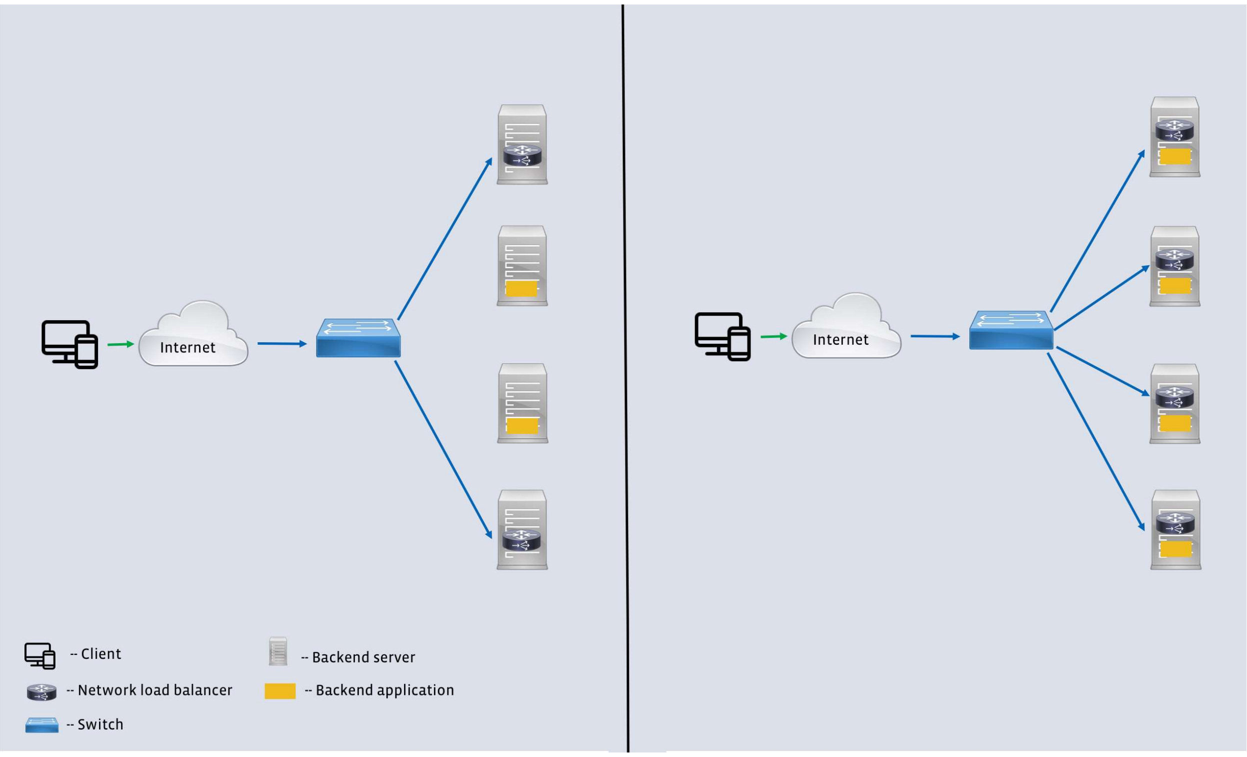
POSTED ON MAY 22, 2018 TO OPEN SOURCE

Open-sourcing Katran, a scalable network load balancer



By Nikita Shirokov Ranjeeth Dasineni

Katran creates a software-based solution to load balancing with a reengineered forwarding plane that takes advantage of recent innovations in kernel engineering.

With billions of people around the globe using Facebook services, our infrastructure engineers have created a range of systems to optimize traffic and to enable fast, reliable access for everyone. Today, we are open-sourcing a component of this work by releasing the Katran forwarding plane software library, which powers the network load balancer used in Facebook's infrastructure. Katran offers a software-based solution to load balancing with a completely reengineered forwarding plane that takes advantage of two recent innovations in kernel engineering: eXpress Data Path (XDP) and the eBPF virtual machine. Katran is deployed today on backend servers in Facebook's points of presence (PoPs), and it has helped us improve the performance and scalability of network load balancing and reduce inefficiencies such as busy loops when there are no incoming packets. By sharing it with the open source community, we hope others can improve the performance of their load balancers and also use Katran as a foundation for future work.

The challenge of serving requests at Facebook scale

To manage traffic at Facebook scale, we have deployed a globally distributed network of points of presence to act as proxies for our data centers. Given the extremely high volume of requests, both PoPs and data centers confront the challenge of making the large fleet of (backend) servers appear as a single virtual unit to the outside world and also distributing the workload efficiently among those backend servers.

These challenges are typically addressed by announcing a virtual IP address (VIP) to the internet at each location. Packets destined to the VIP are then seamlessly distributed among the backend servers. The distribution algorithm, however, needs to account for the fact that the backend servers typically operate at an application layer and terminate the TCP connections. This responsibility is handled by a network load balancer (often called a layer 4 load balancer, or an L4LB, because it operates on packets rather than serving application level requests). Figure 1 illustrates the role of an L4LB in relation to the other network components.

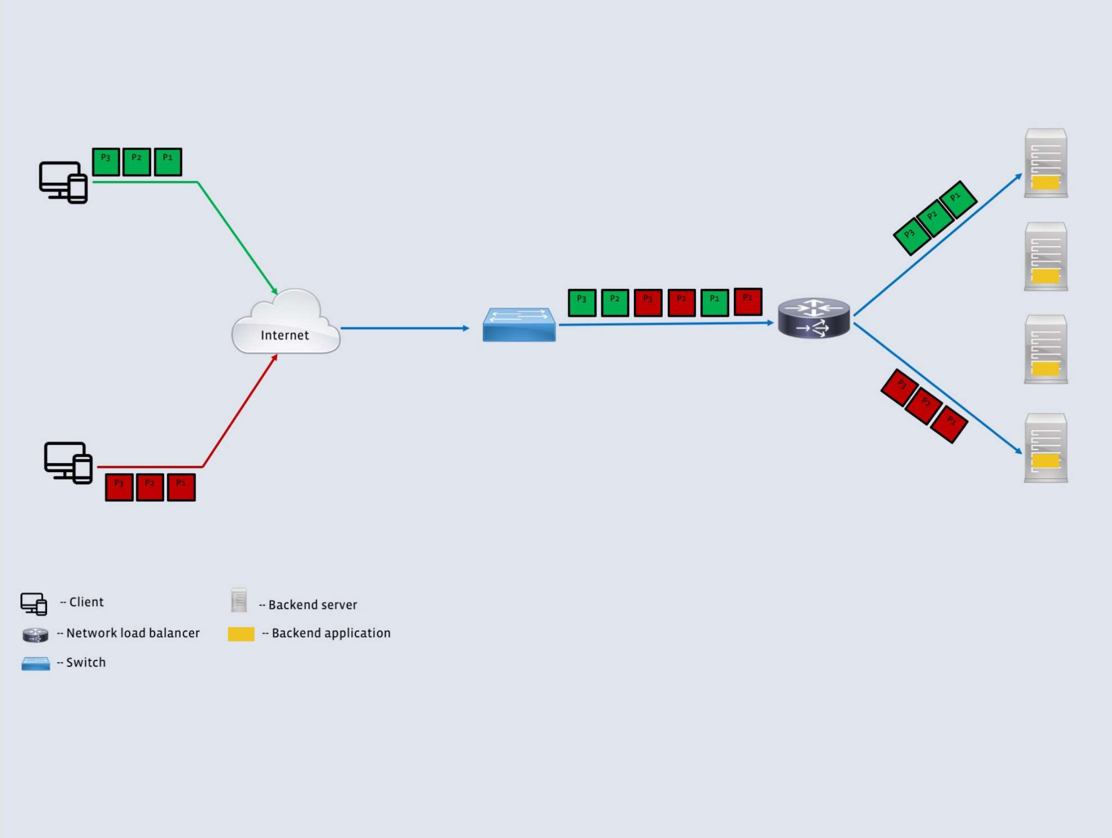


Figure 1: A network load balancer fronts several backend servers running a backend application and consistently sends all packets from each client connection to a unique backend server.

Requirements for a high-performance load balancer

An L4LB's performance is especially important for managing latency and scaling the number of backend servers, because the L4LBs are on a path that needs to process every incoming packet. Performance is typically measured as peak packets per second (pps) that the L4LB can process. Traditionally, engineers have preferred hardware-based solutions for this task because they typically use accelerators such as application-specific integrated circuits (ASICs) or field-programmable gate arrays (FPGAs) to reduce the burden on the main CPU. However, one of the drawbacks of a hardware-centric approach is that it limits the system's flexibility. To effectively serve Facebook's needs, a network load balancer must:

- **Run on commodity Linux servers.** This allows us to run the load balancer on part or all of the large fleet of currently deployed servers. A software-based load balancer satisfies this criteria.
- **Coexist with other services on a given server.** This removes the need for dedicated servers that run the load balancer exclusively, thereby increasing fault tolerance.
- **Allow low-disruption maintenance.** Facebook's software must be able to evolve quickly in order to support new or improved products and services. Maintenance and upgrades are a norm, not exceptions, for the load balancer and backend layers. Minimizing disruption during these events allows us to iterate faster.
- **Offer easy instrumentation and debugging.** All large distributed infrastructures must contend with anomalies and unexpected events, so reducing the time to debug and troubleshoot issues is important. The load balancer needs to be instrumentable and friendly to standard tools like tcpdump.

In order to solve for these requirements, we designed a high-performance software network load balancer. The first generation of our L4LB was based on the IPVS kernel module and served Facebook's needs for well over four years. However, it fell short on the goal of coexistence with other services, specifically the backends. In the second iteration, we leveraged the eXpress Data Path (XDP) framework and the new BPF virtual machine (eBPF) to run the software load balancer together with the backends on a large number of machines. Figure 2 shows the key difference between the two generations.

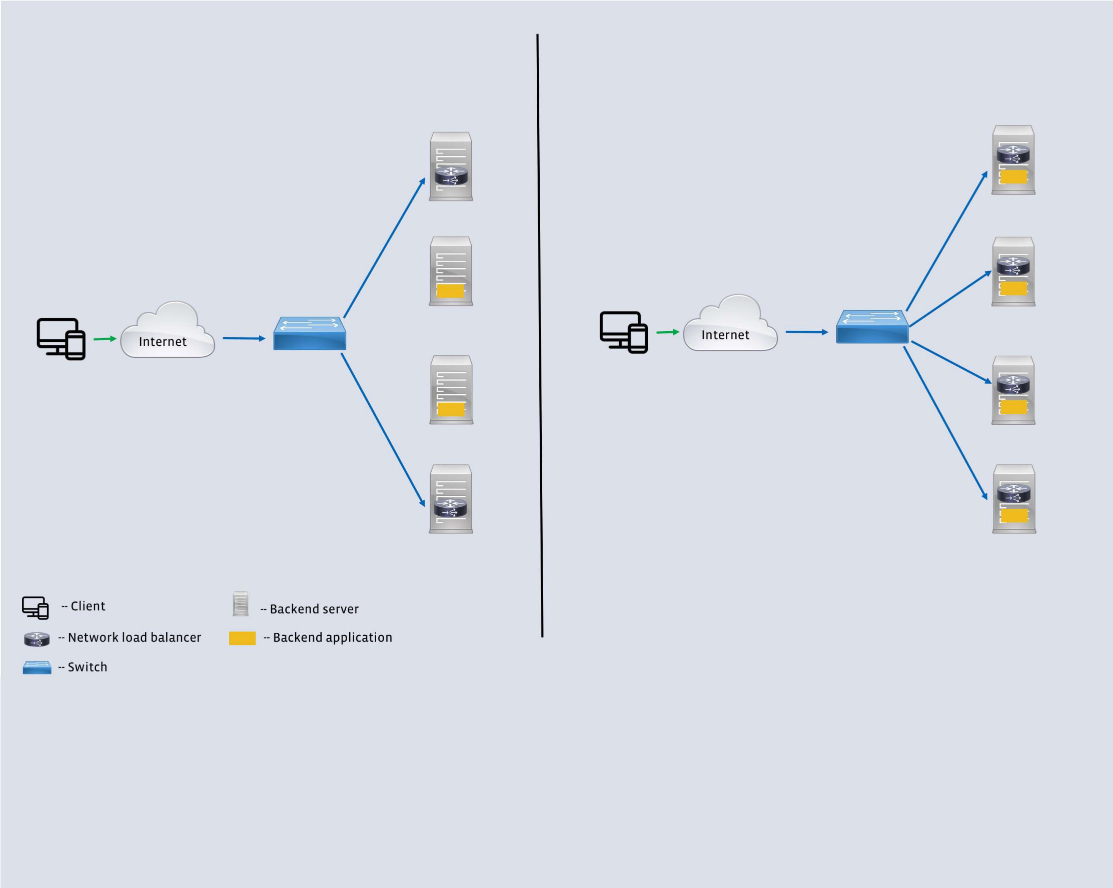


Figure 2: Differences between the two generations of L4LBs. Note that both are software load balancers running on backend servers. Katran (right) allows us to colocate the load balancer with backend application, thus increasing the load balancer capacity.

Our First-generation L4LB: Building on OSS Software

With our first-generation L4LB, we leaned heavily on existing open source components to implement most of the functionality. This approach helped us replace a hardware-based solution across a large deployment in only a few months. The design has four major components:

- **VIP announcement:** This component simply announces the virtual IP addresses that the L4LB is responsible for to the world by peering with the network element (typically a switch) in front of the L4LB. The switch then uses an equal-cost multipath (ECMP) mechanism to distribute packets among the L4LBs announcing the VIP. We used ExaBGP for the VIP announcement because of its lightweight, flexible design.
- **Backend server selection:** In order to send all packets from a client to the same backend, the L4LBs use a consistent hash that depends on the 5-tuple (source address, source port, destination address, destination port, and protocol) of the incoming packet. The use of a consistent hash ensures that all packets that belong to a transport connection are sent to the same backend irrespective of the L4LB receiving the packet. This removes the need for any state synchronization across multiple L4LBs. The consistent hash also guarantees minimal disruption to existing connections when a backend leaves or joins the pool of backends.

- **Forwarding plane:** Once the L4LB picks the appropriate backend, the packets need to be forwarded to that host. To avoid restrictions such as keeping L4LB and backend hosts on the same L2 domain, we use a simple IP-in-IP encapsulation. This allows us to place L4LB and backend hosts in different racks. We used the IPVS kernel module for the encapsulation. The backends are configured to have the corresponding VIP on their loopback interface. This allows the backend to send packets on the return path directly to the client (instead of the L4LB). This optimization, often called direct server return (DSR), allows the L4LB to be constrained only by the incoming packet volume.
- **Control plane:** This component performs various functions, including performing health checks on the backend servers, providing a simple interface (via a configuration file) to add or remove VIPs, and providing simple APIs to examine the state of the L4LB and backend servers. We developed this component in-house.

Each L4LB also stores the backend choice for each 5-tuple as a lookup table to avoid duplicate computation of the hash on future packets. This state is a pure optimization and is not necessary for correctness. This design met several requirements of Facebook's workload listed above, but there was one major drawback: Colocating the L4LB and a backend on a single host increased the chance of device failure. Even with the local state, the L4LB was a CPU-intensive component. To separate the failure domains, we ran the L4LBs and backend servers on a disjointed set of machines. There were fewer L4LBs than backend servers in this setup, which made the L4LBs more vulnerable to a sudden increase in load. The fact that packets had to traverse the regular Linux network stack before being handled by the L4LB exacerbated the problem.

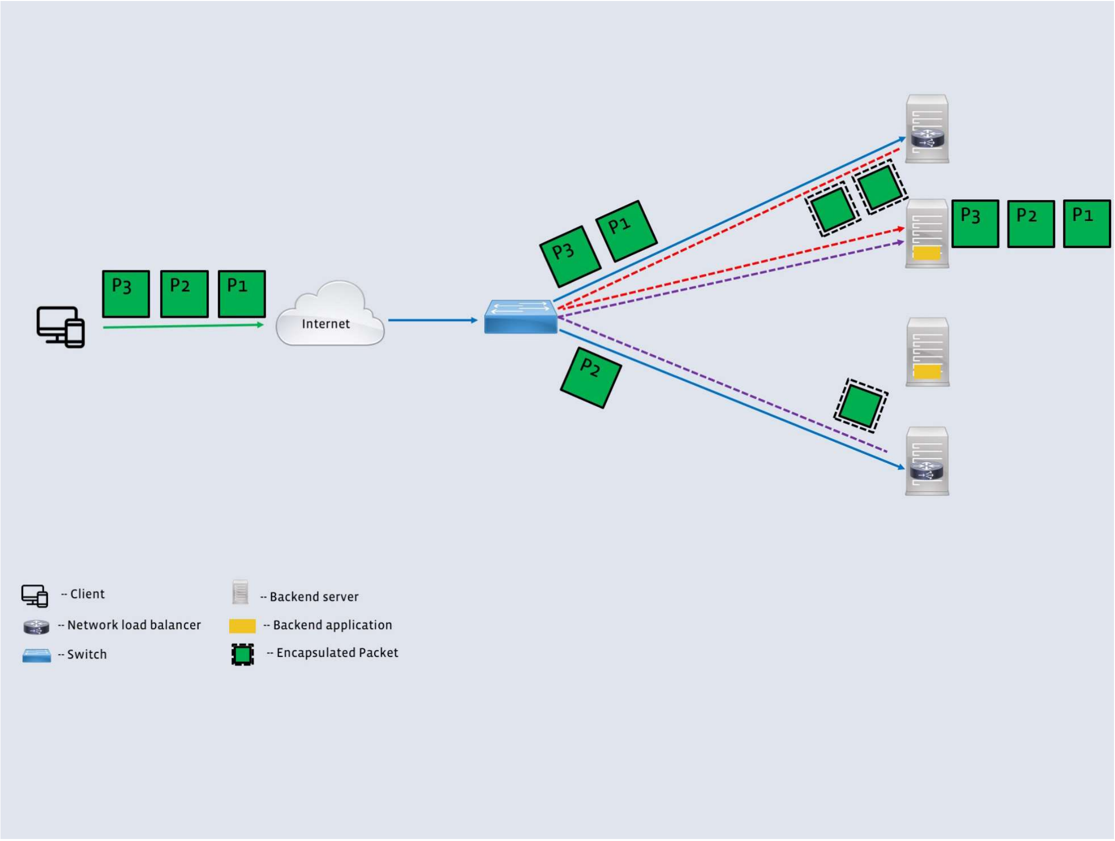


Figure 3: Overview of our first-generation L4LB. Note that the load balancer and the backend application run on different machines. Different load balancers make consistent decisions without any state synchronization. Using packet encapsulation allows the servers running the load balancer and the backend application to be placed in different racks. In a typical deployment, the ratio of the number of L4LBs to the number of backend application servers is very small.

Katran: Reimagining the forwarding plane

Katran, our second-generation L4LB, significantly improves upon the previous version with a completely reengineered forwarding plane. Two recent developments in the kernel world powered the new design:

- The XDP provides a fast, programmable network data path without resorting to a full-fledged kernel bypass method and works in conjunction with the Linux networking stack. (A detailed overview of XDP is available here.)
- The eBPF virtual machine provides a flexible, efficient, and more reliable way to interact with the Linux kernel and to extend its functionality by running user-space supplied programs at specific points in the kernel. eBPF has already brought dramatic improvements to several areas, including tracing and filtering. (More details are available here.)

The overall architecture of the system is similar to that of the first-generation L4LB: First, ExaBGP announces to the world which VIPS a particular Katran instance is responsible for. Second, packets destined to a VIP are sent to Katran instances using an ECMP mechanism. Finally, Katran selects a backend and forwards the packet to the correct backend server. The main differences are in the last step.

Early and efficient packet handling: Katran uses XDP in combination with a BPF program for packet forwarding. When XDP is enabled in driver mode, a packet handling routine (BPF program) is run immediately after a packet is received by the network interface card (NIC) and before the kernel intercepts it. XDP invokes the BPF program on every incoming packet. If the NIC has multiple queues, the program is invoked in parallel for each one them. The BPF program used for handling packets is lockless and uses a per-CPU version of BPF maps. Due to this parallelism, performance scales linearly with the number of the NIC's RX queues. Katran also supports the "generic XDP" mode (instead of driver mode) of operation, at a performance cost.

Inexpensive and more stable hashing: Katran uses an extended version of the Maglev hash to select the backend server. A few features of the extended hash are resilience to backend server failures, more uniform distribution of load, and the ability to set unequal weights for different backend servers. The last of these is an important feature that allows us to handle hardware refreshes in our PoPs and data centers easily: We can absorb the newer generation hardware by simply setting appropriate weights. Despite its being more expressive, the code for computing this hash is small enough to fit entirely in the L1 cache.

More resilient local state: Katran's efficiency at handling packets and computing the hash results in an interesting interaction with the local state table. We observed that, quite often, computing the hash is computationally easier than looking up the local state table for the 5-tuple to backend server choice. This is more visible for cases where the local state table lookup traverses all the way to the shared last level cache. In order to take advantage of this phenomenon in a natural way, we implemented the lookup table as an LRU-evicting cache. The LRU cache size is configurable at startup time and acts as a tunable parameter to strike a balance between computation and lookup. We picked these values empirically to optimize for pps. In addition, Katran provides a runtime "compute only" switch to ignore the LRU cache altogether in the event of catastrophic memory pressure on the host.

RSS-friendly encapsulation: Received Side Scaling (RSS) is an important optimization in NICs that aims to spread load across CPUs uniformly by steering packets from each flow to a separate CPU. Katran crafts its encapsulation to work in conjunction with RSS. Instead of using the same outer source for every IP-in-IP packet, packets in different flows (e.g., with different 5-tuples) are encapsulated using a different outer source IP, but packets in the same flow are always assigned the same outer source IP.

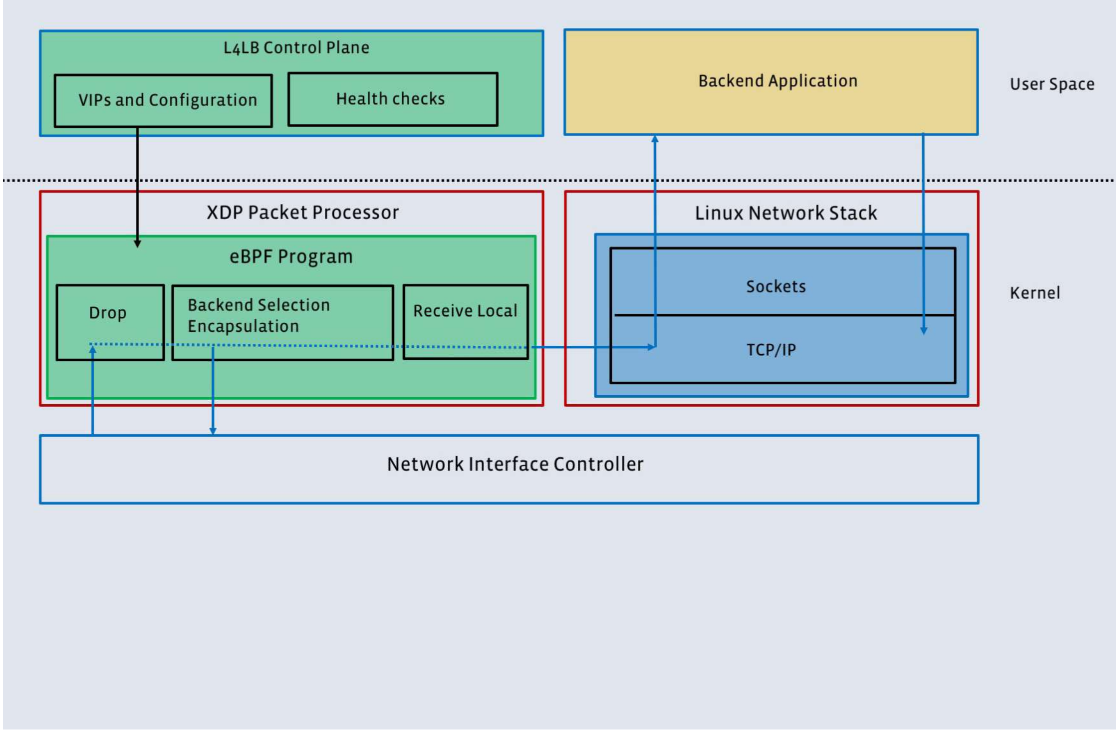


Figure 4: Katran enables a fast path for processing packets at high speed without resorting to a full-fledged kernel bypass. Note that the packets cross the kernel/user-space boundary only once. This allows us to colocate the L4LB and backend application without sacrificing performance.

These features dramatically enhance performance, flexibility and scalability of the L4LB. Katran's design also gets rid of busy loops on receive path barely consuming any CPU if there are no incoming packets. In contrast to a full-fledged Kernel Bypass solution (such as DPDK), using XDP allows us to run Katran alongside any application without any performance penalties on the same host. Katran today runs alongside the backend servers in our PoPs with an improved L4LB-to-backend ratio. This increases resilience to load spikes, host failures, and maintenance, as well. The reengineered forwarding plane was central to this shift. We believe other systems can benefit by using our forwarding plane, so we are open-sourcing our code and including several examples of how to use it to craft an L4LB.

Additional Considerations

Katran operates under certain assumptions and constraints that enable the performance improvements. In practice, we found these constraints to be fairly reasonable, and they did not block our deployment. We believe that most users of our library will find them easy to satisfy. We've listed them below:

- Katran works only in direct service return (DSR) mode.
- Katran is the component that decides the final destination of a packet addressed to a VIP so the network needs to route packets to Katran first. This requires the network topology to be L3 based, e.g., packets are routed by IP rather than by MAC addresses.
- Katran cannot forward fragmented packets, nor can it fragment them by itself. This could be mitigated either by increasing the maximal transmission unit (MTU) inside the network or by changing advertised TCP MSS from the backends. (The latter step is recommended even if you have increased the MTU.)
- Katran doesn't support packets with IP options set. The maximum packet size cannot exceed 3.5 KB.
- Katran was built with the assumption that it's going to be used in a "load balancer on a stick" scenario, where a single interface would be used for traffic both "from user to L4LB (ingress)" and "from L4LB to L7LB (egress)."

Despite these limitations, we believe that Katran offers an excellent forwarding plane to users and organizations who intend to leverage the exciting combination of XDP and eBPF to build efficient load balancers. We look forward to answering any questions from prospective adopters on our GitHub repository — and pull requests are always welcome!

Like Share 10K people like this. Sign Up to see what your friends like.

Related Posts

Oct 30, 2018
Facebook open-sources new suite of Linux kernel components and tools

Jun 20, 2018
2018 Networking @Scale recap

Jan 02, 2019
Open source year in review

Related Positions

Director, Software Engineering
LONDON, UNITED KINGDOM

Instagram - Manager, Software Engineering
MENLO PARK, US

Technical Investigator, Business Integrity
SEATTLE, US

Manager, Performance and Capacity Engineering
MENLO PARK, US

WhatsApp Software Engineer, Android
MENLO PARK, US

See All Jobs

Join Our Engineering Community

Available Positions

Director, Software Engineering
LONDON, UNITED KINGDOM
Instagram - Manager, Software Engineering
MENLO PARK, US
Technical Investigator, Business Integrity
SEATTLE, US
Manager, Performance and Capacity Engineering
MENLO PARK, US
WhatsApp Software Engineer, Android

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

MENLO PARK, US

See All Jobs

Follow

Facebook Research

Like

Facebook Developers

Like

RSS

Subscribe

iOS

WEB

BACKEND

HARDWARE

Learn More