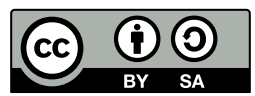
Load balancing using XDP

Luca Bassi

FOSDEM 2024



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.

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.

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.

```
1  #include <linux/bpf.h>
2  #include <bpf/bpf_helpers.h>
3
4  SEC("xdp")
5  int xdp_pass(struct xdp_md *ctx)
6  {
7    return XDP_PASS;
8  }
```

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.

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

Examining a packet

```
 1  void *data_end = (void *)(long)ctx->data_end;
 2  void *pos = (void *)(long)ctx->data;
 3  struct ethhdr *eth = pos;
 4  if (eth + 1 > data_end) {return -1;}
 5  __u16 h_proto = eth->h_proto;
 6  pos = eth + 1;
 7  __u32 key;
 8  if (h_proto == bpf_htons(ETH_P_IP)) {
 9    struct iphdr *ip = pos;
10    if (ip + 1 > data_end) {return -1;}
11    __u8 protocol = ip->protocol;
12    if (protocol == IPPROTO_ICMP) {
13      key = 0;
14      long *value = bpf_map_lookup_elem(&count, &key);
15      if (value) {*value += 1;}
16    }
17  } else if (h_proto == bpf_htons(ETH_P_IPV6)) {
18    /* ... */
19  }
20  return XDP_PASS;
```

Reading a map from user space

```
 1  int fd = bpf_obj_get("/sys/fs/bpf/test/count");
 2  if (fd < 0) {
 3    printf("Error bpf_obj_get\n");
 4    return fd;
 5  }
 6  int nr_cpus = libbpf_num_possible_cpus();
 7  long sum[] = { 0, 0 };
 8  for (__u32 key = 0; key <= 1; ++key) {
 9    long values[nr_cpus];
10    if ((bpf_map_lookup_elem(fd, &key, values)) != 0) {
11      printf("Error bpf_map_lookup_elem\n");
12      return -1;
13    }
14    for (int i = 0; i < nr_cpus; ++i) {
15      sum[key] += values[i];
16    }
17  }
18  printf("IPv4: %d\nIPv6: %d\n", sum[0], sum[1]);
```

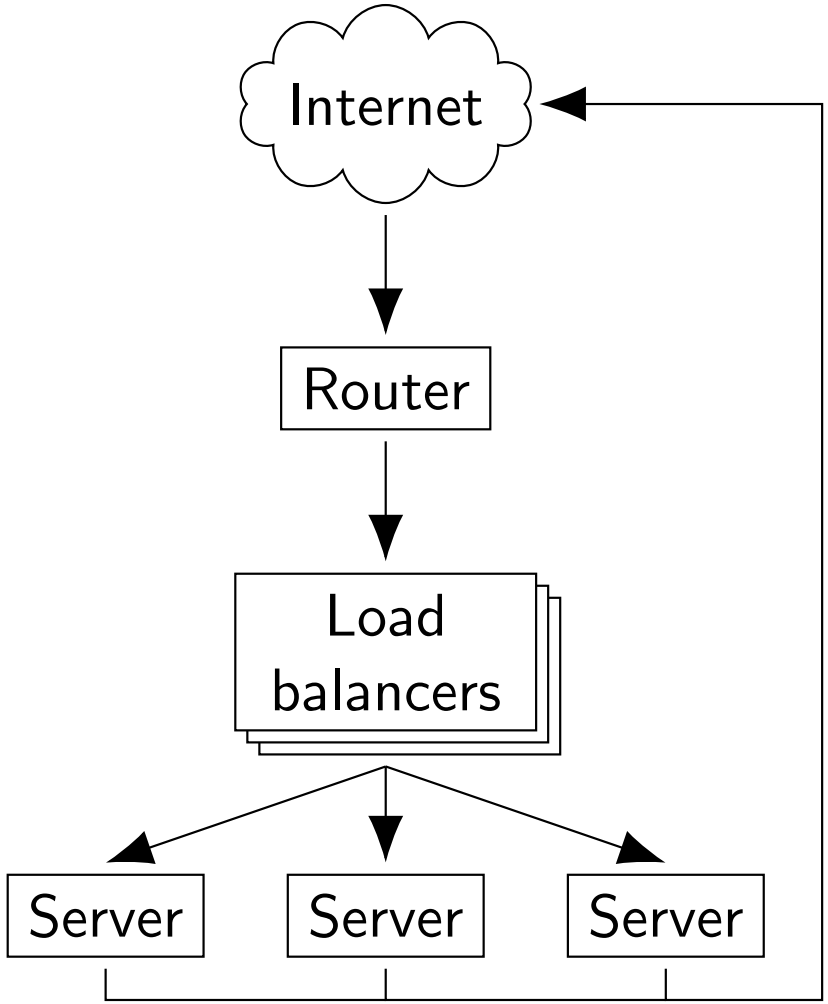
Redirecting packets

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

```
1  __builtin_memcpy(eth->h_dest, destination_mac,
2    sizeof(destination_mac));
3  ip->daddr = destination_ip;
4  __builtin_memcpy(eth->h_source, load_balancer_mac,
5    sizeof(load_balancer_mac));
6  ip->saddr = load_balancer_ip;
7  ip->check = iph_csum(ip);
8  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.



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.

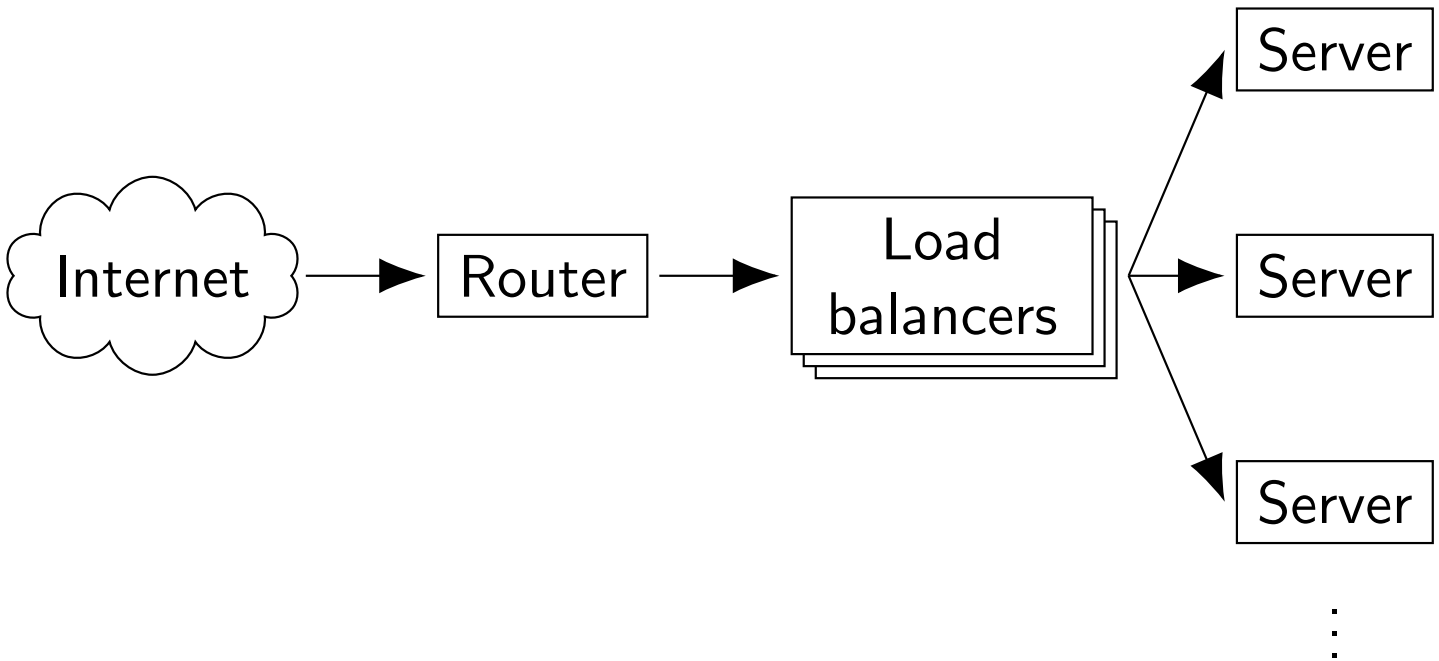
ECMP

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.

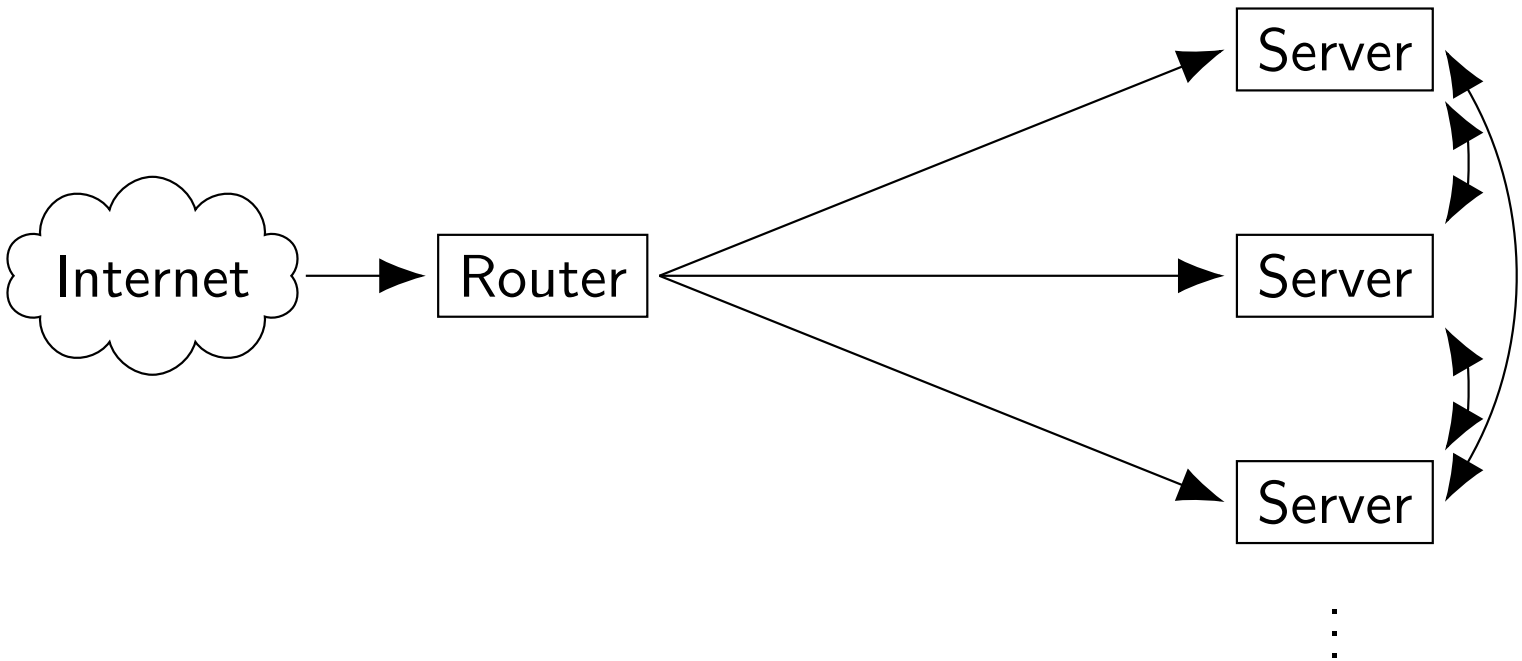
A distributed load balancer

From a classic architecture with dedicated load balancers...



A distributed load balancer

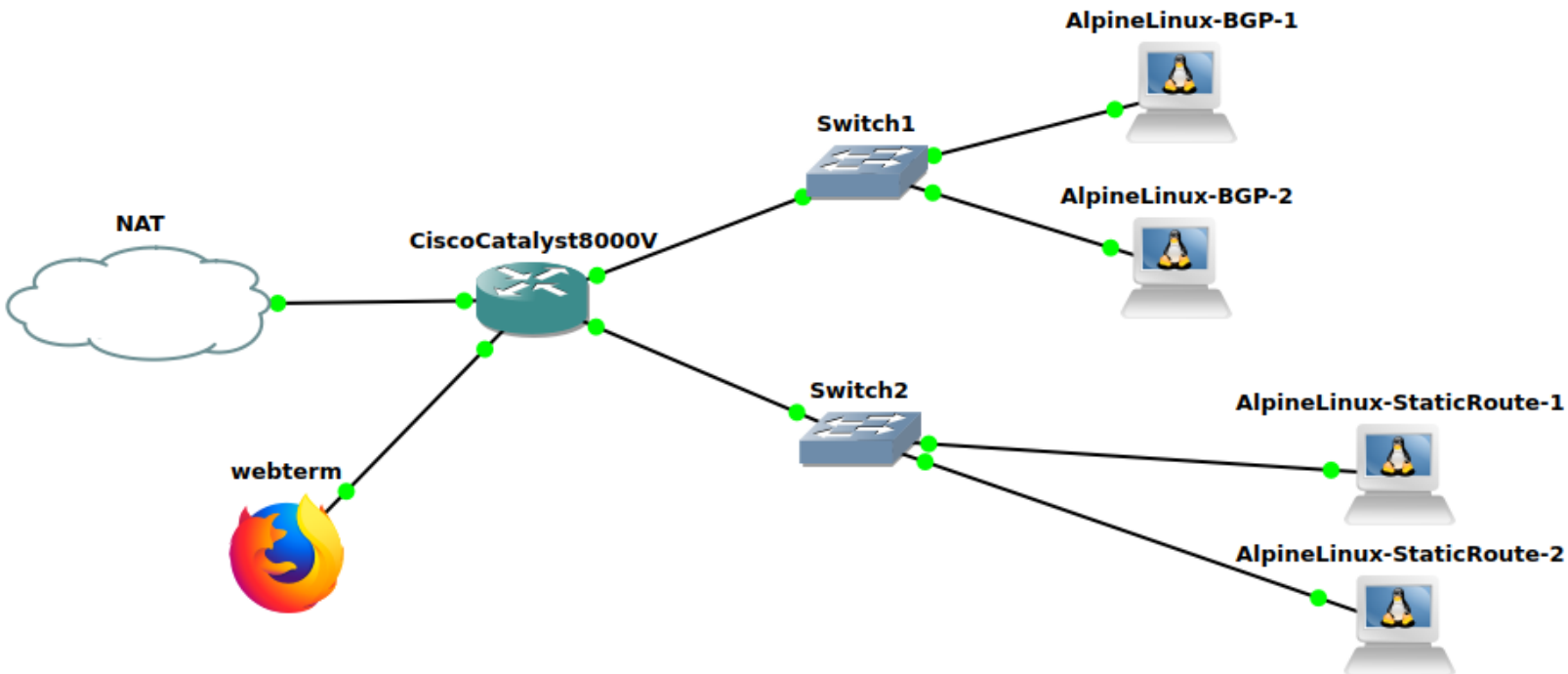
... to a distributed load balancer



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)

Conclusions

XDP permits to develop high-efficiency load balancers.

Direct Server Return can increase the throughput.

To redirect all the packets of a specific connection to the same backend server, a consistent hashing algorithm can be used.

It's possible to leverage the ECMP routing to distribute packets between servers and deploy directly to backend servers without the need for a dedicated machine.

Source code:
`https://gitlab.com/argoware/xdp-load-balancer`

Thank you for your attention

Questions?