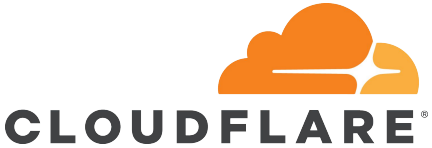


Programmable socket lookup with BPF

Jakub Sitnicki, Lorenz Bauer, Marek Majkowski; Linux Plumbers Conf ‘19

Agenda

- Socket dispatch woes
- Proposal: program socket lookup with BPF
- User-space API (RFCv2)
- Next steps

Cloudflare: many IPs, many ports

- We use millions of anycasted IPs
  - ~160 prefixes
- Exposing ~10 daemons
  - All servers run all daemons
- Services run on certain prefixes, not IPs

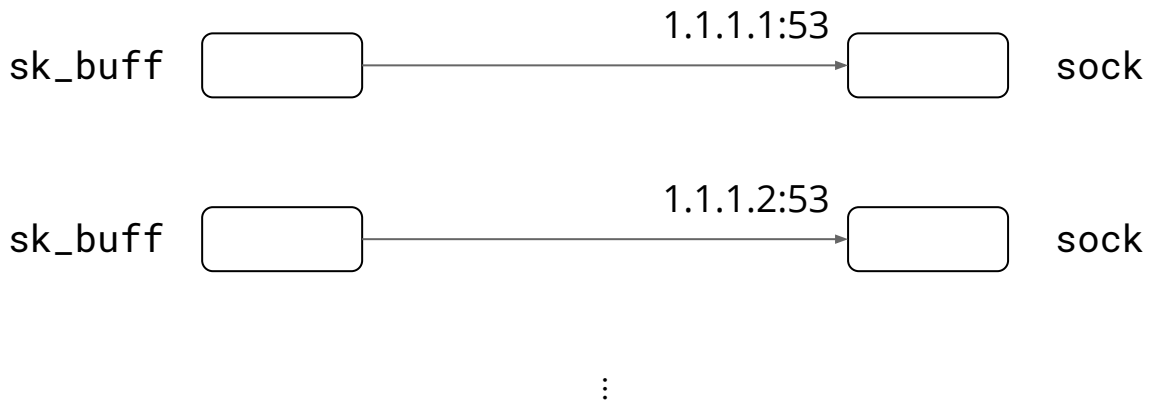
CLOUDFLARE

This breaks traditional socket API

bind / listen on single IP

1:1 mapping

bind / listen on a prefix



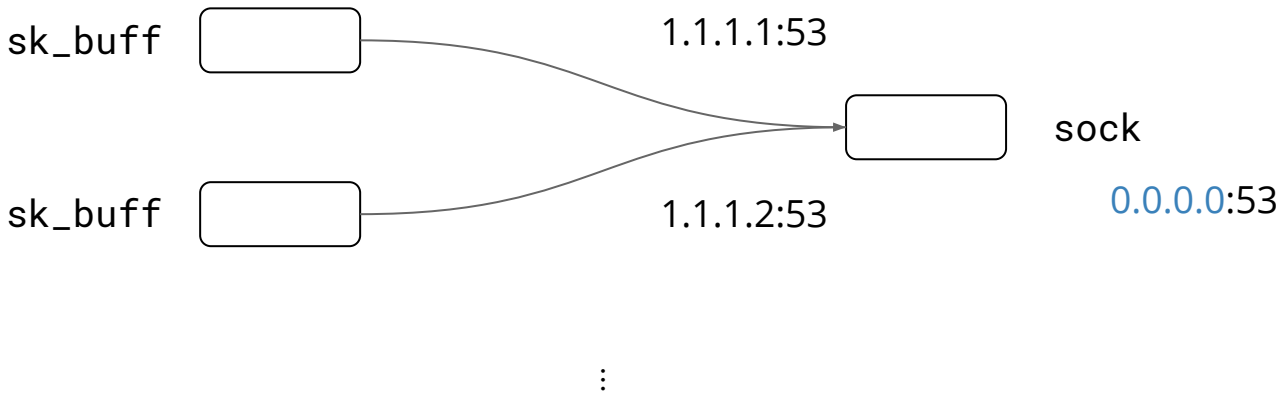
and ~250 more for /24

many 1:1 mappings

Too many listening sockets

- Creates latency spikes
- See Revenge of the listening sockets

Solution: INADDR_ANY?



CLOUDFLARE

catch-all M:1 mapping

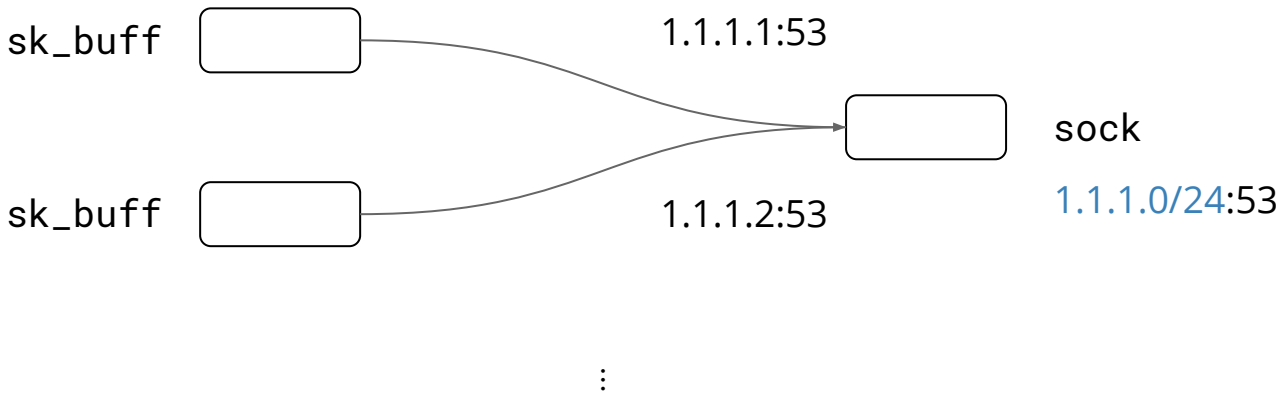
INADDR_ANY: port sharing is impossible

- 1.1.1.0/24 port 53: public DNS resolver
- 2.2.2.0/24 port 53: authoritative DNS

Problems so far

- Can’t bind to prefix

Solution: SO_BINDTOPREFIX sockopt



M:1 mapping

Solution: SO_BINDTOPREFIX sockopt

[net-next RFC 0/4] SO_BINDTOPREFIX

[Date Prev][Date Next][Thread Prev][Thread Next][Date Index][Thread Index]

CLOUDFLARE

Solution: SO_BINDTOPREFIX sockopt

[net-next RFC 0/4] SO_BINDTOPREFIX

[Date Prev][Date Next][Thread Prev][Thread Next][Date Index][Thread Index]

Please do not add such monolithic option.

CLOUDFLARE

Problems contd.

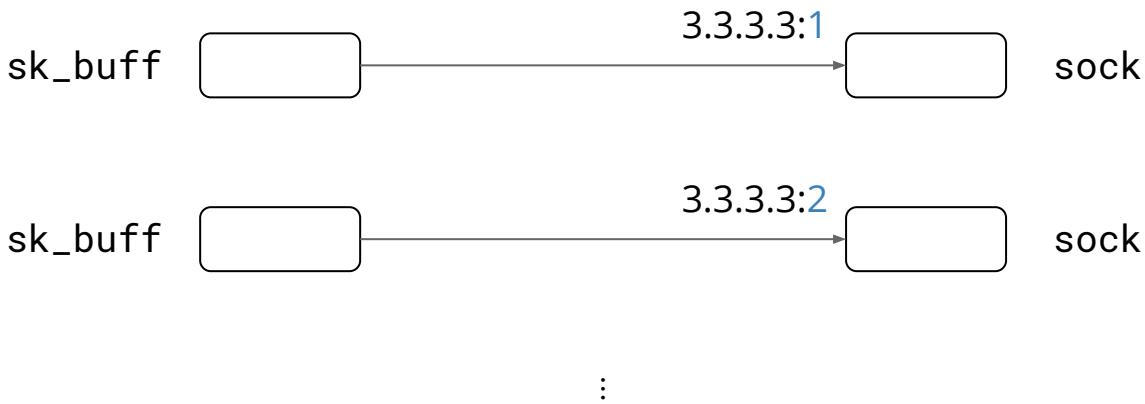
- Can’t bind to prefix
- Can’t upstream SO_BINDTOPREFIX

CLOUDFLARE

Bind to all the ports

- We offer a TCP reverse proxy
- Needs to work on all ports for a given IP address

Naive: listen on 65k ports



and ~65k more

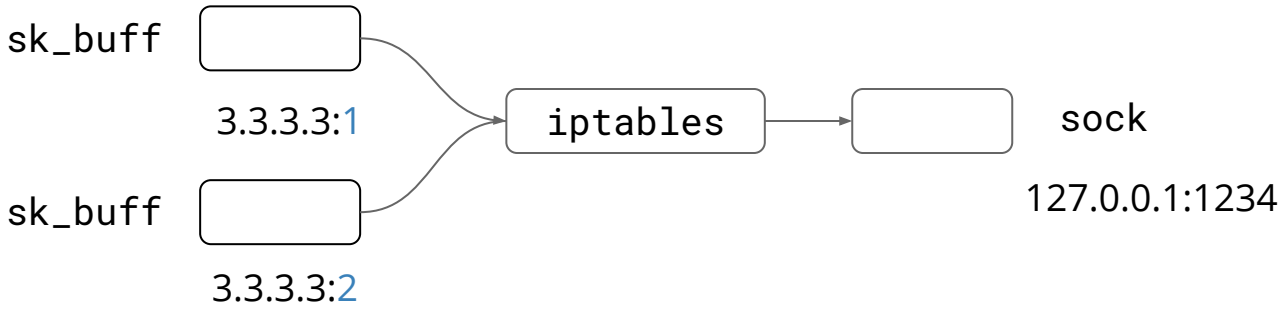
1:1 mapping

Problems contd.

- Can’t bind to prefix
- Can’t upstream SO_BINDTOPREFIX
- Can’t bind to PORT_ANY

CLOUDFLARE

Solution: TPROXY redirect



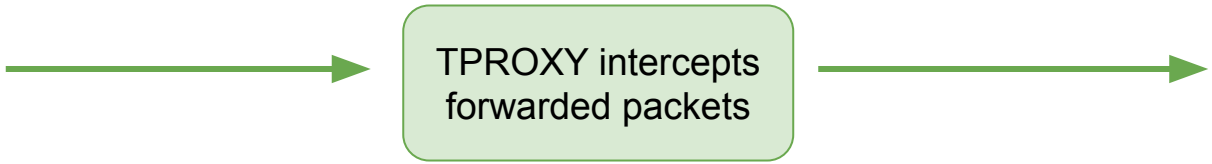
M:1 mapping

See Abusing Linux's firewall: the hack that allowed us to build Spectrum

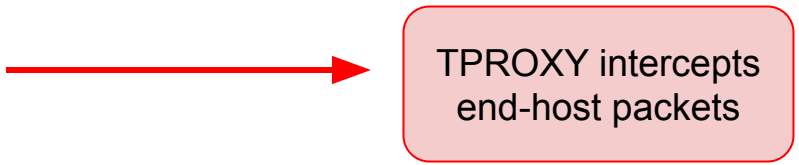
CLOUDFLARE

TPROXY was not designed for this

What it was meant to do:



How we use it:



CLOUDFLARE

TPROXY drawbacks

- Forces IP_TRANSPARENT sockopt on listening socket
  - Requires CAP_NET_ADMIN
- Difficult iptables set up
- Problematic performance under SYN flood
- Creates inconsistent view of socket bindings

Problems contd.

- Can’t bind to prefix
- Can’t upstream SO_BINDTOPREFIX
- Can’t bind to PORT_ANY
- TPROXY: massive hack (in our case)

I lied a little

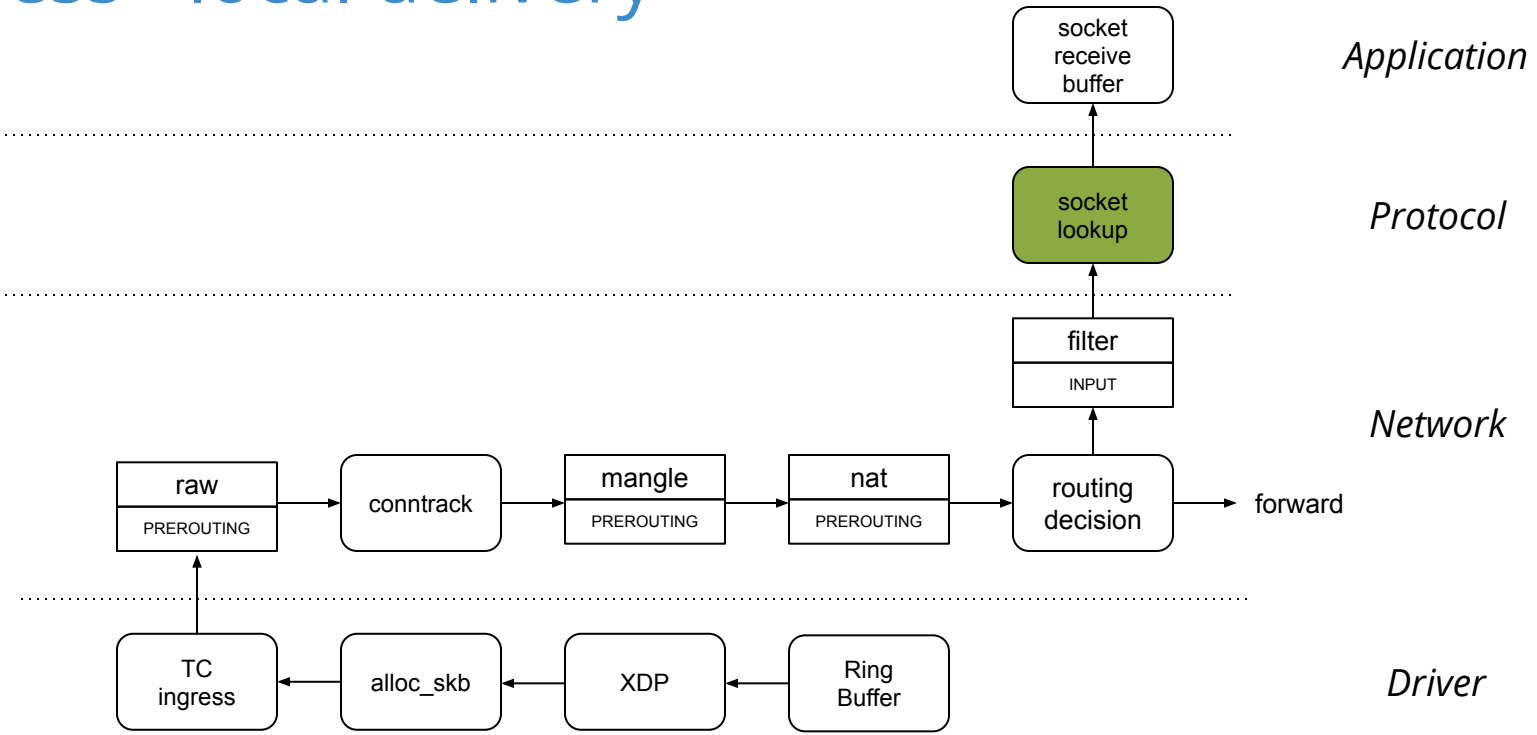
I lied a little

```
Please do not add such monolithic option.

BPF is absolutely the way to go here, as it allows for whatever user
specified tweaks, like a list of destination subnetwork, or/and a list
of source network, or the date/time of the day, or port knocking without
netfilter, or ... you name it.

Simply add an option to load a BPF filter on a socket, used to vary the
various compute_score() functions.

No hard coded knowledge in the kernel, but a generic interface.
```

See https://marc.info/?l=linux-netdev&m=145926190805592&w=2

Proposal:
program socket lookup with BPF

Ingress - local delivery

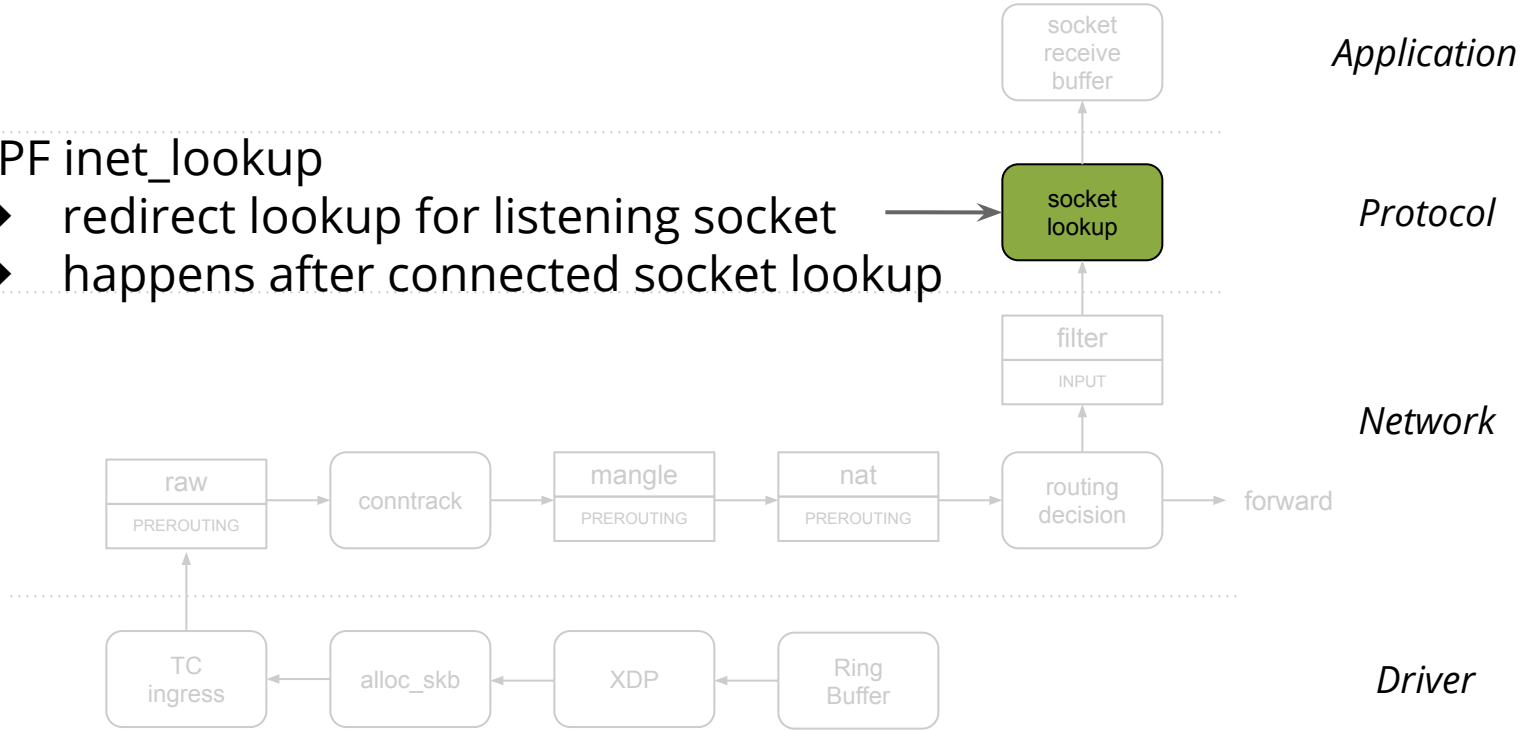


CLOUDFLARE

[Wikipedia - Packet flow in Netfilter and General Networking]

Ingress - local delivery & BPF inet_lookup

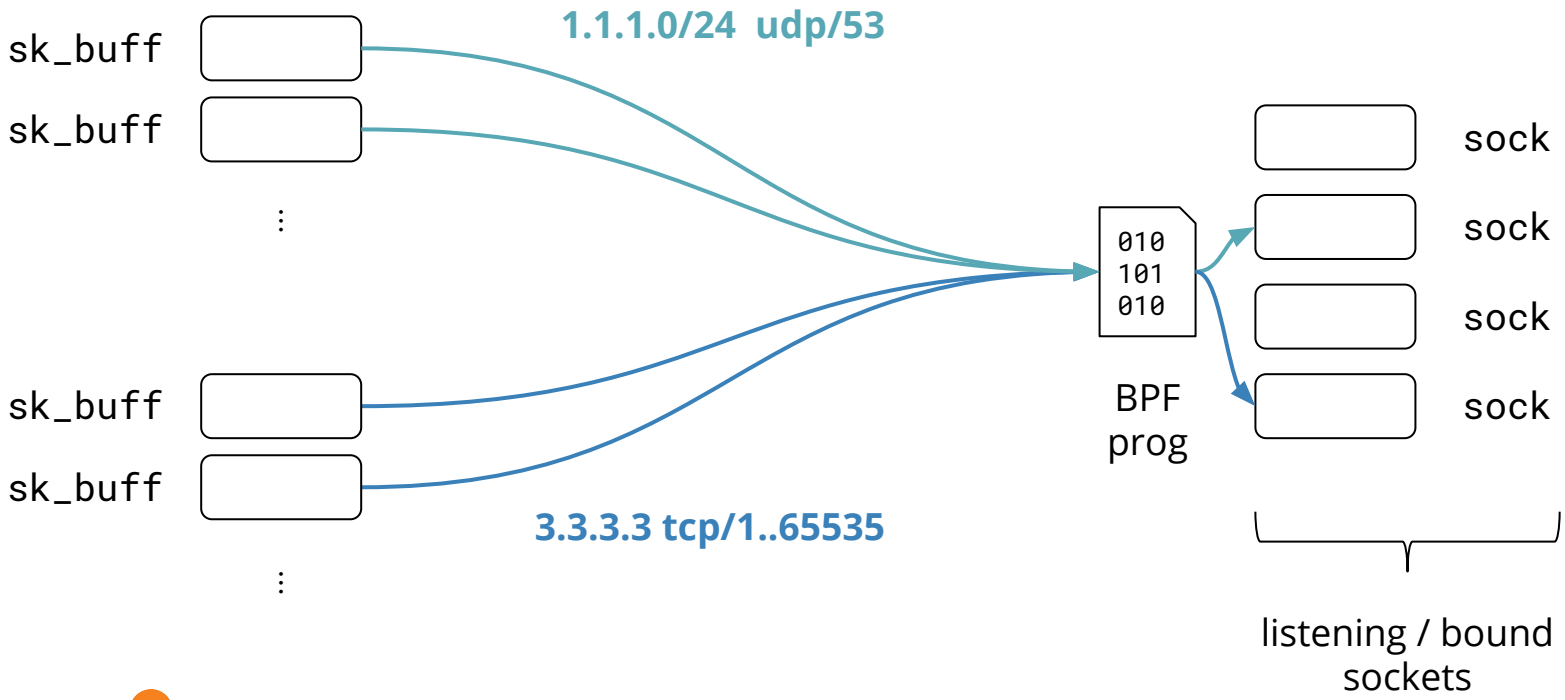
BPF inet_lookup
➔ redirect lookup for listening socket
➔ happens after connected socket lookup



CLOUDFLARE

[Wikipedia - Packet flow in Netfilter and General Networking]

Socket lookup - BPF inet_lookup redirect

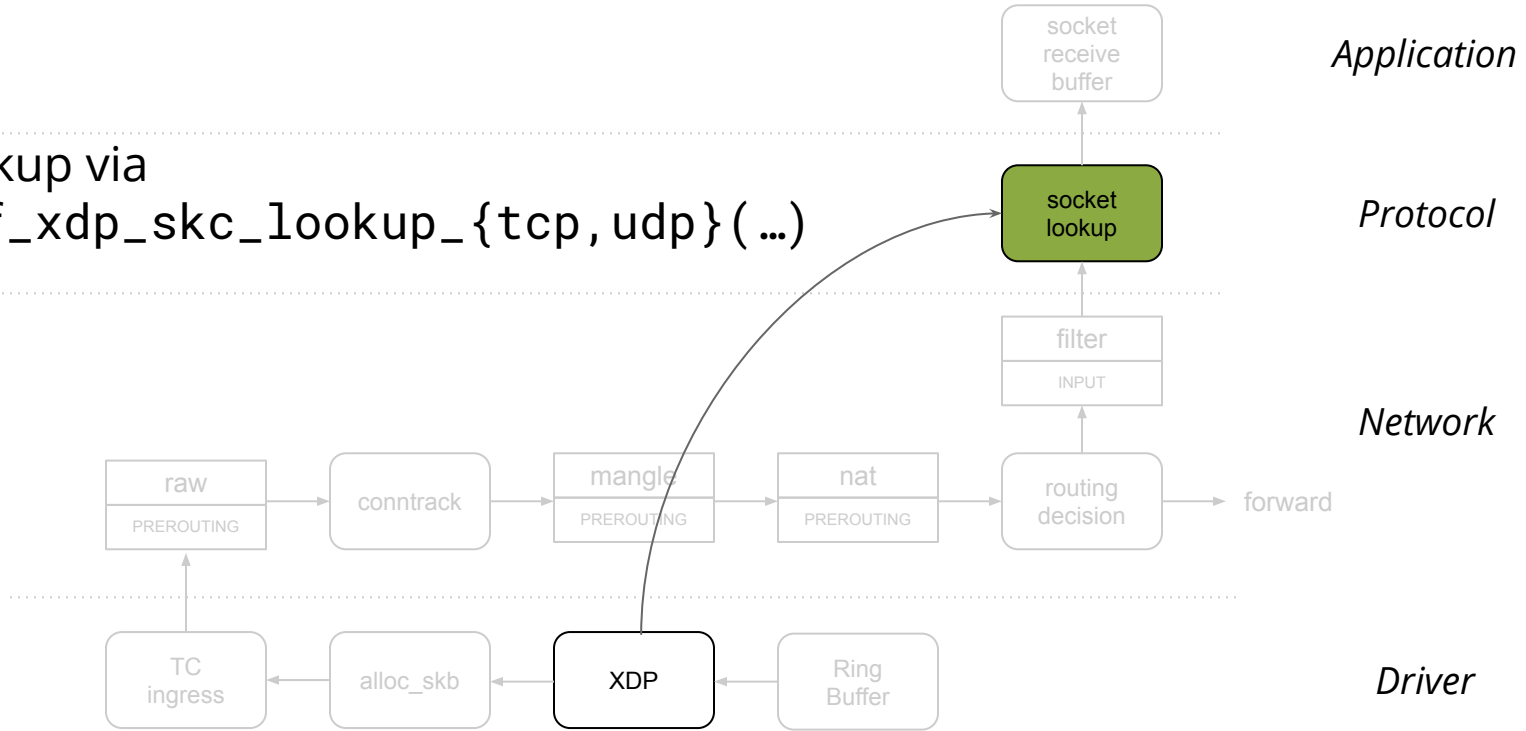


CLOUDFLARE

M:N mapping

Ingress - local delivery & XDP programs

➔ lookup via `bpf_xdp_skc_lookup_{tcp,udp}(…)`

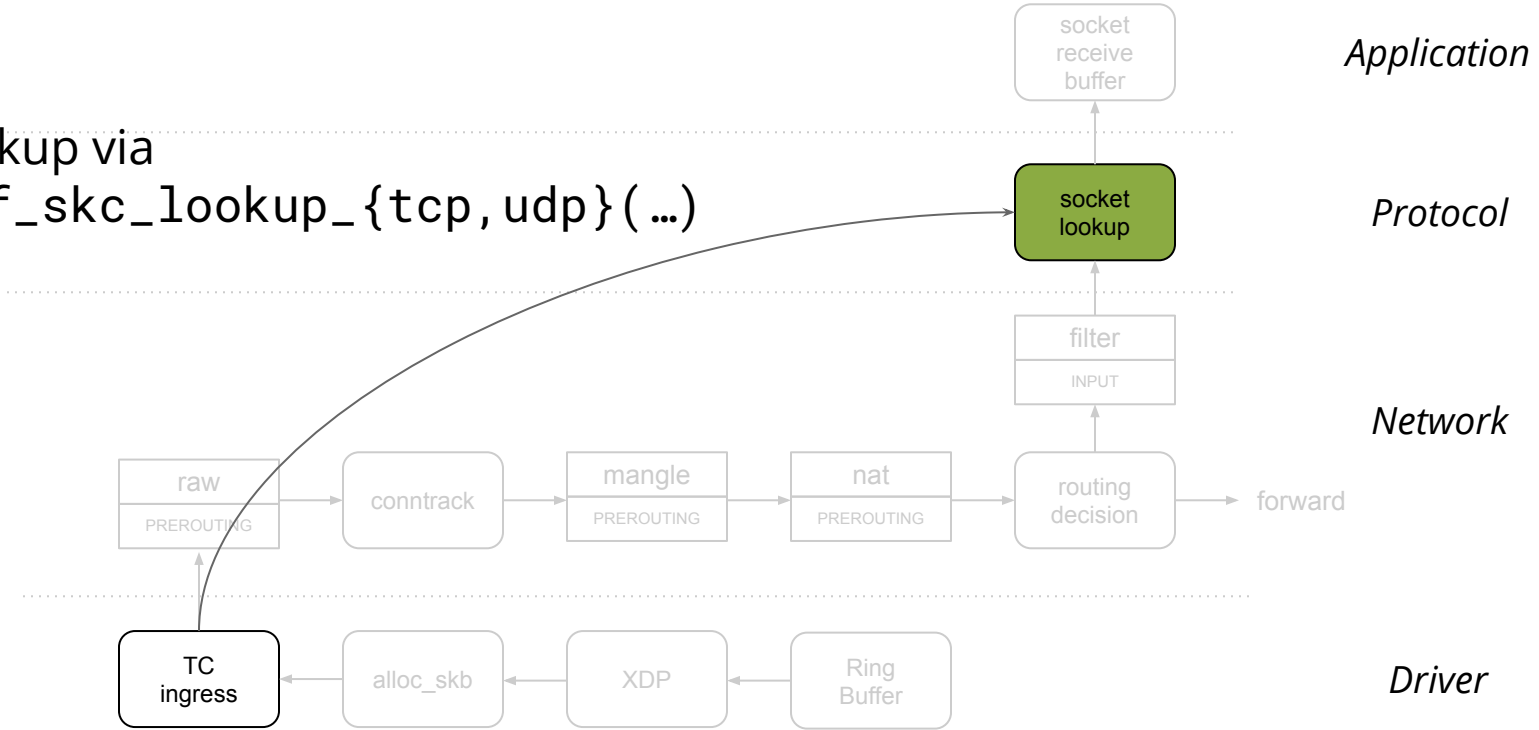


CLOUDFLARE

[Wikipedia - Packet flow in Netfilter and General Networking]

Ingress - local delivery & TC ingress cls_act

➔ lookup via `bpf_skc_lookup_{tcp,udp}(…)`

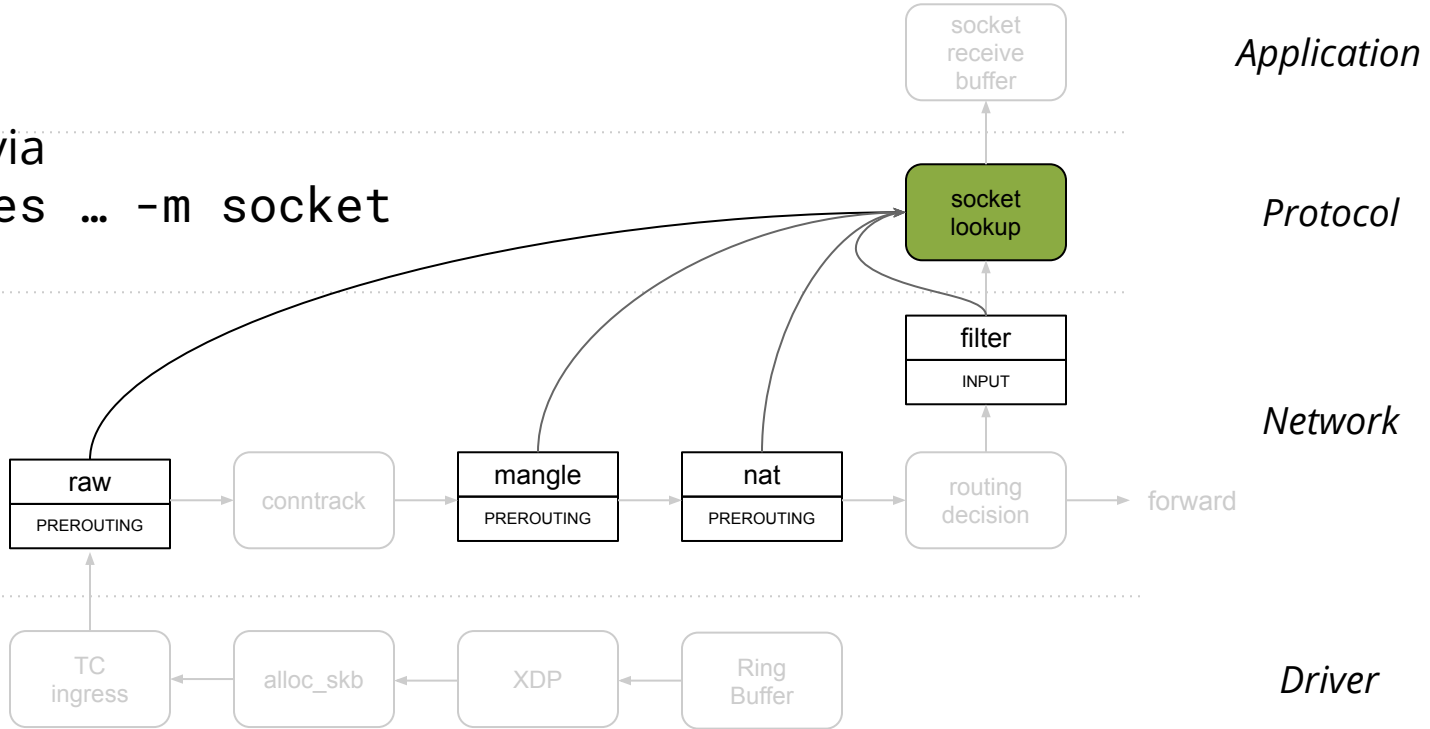


CLOUDFLARE

[Wikipedia - Packet flow in Netfilter and General Networking]

Ingress - local delivery & Netfilter hooks

➔ lookup via
`iptables … -m socket`

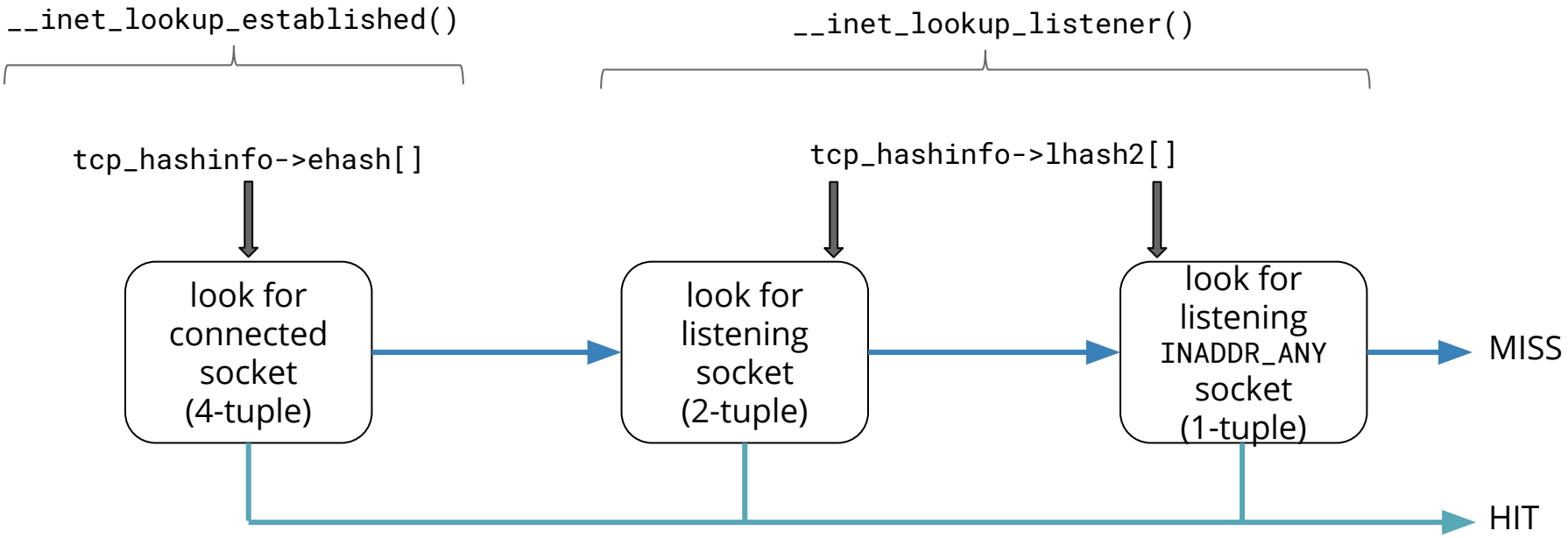


CLOUDFLARE

[Wikipedia - Packet flow in Netfilter and General Networking]

Implementation... details
RFCv2

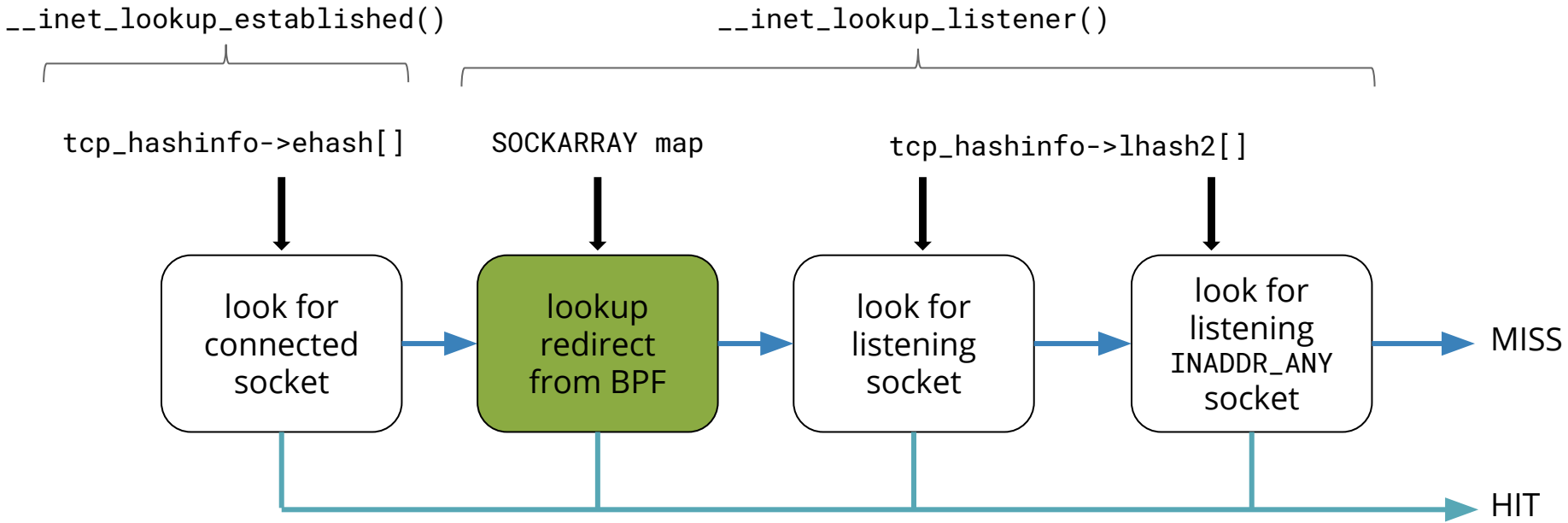
TCPv4 socket lookup - `__inet_lookup()`



* since v5.0

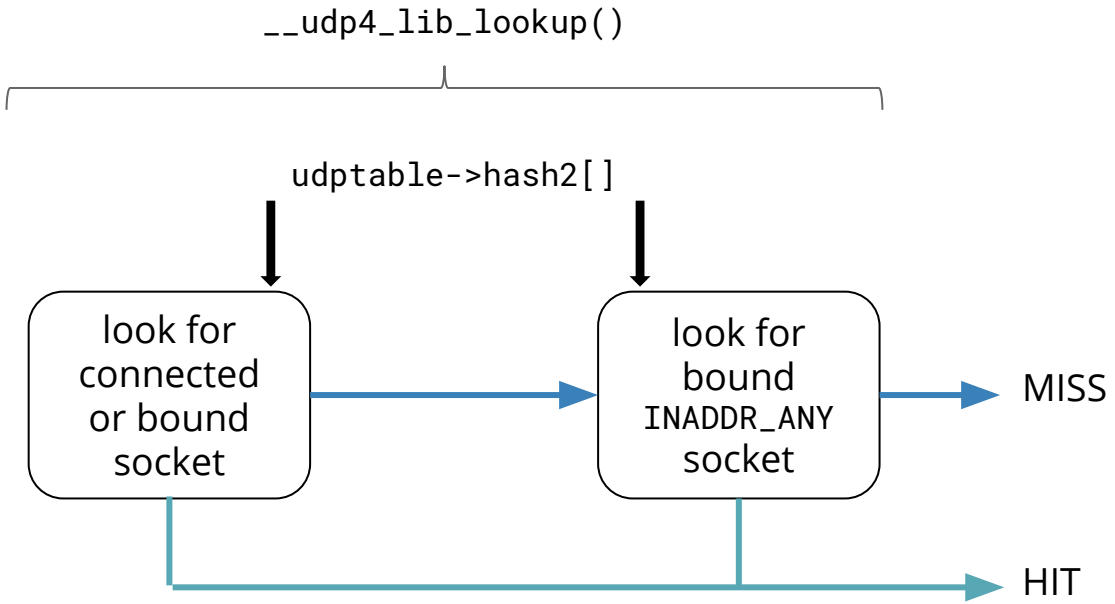
CLOUDFLARE

TCPv4 socket lookup with BPF redirect



CLOUDFLARE

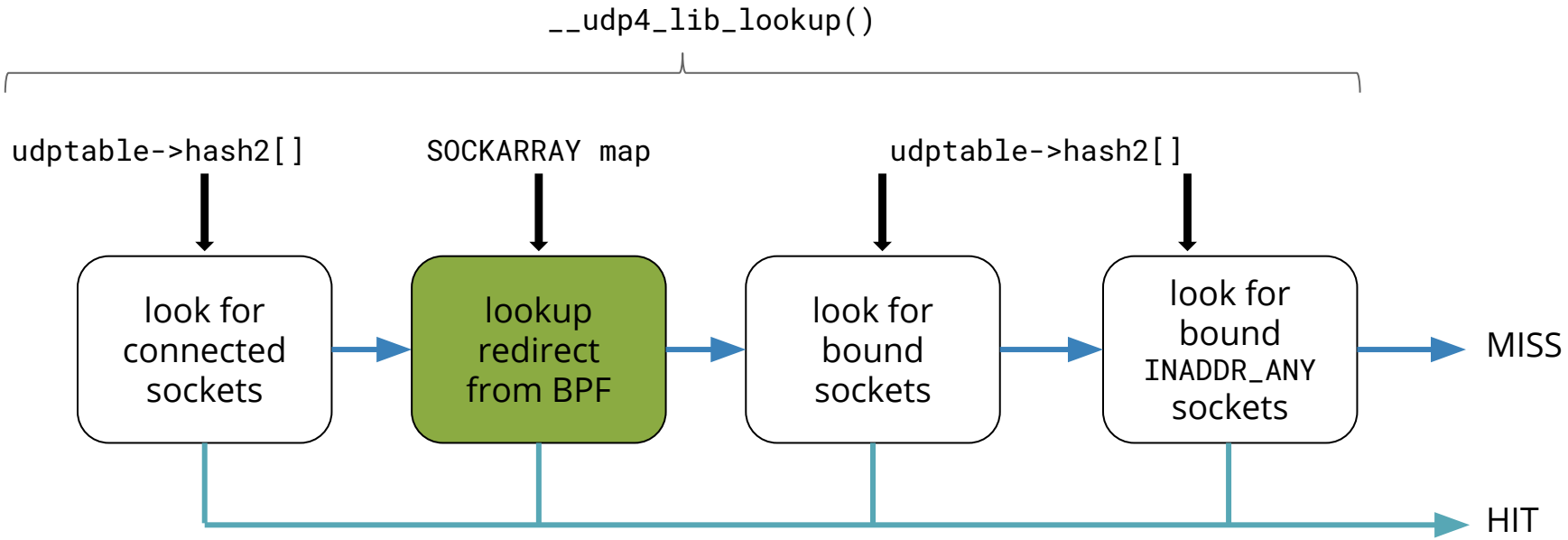
UDPv4 socket lookup - __udp4_lib_lookup



* since v5.0

CLOUDFLARE

UDPv4 socket lookup with BPF redirect

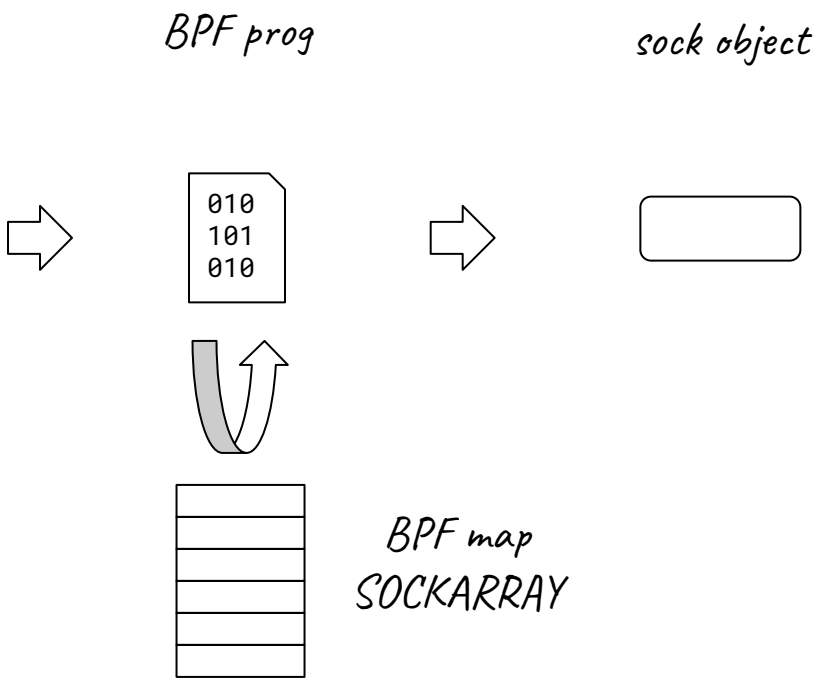


CLOUDFLARE

BPF inet_lookup prog - overview

socket lookup key

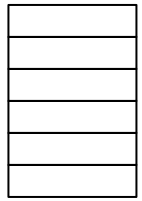
```
struct bpf_inet_lookup {
    __u32 family;
    __u32 protocol;
    __u32 remote_ip4;
    __u32 local_ip4;
    __u32 remote_port;
    __u32 local_port;
    /* ... */
};
```



CLOUDFLARE

inet_lookup - BPF map

```
struct {
    __uint(type, BPF_MAP_TYPE_REUSEPORT_SOCKARRAY);
    __uint(max_entries, MAX_SERVERS);
    __type(key, __u32);
    __type(value, __u64);
} redir_map SEC(".maps");
```



BPF map
SOCKARRAY

array index

struct sock ref

CLOUDFLARE

inet_lookup - BPF program

```
SEC("inet_lookup/redir_prefix")
int demo_redir_prefix(struct bpf_inet_lookup *ctx)
{
    __u32 index = 0;
    __u64 flags = 0;

    if (ctx->family != AF_INET)
        return BPF_OK;
    if (ctx->protocol != IPPROTO_TCP)
        return BPF_OK;
    if (ctx->local_port != 80)
        return BPF_OK;
    if ((bpf_ntohl(ctx->local_ip4) >> 8) != (IP4(192, 0, 2, 0) >> 8))
        return BPF_OK;

    return bpf_redirect_lookup(ctx, &redir_map, &index, flags);
}
```

CLOUDFLARE

inet_lookup - BPF program

```
SEC("inet_lookup/redir_prefix")
int demo_redir_prefix(struct bpf_inet_lookup *ctx)
{
    __u32 index = 0;
    __u64 flags = 0;

    if (ctx->family != AF_INET)
        return BPF_OK;
    if (ctx->protocol != IPPROTO_TCP)
        return BPF_OK;
    if (ctx->local_port != 80)
        return BPF_OK;
    if ((bpf_ntohl(ctx->local_ip4) >> 8) != (IP4(192, 0, 2, 0) >> 8))
        return BPF_OK;

    return bpf_redirect_lookup(ctx, &redir_map, &index, flags);
}
```

match on 192.0.2.0/8 tcp/80

CLOUDFLARE

inet_lookup - BPF program

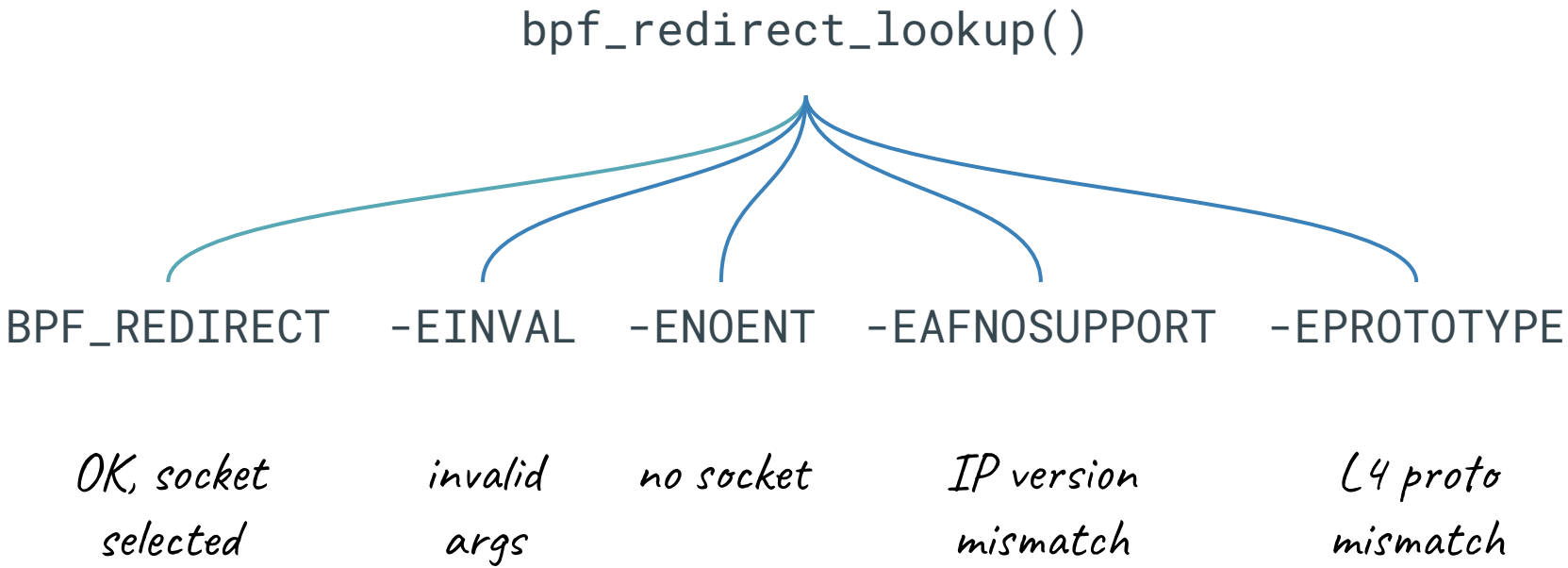
```
SEC("inet_lookup/redir_prefix")
int demo_redir_prefix(struct bpf_inet_lookup *ctx)
{
    __u32 index = 0;
    __u64 flags = 0;

    if (ctx->family != AF_INET)
        return BPF_OK;
    if (ctx->protocol != IPPROTO_TCP)
        return BPF_OK;
    if (ctx->local_port != 80)
        return BPF_OK;
    if ((bpf_ntohl(ctx->local_ip4) >> 8) != (IP4(192, 0, 2, 0) >> 8))
        return BPF_OK;

    return bpf_redirect_lookup(ctx, &redir_map, &index, flags);
}
```

action - redirect

map-based socket selection
returns BPF_REDIRECT

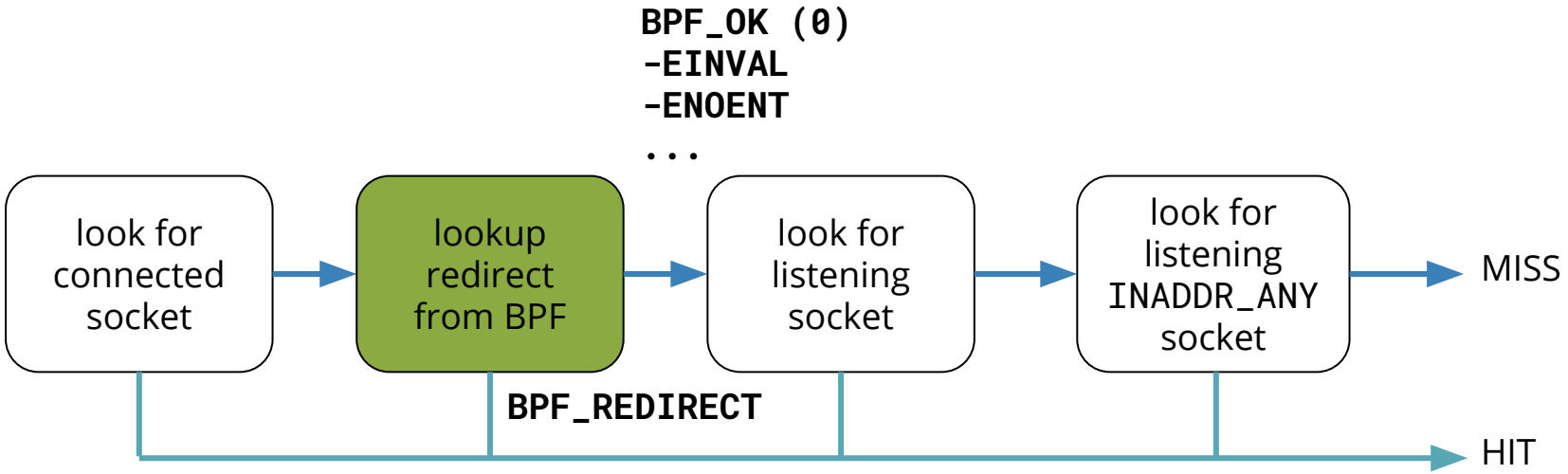
CLOUDFLARE

inet_lookup - BPF helper



CLOUDFLARE

inet_lookup - return value



User-space API RFCv2

User-space API

1. create SOCKARRAY map

```
union bpf_attr attr = {
    .map_type = BPF_MAP_TYPE_REUSEPORT_SOCKARRAY,
    /* ... */
};

map_fd = bpf(BPF_MAP_CREATE, &attr, sizeof(attr));
```

CLOUDFLARE

User-space API

1. create SOCKARRAY map

```
union bpf_attr attr = {
    .map_type = BPF_MAP_TYPE_REUSEPORT_SOCKARRAY,
    /* ... */
};

map_fd = bpf(BPF_MAP_CREATE, &attr, sizeof(attr));
```

misleading type name

User-space API

2. insert sockets into SOCKARRAY map

```
uint32_t index = 0;
uint64_t value = sock_fd;
union bpf_attr attr = {
    .map_fd = map_fd,
    .key = (unsigned long)&index,
    .value = (unsigned long)&value,
};
err = bpf(BPF_MAP_UPDATE_ELEM, &attr, sizeof(attr));
```

User-space API

2. insert sockets into SOCKARRAY map

```
uint32_t index = 0;
uint64_t value = sock_fd;
union bpf_attr attr = {
    .map_fd = map_fd,
    .key = (unsigned long)&index,
    .value = (unsigned long)&value,
};
err = bpf(BPF_MAP_UPDATE_ELEM, &attr, sizeof(attr));
```

must be a listening socket
must have SO_REUSEPORT set
map can't be updated from BPF

CLOUDFLARE

User-space API

2. insert sockets into SOCKARRAY map

```
uint32_t index = 0;
uint64_t value = sock_fd;
union bpf_attr attr = {
    .map_fd = map_fd,
    .key = (unsigned long)&index,
    .value = (unsigned long)&value,
};
err = bpf(BPF_MAP_UPDATE_ELEM, &attr, sizeof(attr));
```

how to get the FD???

User-space API

3. load BPF inet_lookup program

```
union bpf_attr attr = {
    .prog_type = BPF_PROG_TYPE_INET_LOOKUP,
    /* ... */
};

prog_fd = bpf(BPF_PROG_LOAD, &attr, sizeof(attr));
```

new program type

CLOUDFLARE

User-space API

4. attach BPF inet_lookup program

```
union bpf_attr attr = {
    .attach_type    = BPF_INET_LOOKUP,
    .attach_bpf_fd = prog_fd,
    .attach_flags  = 0,
    .target_fd     = -1,
};
err = bpf(BPF_PROG_ATTACH, &attr, sizeof(attr));
```

new attach point
tied to network namespace

CLOUDFLARE

User-space API

4. attach BPF inet_lookup program

```
union bpf_attr attr = {
    .attach_type    = BPF_INET_LOOKUP,
    .attach_bpf_fd = prog_fd,
    .attach_flags   = 0,
    .target_fd      = -1,
};
err = bpf(BPF_PROG_ATTACH, &attr, sizeof(attr));
```

current netns always used
target FD ignored

CLOUDFLARE

User-space API

4. attach BPF inet_lookup program

```
union bpf_attr attr = {
    .attach_type    = BPF_INET_LOOKUP,
    .attach_bpf_fd = prog_fd,
    .attach_flags   = 0,
    .target_fd      = -1,
};
err = bpf(BPF_PROG_ATTACH, &attr, sizeof(attr));
```

flags ignored
BPF_F_ALLOW_OVERRIDE
not supported yet

CLOUDFLARE

User-space API

2. insert sockets into SOCKARRAY map

```
uint32_t index = 0;
uint64_t value = sock_fd;
union bpf_attr attr = {
    .map_fd = map_fd,
    .key = (unsigned long)&index,
    .value = (unsigned long)&value,
};
err = bpf(BPF_MAP_UPDATE_ELEM, &attr, sizeof(attr));
```

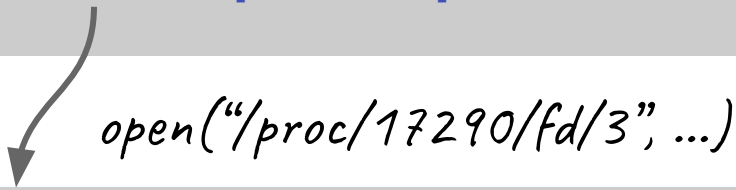
how to get the FD???

Populate the SOCKARRAY - problem

SOCKARRAY owner != listening socket owner

Populate the SOCKARRAY - problem

```
$ ls -lG /proc/17290/fd/
total 0
lrwx------. 1 jkbs 64 Aug 31 20:49 0 -> /dev/pts/4
lrwx------. 1 jkbs 64 Aug 31 20:49 1 -> /dev/pts/4
lrwx------. 1 jkbs 64 Aug 31 20:49 2 -> /dev/pts/4
lrwx------. 1 jkbs 64 Aug 31 20:49 3 -> 'socket:[23103837]'
```

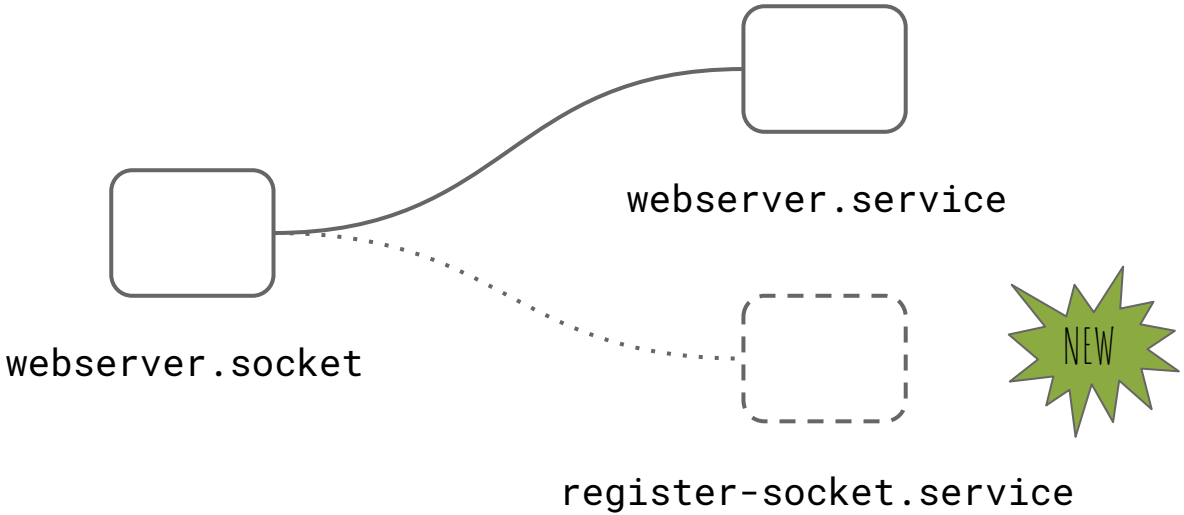



```
openat(AT_FDCWD, "/proc/17290/fd/3", O_RDONLY) = -1 ENXIO
                                                 (No such device or address)
```

CLOUDFLARE

Getting the socket - solutions

1. systemd socket activation + socket registering service



CLOUDFLARE

Getting the socket - solutions

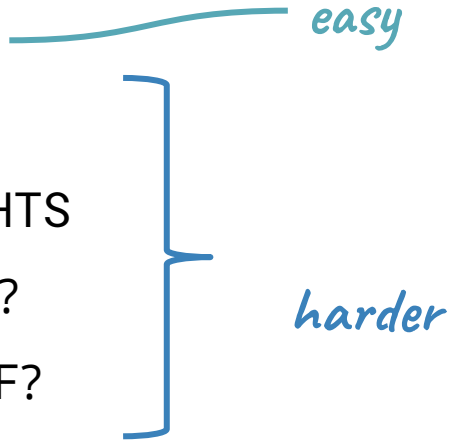
1. systemd socket activation + socket registering service

```
$ cat /etc/systemd/system/register-socket.service
[Unit]
Description=Register Webserver Socket
After=network.target
Requires=webserver.socket

[Service]
Sockets=webserver.socket
ExecStart=inet-tool register “webserver”
```

CLOUDFLARE

Getting the socket - solutions

1. systemd socket activation + socket registering service
2. overload `listen(2)` / `bind(2)` with `LD_PRELOAD`
3. adapt the application & send socket FD with `SCM_RIGHTS`
4. extend `SOCKARRAY` `map_update` to take socket cookie?
5. let BPF programs update `SOCKARRAY & use TCP-BPF`?



Live demo?
or pre-recorded

Performance - TCP SYN flood

CLOUDFLARE

CC0 Public Domain photo, https://pxhere.com/en/photo/660445

Performance - UDP flood

CLOUDFLARE

CC0 Public Domain photo, https://pxhere.com/en/photo/660445

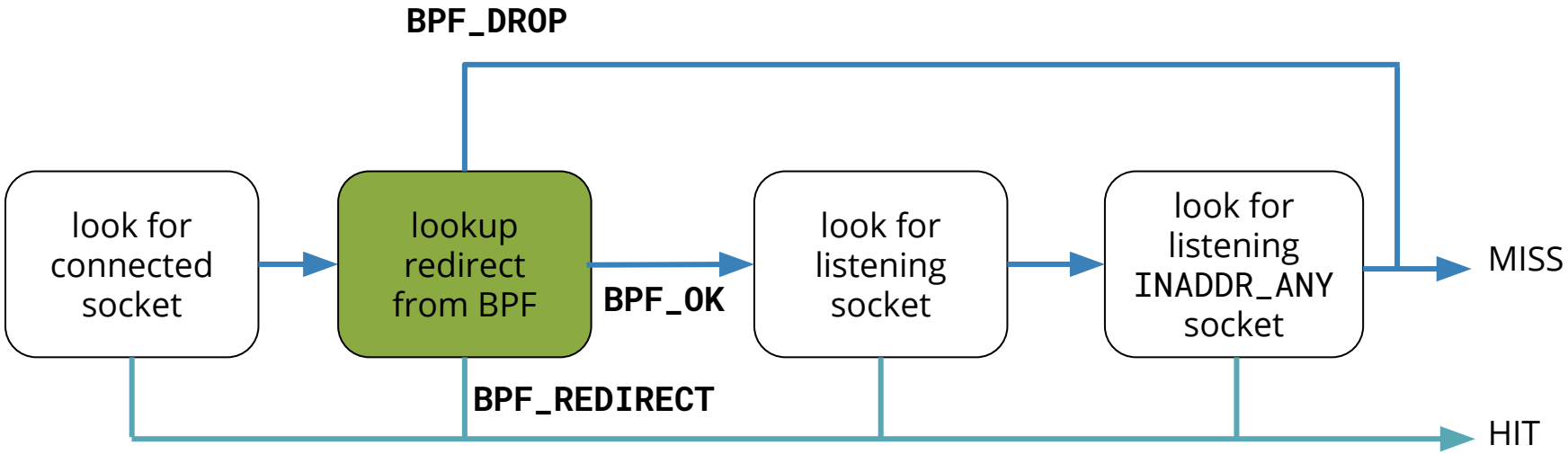
What’s next?
After RFCv2...

Next steps - add ingress device to context

```
struct bpf_inet_lookup {
    __u32 family;
    __u32 protocol;
    __u32 remote_ip4;
    __u32 local_ip4;
    /* ... */
    __u32 ifindex;  /* ingress device index */
    __u32 sifindex; /* enslaved ingress device index (VRF) */
};
```

CLOUDFLARE

Next steps - fail the lookup on BPF_DROP



CLOUDFLARE

Next steps -
atomic re-attach & target netns support

```
union bpf_attr attr = {
    .attach_type    = BPF_INET_LOOKUP,
    .attach_bpf_fd = prog_fd,
    .attach_flags   = BPF_F_ALLOW_OVERRIDE,
    .target_fd      = netns_fd,
};
err = bpf(BPF_PROG_ATTACH, &attr, sizeof(attr));
```

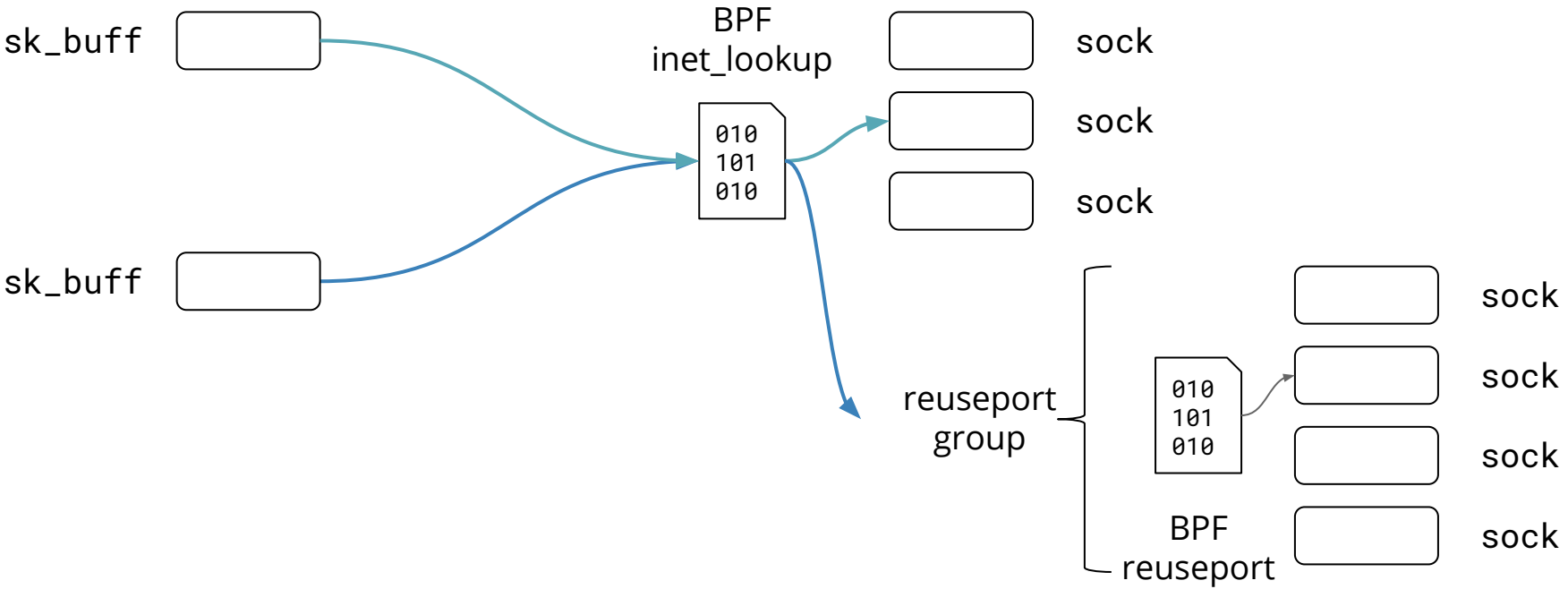
CLOUDFLARE

Next steps -
lift SO_REUSEPORT check for SOCKARRAY

```
union bpf_attr attr = {
    .map_type  = BPF_MAP_TYPE_REUSEPORT_SOCKARRAY,
    .map_flags = BPF_F_SOCKARRAY_NO_REUSEPORT,
    /* ... */
};

map_fd = bpf(BPF_MAP_CREATE, &attr, sizeof(attr));
```

CLOUDFLARE

Next steps - support reuseport groups



CLOUDFLARE

Next steps -
use socket cookie to insert into SOCKARRAY

```
uint32_t index = 0;
uint64_t value = sock_cookie;
union bpf_attr attr = {
    .map_fd = map_fd,
    .key = (unsigned long)&index,
    .value = (unsigned long)&value,
    .flags = BPF_F_SOCKARRAY_COOKIE,
};
err = bpf(BPF_MAP_UPDATE_ELEM, &attr, sizeof(attr));
```

getsockopt(..., SO_COOKIE, ...)
or sock_diag(7) / ss tool

CLOUDFLARE

Thank you!

Questions or feedback?

Resources - BPF inet_lookup

Kernel code

- RFCv2 tag on GitHub or patches at lore.kernel.org for bpf-next
- backport to 5.2 stable

User-space tool from demo

- inet-tool repo

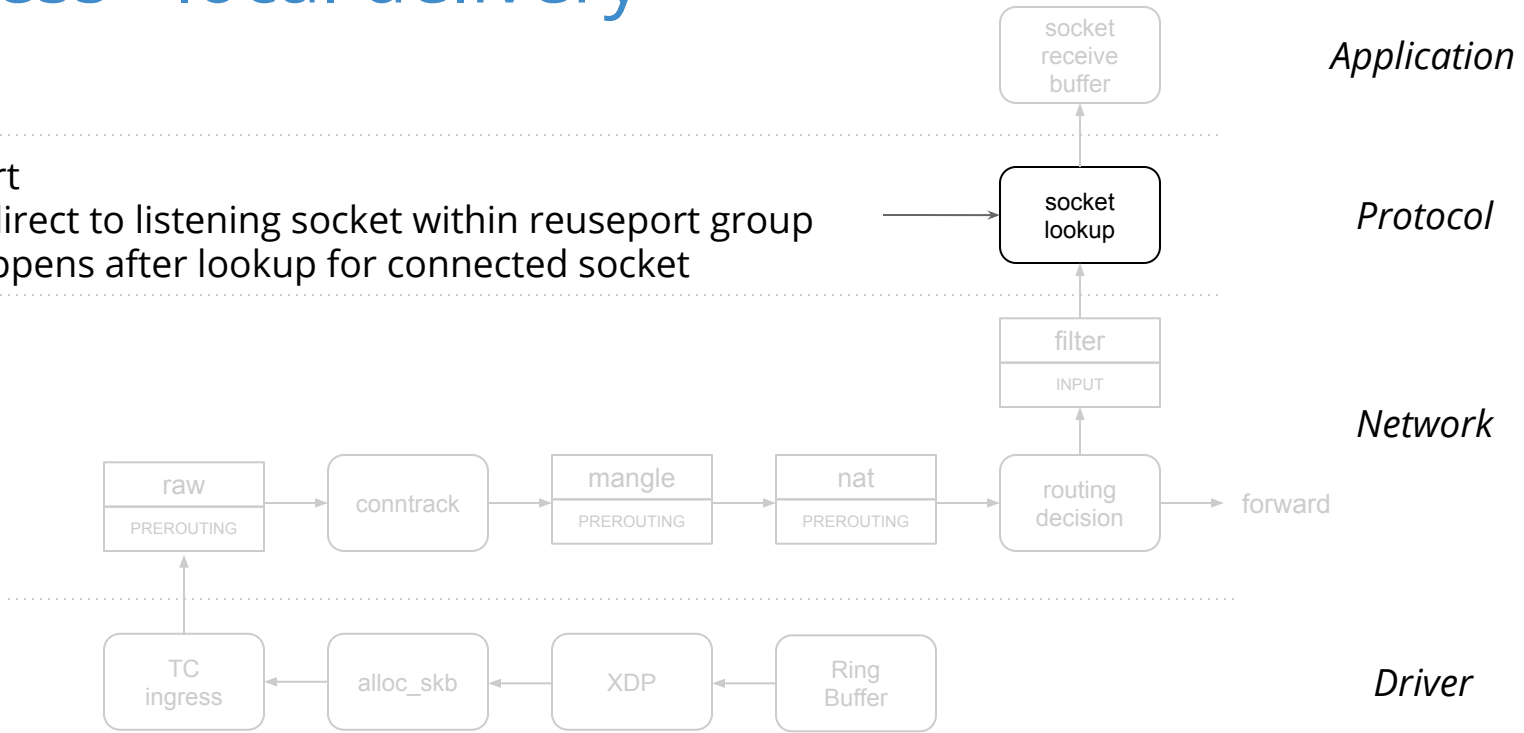
Background

- Presentation from Netconf 2019
- Abusing Linux's firewall: the hack that allowed us to build Spectrum blog
- Revenge of the listening sockets blog
- RFC SO_BINDTOPREFIX patches

-EOVERFLOW

Ingress - local delivery

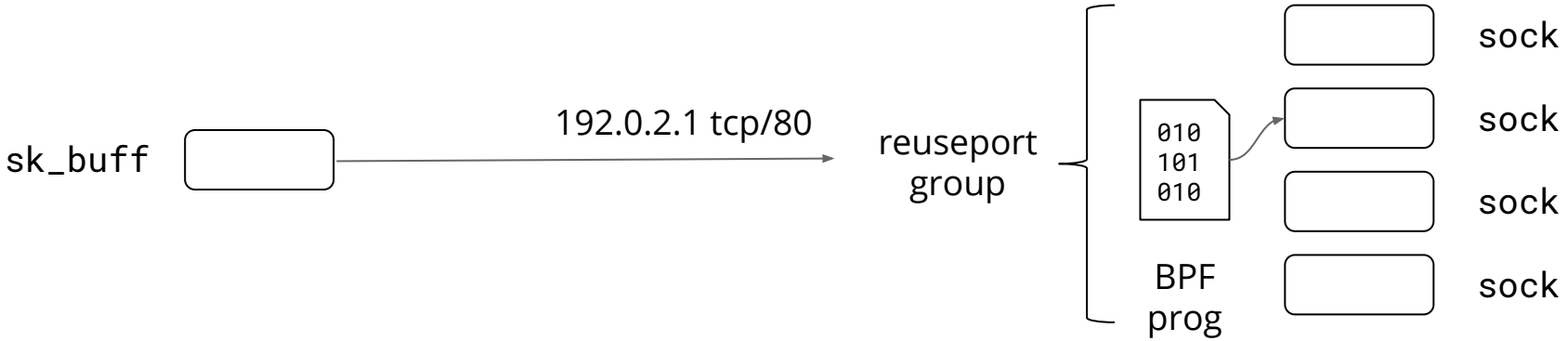
reuseport
- ➔ redirect to listening socket within reuseport group
- ➔ happens after lookup for connected socket



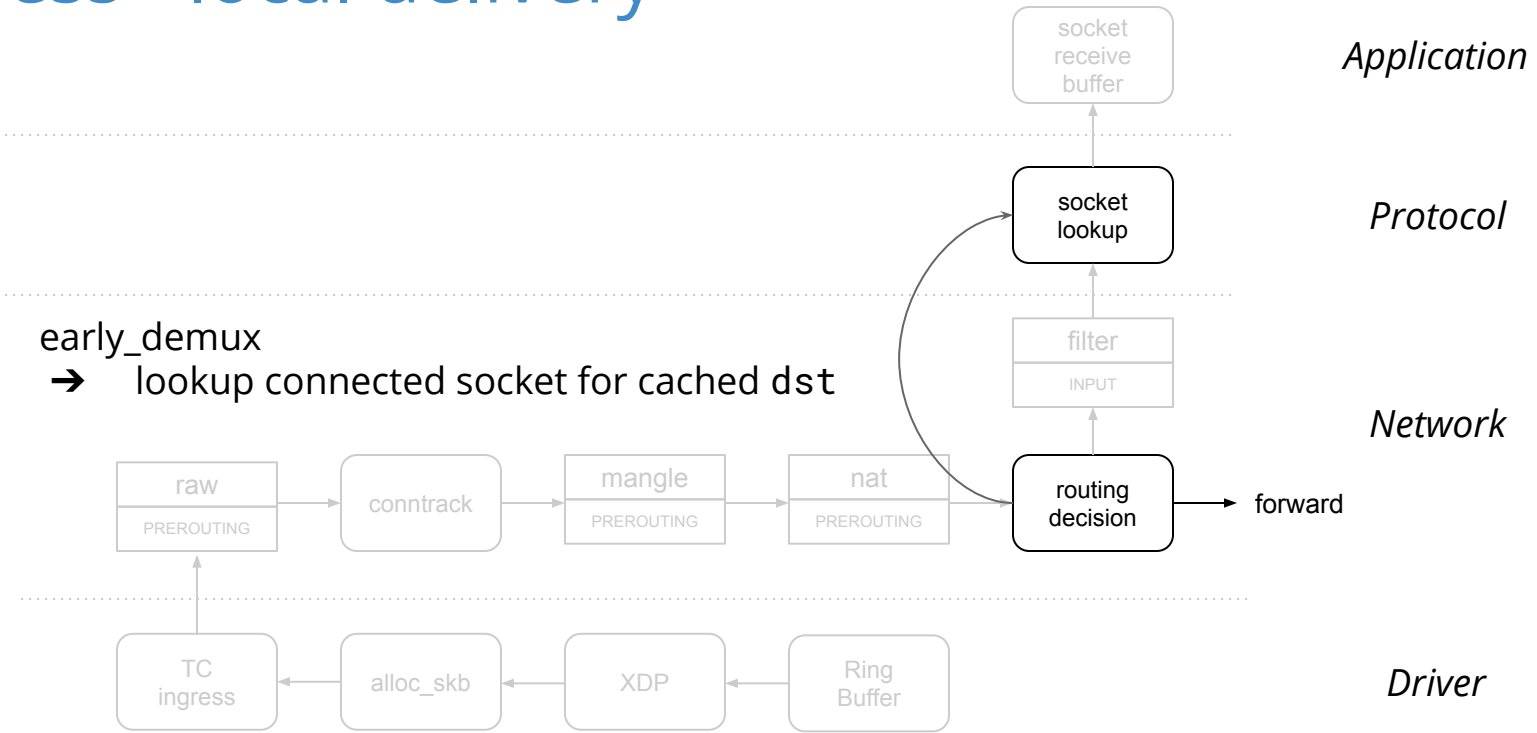
CLOUDFLARE

[Wikipedia - Packet flow in Netfilter and General Networking]

Ingress - socket lookup - reuseport redirect



Ingress - local delivery



CLOUDFLARE

[Wikipedia - Packet flow in Netfilter and General Networking]