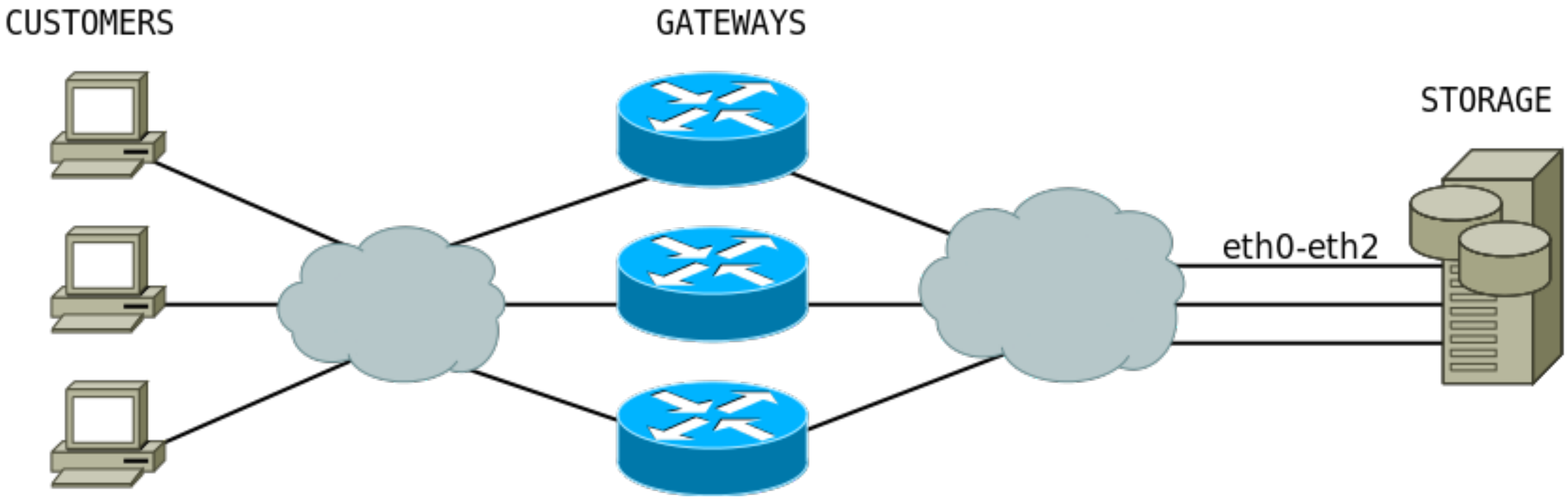
IP Address Reflection Scheme Implementation for Linux

M. Krinkin, SPbAU
K. Krinkin, OSLL

Motivation



- Provide required quality of service
- Isolate customers data flows

What does Linux provide for us?

```
struct ifreq ifr;
...
int s = socket(AF_INET, SOCK_STREAM, 0);
...
strncpy(ifr.ifr_name, “eth0”, sizeof(ifr.
ifr_name));
setsockopt(s, SOL_SOCKET, SO_BINDTODEVICE,
        &ifr, sizeof(ifr));
```

First attempt

```
int tcp_v4_rcv(struct sk_buff *skb)
{
   struct sock *sk;
   ...
   sk = __inet_lookup_skb(&tcp_hashinfo,
        skb, th->source, th->dest);
   ...
   sk->sk_bound_dev_if = inet_iif(skb);
}
```

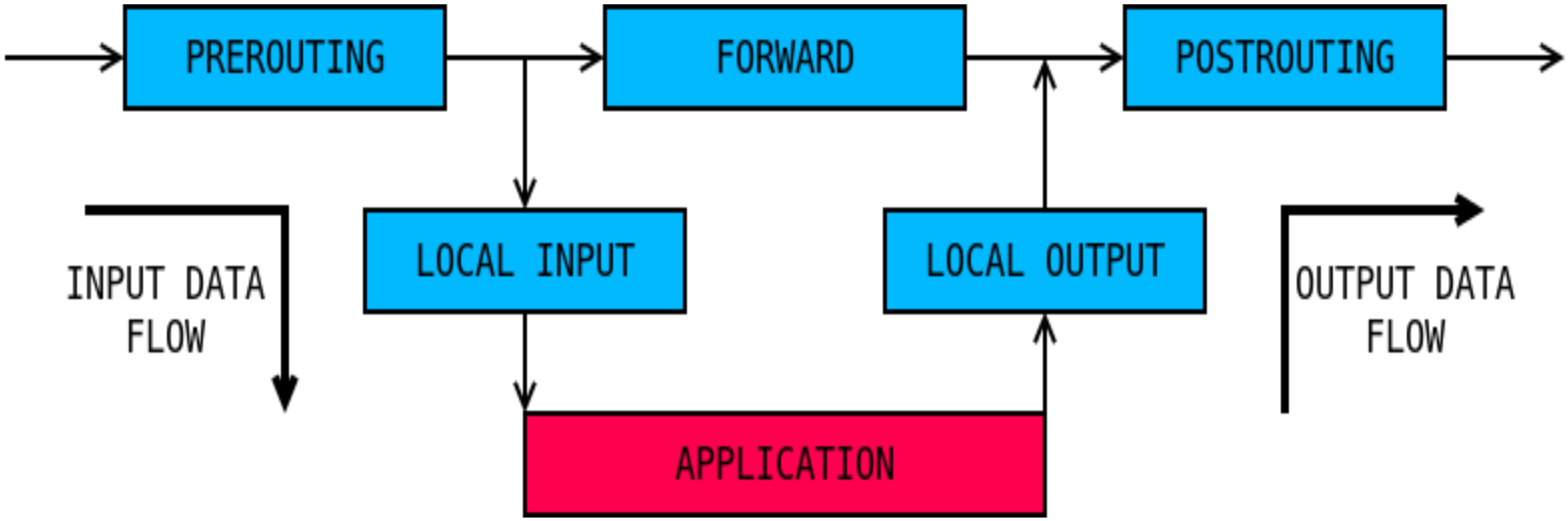
Control interface

```
int s = socket(AF_INET, SOCK_STREAM, 0);
int opt = 1; /* enable */
setsockopt(s, SOL_SOCKET, SO_IPREFLECT,
        &opt, sizeof(opt));
opt = 0; /* disable */
setsockopt(s, SOL_SOCKET, SO_IPREFLECT,
        &opt, sizeof(opt));
```

Pros and cons

- pros:
  - solution is rather simple
  - fine grained control interface
  - almost no performance overhead
- cons:
  - every protocol needs specific support
  - control interface is not suitable for ICMP
  - solution fixes network interface only

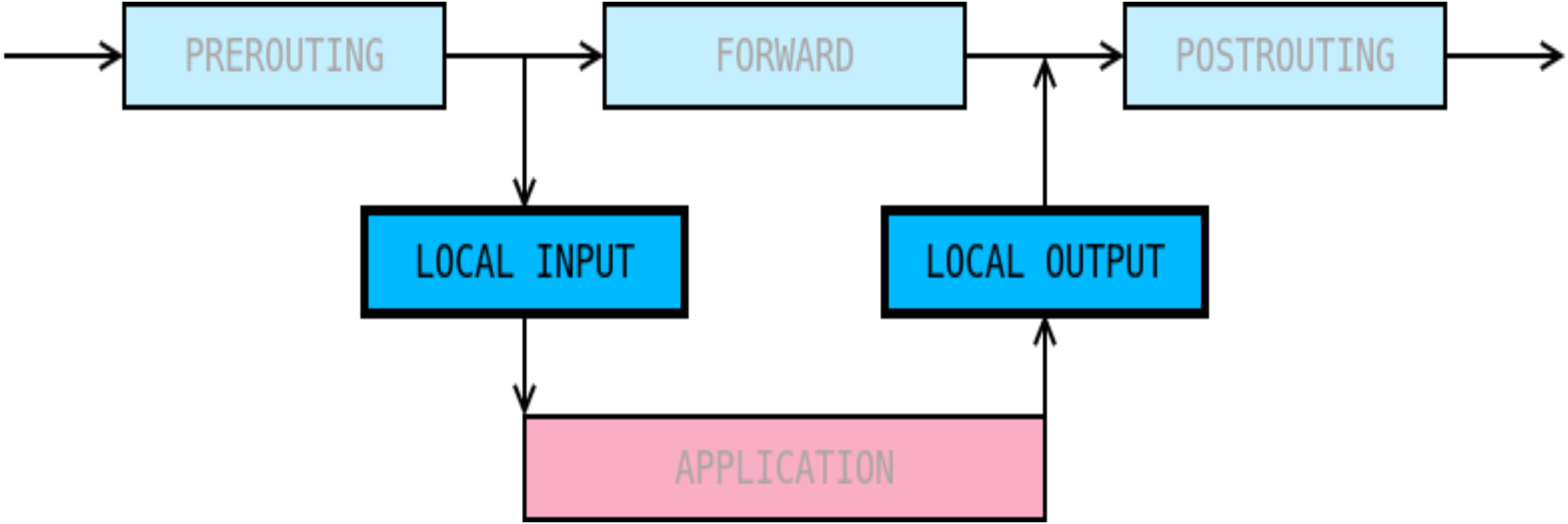
Netfilter API



Netfilter callback

```
unsigned callback(
        struct nf_hook_ops const *ops,
        struct sk_buff *skb,
        struct net_device const *in,
        struct net_device const *out,
        int (*okfn)(struct sk_buff *)
)
```

Second attempt



Input handler

- extract previous hop link layer address (gateway)
- extract client ip address (source ip)
- extract local ip address (destination ip)
- save data in hashtable (cache)

Output handler

- find cache entry for destination ip address
- create new route through right gateway
- ensure ARP entry for destination ip

Pros and cons

- pros:
    - implemented as loadable module
    - works for all IPv4 network protocols without special support
    - fixes gateway
- cons:
    - implies additional performance overhead due to rerouting

Next steps

- define and implement control interface
- measure performance overhead
- cache routes

Contact information and links

- Sources:
  - first solution https://github.com/OSLL/ipreflect
  - second solution https://github.com/OSLL/hwaddr-cache
- Contacts:
  - Mike Krinkin krinkin.m.u@gmail.com
  - Kirill Krinkin kirill.krinkin@fruct.org

Q&A

Input handler (source code)

```
unsigned in_hook_fn(...)
{
   struct ethhdr *lhdr = eth_hdr(skb);
   struct iphdr *nhdr = ip_hdr(skb);
   ...
   /* store in hashtable */
   hwaddr_update(nhdr->saddr, nhdr->daddr,
                  lhdr->h_source, ETH_ALEN);
}
```

Output handler (source code)

```
unsigned out_hook_fn(...)
{
   /* lookup hashtable entry */
   struct hwaddr_entry *entry =
               hwaddr_lookup(nhdr->daddr);
   ...
   /* reroute network packet */
   rt = update_route(skb, ..., entry);
   ...
}
```

Ensure ARP entry (source code)

```
void ensure_neigh(struct hwaddr_entry *entry)
{
  neigh = __ipv4_neigh_lookup_noref(
          rt->dst.dev, entry->local);
  if (!neigh)
     neigh = __neigh_create(&arp_tbl,
             &entry->local, ...);
  neigh_update(neigh, entry->ha, NUD_NOARP,
          NEIGH_UPDATE_F_WEAK_OVERRIDE);
}
```