime and get metrics about existing virtual services. It's totally **open source and free** but might not work (as everything open source and free).

GORB

Go Routing and **B**alancing.

- It’s here: https://github.com/kobolog/gorb
- Based on native Go `netlink` library – https://github.com/tehnerd/gnl2go. The library itself talks directly to the Kernel and already supports the majority of IPVS operations. This library is a port of Facebook’s gnl2py – https://github.com/facebook/gnlpy
- It exposes a very straightforward JSON-based REST API:
    - `PUT or DELETE /service/<vsID>` – create or remove a virtual service.
    - `PUT or DELETE /service/<vsID>/<rsID>` – add or remove a backend for a virtual service
    - `GET /service/<vsID>` – get virtual service metrics (incl. health checks).
    - `GET /service/<vsID>/<rsID>` – get backend configuration.
    - `PATCH /service/<vsID>/<rsID>` – update backend weight and other parameters without restarting anything.

GORB

And why is it cool for Docker Containers.

- Every time you spin up a new container, it can be **magically automatically registered with GORB**, so that your application's clients could reach it via the balancer-provided endpoint.
- GORB will **automatically do TCP checks** (if it's a TCP service) on your container and inhibit all traffic coming to it if, for some reason, it disappeared without gracefully de-registering first: network outage, oom-killer or whatnot.
- **HTTP checks are also available**, if required – e.g. your app can have a `/health` endpoint which will respond `200` only if all downstream dependencies are okay as well.
- Essentially, the only thing you need to do is to **start a tiny little daemon on Docker Engine boxes** that will listen for events on Events API and send commands to GORB servers.
- More checks can be added in the future, e.g. Consul, ZooKeeper or etcd!

GORB

And how do I use it? Live demo or GTFO!

```
kobolog@bulldozer:~$ docker-machine create -d virtualbox gorb

kobolog@bulldozer:~$ docker-machine ssh gorb sudo modprobe ip_vs

kobolog@bulldozer:~$ eval $(docker-machine env gorb)

kobolog@bulldozer:~$ docker build -t gorb src/gorb

kobolog@bulldozer:~$ docker run -d —net=host --privileged gorb -f -i eth1

kobolog@bulldozer:~$ docker build -t gorb-link src/gorb/gorb-docker-link

kobolog@bulldozer:~$ docker run -d --net=host -v /var/run/docker.sock:/var/run/
docker.sock gorb-link -r $(docker-machine ip default):4672 -i eth1

kobolog@bulldozer:~$ docker run -d -p 80 nginx (repeat 4 times)

kobolog@bulldozer:~$ curl -i http://$(docker-machine ip gorb):80
```

A few words about BGP

Black belt in networking is not complete without a few words about BGP

- Once you have a few router boxes to **avoid having a SPOF**, you might wonder how clients would find out which one to use. One option could be **DNS RR**, where you put them all behind one domain name and let DNS resolvers to do their job.
- But there's a better way – BGP host routes, also known as **anycast routing**:
    - It's a protocol used by network routers to agree on network topologies.
    - The idea behind anycast is that each router box announces the **same IP address** on the network via BGP host route advertisements.
    - When a client hits this IP address, it's automatically routed to one of the GORB boxes based on configured routing metrics.
    - You don't need any special hardware – it's already built into your routers.
    - You can do this using one of the BGP packages, such as **Bird** or **Quagga**. Notable ecosystem projects in this area: Project Calico.
    - Also can be done with **IPv6 anycast**, but I've never seen it implemented.

GORB

Is it stable? Is it production-ready? Can I blame you if it doesn’t work?

- No, you cannot blame me, but I’m here to help and answer questions!
- **IPVS is production ready since I was in college**.
- GORB is not production ready and is not tested in production, but since it’s just a configuration daemon, it won’t actually affect your traffic, I hope.
- Here are some nice numbers to think about:
    - 1 GBPS line rate uses 1% CPU in DR mode.
    - Can utilize 40G/100G interface.
    - Costs you €**0**.
    - Typical *enterprise* hardware load-balancer – €**25,000**.
- Requires **generic hardware**, software, tools – also easy to learn. Ask your Ops whether they want to read a 1000-page vendor manual or a 1000-word manpage.
- Also, **no SNMP** involved. Amazing and unbelievable!

This guy on the stage

Who the hell are you and why should I believe a Russian?

- **You shouldn't**. In fact, don't trust anybody on this stage – get your hands dirty and verify everything yourself. Although last month I turned 30, so I'm trustworthy!
- I've been building distributed systems and networking for **more than 7 years** in companies with worldwide networks and millions of users, multiple datacenters and other impressive things.
- These distributed systems and networks are still operational and serve **billions of user requests** each day.
- I'm the author of IPVS-based routing and load balancing framework for Yandex's very own open-source platform called **Cocaine**: https://github.com/cocaine.
- As of now, I'm a Senior Infrastructure Software Engineer in **Uber** in NYC, continuing my self-proclaimed crusade to make magic infrastructure look less magical and more useful for humans.

Gracias, Cataluña

Some links, more links, some HR and **questions**!

- **This guy**:
  - Twitter (it's boring and mostly in Russian): @kobolog
  - Questions when I'm not around: me@kobology.ru
  - My name is Andrey Sibiryov.
- **IPVS**: http://www.linuxvirtualserver.org/software/ipvs.html
- **GORB sources**: https://github.com/kobolog/gorb
- **Bird BGP**: http://bird.network.cz
- **Stage version of this slide deck:** http://bit.ly/1S1A3cT
- Also, it's right about time to **ask your questions**!