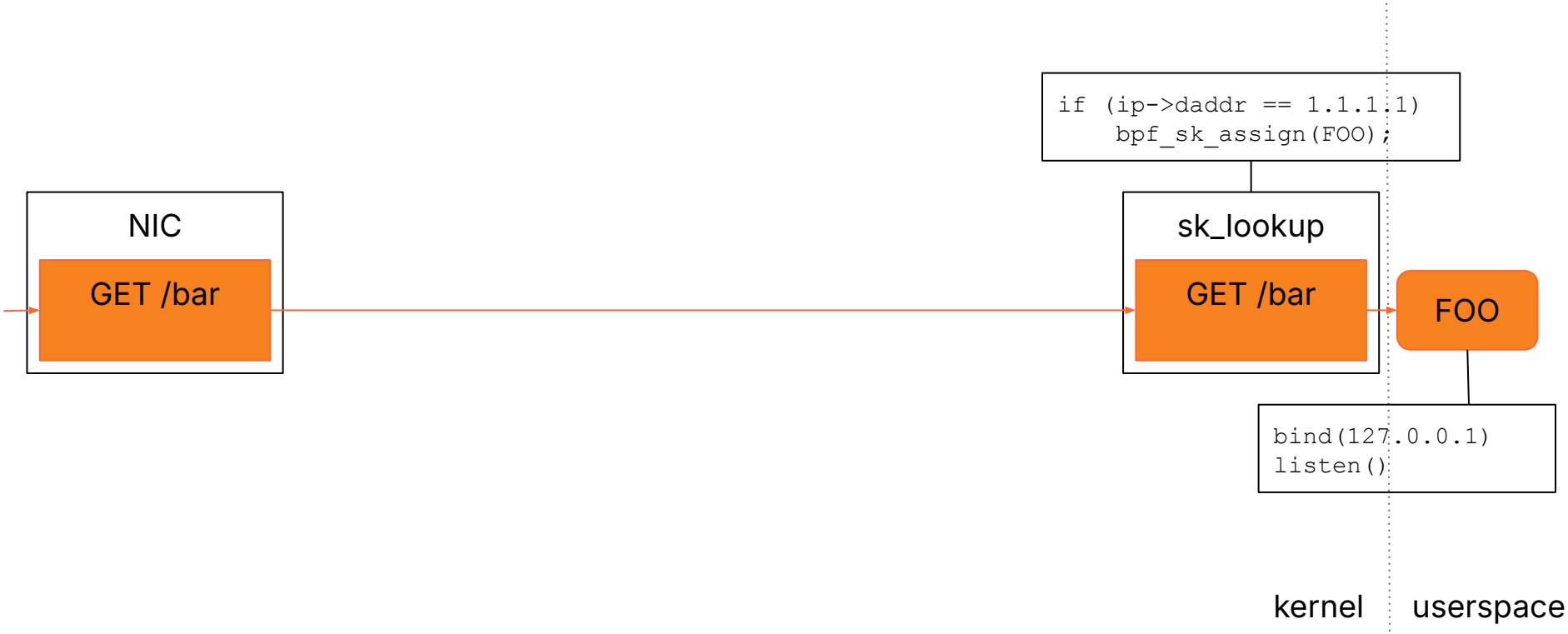
Traits: Rich Packet Metadata

Arthur Fabre
Jakub Sitnicki
Jesper Dangaard Brouer

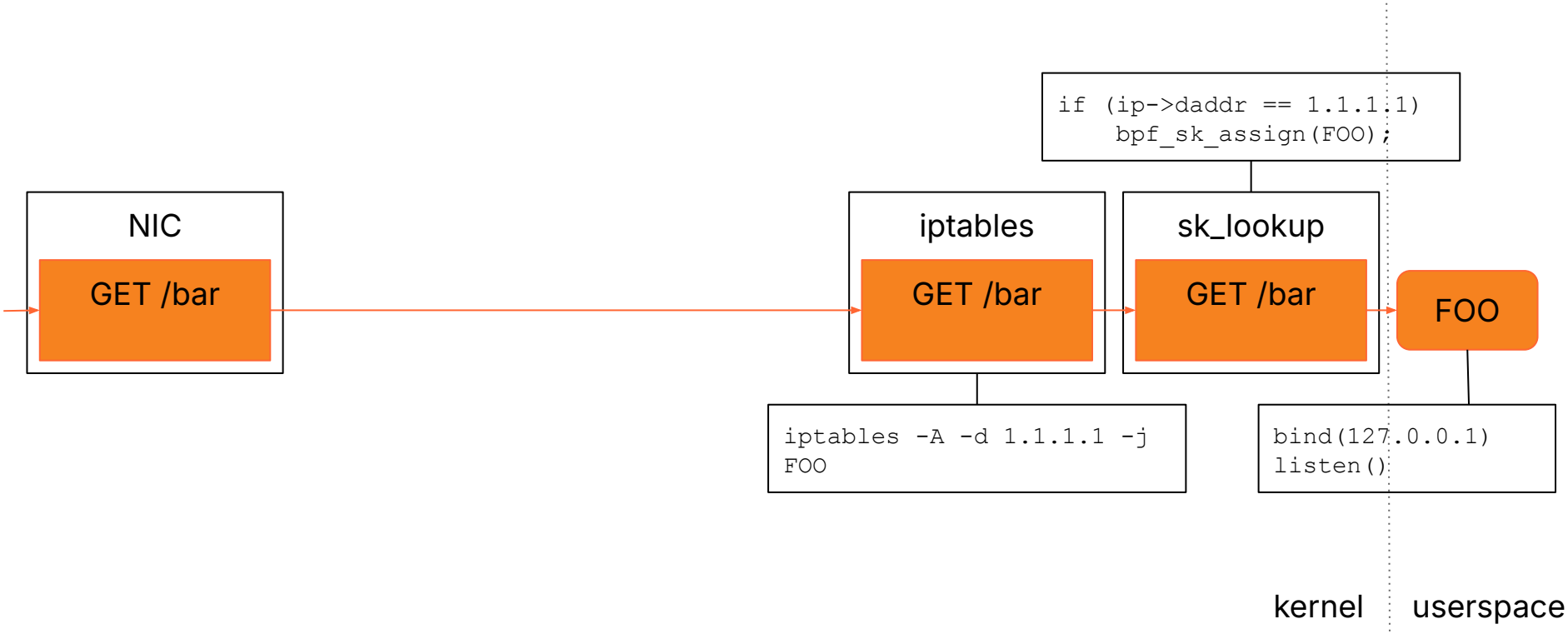
Netdev 0x19
March 11, 2025
Zagreb, Croatia

Life of a Packet

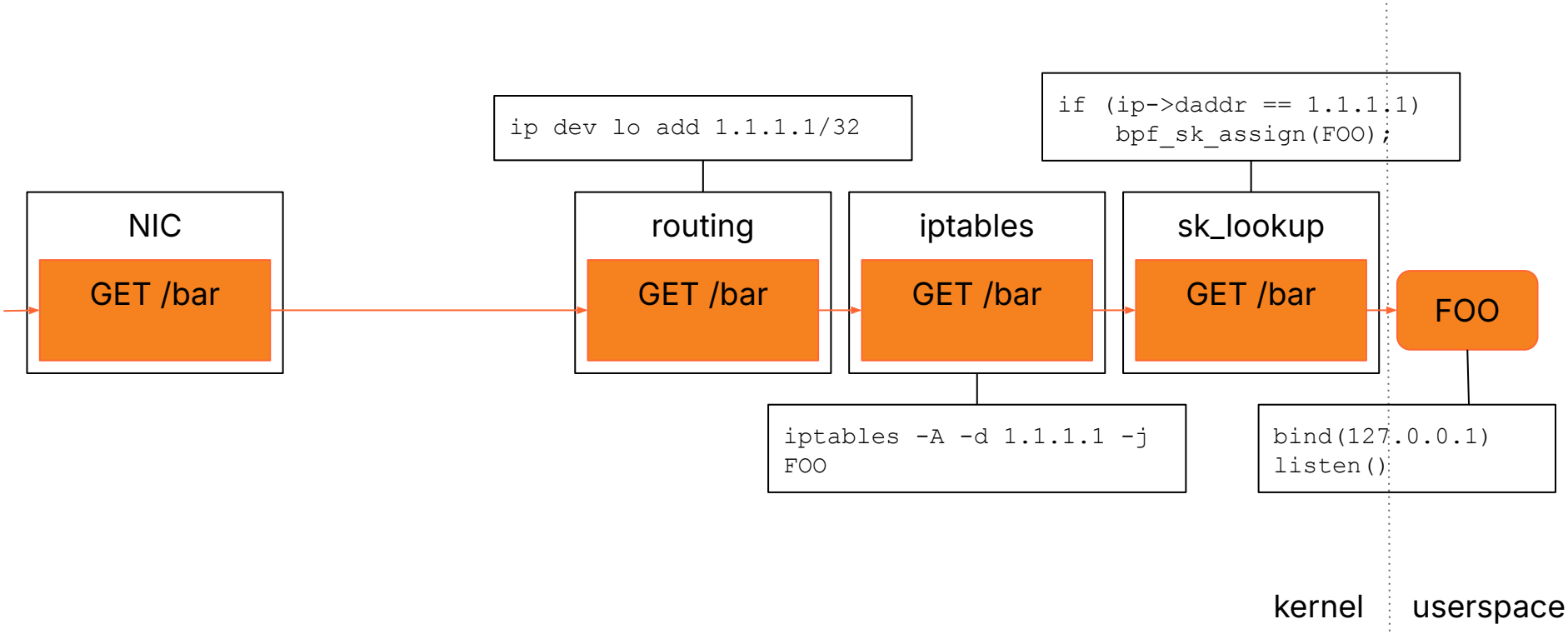
Life of a Packet: Sockets



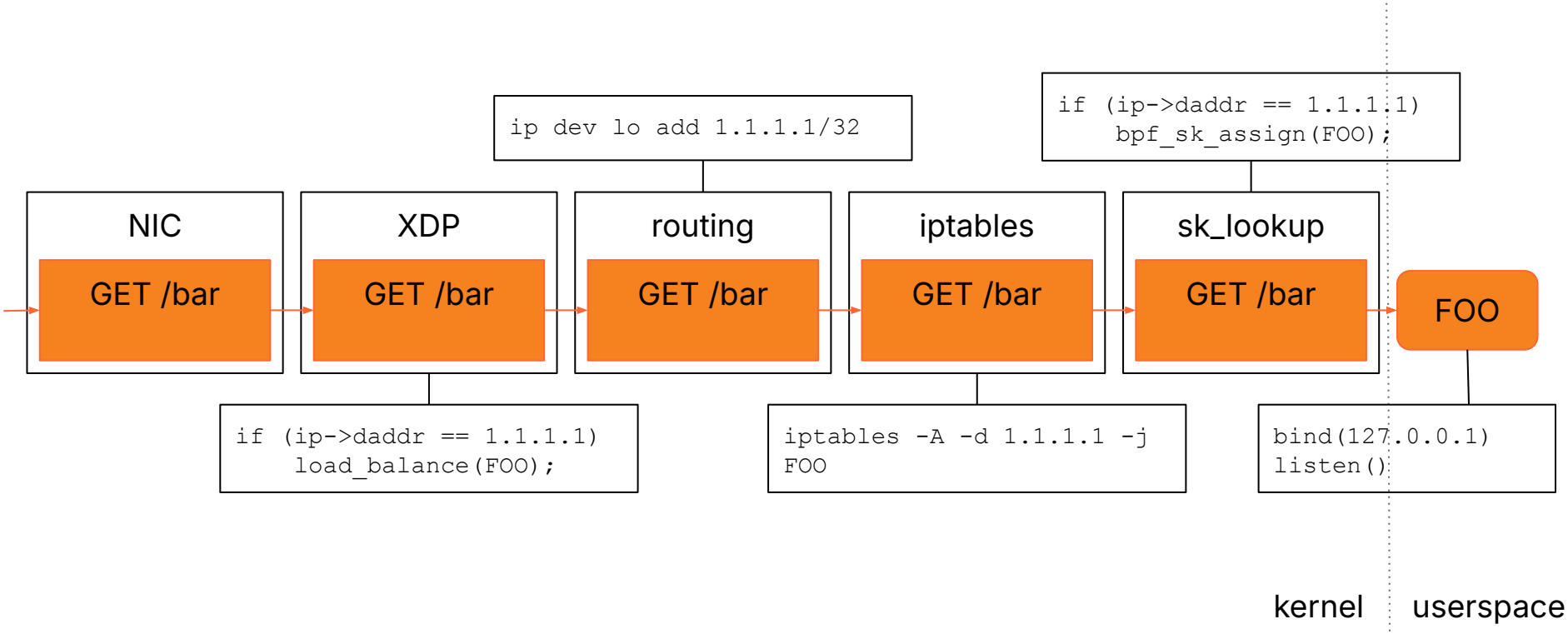
Life of a Packet: iptables



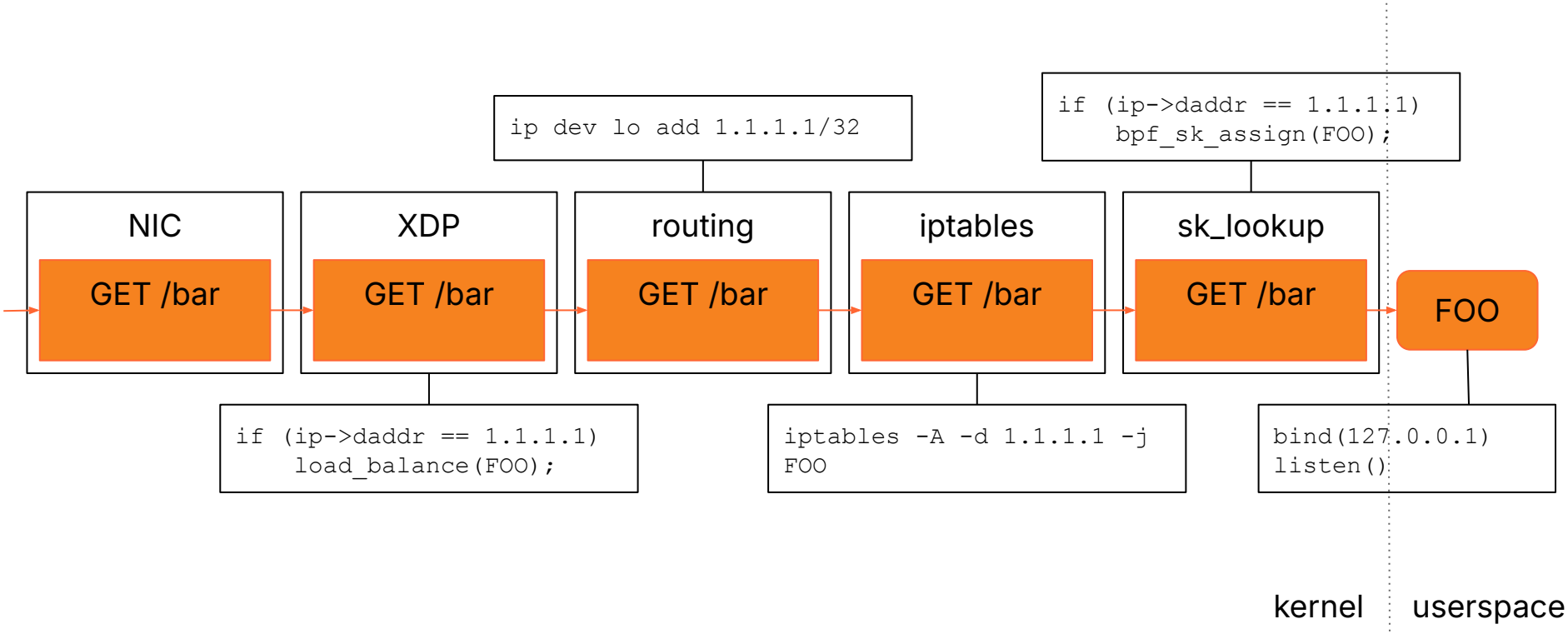
Life of a Packet: Routing



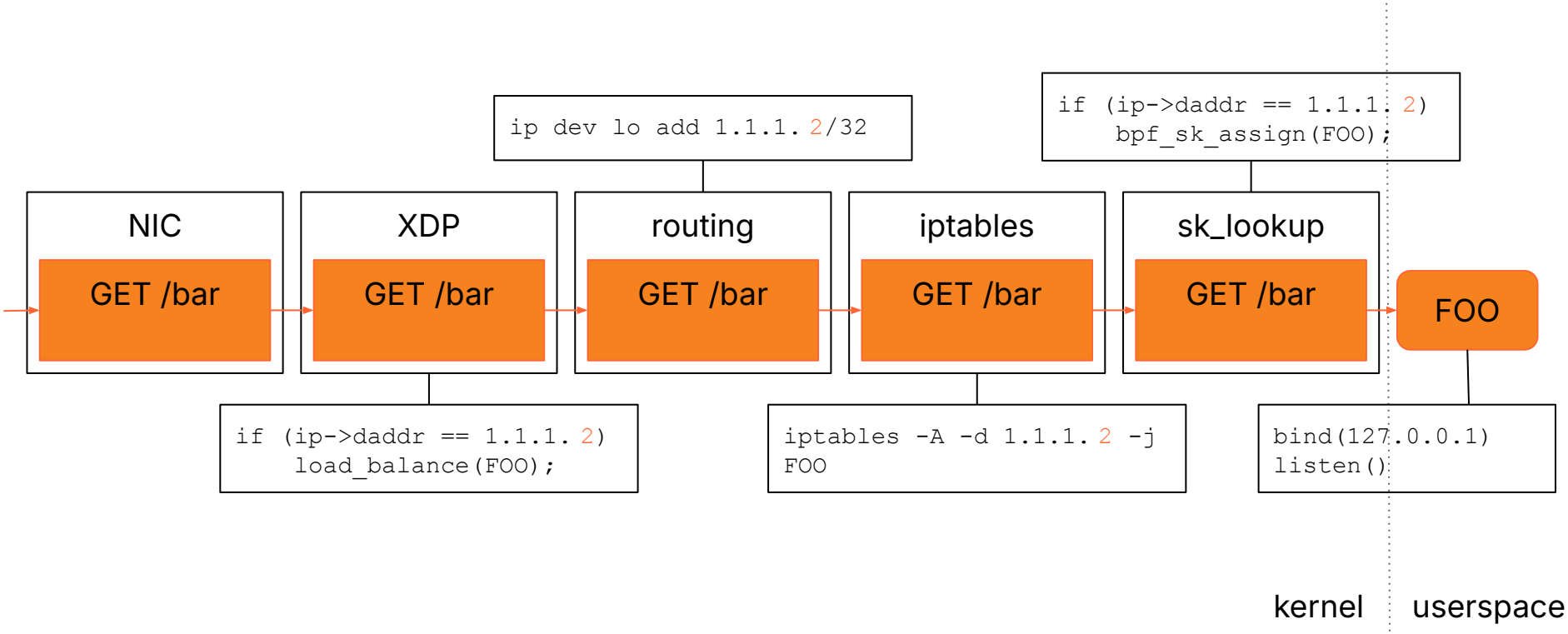
Life of a Packet: XDP



Configuration Everywhere



Inconsistency Everywhere

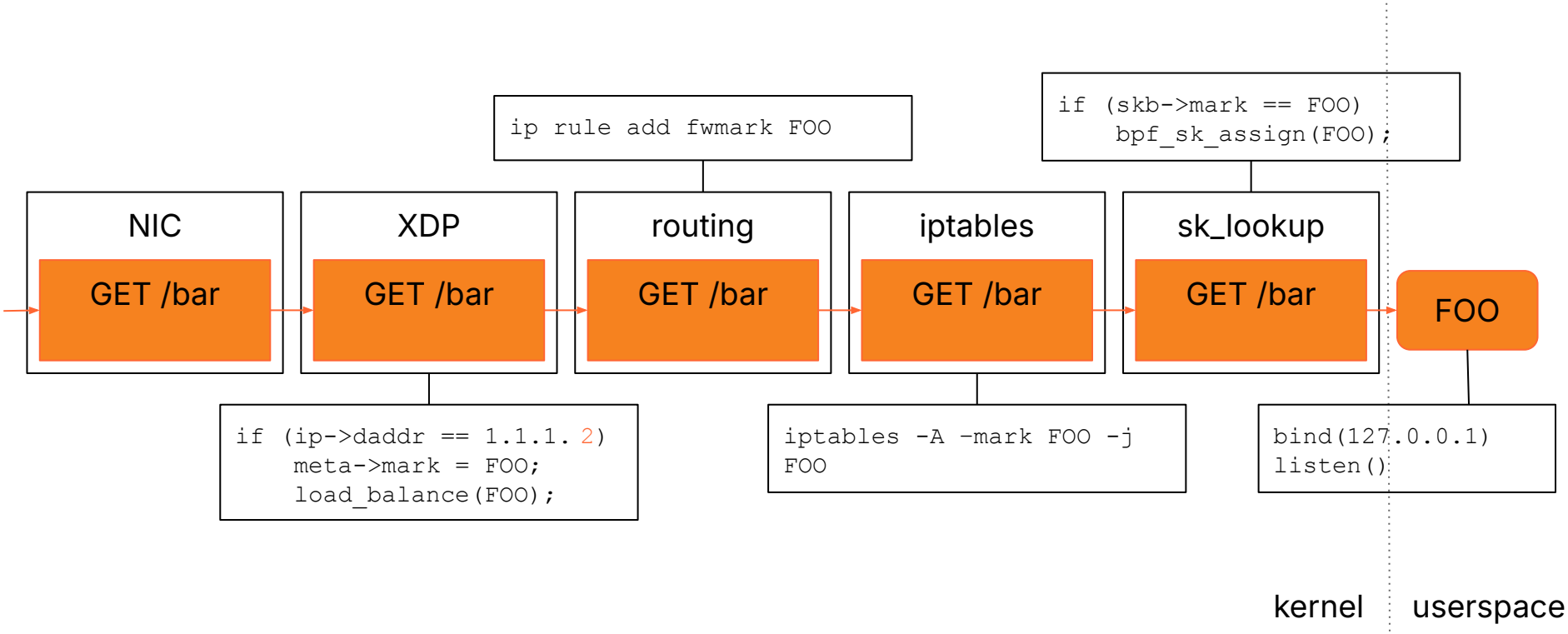


CLOUDFLARE

Mark?

- `sk_buff->mark`
- `fw_mark`
- `ct_mark`
- `SO_MARK`

Mark Everywhere



And Much More

- Packet tracing
  - Unique ID in mark
- Hardware metadata
  - Receive timestamp, RX-hash, checksum
- Network metadata
  - Encapsulated?

Everyone Wants a Bit

- 32 bits ought to be enough for anyone

Everyone Use the Same Bits

Bitwise Mark Registry

Bits are numbered from 0 to 31, least-significant bit (LSB) to most-significant (MSB). For example, if only mark bit number 3 is set, the overall packet mark is 0x8. For search engine discoverability, the full mark value with individual bits set is also listed in the form that people are likely to search for.

Bits	Mark mask	Software	Source
0-12,16-31	0xFFFF1FFF	Cilium	Source code
7	0x00000080	AWS CNI	Source code
13	0x00002000	CNI Portmap	Documentation
14-15	0x0000C000	Kubernetes	Source code
16-31	0xFFFF0000	Calico	Documentation
17-18	0x60000	Weave Net	Source code
18-19	0xC0000	Tailscale	Source code

Non-Bitwise Mark Registry

Some software treats the packet mark as a simple integer, and so sets/clears all bits at once whenever it touches a packet. Such software is likely to be broadly incompatible with "bitwise" users of the packet mark.

Mark value	Software	Source
Any	OpenShift	Source code
0x00000800	Antrea	Documentation
0x1337	Istio	Documentation
0x1e7700ce	AWS AppMesh	Documentation

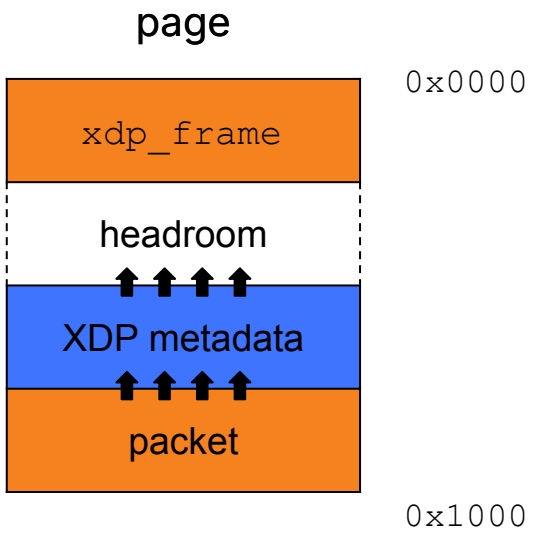
https://github.com/fwmark/registry

Alternatives

	size	allocation-free	persistent
mark	32 bits	yes	yes
skb->ext	∞	no	yes
tc_classid / skb->cb	48 bytes	yes	no
XDP metadata	256 bytes	yes	possible

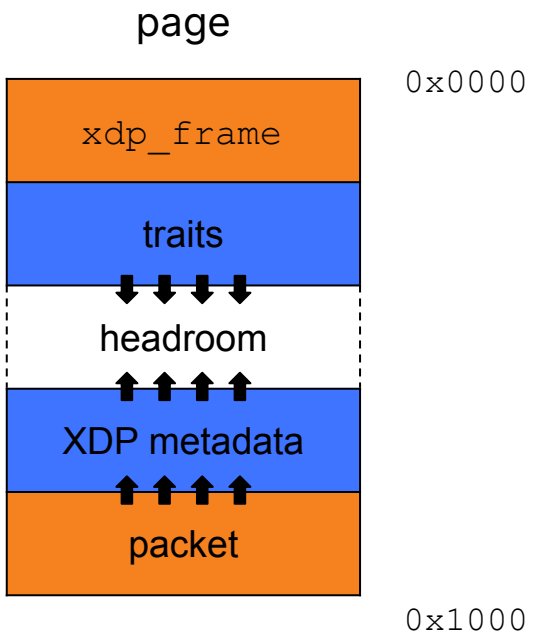
XDP Metadata

- Headroom of packet
- XDP: ~256 bytes
  - Lots of space!
- Binary blob
  - Hard to share...



Traits: Storage

- Front of headroom
- Coexist with XDP metadata
- Fast adjust_head()

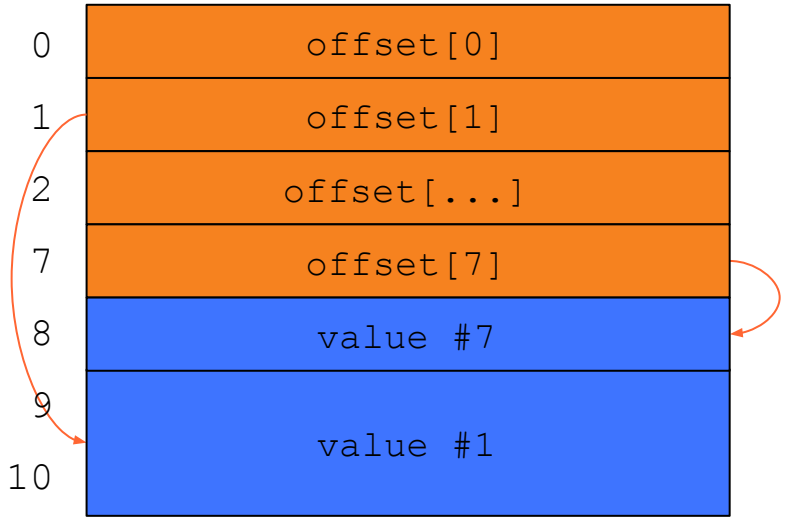


Traits: Encoding

- KV store
- Keys per service
- Nicer API
  - `get()`
  - `set()`
  - `del()`
  - BPF & Userspace

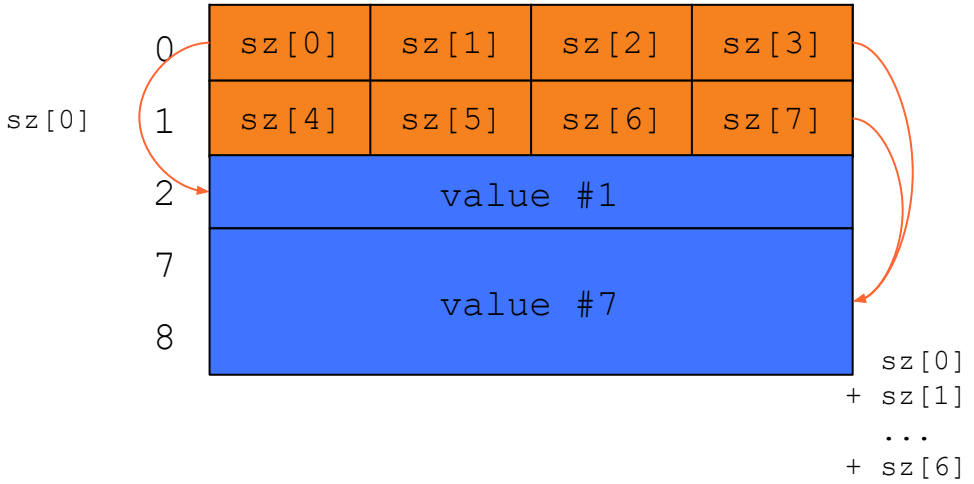
Traits: Store Offsets

- Store absolute offset of values
  - ~200 bytes: `u8` offset
- Waste one byte per KV
- *O(1)* `get()`
- *O(1)* `set()`



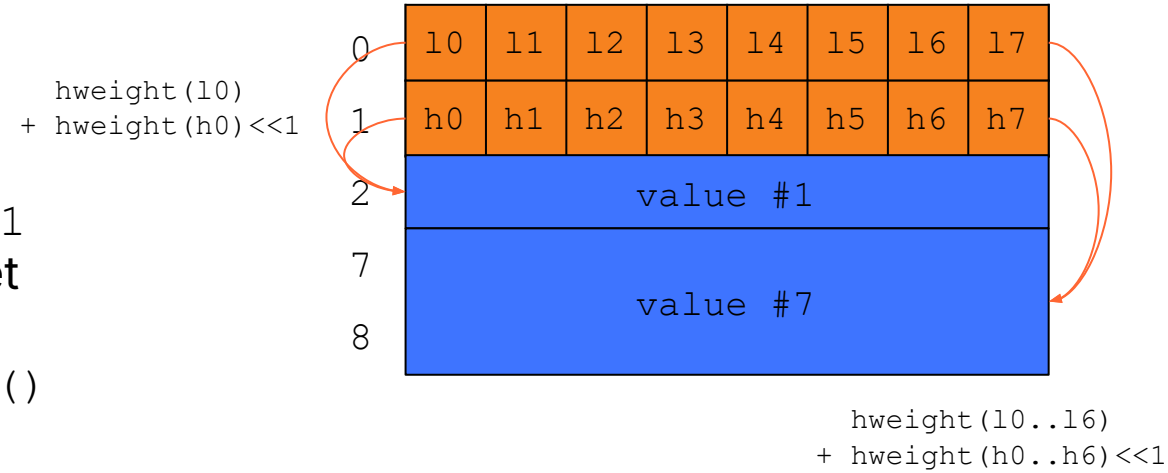
Traits: Relative Offsets

- Store size
  - 3 sizes? 2 bits!
- Waste 2 bits per KV
- *O(n)* `get()`
- *O(n)* `set()`
  - Move higher KVs over



Traits: Hamming Weight

- Split size into:
  - low bits word `l`
  - high bits word `h`
- `sz[0] = l[0] + h[0]<<1`
- Hweight: number of bits set
  - 1 instruction! (popcnt)
- *O(1/word_size) ≈ O(1)* `get()`
- *O(n)* `set()`



Traits

- 64 keys
- 3 value sizes

```
int bpf_xdp_trait_set(struct xdp_buff *xdp, u64 key,
                      void *val, u64 val__sz, u64 flags);

int bpf_xdp_trait_get(struct xdp_buff *xdp, u64 key,
                      void *val, u64 val__sz);
```

Benchmarks

- ~1 indirect call
- Jesper's writeup

	Intel E5-1650 v4	AMD 9684X SRSO	AMD 9684X SRSO=IBPB
xdp-trait-get (ns/op)	5.530	3.901	9.188
xdp-trait-set (ns/op)	7.538	4.941	10.050
xdp-trait-move (ns/op)	14.245	8.865	14.834
function call (ns/op)	1.319	1.359	5.703
indirect call (ns/op)	8.922	6.251	10.329

Status: RFC

[PATCH RFC bpf-next 00/20] traits: Per packet metadata KV store

Open Question: Value Sizes

- 3 value sizes only...

- 4 bytes (IPv4 addr)
- 8 bytes
- 16 bytes (IPv6 addr, UUID)

Open Question: Value Sizes

- Smaller values
  - 0 / flags
  - 4
  - 8
- Batch API
  - Set consecutive keys

```
int bpf_xdp_trait_set(u64 key_from, u64 key_to,
                      void *val, u64 val__sz, u64 flags);

int bpf_xdp_trait_get(u64 key_from, u64 key_to,
                      void *val, u64 val__sz);
```

Open Question: SKB Clones

- Shallow copy
  - Same headroom, same traits
- Mark is deep copied

- Does it matter?
- Could copy traits when cloning
  - Store them somewhere else, `skb->ext`?

Open Question: SKB Clones

- Ok:
  - TCP retransmits
  - Packet capture
- Maybe ok?
  - REUSEPORT and multicast / broadcast
- Others?

Open Question: Registration API

- Only 64 keys
- Rosy future:
  - Keys configured in config files
  - Like TCP ports
  - Admin allocates them

```
$ cat load-balancer.toml
service_trait = 4;
```

- Gotcha: no `EADDRINUSE`

Open Question: Registration API

- Sad future:
  - Services hardcode keys
  - Someone makes a trait registry on github

Open Question: Registration API

- Dynamic key allocation?
  - `register("my_key_name") -> key`
- Only kernel deals with integer keys?

- Lifetime of keys is tricky…
- Unregister?
  - Is key in use?
  - In flight packets?

- Can't retrofit

Open Question: GRO

- Set policy for merging values?
  - `KEEP_FIRST`
  - `KEEP_LAST`
- API for userspace to signal behavior
  - Piggyback on registration API?

- Is `KEEP_FIRST` as default good enough for now?

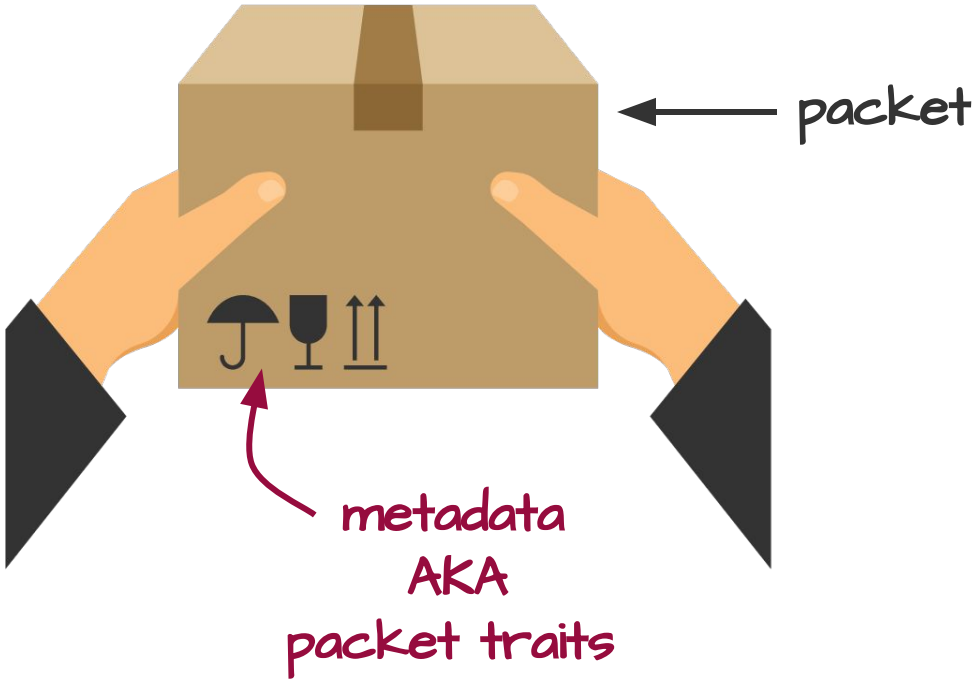
Open Question: Name

- Metadata is overloaded

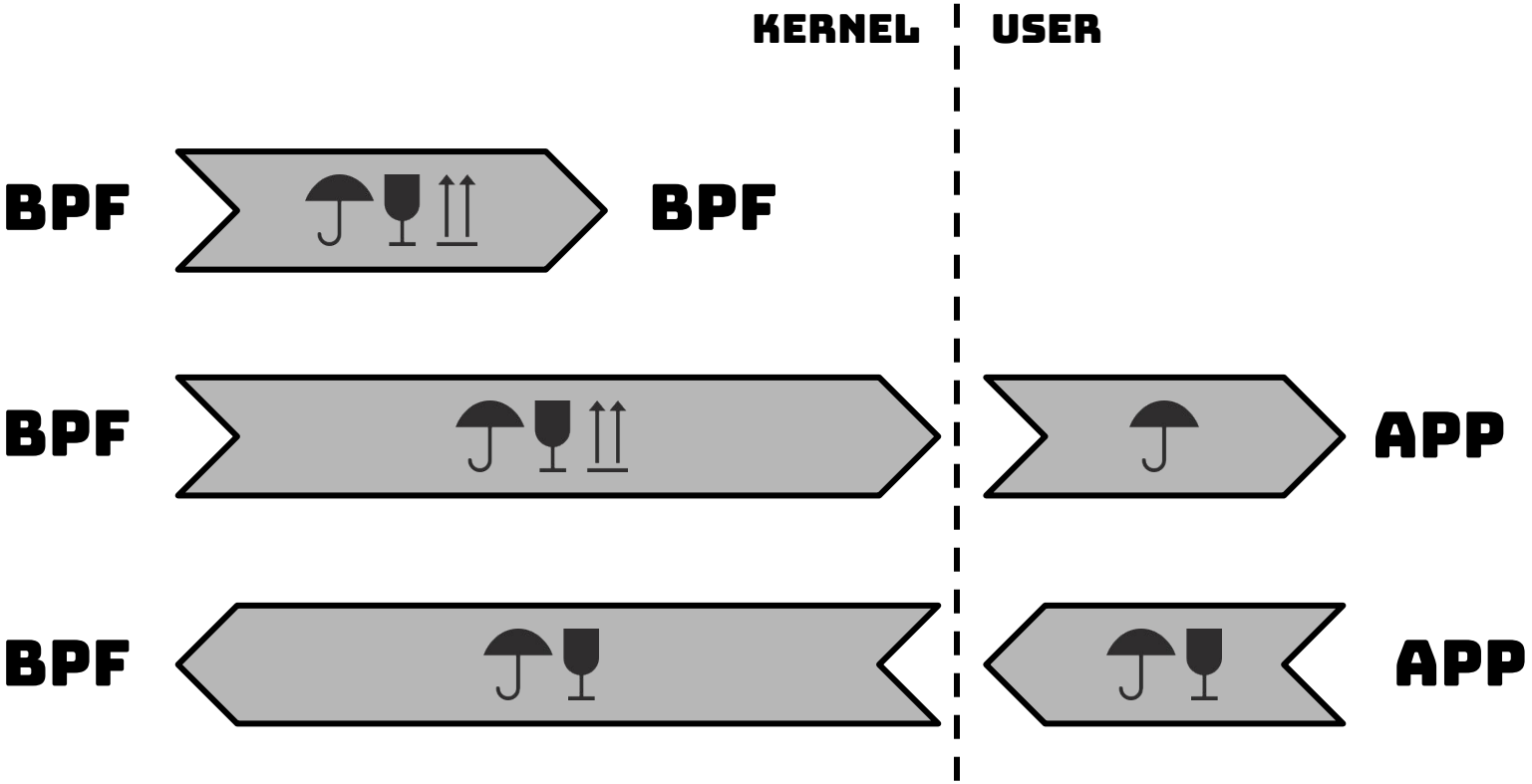
- Traits?
- Hints?
- Stamps?
- Tidbits?
- Labels?

https://forms.gle/VcMuz8hfhgRVkfa49

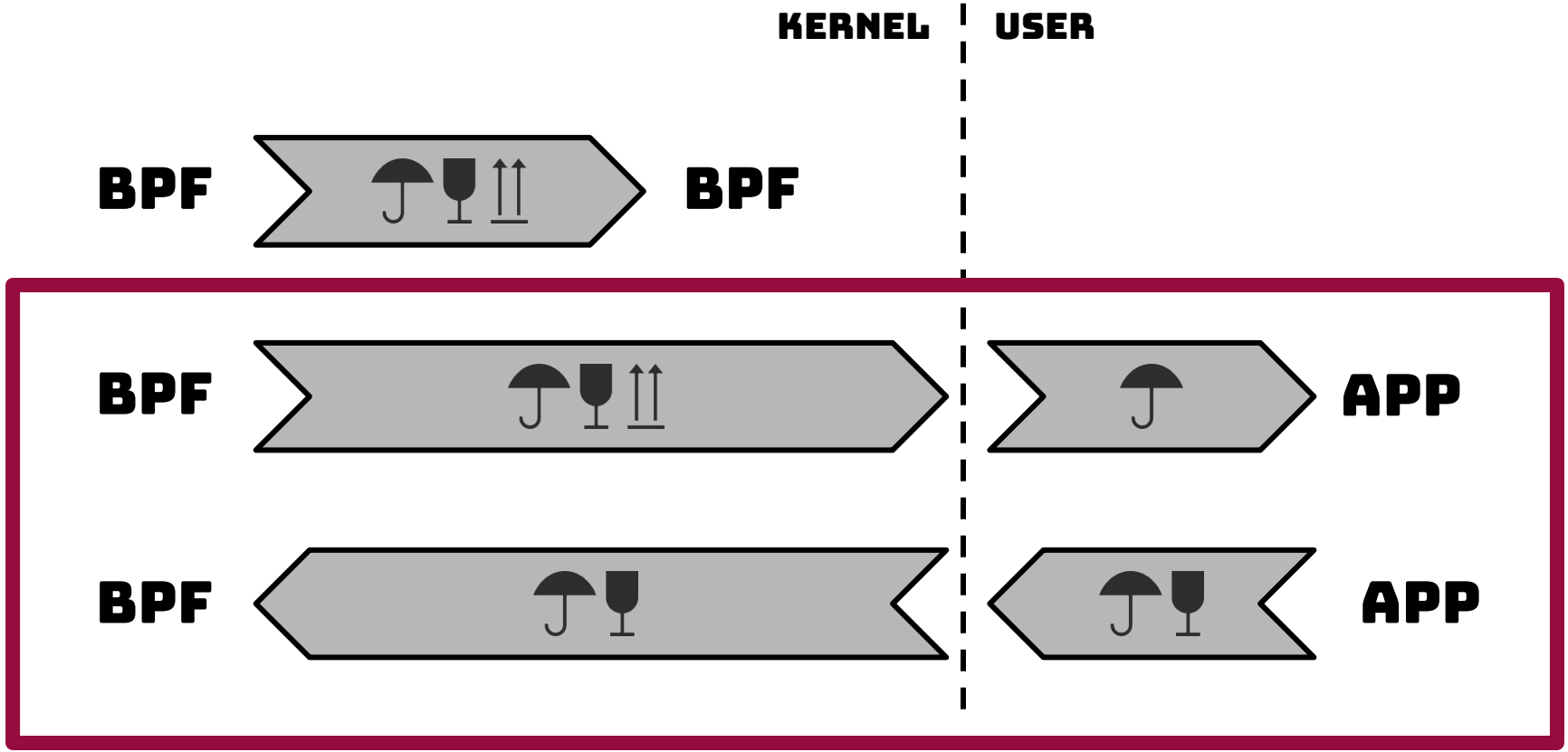
Passing packet metadata to / from application



Use cases



Use cases

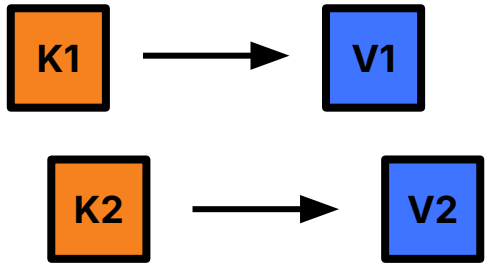


Design considerations

CLOUDFLARE

We have two kinds of users

app wants to get/set *one* or *two* keys
AND
can interpret their values



app is a metadata consumer / producer

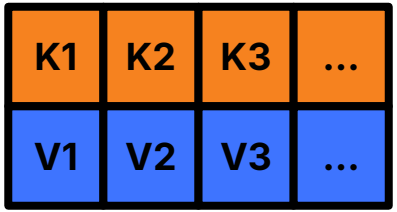
★ bespoke encap with metadata (XDP decap → userland app)
★ set customer ID (userland app → BPF drop monitor)

Design considerations

CLOUDFLARE

We have two kinds of user

app wants to get/set *all* keys
AND
treats them as opaque values



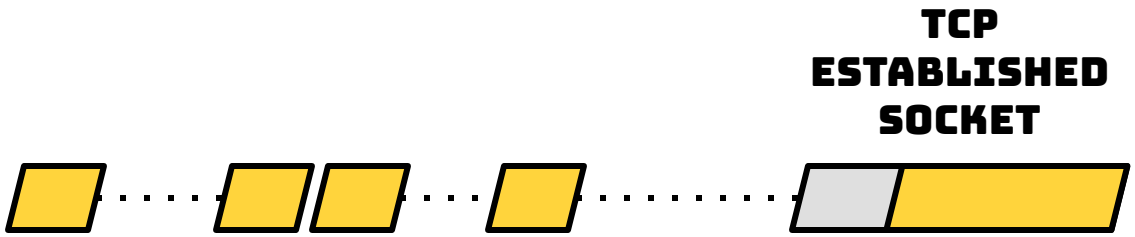
app is a metadata (transparent) proxy

★ intermediate L7 proxies which forward requests
★ “sniffers” - intercept & reinject apps (AF_XDP proxies, Open vSwitch)

Design considerations

CLOUDFLARE

We have two types of sockets



EXPECTED USE:

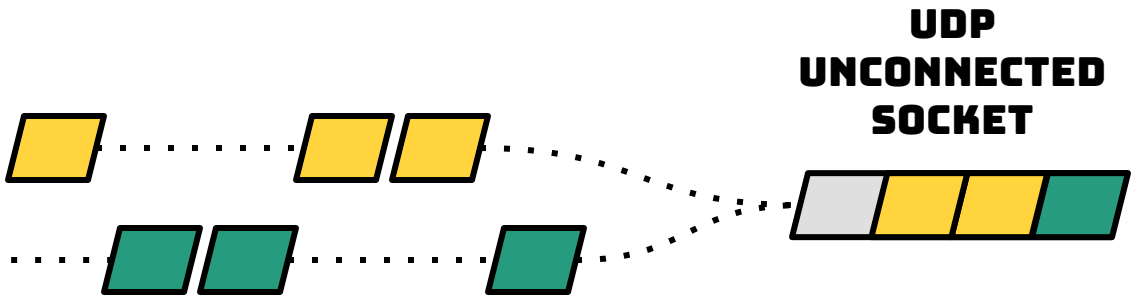
metadata get / set once per socket lifetime

★ initial packet (SYN) not read / written by app
★ message boundaries not preserved (queue coalescing)

Design considerations

CLOUDFLARE

We have two types of sockets



EXPECTED USE:

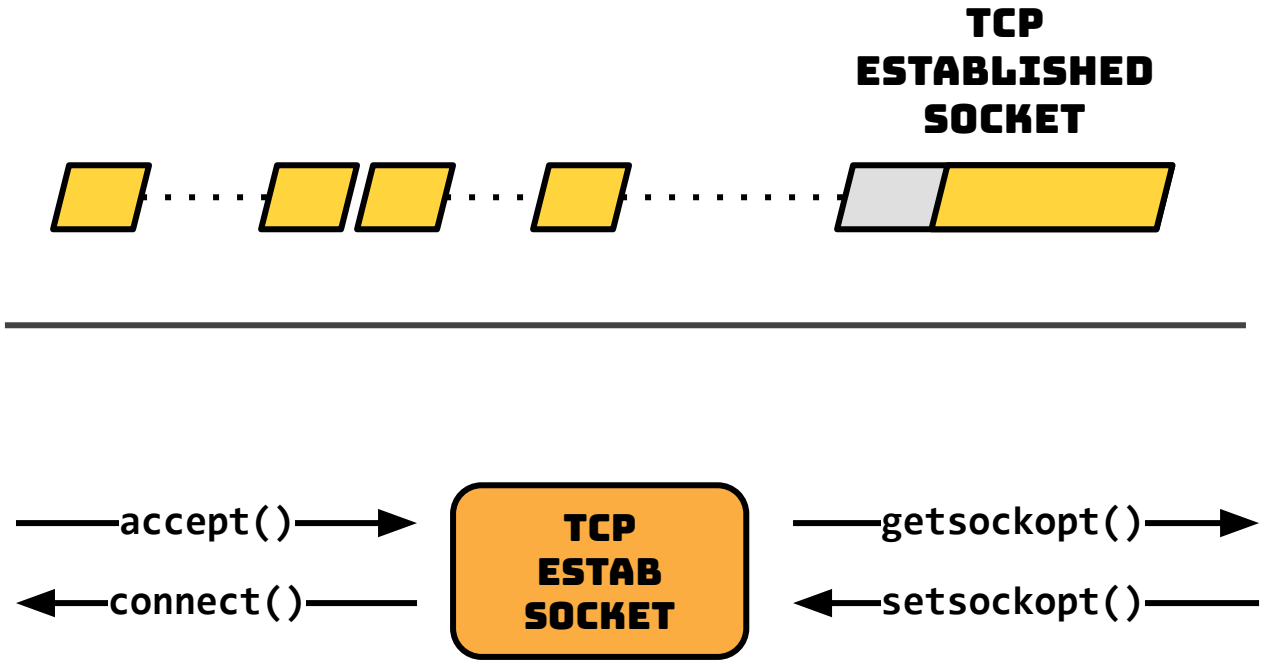
metadata get / set many times per socket lifetime

★ multiplexing – one socket, many QUIC connections
★ message boundaries preserved
(receive one packet, read one message)

CLOUDFLARE

Linux socket API extension proposal for packet metadata access

Metadata access » Stream sockets



Metadata access » Stream sockets

CLOUDFLARE

Follow an existing API pattern – TCP_SAVE_SYN

```
// Save TCP SYN packet within connection socket
setsockopt(listener_fd, SOL_TCP, TCP_SAVE_SYN,
           &(int){ 1 /* enable */ }, …);

conn_fd = accept(listener_fd, …);

// Retrieve the saved SYN packet content
getsockopt(conn_fd, SOL_TCP, TCP_SAVED_SYN,
           &buf, sizeof(buf));
```

Metadata access » Stream sockets

CLOUDFLARE

Incoming connection – reading metadata

```
// Save metadata from TCP SYN packet
setsockopt(listener_fd, SOL_TCP, TCP_SAVE_SYN_TRAITS,
           &(int){ 1 /* enable */ }, …);

conn_fd = accept(listener_fd, …);

// Retrieve metadata from TCP SYN
getsockopt(conn_fd, SOL_TCP, TCP_SYN_TRAITS,
           &buf, sizeof(buf));
```

Metadata access » Stream sockets

CLOUDFLARE

Incoming connection – reading metadata

```
// Save metadata from TCP SYN packet
setsockopt(listener_fd, SOL_TCP, TCP_SAVE_SYN_TRAITS,
           &(int){ 1 /* enable */ }, …);

conn_fd = accept(listener_fd, …);

// Retrieve metadata from TCP SYN
getsockopt(conn_fd, SOL_TCP, TCP_SYN_
           &buf, sizeof(buf));
```

Metadata access » Stream sockets

CLOUDFLARE

Outgoing connection – writing metadata

```
conn_fd = socket(AF_INET, SOCK_STREAM, 0);

// Set metadata for TCP SYN packet
setsockopt(conn_fd, SOL_TCP, TCP_SYN_TRAITS,
           &buf, sizeof(buf));
// … and then initiate connection
connect(conn_fd, …);
```

What should be the format of buf?

CLOUDFLARE

```
getsockopt(conn_fd, SOL_TCP, TCP_SYN_TRAITS, &buf, sizeof(buf));
```

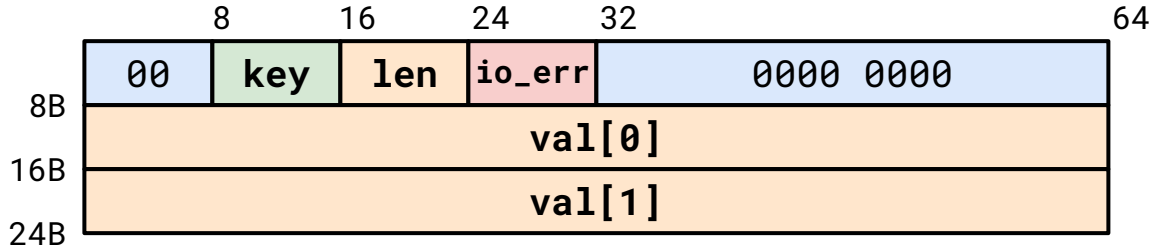

```
setsockopt(conn_fd, SOL_TCP, TCP_SYN_TRAITS, &buf, sizeof(buf));
```

buf format » Guidelines

CLOUDFLARE

☐ fit for use cases	➡	access to one or many keys
☐ hard to misuse	➡	no holes, natural field alignment
☐ future-proof	➡	has wiggle room for extensions
☐ non C-specific	➡	no bitfields, anon members, trailing arrays, …

buf format » Layout



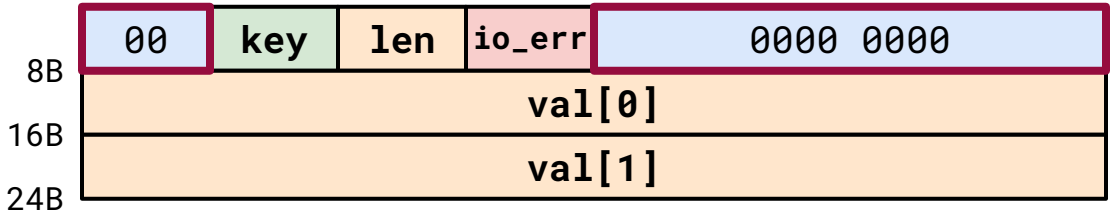
unused

input on get / set

output on get, input on set

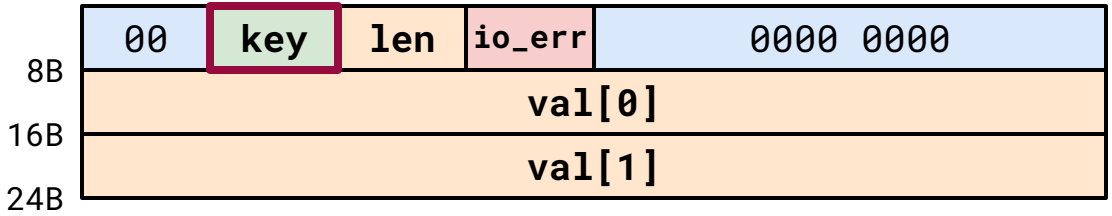
output on get / set

buf format » Padding



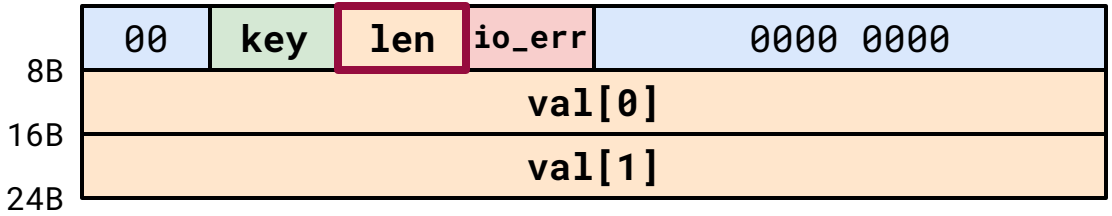
* padding
* must be zero
* future use

buf format » key field



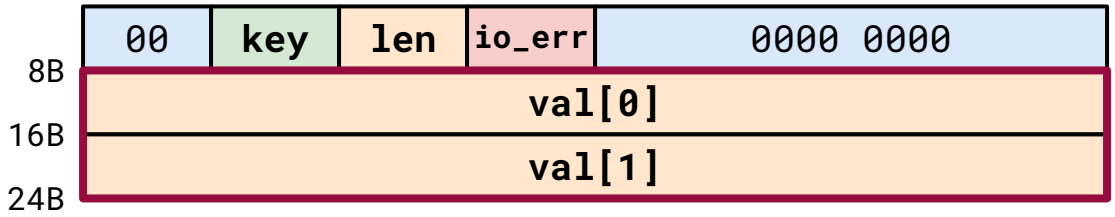
* trait identifier
* 0..63 range today
* future - up to 256 keys
* future - ~0U means "any key" on get?

buf format » len field



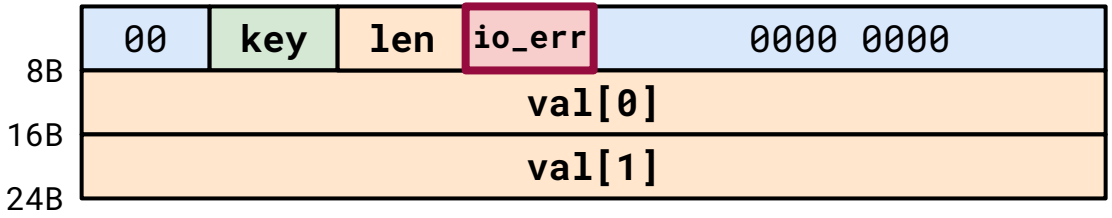
* value length in bytes
* 2, 4, 8 in Arthur's RFC
* likely to change to 4, 8, 16
* 0 if trait absent
* future - other sizes, up to 255 bytes
* future - ~0U means u1 (one bit) value?

buf format » val field



* trait value
* two 64-bit words
* 16 bytes total
* no assumptions on word byte order - producer decides
* unused bytes set to zero

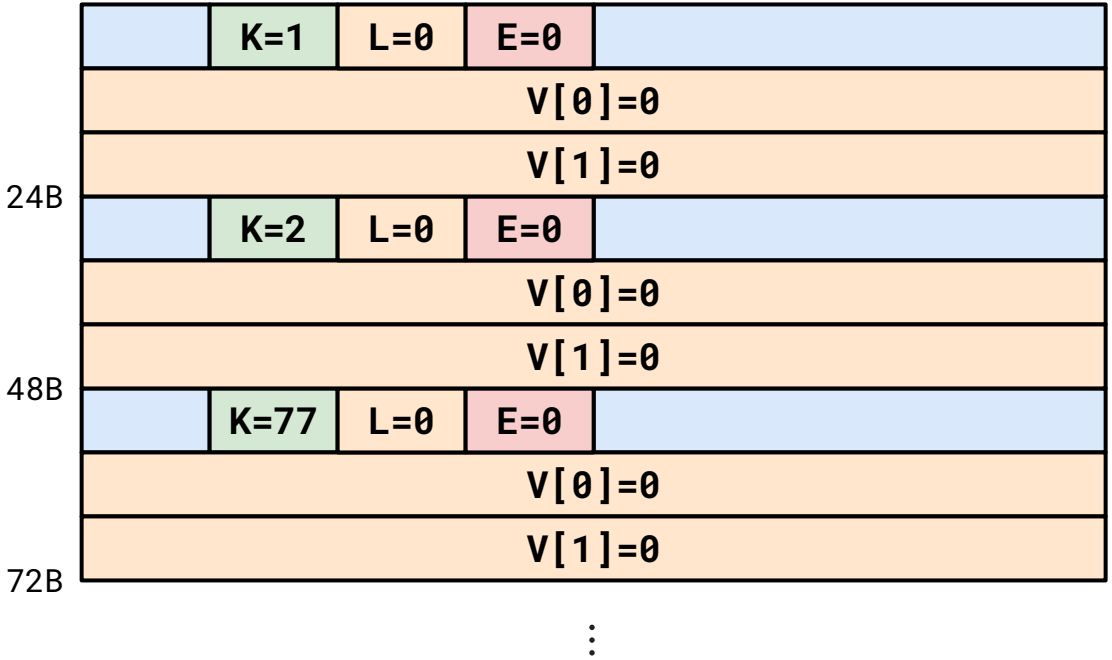
`buf` format » `io_err` field



* errno from read / write for this key
* allows batch ops - avoids error hiding
* user-space must set to zero on syscall
* kernel sets to errno on failure

buf format » batch get » input

pass an array of structures to `getsockopt(TCP_SYN_TRAITS)`

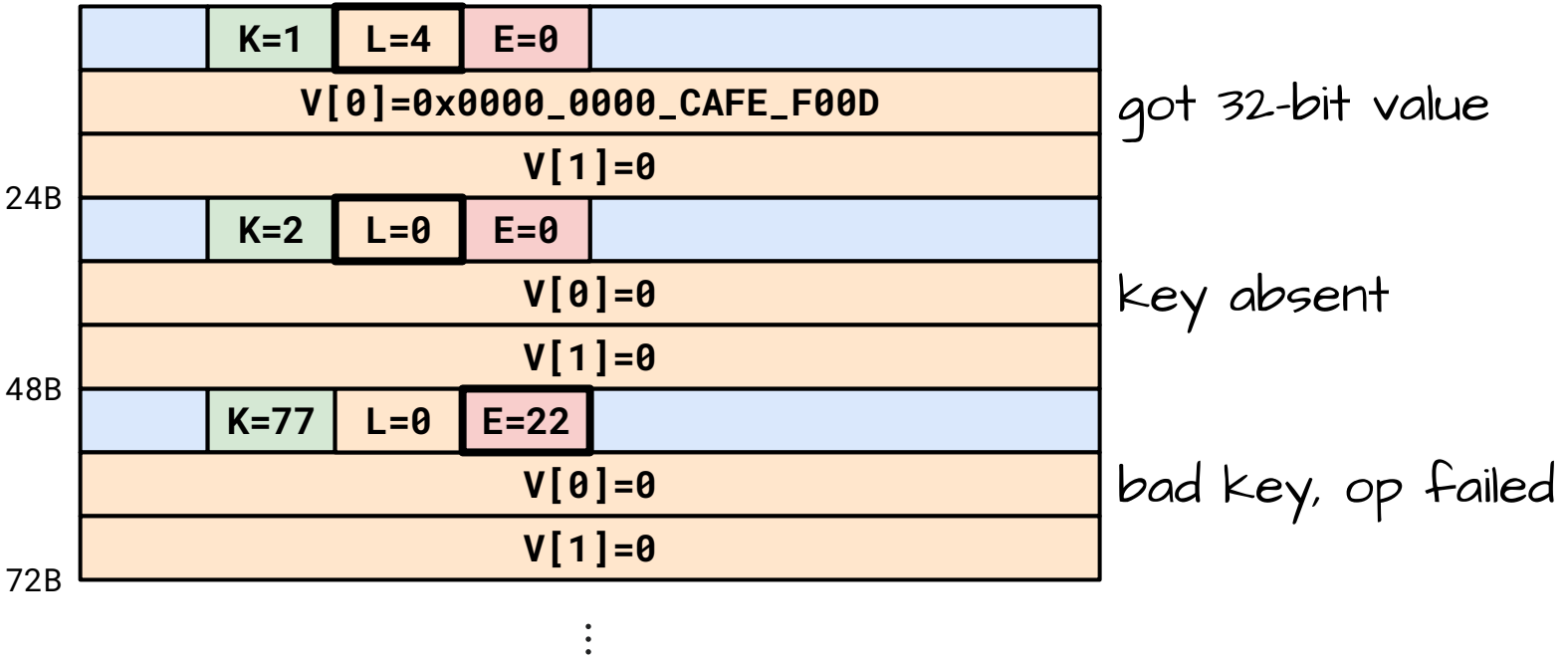


buf format » batch get » output

CLOUDFLARE

`getsockopt(TCP_SYN_TRAITS)` =>

* `0` - success
* `EIO` - partial success, some errors



A word about errors…

```
getsockopt(…, SOL_TCP, TCP_SYN_TRAITS, &traits, …) => EINVAL
```

An error != EIO from syscall => **general problem**

Example: Bad size of the container buffer

vs

```
getsockopt(…, SOL_TCP, TCP_SYN_TRAITS, &traits, …) => EIO
traits[i].io_err == EINVAL
```

EIO from syscall and io_err set => **element-specific problem**

Example: Key out of range.

A word about errors…

```
getsockopt(…, SOL_TCP, TCP_SYN_TRAITS, &traits, …) => EINVAL
```

An error

Example:

> **(!) IMPORTANT**
>
> Get on absent key is **not an error**.
> But length is set to zero.
>
> Buffer can be passed as is from `getsockopt` to `setsockopt`.

```
getsockop
traits[i]
```

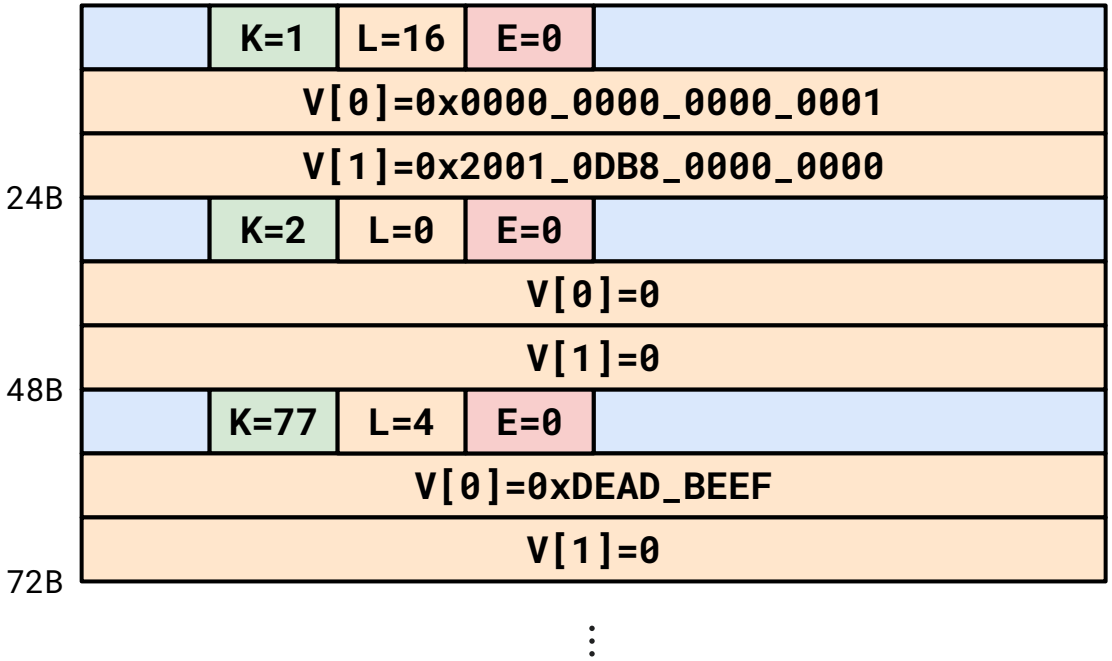
EIO

`EIO` from syscall and `io_err` set => **element-specific problem**

Example: Key out of range.

buf format » batch set » input

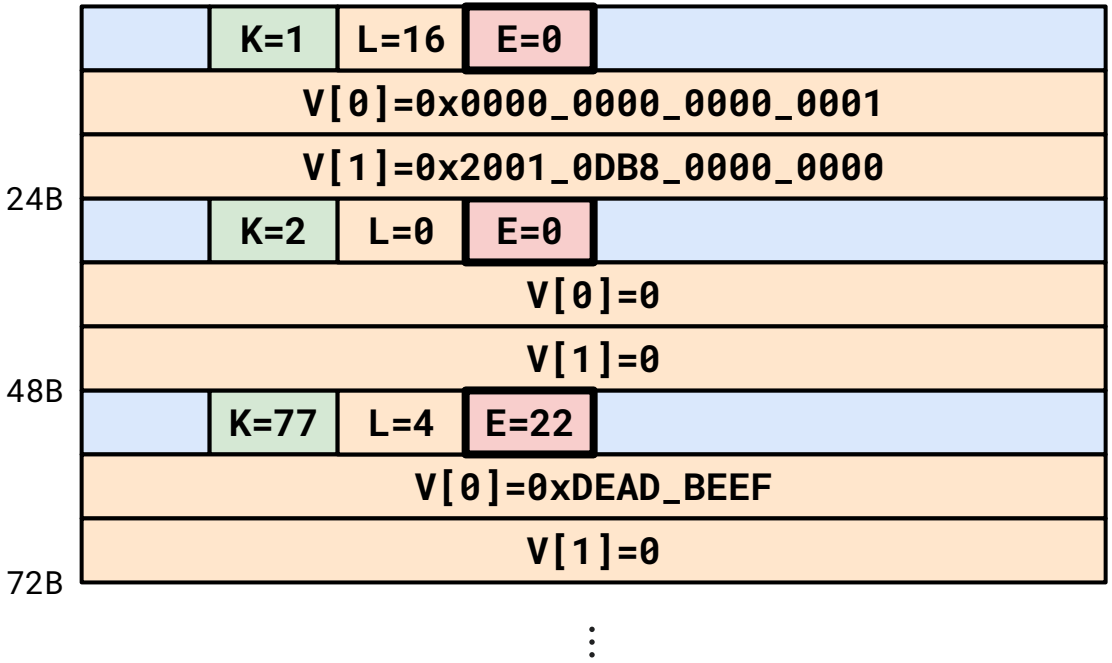
pass an array of structures to `setsockopt(TCP_SYN_TRAITS)`



buf format » batch set » output

`setsockopt(TCP_SYN_TRAITS)` =>

* `0` - success
* `EIO` - partial success, some errors



set 16B value

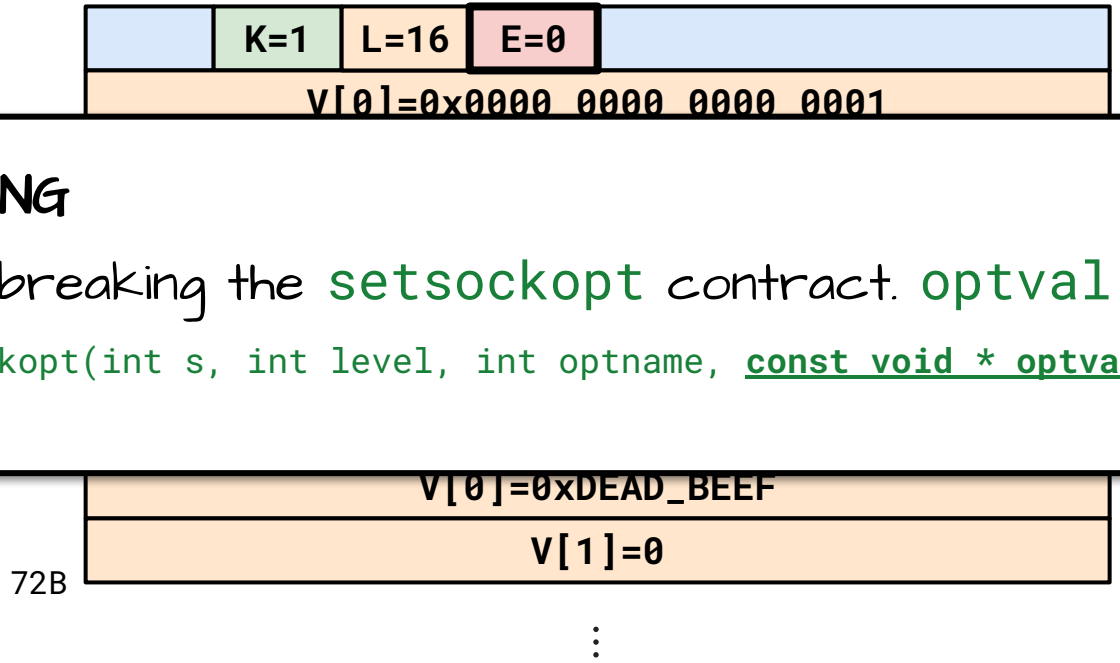
len 0, ignored

bad key, op fail

buf format » batch set » output

`setsockopt(TCP_SYN_TRAITS)` =>

* `0` - success
* `EIO` - partial success, some errors



bad key, op fail

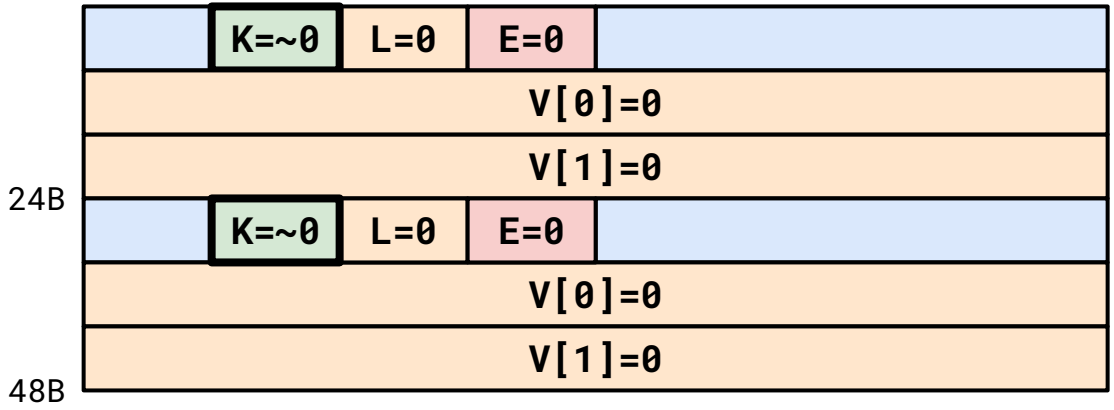
WARNING

We are breaking the `setsockopt` contract. `optval` is `const`.

```
int setsockopt(int s, int level, int optname, const void * optval, int optlen);
```

buf format » batch dump [future]

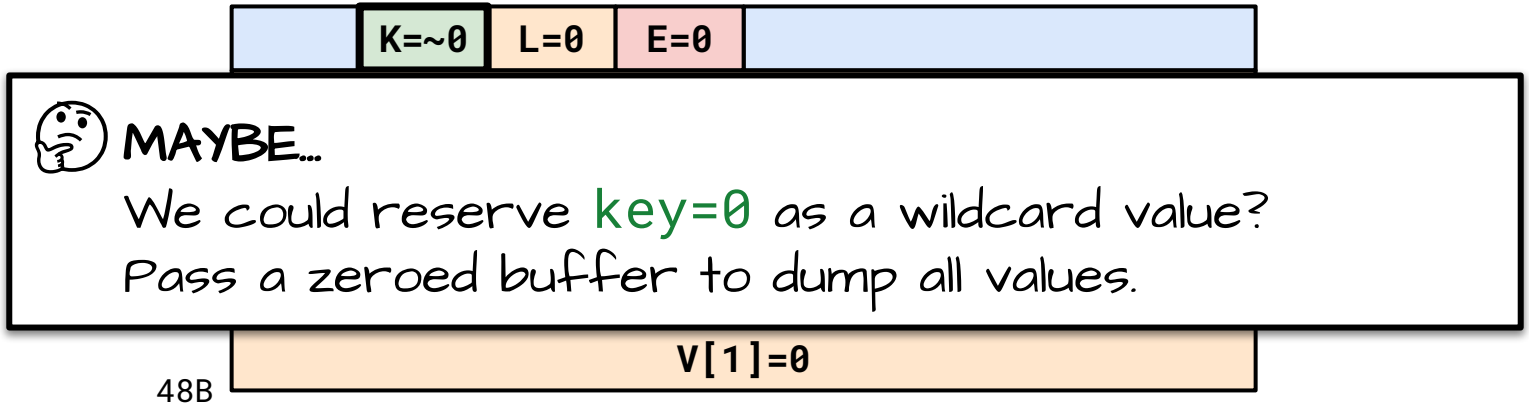
If app is a transparent proxy which doesn't know/care about key IDs, pass "wildcard" value as key ID to `getsockopt(TCP_SYN_TRAITS)`



Kernel fills in `key`, `len`, `val`

buf format » batch dump [future]

If app is a transparent proxy which doesn't know/care about key IDs, pass "wildcard" value as key ID to `getsockopt(TCP_SYN_TRAITS)`



Kernel fills in key, len, val

buf format » C definition

CLOUDFLARE

```
struct pkt_trait {
    __u8 _zpad_1;  /* padding; must be zero; future use */
    __u8 key;      /* trait identifier in 0..63 range */
    __u8 len;      /* value length in bytes; zero if trait absent */
    __u8 io_err;   /* errno from read/write for this key; zero on success */
    __u32 _zpad_2; /* padding; must be zero; future use */
    __u64 val[2];  /* trait value; unused bytes must be zero */
};
```

- ☑️ fit for use cases
- ☑️ hard to misuse
- ☑️ future-proof
- ☑️ non C-specific

TIMTOWTDI - Alternative API ideas

CLOUDFLARE

```
/* A dedicated socket option for each trait (key) */

uint64_t val64;
getsockopt(…, SOL_TCP, TCP_SYN_TRAIT_0, &val64, …);

uint32_t val32 = 42;
setsockopt(…, SOL_TCP, TCP_SYN_TRAIT_63, &val32, …);
```

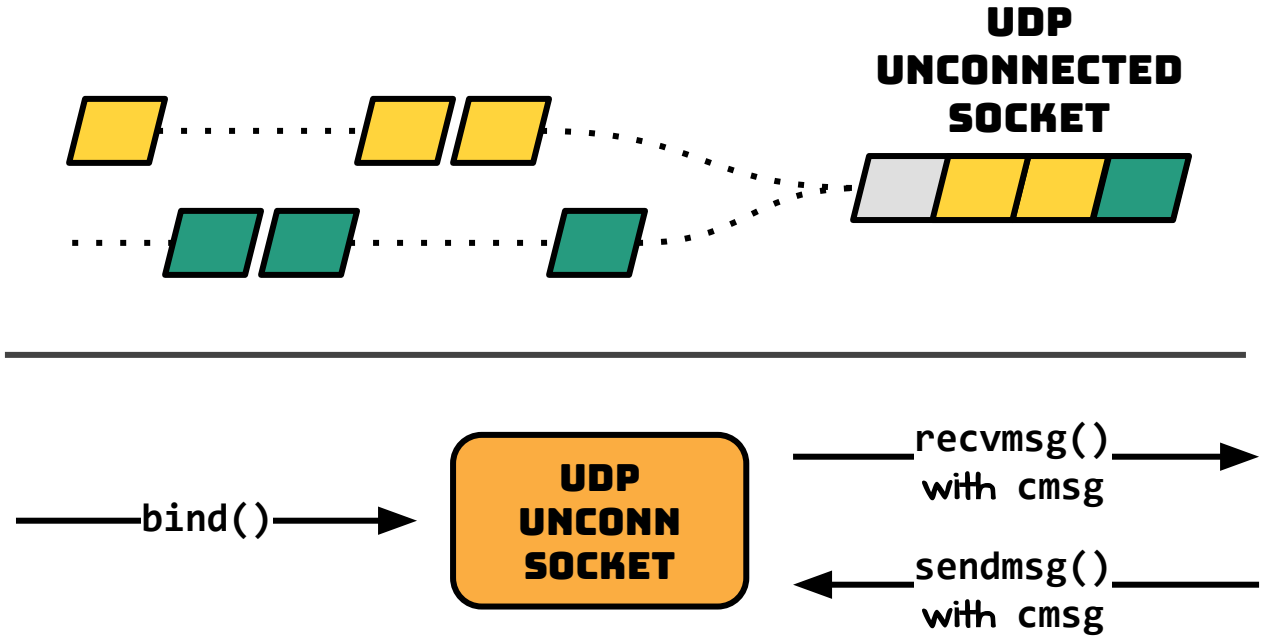
👍 simple & familiar

👍 suitable for any value length

👎 not clear how to handle batch get/set for TCP

👎 cast needed on BE if you don't know length

Metadata access » Datagram sockets



https://man7.org/linux/man-pages/man3/cmsg.3.html

Metadata access » Datagram sockets

CLOUDFLARE

Follow an existing API pattern

New timestamping API – `SO_TIMESTAMPING`

➔ Timestamping — The Linux Kernel documentation

➔ Netdev 0x17: SO_TIMESTAMPING: powering fleetwide RPC monitoring

UDS FD passing API – `SCM_RIGHTS`

➔ https://man7.org/linux/man-pages/man7/unix.7.html

Metadata access » Datagram sockets

CLOUDFLARE

Receiving metadata

```
/* Opt in to receive packet traits */
setsockopt(listener_fd, SOL_SOCKET, SO_PKT_TRAITS, &one, …);

/* Receive packet with ancillary message */
assert( recvmsg(fd, &msg, 0) >= 0 );

/* Retrieve traits */
for (cm = CMSG_FIRSTHDR(msg); cm; cm = CMSG_NXTHDR(msg, cmsg)) {
        if (cm->cmsg_level == SOL_SOCKET &&
            cm->cmsg_type == SCM_PKT_TRAITS) {
                memcpy(traits, CMSG_DATA(cm), sizeof(traits));
                for (i = 0; i < ARRAY_LEN(traits); i++)
                        /* access traits[i]->{key, len, io_err, val} */;
        }
}
```

Metadata access » Datagram sockets

CLOUDFLARE

Sending metadata

```
struct pkt_trait traits[] = {
        { .key = 0, .len = sizeof(__u32), .val = 0xcafe },
        { .key = 2, .len = sizeof(__u64), .val = 0xdeadbeef },
};

struct msghdr *msg;
// …
cmsg             = CMSG_FIRSTHDR(msg);
cmsg->cmsg_level = SOL_SOCKET;
cmsg->cmsg_type  = SCM_PKT_TRAITS;
cmsg->cmsg_len   = CMSG_LEN(sizeof(traits));
memcpy(CMSG_DATA(cmsg), traits, sizeof(traits));

assert( sendmsg(fd, msg, 0) >= 0 );
```

Show me the code!

CLOUDFLARE

Traits

- [PATCH RFC bpf-next 00/20] traits: Per packet metadata KV store
- Benchmark results

uAPI

- Dev branch with socket glue code and selftests
  https://github.com/jsitnicki/linux/commits/dev/skb-traits-uapi/
- Playground repo with tests and bindings for Rust (planned - Golang, Python, Zig)
  https://github.com/jsitnicki/skb-traits-uapi

Prior Art

- Upstream discussion on integrating this with NIC hardware metadata
- Linux Plumbers Conference talk

Thank you!

arthur@arthurfabre.com

jakub@cloudflare.com

jesper@cloudflare.com

Vote on the name
or forever hold your peace
;-)

Questions? Now or over pizza 🍕

CLOUDFLARE

Traits

- Values sizes?
- SKB clones?
- Registration API?
- GRO?

uAPI

- on/off switch or subscribe to keys?
- can `setsockopt` mutate value?
- reserve key `0` as wildcard?
- TLV array or opt per key?

Vote on the name
or forever hold your peace
;-)