IPv6 Segment Routing integration with TCP

Mathieu Jadin, Fabien Duchêne
and Olivier Bonaventure

IPv6 Segment Routing (SRv6) is available in the Linux kernel since version 4.10

4.10:
- SRH parsing, insertion and encapsulation
- Socket API

4.12:
- iproute2 modification

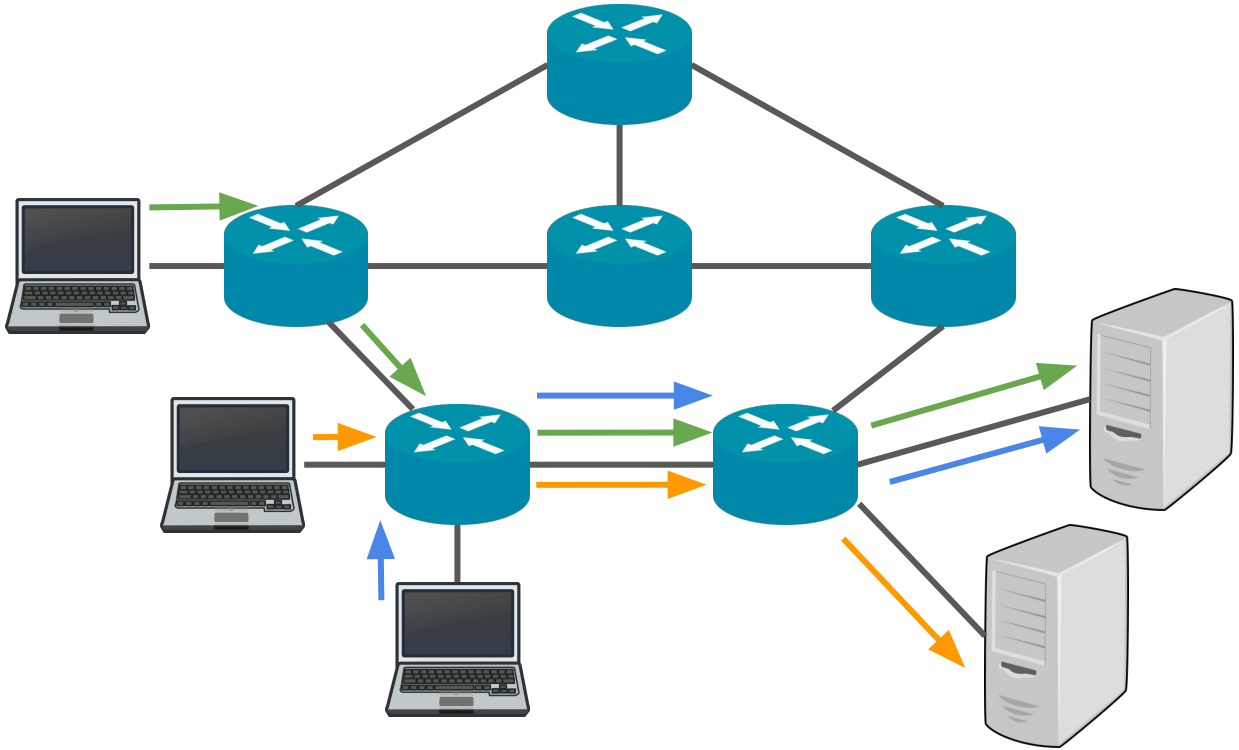
4.18:
- iptables modification
- Custom SRv6 network rules

The reference website is
https://segment-routing.org

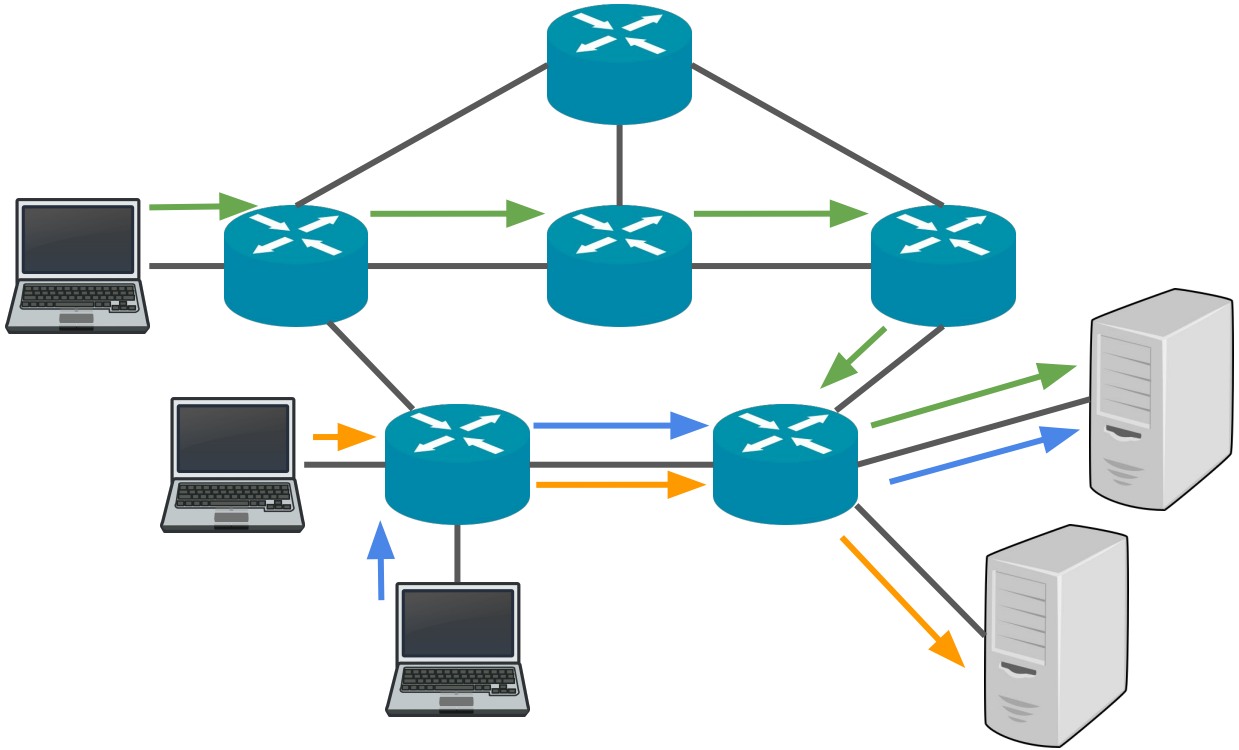
Outline

- Linux implementation
- Motivation
- Path injections
- Flexible and application-agnostic path management
- Network events
- Evaluation

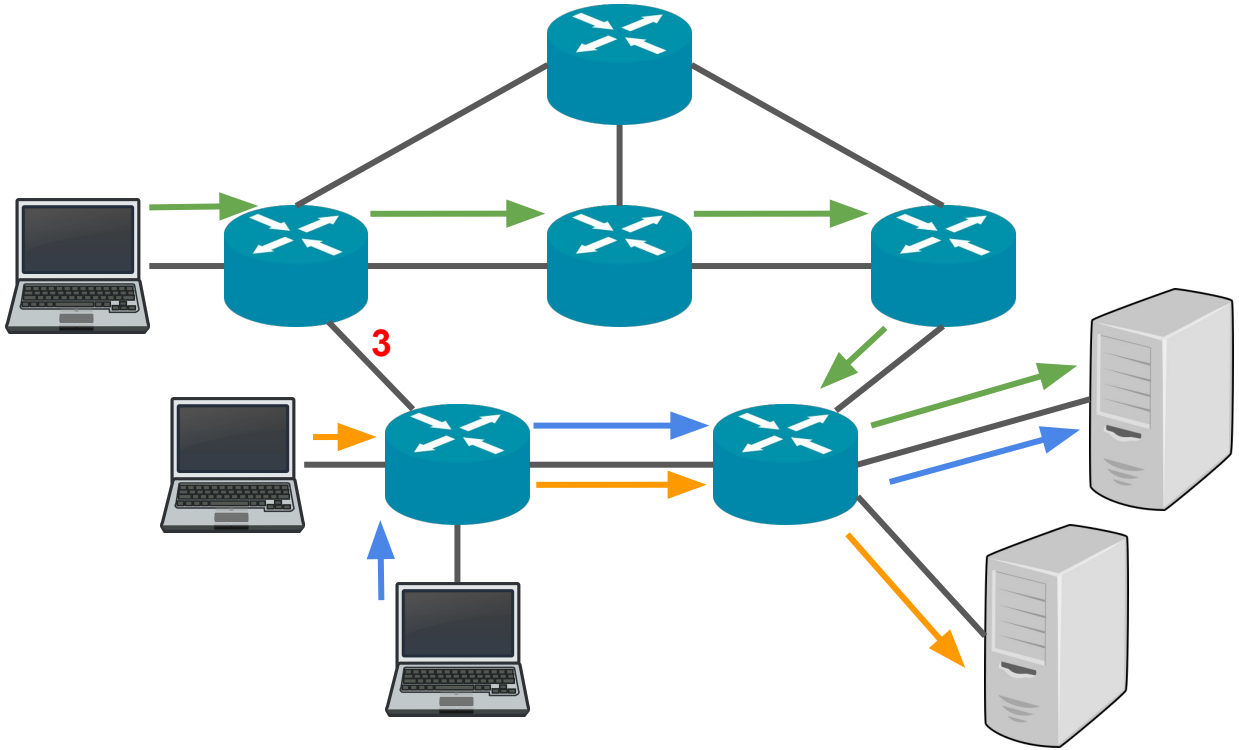
Using the whole network is hard



Using the whole network is hard



Tweaking IGP weights can change the paths

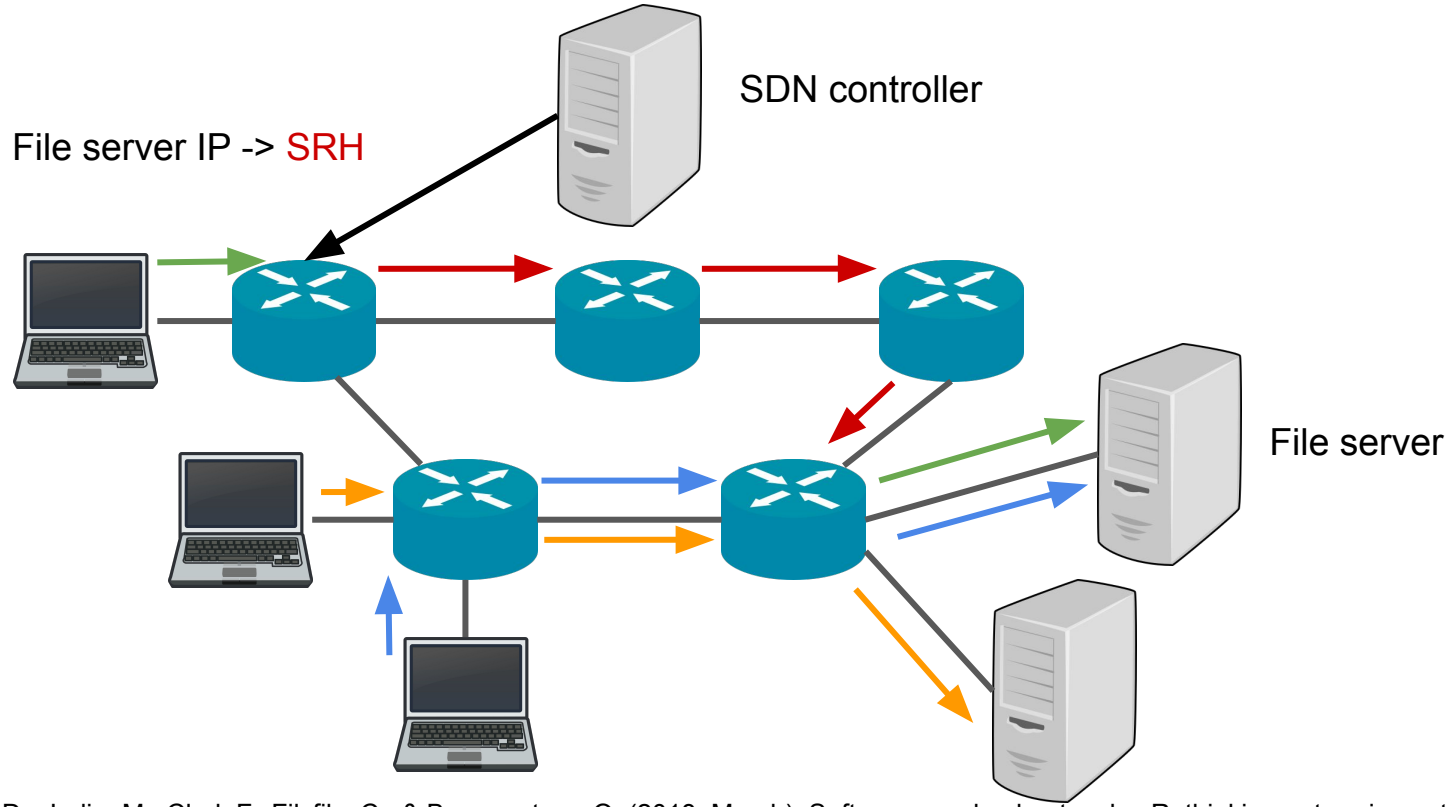


Tweaking IGP weights can change the paths

This solution lacks of flexibility!

- Hard to control the side effects
- Cannot split flows that goes to the same destination

IPv6 Segment Routing* brings more flexibility!

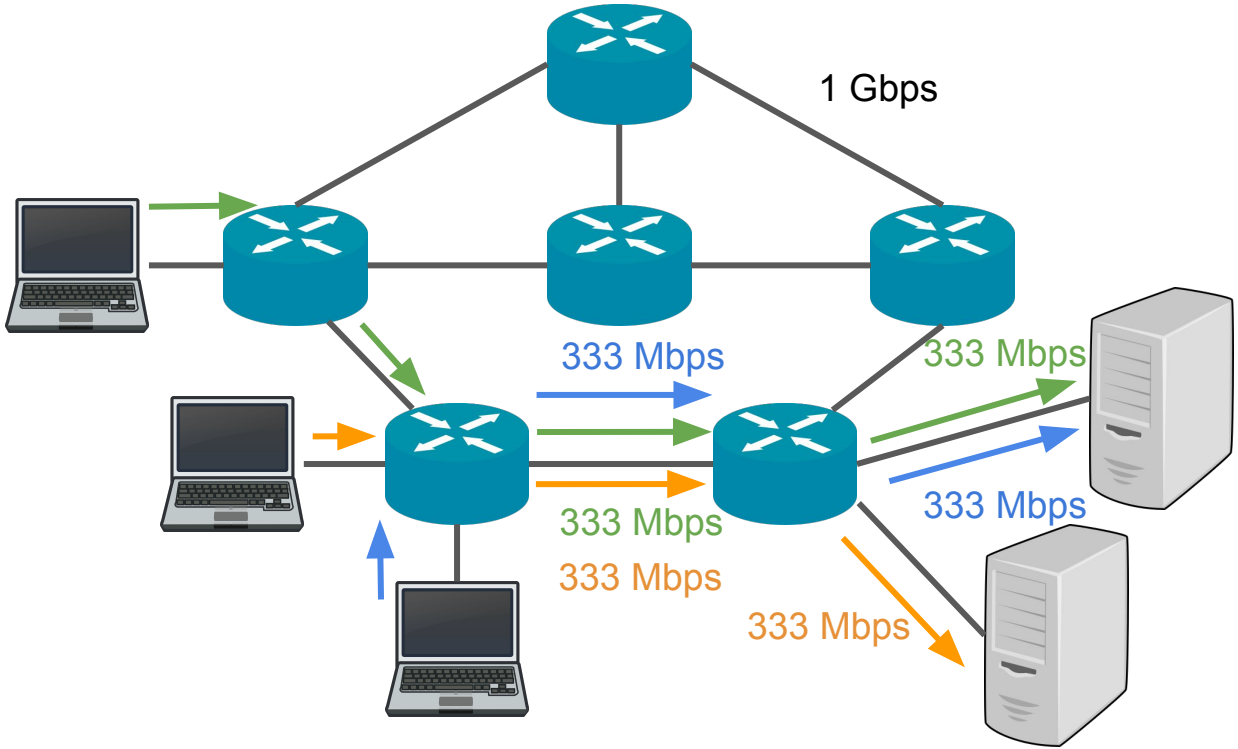


* Lebrun, D., Jadin, M., Clad, F., Filsfils, C., & Bonaventure, O. (2018, March). Software resolved networks: Rethinking enterprise networks with ipv6 segment routing. In *Proceedings of the Symposium on SDN Research* (p. 6). ACM.

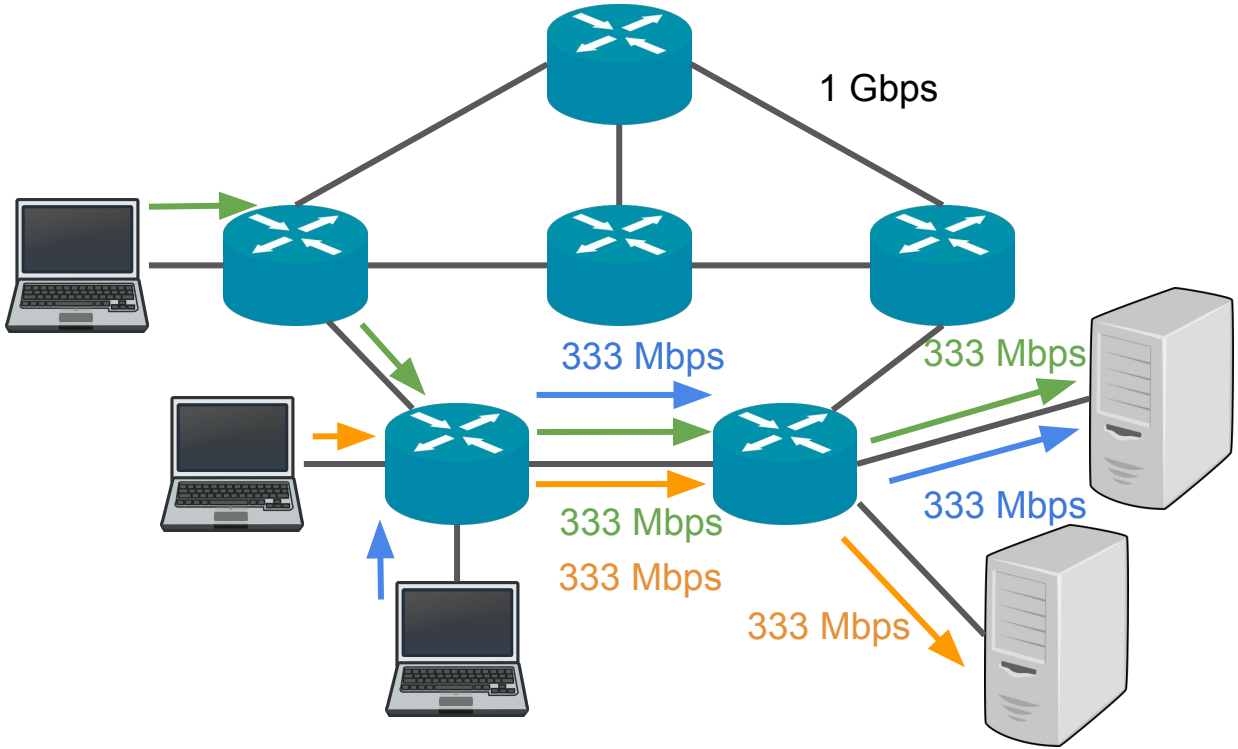
Still, there are some limitations

1. Granularity: we cannot consider flows in isolation for scalability
2. Reactivity: our first prototype does not change the SRH once assigned

TCP is very reactive to congestion through congestion control: Reno, Cubic, BBR,...



They reduce queuing delay
but they cannot change to an non-congested path



Can we better integrate TCP with SRv6?

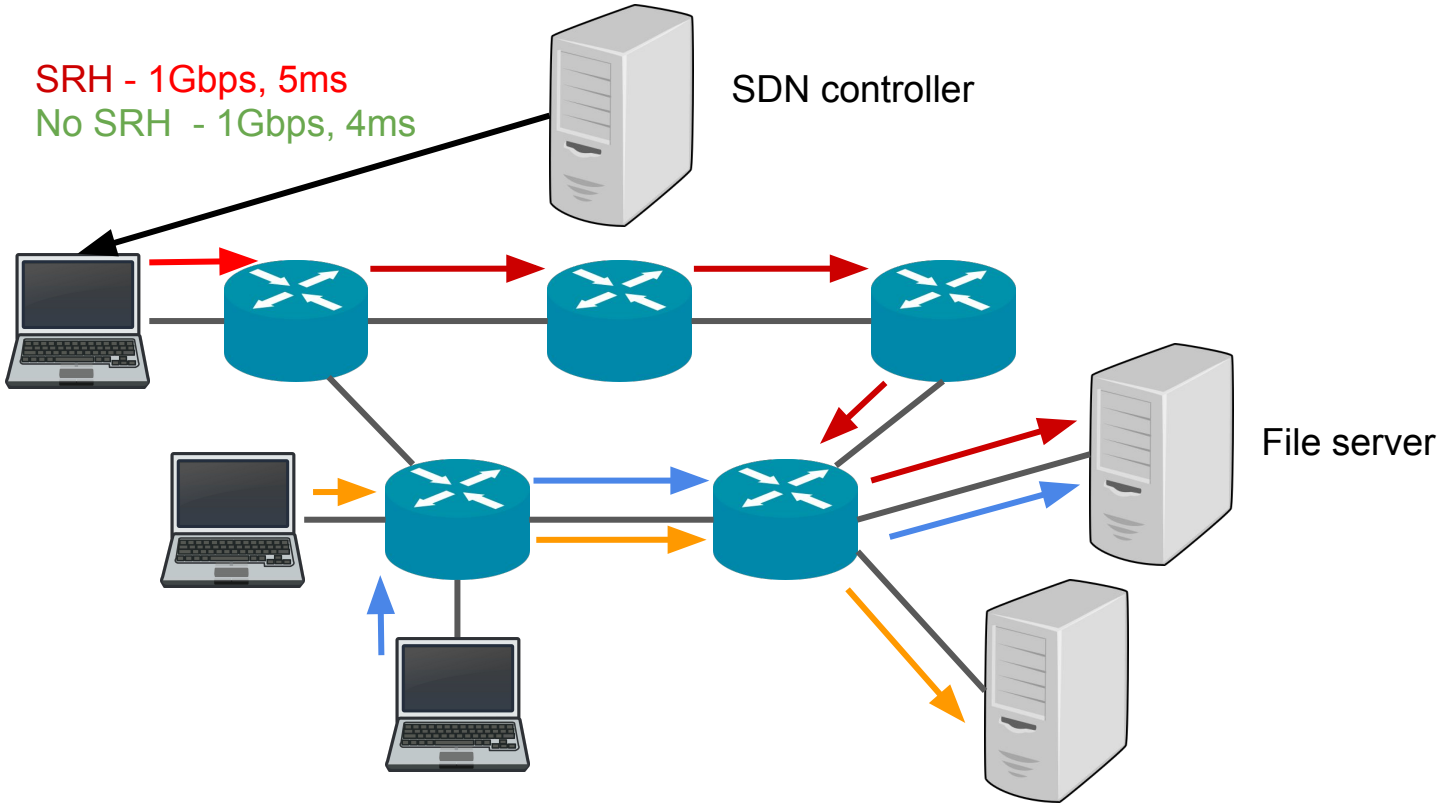
SRv6 enables endhost apps to choose their paths:

1. Granularity: each app can set its own SRH in Linux
2. Reactivity: each app can rank SRHs with their CC state

Outline

- Linux implementation
- Motivation
- Path injections
- Flexible and application-agnostic path management
- Network events
- Evaluation

The controller can provide a list of SRHs



The controller can choose any type of paths

Disjoint paths

Paths with similar latencies

Paths with a limit on the number of segments

Paths avoiding overloaded links

...

Outline

- Linux implementation
- Motivation
- Path injections
- Flexible and application-agnostic path management
- Network events
- Evaluation

How do we add a path management on all endhosts?

Modify the kernel:

- Hard
- Takes time for merging and updating endhosts

Use a library:

- Cannot modify every application

=> We want to inject code in the kernel space from user space

eBPF to the rescue

Lightweight, integrated in Linux kernel since 2014

Bytecode recompiled to native architecture

Dedicated, isolated stack memory

Rely on a verifier to check code termination and memory accesses

Hooks in the kernel trigger the injected code

```
int tcp_connect(struct sock *sk) {

 tcp_call_bpf(sk, BPF_SOCK_OPS_TCP_CONNECT_CB,
              0, NULL);



 // Some processing

}
```

```
SEC("sockops")
int handle_sockop(struct bpf_sock_ops *skops)
{
```

VM

```
// Processing
bpf_setsockopt(skops, SOL_IPV6,
     IPV6_RTHDR, srh, sizeof(*srh));
```

```
return 0;

}
```

Exclusive access to connection state

Most of the hooks already exist in the kernel

BPF_SOCK_OPS_TCP_CONNECT_CB
BPF_SOCK_OPS_ACTIVE_ESTABLISHED_CB
BPF_SOCK_OPS_PASSIVE_ESTABLISHED_CB
BPF_SOCK_OPS_NEEDS_ECN
BPF_SOCK_OPS_BASE_RTT
BPF_SOCK_OPS_RTO_CB
BPF_SOCK_OPS_RETRANS_CB
BPF_SOCK_OPS_STATE_CB
BPF_SOCK_OPS_TCP_LISTEN_CB
BPF_SOCK_OPS_RTT_CB
BPF_SOCK_OPS_TCP_XMIT
BPF_SOCK_OPS_UDP_XMIT
BPF_SOCK_OPS_ECN_CE

BPF maps enable communication between injected code and user space daemon

Example

```
int handle_sockop(struct bpf_sock_ops *skops) {
    // initialization of variables
    int hook = (int) skops->op;
    get_flow_id_from_sock(&five_tuple, skops);
    flow_info = (void *)bpf_map_lookup_elem(&conn_map, &five_tuple);

    switch (hook) {
        case BPF_SOCK_OPS_RTO_CB:
            key = get_better_dest_path(&map, flow_info, 0, flow_id.remote_addr);
            srhs = (void *) bpf_map_lookup_elem(&map, flow_id.remote_addr);
            srh = srhs[key];
            bpf_setsockopt(skops, SOL_IPV6, IPV6_RTHDR, srh, sizeof(*srh));
            flow_info->srh_id = key;
            bpf_map_update_elem(&conn_map, &flow_id, flow_info, BPF_ANY);
            break;
        // Other hooks handling
    }
    return 0;
}
```

Outline

- Linux implementation
- Motivation
- Path injections
- Flexible and application-agnostic path management
- Network events
- Evaluation

Various network events can trigger a change of path

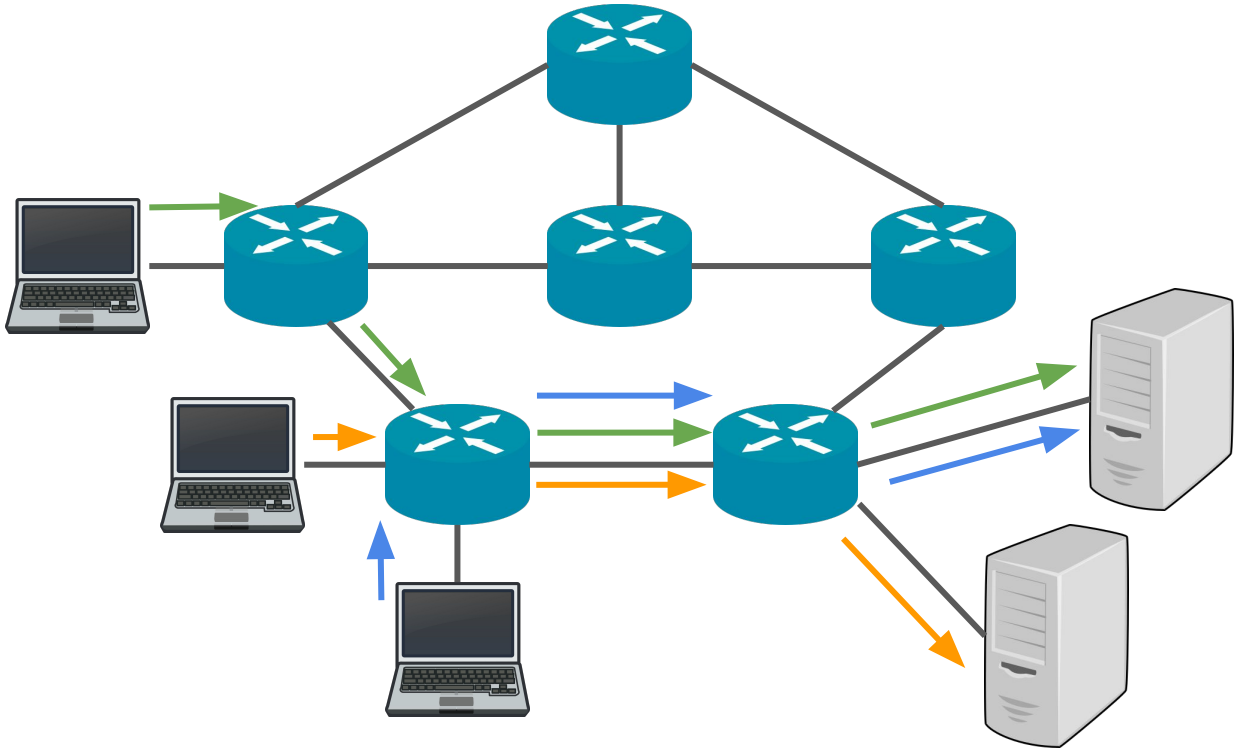
Retransmission Timer Expiration

Fast Retransmit

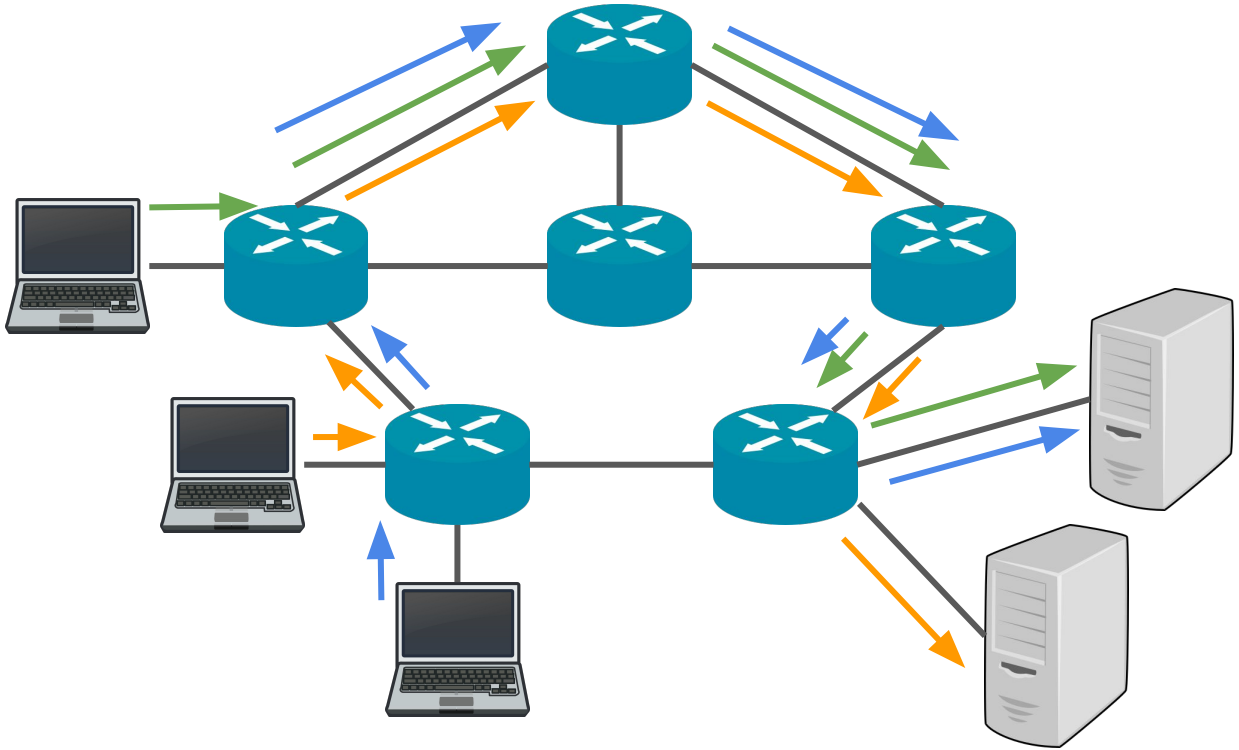
Explicit Congestion Notification

Update of the congestion state from the controller

We need a way to stabilize the path changes



We need a way to stabilize the path changes



Outline

- Linux implementation
- Motivation
- Path injections
- Flexible and application-agnostic path management
- Network events
- Evaluation

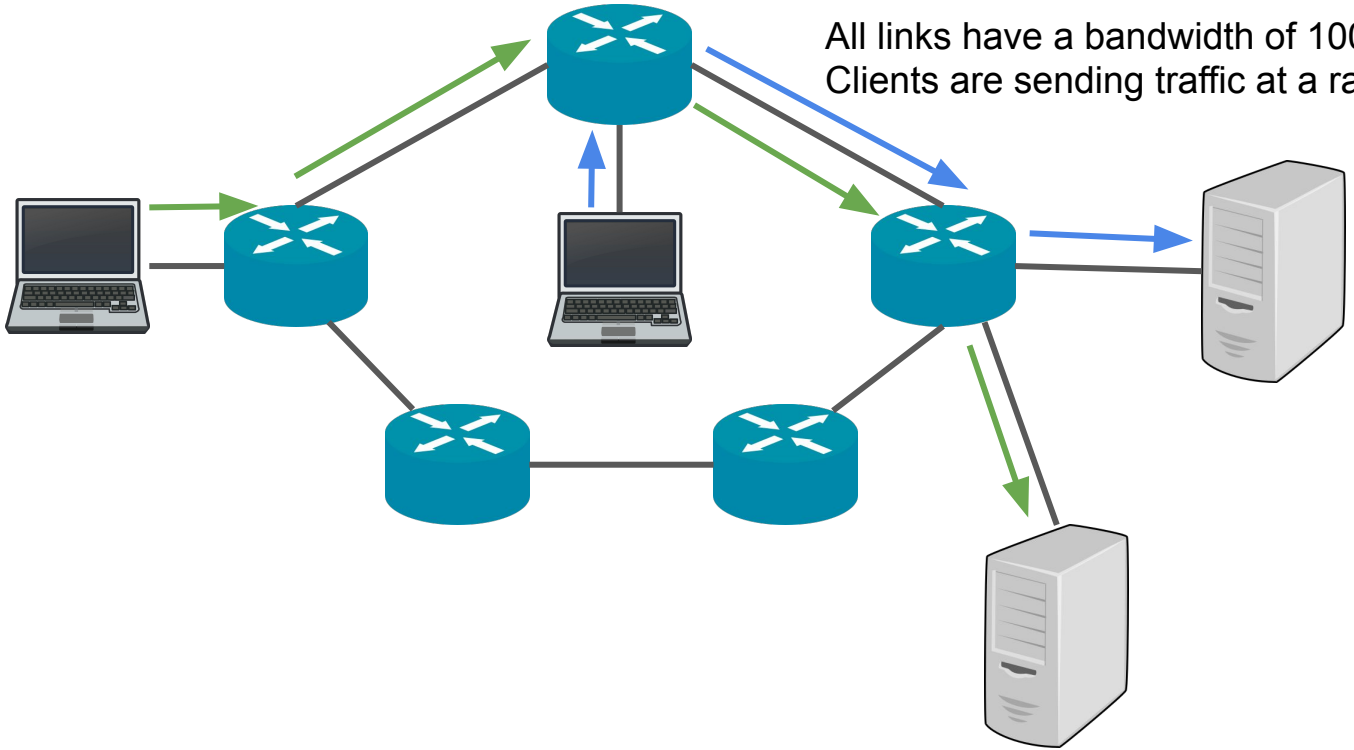
Experimental setup

Mininet emulation

Linux kernel v5.3

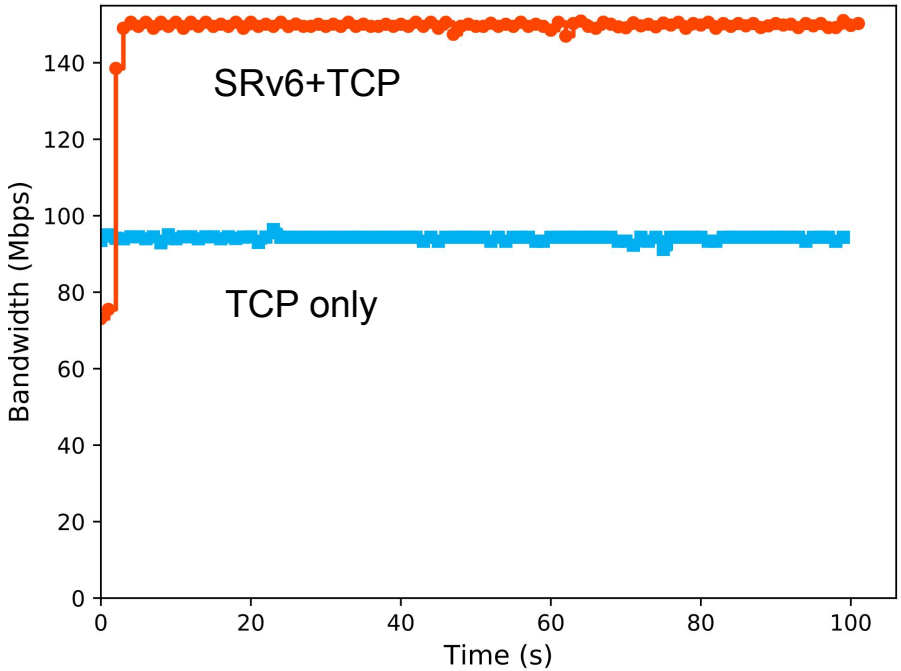
Topologies from TopologyZoo

Example of topology

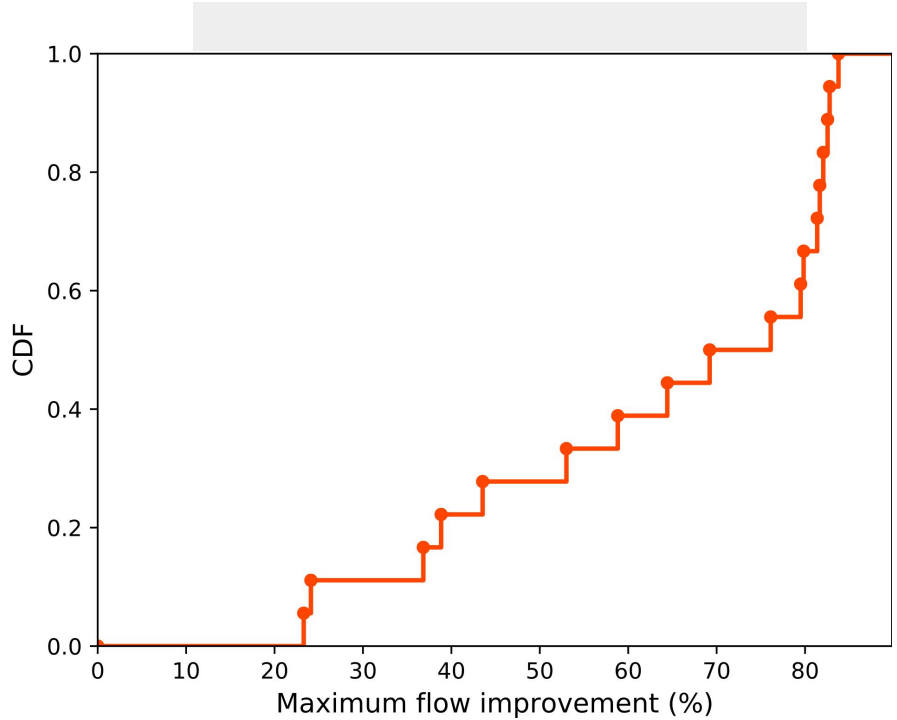


All links have a bandwidth of 100 Mbps
Clients are sending traffic at a rate of 75 Mbps

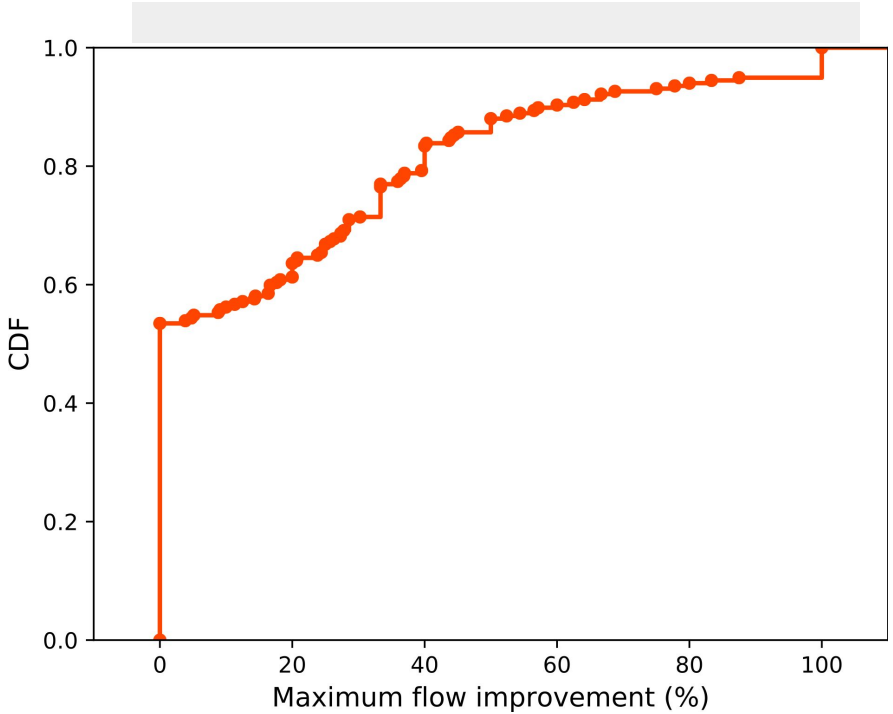
We can observe that one of the connections changes its path and their bandwidths improve



For 50% of our topo subset
we have an improvement of 65%



In theory, we can improve 45% of topologies of TopologyZoo set



Conclusion

Motivation
- We can improve the network usage by mixing SRv6 and TCP

Path injections
- SDN controllers can compute and inject paths to an endhost

Flexible and application-agnostic path management
- eBPF enables us to inject path management in the kernel

Network events
- We can react to various network events

Evaluation
- Mixing SRv6 and TCP does improve the network usage

Backup slides

Discussion: eBPF code injection

We could dynamically inject the eBPF code to each endhost:

- From the controller
- From Router Advertisement or DHCP
- From the server

Future work: Study SRv6 with (MP)QUIC

It is simpler to integrate QUIC and SRv6:

- QUIC is user space and thus, easier to modify
- QUIC also have a multipath extension
- The server could push eBPF code directly to the client through a separate stream

We use two maps: one for path list and one for connection infos

```
struct bpf_elf_map SEC("maps") conn_map = {
    .type           = BPF_MAP_TYPE_HASH,
    .size_key    = sizeof(struct flow_tuple),
    .size_value    = sizeof(struct flow_infos),
    .pinning    = PIN_NONE,
    .max_elem    = MAX_FLOWS,
};

struct bpf_elf_map SEC("maps") dest_map = {
    .type           = BPF_MAP_TYPE_HASH,
    .size_key    = sizeof(struct in6_addr),
    .size_value    = sizeof(struct dst_infos),
    .pinning    = PIN_NONE,
    .max_elem    = MAX_FLOWS,
};
```