Load balancing using eXpress Data Path

Luca Bassi
luca@argoware.com

September 27, 2023

Outline

eBPF and XDP

XDP programs

Direct Server Return

Hashing

Test results

ECMP

A distributed load balancer

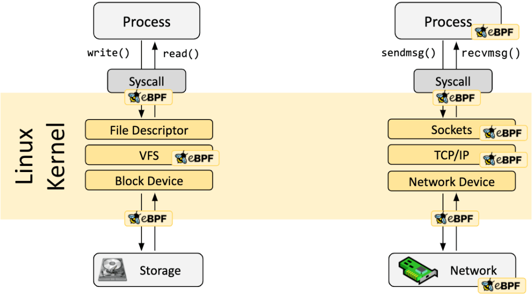
Thesis and acknowledgements

eBPF

eBPF permits to run sandboxed programs in the OS kernel, mainly event-driven.

It's used to extend the kernel without recompiling it or use modules.

Safety is provided through static code analysis.

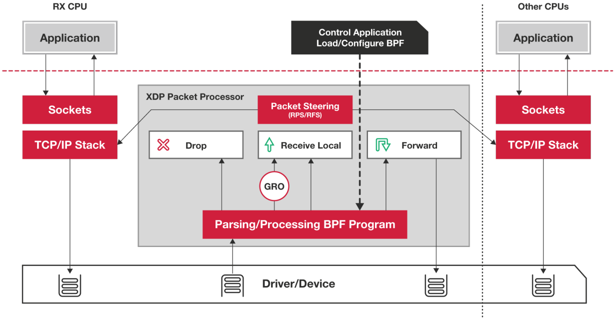


Copyright eBPF.io authors

eXpress Data Path

eXpress Data Path provides a high performance, programmable network data path in the Linux kernel.

It provides bare metal packet processing at the lowest point in the software stack, which makes it ideal for speed.



Copyright IO Visor

eXpress Data Path

The XDP program returns an action code to tell the kernel what to do with the packet:

- `XDP_PASS`
- `XDP_DROP` and `XDP_ABORTED`
- `XDP_TX` and `XDP_REDIRECT`

First XDP program

This program lets all packets pass to the kernel network stack.

```
#include <linux/bpf.h>
#include <bpf/bpf_helpers.h>

SEC("xdp")
int xdp_pass(struct xdp_md *ctx)
{
  return XDP_PASS;
}
```

First XDP program

We can compile it with clang using the -target bpf option, for example:

```
clang -g -Wall -Wno-compare-distinct-pointer-types \
    -target bpf -O2 -c xdp_pass.c -o xdp_pass.o
```

And load it with xdp-loader included in the xdp-tools:

```
sudo xdp-loader load interface_name xdp_pass.o
```

First XDP program

If we replace XDP_PASS with XDP_DROP, all incoming packets will be dropped.

This will happen before the kernel network stack, so for example these packets will be "invisible" also to tcpdump.

Fortunately, we can use xdpdump for debugging XDP programs.

```
xdpdump -i interface_name --rx-capture entry,exit -x
```

Maps

Maps are the method used by eBPF programs to store and retrieve data.

Maps can be accessed from applications in user space via syscalls.

```
struct {
  __uint(type, BPF_MAP_TYPE_PERCPU_ARRAY);
  __uint(max_entries, 2);
  __type(key, __u32);
  __type(value, long);
  //__uint(pinning, LIBBPF_PIN_BY_NAME);
} count SEC(".maps");
```

Examining a packet

```
void *data_end = (void *)(long)ctx->data_end;
void *pos = (void *)(long)ctx->data;
struct ethhdr *eth = pos;
if (eth + 1 > data_end) {return -1;}
__u16 h_proto = eth->h_proto;
pos = eth + 1;
__u32 key;
if (h_proto == bpf_htons(ETH_P_IP)) {
  struct iphdr *ip = pos;
  if (ip + 1 > data_end) {return -1;}
  __u8 protocol = ip->protocol;
  if (protocol == IPPROTO_ICMP) {
    key = 0;
    long *value = bpf_map_lookup_elem(&count, &key);
    if (value) {*value += 1;}
  }
} else if (h_proto == bpf_htons(ETH_P_IPV6)) {
  /* ... */
}
return XDP_PASS;
```

Reading a map from user space

```c
int fd = bpf_obj_get("/sys/fs/bpf/test/count");
if (fd < 0) {
  printf("Error bpf_obj_get\n");
  return fd;
}
int nr_cpus = libbpf_num_possible_cpus();
long sum[] = { 0, 0 };
for (__u32 key = 0; key <= 1; ++key) {
  long values[nr_cpus];
  if ((bpf_map_lookup_elem(fd, &key, values)) != 0) {
    printf("Error bpf_map_lookup_elem\n");
    return -1;
  }
  for (int i = 0; i < nr_cpus; ++i) {
    sum[key] += values[i];
  }
}
printf("IPv4: %d\nIPv6: %d\n", sum[0], sum[1]);
```

Redirecting packets

Redirecting packets, using the XDP_TX (or XDP_REDIRECT) return code, can be used for implementing a load balancer.

```
__builtin_memcpy(eth->h_dest, destination_mac,
  sizeof(destination_mac));
ip->daddr = destination_ip;
__builtin_memcpy(eth->h_source, load_balancer_mac,
  sizeof(load_balancer_mac));
ip->saddr = load_balancer_ip;
ip->check = iph_csum(ip);
return XDP_TX;
```

Direct Server Return

We can use direct routing to have outbound traffic served directly from backend servers.
All the backend servers and the load balancer need to use the same virtual IP.
The backend servers must not announce the virtual IP, otherwise some packets may go directly to the server, bypassing the load balancer.

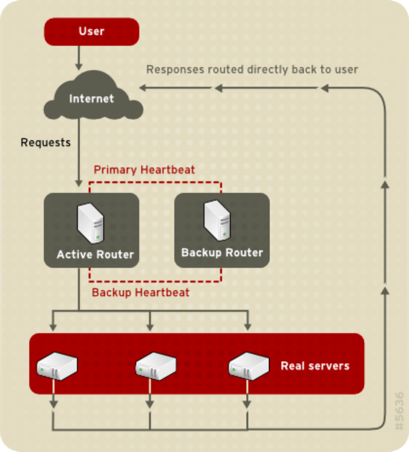


Image by Red Hat / CC BY-SA 3.0

Hashing

We want that all the packets of a specific connection go to the same backend server.

We can hash some information of the packet and use the resulting hash to select the backend server.

For example, we can hash: source and destination IPs and ports.

To minimise the number of reassignments in case of adding or removing a server, we must use a consistent hashing algorithm, for example rendezvous hashing.

Rendezvous hashing

Rendezvous or highest random weight (HRW) hashing is an algorithm that allows clients to achieve distributed agreement on a set of k options (in this case 1 server) out of a possible set of n options (the backend servers).

The idea is to assign each server a score for each request and assign that request to the server with the highest score.

You can assign a weight that acts as a multiplier to each server.

If a server is removed, the algorithm will simply select the server with the second-highest score in cases where the server removed was the one with the highest score, while the selection will remain unchanged in other cases.

Test results

100 000 GET requests per second

Latency	Without LB	LB (MurmurHash3)	LB (xxHash)
min	174 µs	205 µs	183 µs
mean	365 µs	410 µs	366 µs
50	285 µs	321 µs	301 µs
90	469 µs	509 µs	424 µs
95	783 µs	832 µs	609 µs
99	1864 µs	2040 µs	1586 µs

Comparison of latency without load balancer, with load balancer using MurmurHash3 as the hashing function, and with load balancer using xxHash instead.

Test results

10 requests per second of upload of a 100 MB file

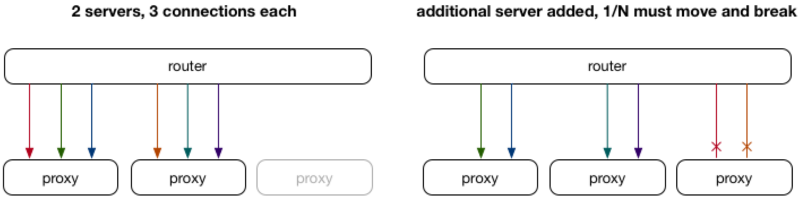
	Without LB	LB (MurmurHash3)	LB (xxHash)
Mean latency	111.712 ms	100.203 ms	108.317 ms
Mean speed	7.83 Gpbs	7.82 Gbps	7.82 Gbps
Max speed	8.11 Gbps	8.08 Gbps	8.13 Gpbs

The mean latency row reports the data from Vegeta, instead the speeds are obtained with nload.

ECMP

Routers have a feature called Equal-Cost Multi-Path (ECMP) routing, which is designed to split traffic destined for a single IP across multiple links of equal cost.

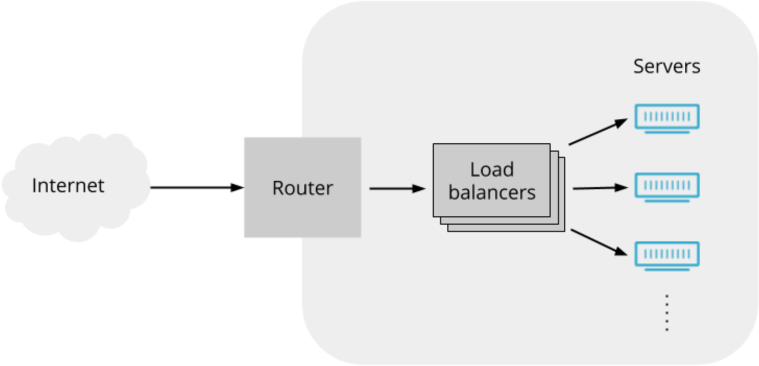
An alternative use of ECMP can come in to play when we want to shard traffic across multiple servers rather than to the same server over multiple paths.


2 servers, 3 connections each
router
proxy
proxy
proxy
additional server added, 1/N must move and break
router
proxy
proxy
proxy

Copyright GitHub

A distributed load balancer

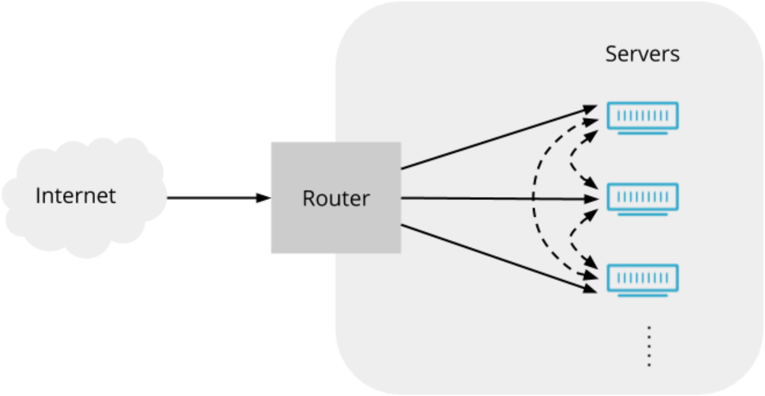
From a classic architecture with dedicated load balancers...



Copyright Cloudflare

A distributed load balancer

... to a distributed load balancer

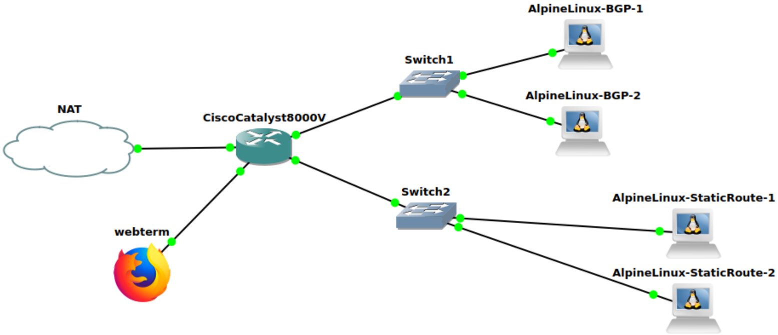


Copyright Cloudflare

A distributed load balancer

In this topology, two servers announce the same IP address with BGP, while the other two are using static routes.

The router is blissfully unaware that the connections are being handled in different places.



Network topology created with GNS3
(Luca Bassi, CC BY-SA 4.0)

Thesis and acknowledgements

My thesis: Bilanciamento del carico per servizi di accesso ai dati a elevata efficienza utilizzando eXpress Data Path

Thanks to: Renzo Davoli, Francesco Giacomini, Antonio Falabella, Lorenzo Chiarelli, Diego Michelotto, Carmelo “budda” Pellegrino

... and all the CNAF staff who made this experience unforgettable!