BPF Extensible Network

TCP Congestion Control, TCP Header option, sk local storage, and...?

Martin Lau
Software Engineer

8/28/2020

Once upon a time...

In 2015 LPC, Protocol Ossification was brought up

- How easy to test/deploy a new TCP CC idea
  - How easy to make kernel changes? No kernel panic!
  - How quick is the turnaround time (deploy, gather data, and re-iterate)?
    - As quick as a kernel can be upgraded or kernel module can be deployed
    - Some environment has long tail of kernel versions
- The answer to the above is usually discouraging to many network/protocol experts

One idea was,

- Can TCP Congestion Control be written in BPF? 🤔

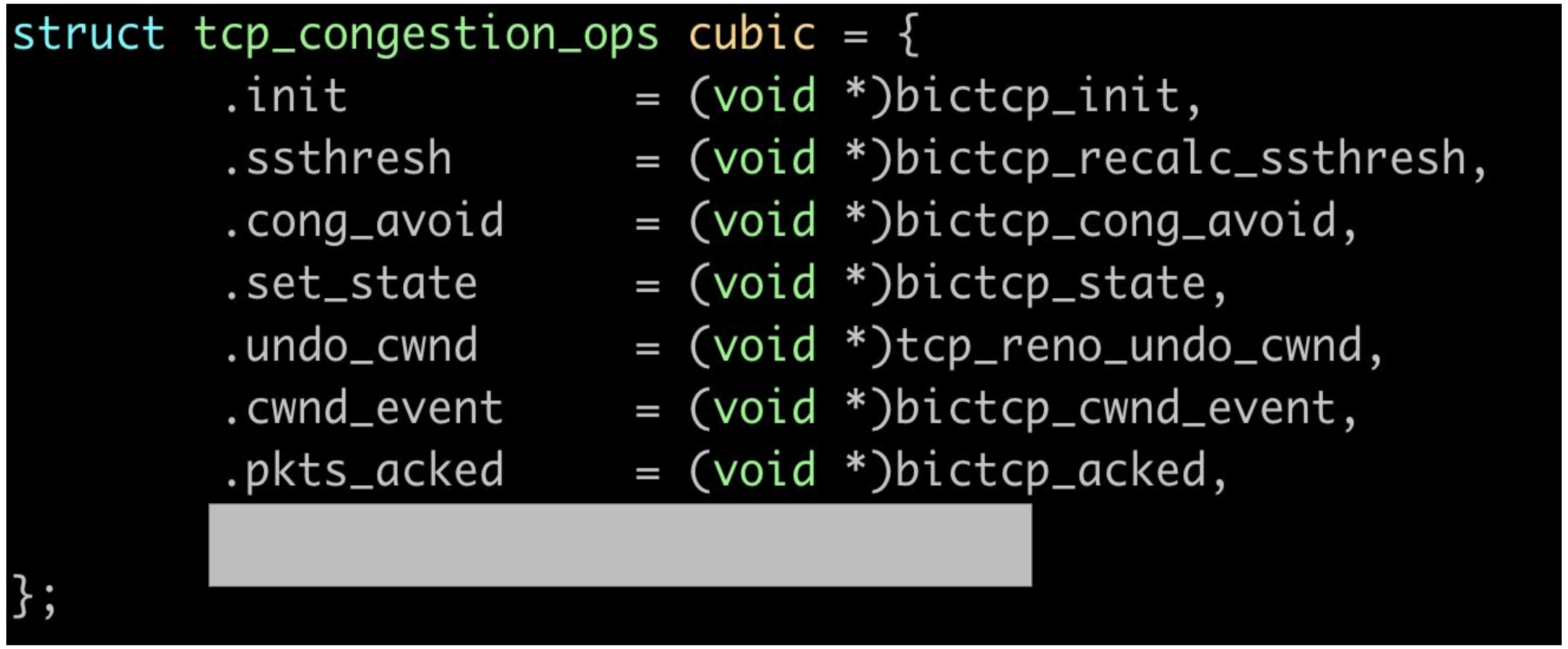
Recent BPF works in networking

- TCP Congestion Control
- TCP Header Option
- SK local storage

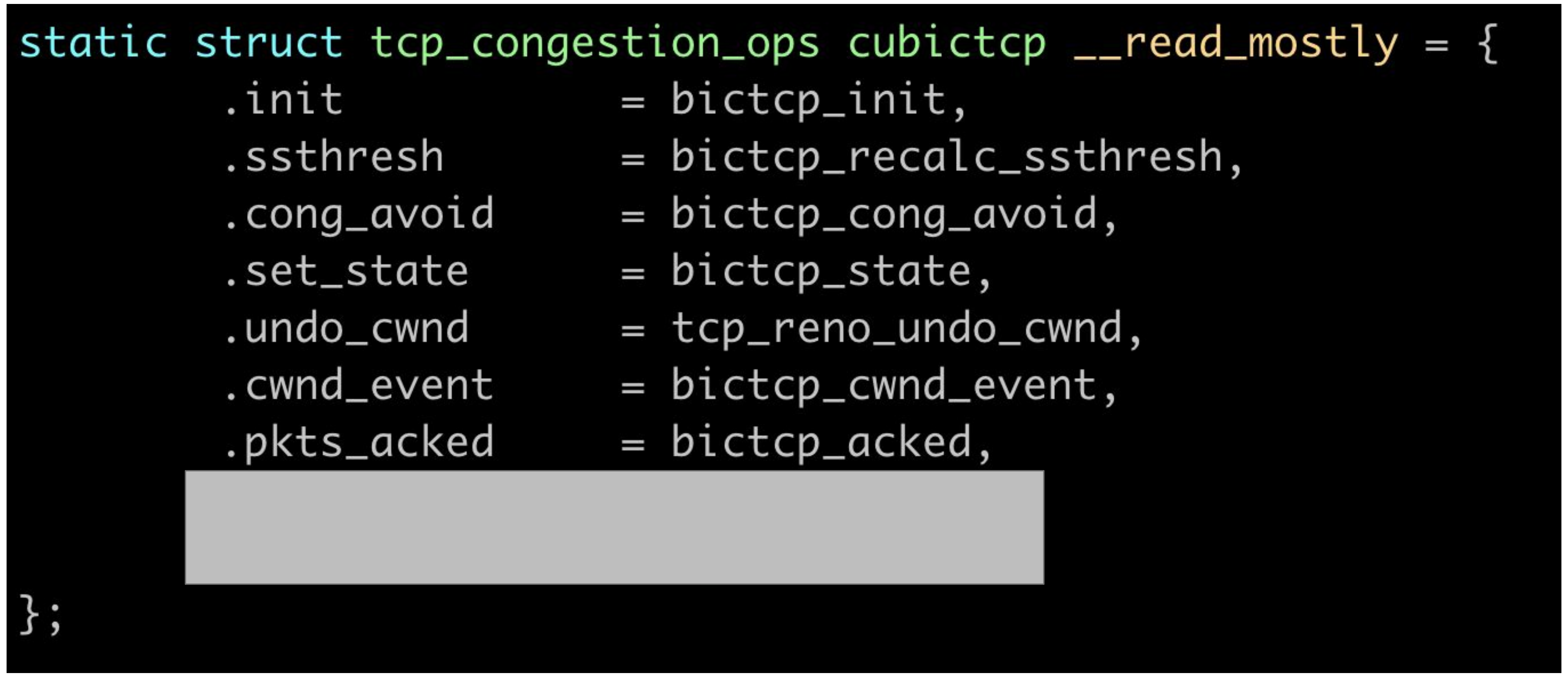
BPF TCP CC

How to write one?

- Which one of them below is a bpf program?



```
struct tcp_congestion_ops cubic = {
        .init           = (void *)bictcp_init,
        .ssthresh       = (void *)bictcp_recalc_ssthresh,
        .cong_avoid     = (void *)bictcp_cong_avoid,
        .set_state      = (void *)bictcp_state,
        .undo_cwnd      = (void *)tcp_reno_undo_cwnd,
        .cwnd_event     = (void *)bictcp_cwnd_event,
        .pkts_acked     = (void *)bictcp_acked,
        [redacted]
};
```

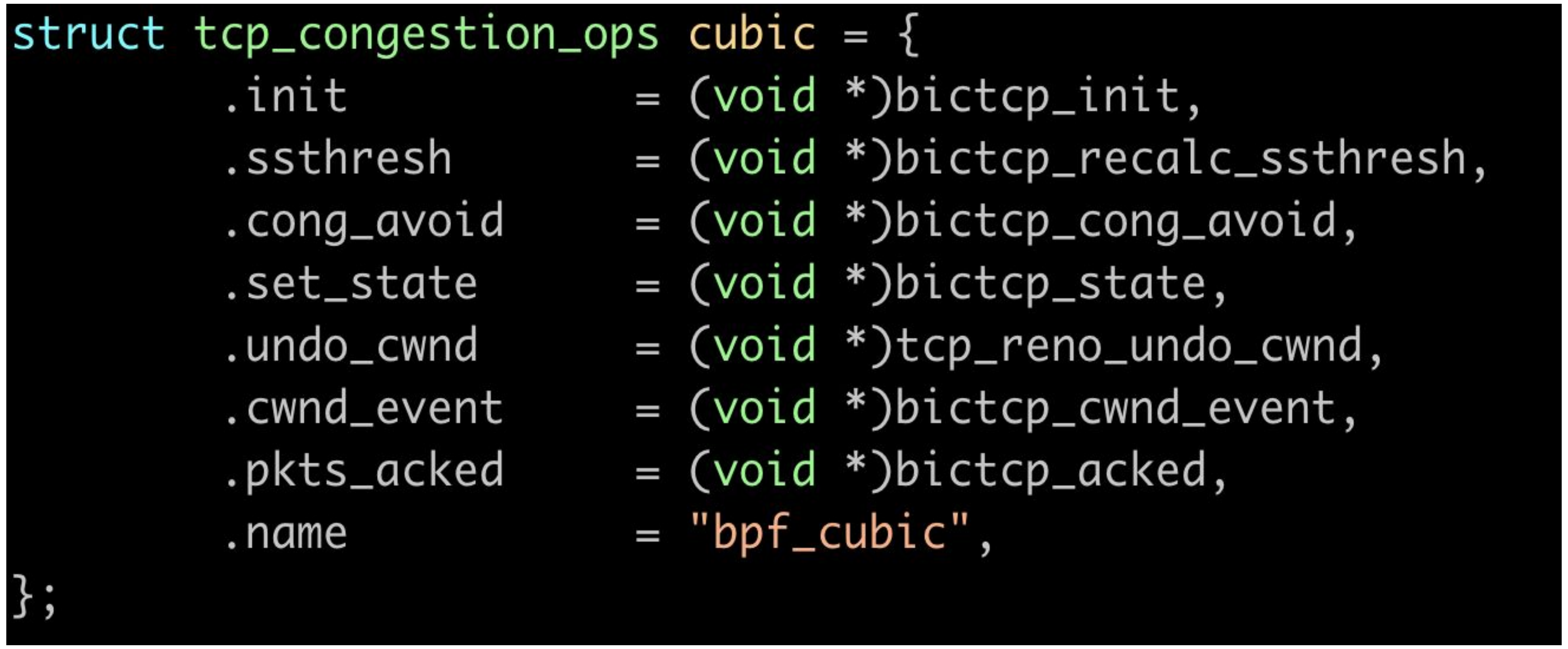


```
static struct tcp_congestion_ops cubictcp __read_mostly = {
        .init           = bictcp_init,
        .ssthresh       = bictcp_recalc_ssthresh,
        .cong_avoid     = bictcp_cong_avoid,
        .set_state      = bictcp_state,
        .undo_cwnd      = tcp_reno_undo_cwnd,
        .cwnd_event     = bictcp_cwnd_event,
        .pkts_acked     = bictcp_acked,
       [redacted]
};
```

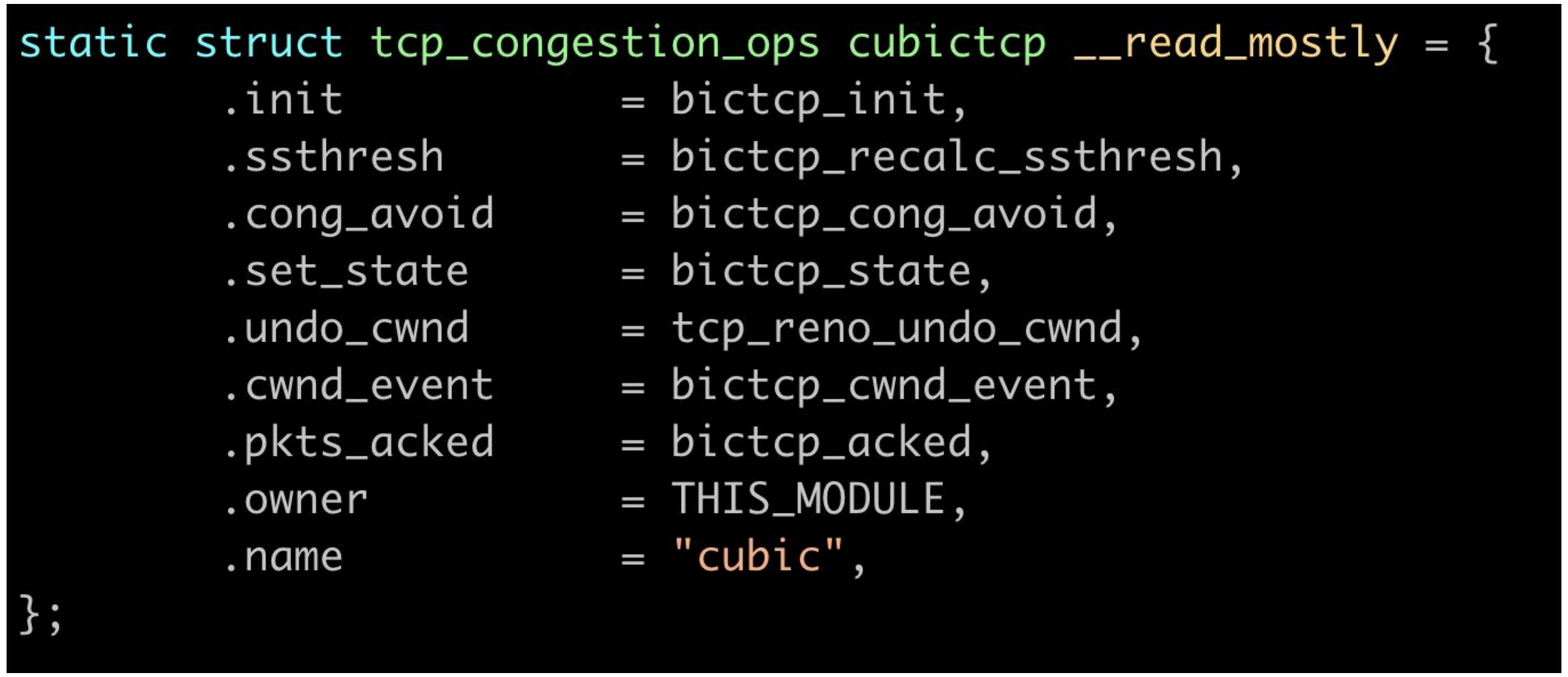
BPF TCP CC

How to write one?

- Which one of them below is a bpf program?



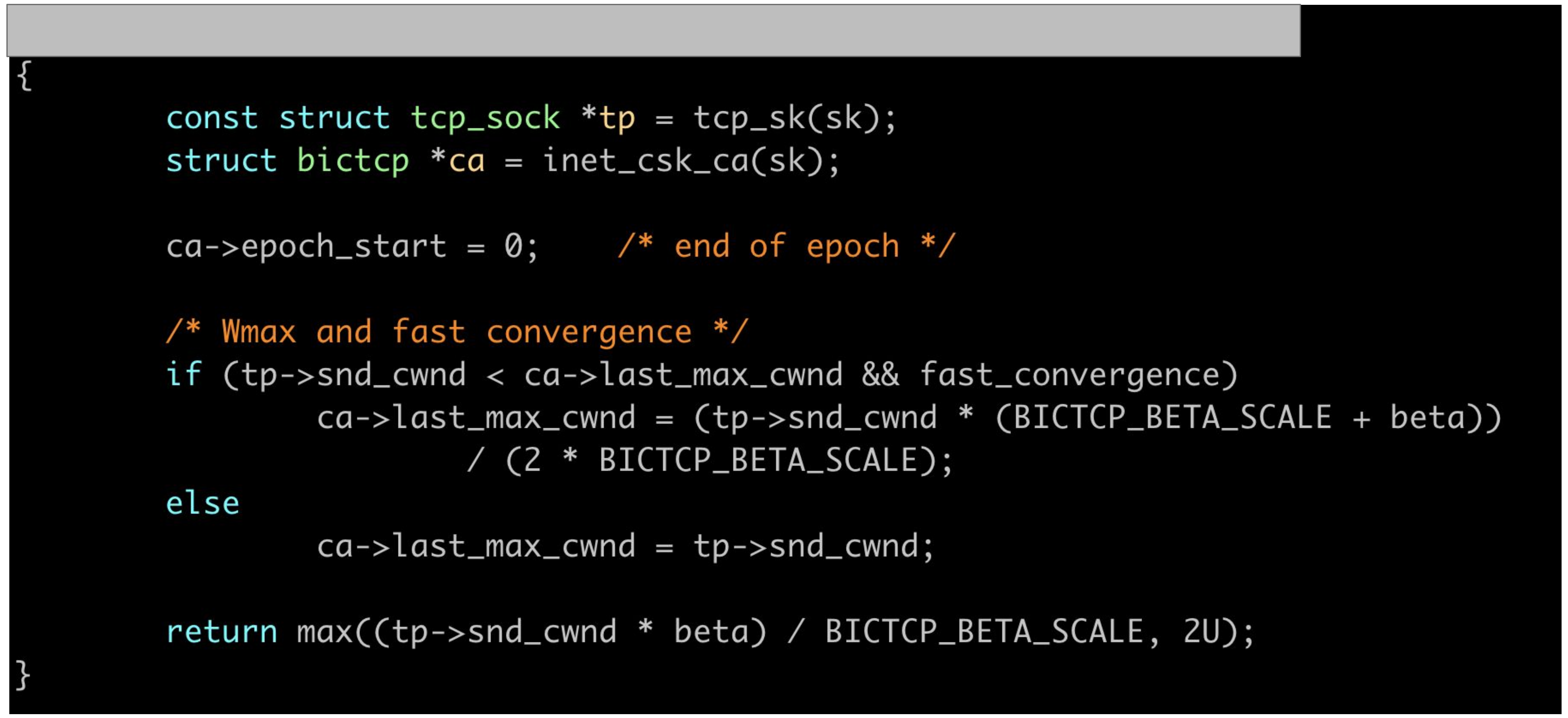
```
struct tcp_congestion_ops cubic = {
        .init           = (void *)bictcp_init,
        .ssthresh       = (void *)bictcp_recalc_ssthresh,
        .cong_avoid     = (void *)bictcp_cong_avoid,
        .set_state      = (void *)bictcp_state,
        .undo_cwnd      = (void *)tcp_reno_undo_cwnd,
        .cwnd_event     = (void *)bictcp_cwnd_event,
        .pkts_acked     = (void *)bictcp_acked,
        .name           = "bpf_cubic",
};
```



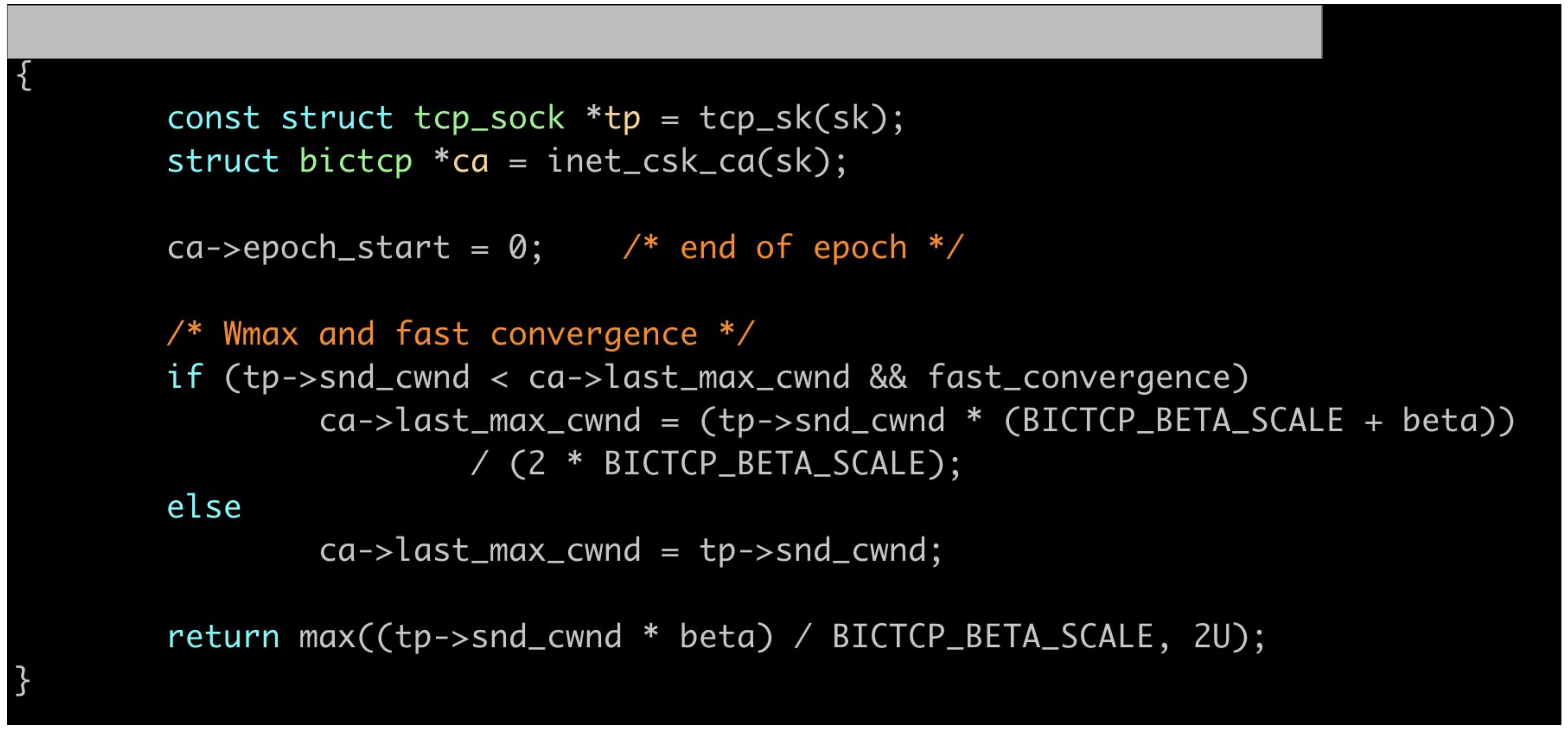
```
static struct tcp_congestion_ops cubictcp __read_mostly = {
        .init           = bictcp_init,
        .ssthresh       = bictcp_recalc_ssthresh,
        .cong_avoid     = bictcp_cong_avoid,
        .set_state      = bictcp_state,
        .undo_cwnd      = tcp_reno_undo_cwnd,
        .cwnd_event     = bictcp_cwnd_event,
        .pkts_acked     = bictcp_acked,
        .owner          = THIS_MODULE,
        .name           = "cubic",
};
```

BPF TCP CC

How to write one? (contd)



```
{
        const struct tcp_sock *tp = tcp_sk(sk);
        struct bictcp *ca = inet_csk_ca(sk);

        ca->epoch_start = 0;    /* end of epoch */

        /* Wmax and fast convergence */
        if (tp->snd_cwnd < ca->last_max_cwnd && fast_convergence)
                ca->last_max_cwnd = (tp->snd_cwnd * (BICTCP_BETA_SCALE + beta))
                        / (2 * BICTCP_BETA_SCALE);
        else
                ca->last_max_cwnd = tp->snd_cwnd;

        return max((tp->snd_cwnd * beta) / BICTCP_BETA_SCALE, 2U);
}
```

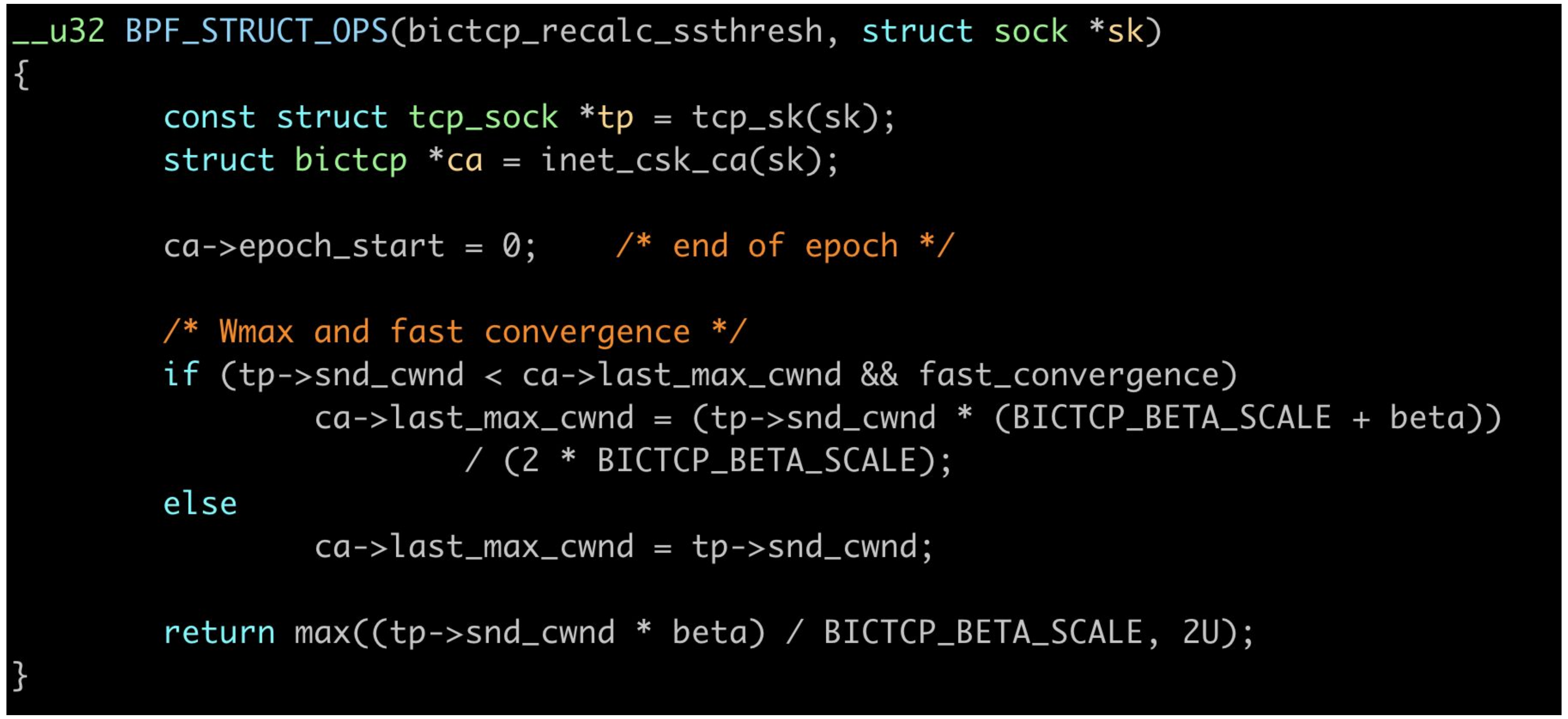


```
{
        const struct tcp_sock *tp = tcp_sk(sk);
        struct bictcp *ca = inet_csk_ca(sk);

        ca->epoch_start = 0;    /* end of epoch */

        /* Wmax and fast convergence */
        if (tp->snd_cwnd < ca->last_max_cwnd && fast_convergence)
                ca->last_max_cwnd = (tp->snd_cwnd * (BICTCP_BETA_SCALE + beta))
                        / (2 * BICTCP_BETA_SCALE);
        else
                ca->last_max_cwnd = tp->snd_cwnd;

        return max((tp->snd_cwnd * beta) / BICTCP_BETA_SCALE, 2U);
}
```

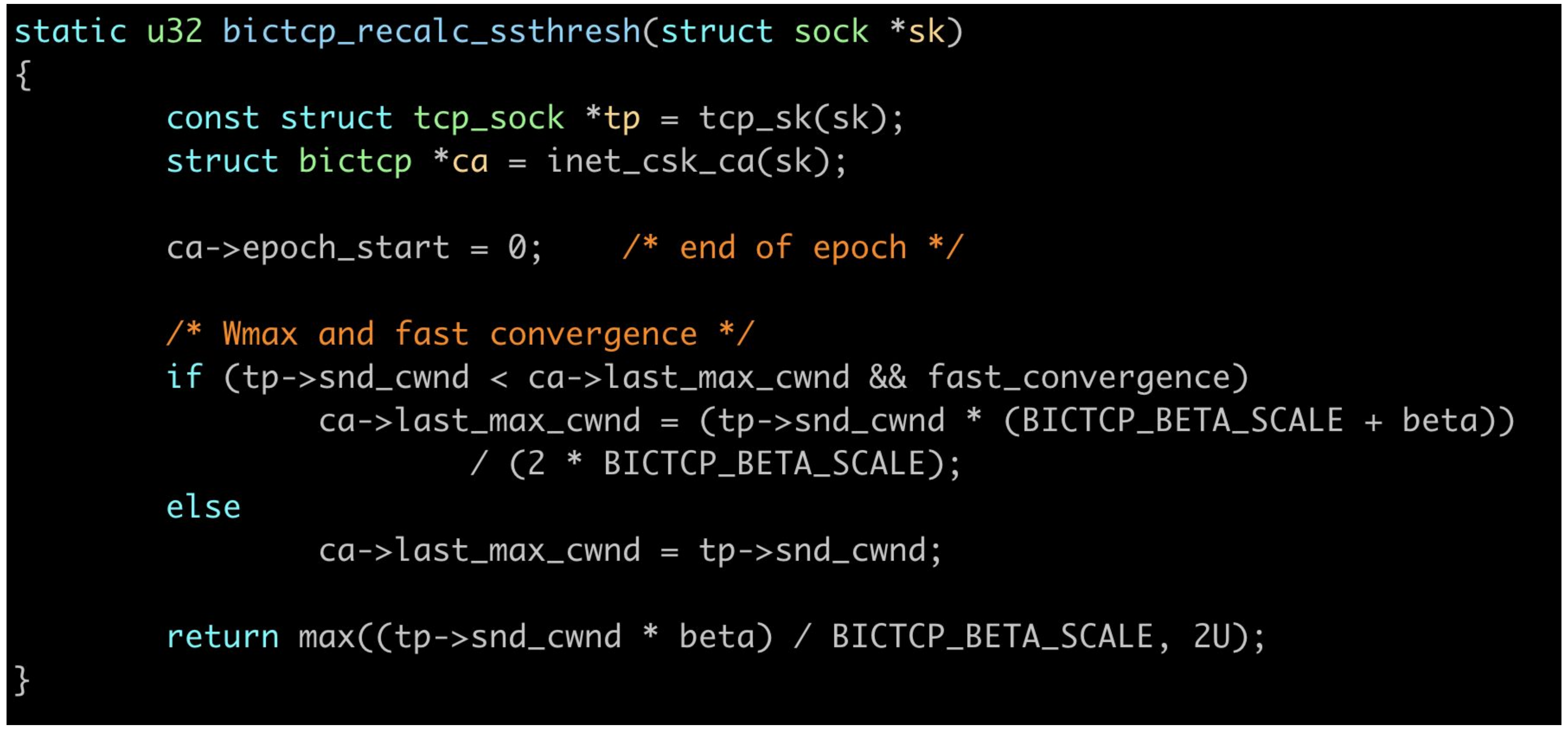
BPF TCP CC

How to write one? (contd)

😁



```c
__u32 BPF_STRUCT_OPS(bictcp_recalc_ssthresh, struct sock *sk)
{
        const struct tcp_sock *tp = tcp_sk(sk);
        struct bictcp *ca = inet_csk_ca(sk);

        ca->epoch_start = 0;    /* end of epoch */

        /* Wmax and fast convergence */
        if (tp->snd_cwnd < ca->last_max_cwnd && fast_convergence)
                ca->last_max_cwnd = (tp->snd_cwnd * (BICTCP_BETA_SCALE + beta))
                        / (2 * BICTCP_BETA_SCALE);
        else
                ca->last_max_cwnd = tp->snd_cwnd;

        return max((tp->snd_cwnd * beta) / BICTCP_BETA_SCALE, 2U);
}
```



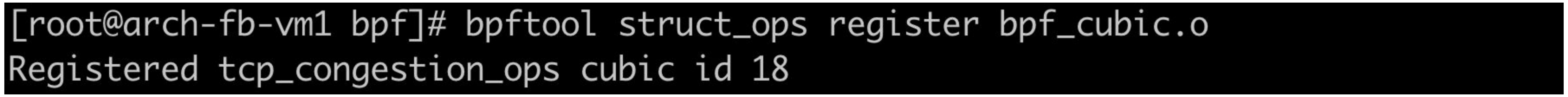
```c
static u32 bictcp_recalc_ssthresh(struct sock *sk)
{
        const struct tcp_sock *tp = tcp_sk(sk);
        struct bictcp *ca = inet_csk_ca(sk);

        ca->epoch_start = 0;    /* end of epoch */

        /* Wmax and fast convergence */
        if (tp->snd_cwnd < ca->last_max_cwnd && fast_convergence)
                ca->last_max_cwnd = (tp->snd_cwnd * (BICTCP_BETA_SCALE + beta))
                        / (2 * BICTCP_BETA_SCALE);
        else
                ca->last_max_cwnd = tp->snd_cwnd;

        return max((tp->snd_cwnd * beta) / BICTCP_BETA_SCALE, 2U);
}
```

🐝🫛f

BPF TCP CC

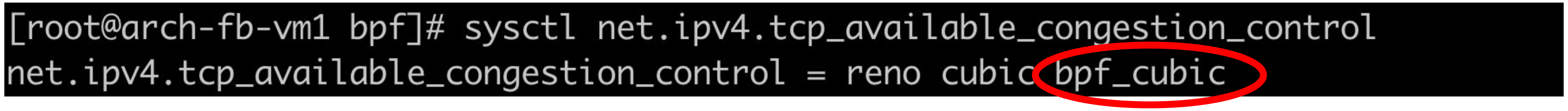
How to use it in production?

- Load the bpf prog



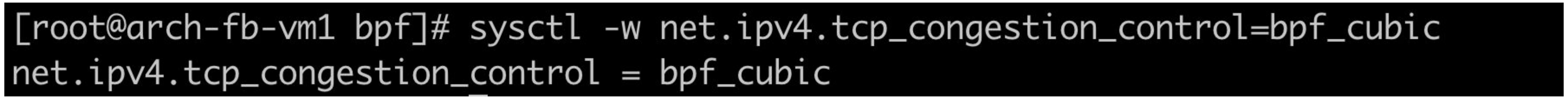
```
[root@arch-fb-vm1 bpf]# bpftool struct_ops register bpf_cubic.o
Registered tcp_congestion_ops cubic id 18
```

- Available in sysctls as any native kernel TCP CC



```
[root@arch-fb-vm1 bpf]# sysctl net.ipv4.tcp_available_congestion_control
net.ipv4.tcp_available_congestion_control = reno cubic bpf_cubic
```

- Can be used as other native kernel TCP CC



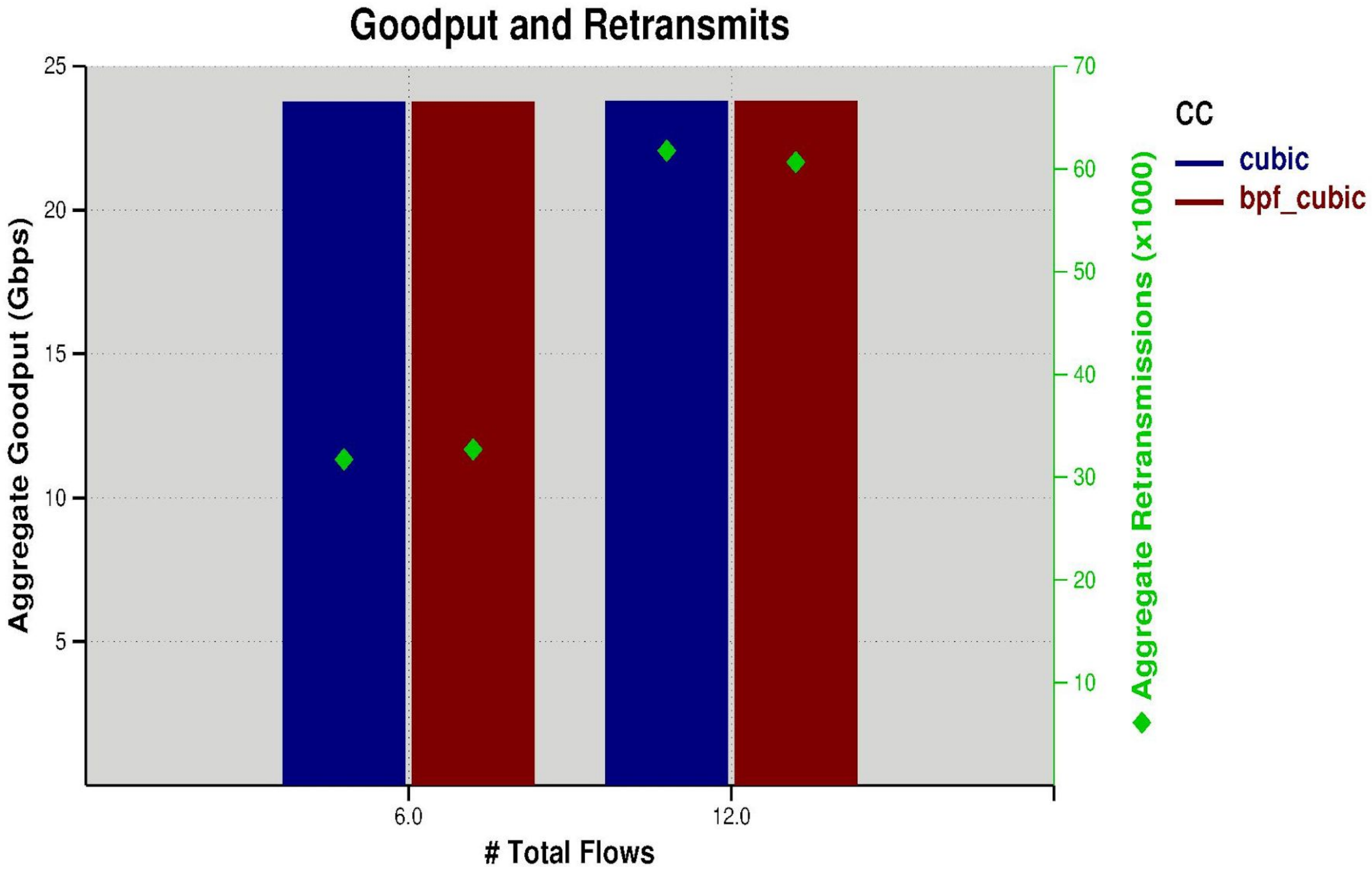
```
[root@arch-fb-vm1 bpf]# sysctl -w net.ipv4.tcp_congestion_control=bpf_cubic
net.ipv4.tcp_congestion_control = bpf_cubic
```

BPF TCP CC

How to use it in existing program?

- setsockopt() works as-is also. For example:

```
setsockopt(fd, IPPROTO_TCP, TCP_CONGESTION, "bpf_cubic",
           strlen("bpf_cubic"));
```


Goodput and Retransmits
Aggregate Goodput (Gbps)
25
20
15
10
5
70
60
50
40
30
20
10
◆ Aggregate Retransmissions (x1000)
6.0
12.0
# Total Flows
CC
cubic
bpf_cubic

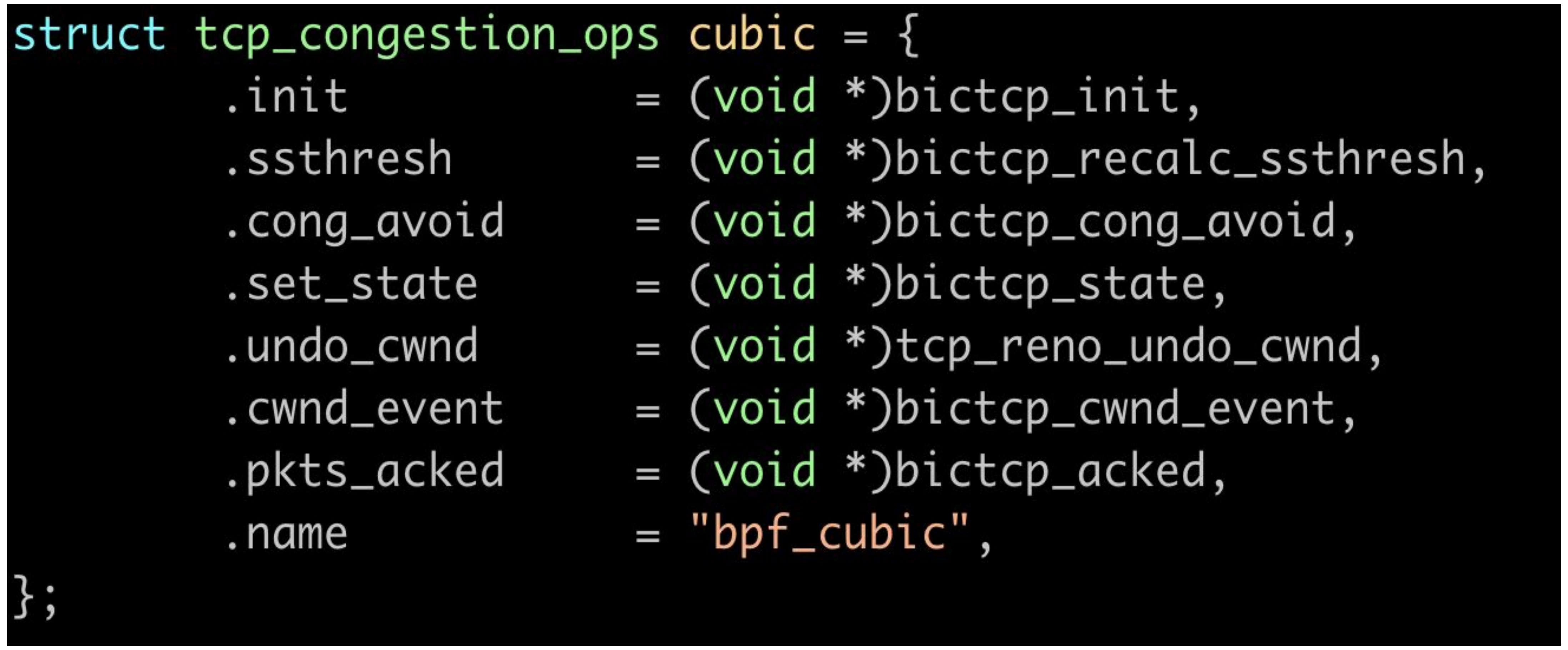
BPF struct_ops

What BPF TCP CC is built upon

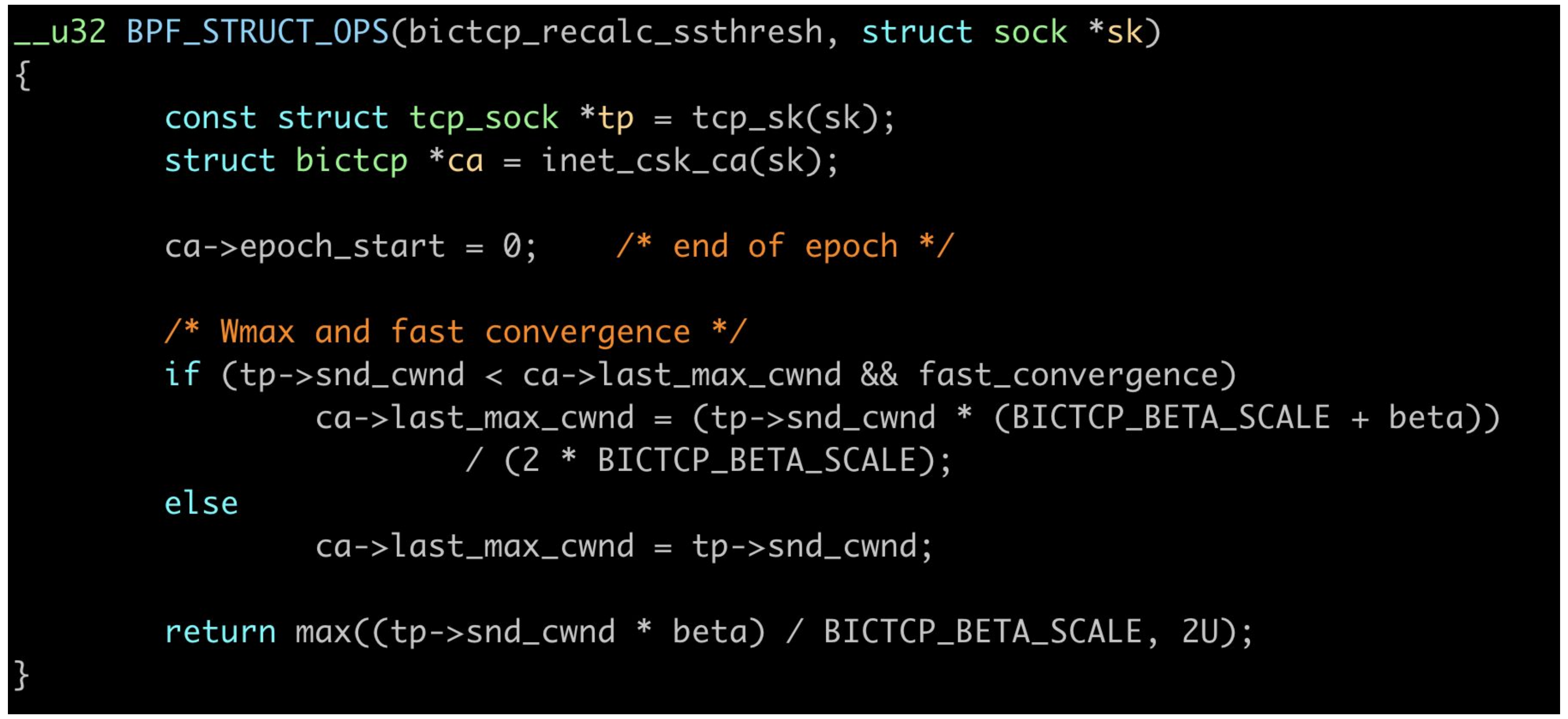
- A kernel “C” struct with a few function pointers
  - kernel module, tcp_congestion_ops, Qdisc_ops, proto…etc.
- bpf_struct_ops
  - An API to implement function pointers (of a kernel struct) in BPF
  - Each function pointer is implemented in a bpf prog in BPF_PROG_TYPE_STRUCT_OPS
  - struct_ops bpf program does not have a static running ctx
    - BTF of kernel: Get the function signature. Only push the needed args to the stack
- Leveraged BTF aware verifier, Trampoline, and CO-RE.

BPF struct_ops

What BPF TCP CC is built upon



```
struct tcp_congestion_ops cubic = {
        .init           = (void *)bictcp_init,
        .ssthresh       = (void *)bictcp_recalc_ssthresh,
        .cong_avoid     = (void *)bictcp_cong_avoid,
        .set_state      = (void *)bictcp_state,
        .undo_cwnd      = (void *)tcp_reno_undo_cwnd,
        .cwnd_event     = (void *)bictcp_cwnd_event,
        .pkts_acked     = (void *)bictcp_acked,
        .name           = "bpf_cubic",
};
```



```
__u32 BPF_STRUCT_OPS(bictcp_recalc_ssthresh, struct sock *sk)
{
        const struct tcp_sock *tp = tcp_sk(sk);
        struct bictcp *ca = inet_csk_ca(sk);

        ca->epoch_start = 0;    /* end of epoch */

        /* Wmax and fast convergence */
        if (tp->snd_cwnd < ca->last_max_cwnd && fast_convergence)
                ca->last_max_cwnd = (tp->snd_cwnd * (BICTCP_BETA_SCALE + beta))
                        / (2 * BICTCP_BETA_SCALE);
        else
                ca->last_max_cwnd = tp->snd_cwnd;

        return max((tp->snd_cwnd * beta) / BICTCP_BETA_SCALE, 2U);
}
```

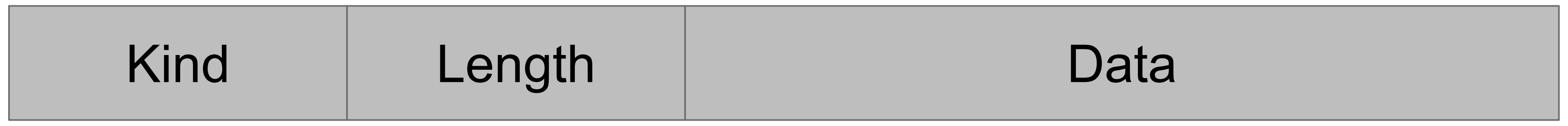
- libbpf
  - Load all the BPF_PROG_TYPE_STRUCT_OPS programs
  - Create the “struct tcp_congestion_ops” object
    - function pointers pointing to the bpf prog fds
  - Load this kernel object to the kernel
- Use bpftools instead!
  - “bpftools struct_ops register bpf_cubic.o” does all the above

BPF TCP Header Option

- Allow BPF prog to write and parse TCP header option
  - Write max delay ack in header and the receiver set a lower RTO
  - NIC speed
  - Preferred CC
  - …etc.
- The bpf prog can write any header option kind. The kernel will check for duplicated option.
  - A lot of flexibility for datacenter internal traffic
  - Potentially support the new standard option in an older kernel
- Commonly used during 3-way handshake
- Can also parse and write option in data, pure-ack, and FIN header

BPF TCP Header Option

Write SYNACK



Kind	Length	Data

```
static int write_synack_opt(struct bpf_sock_ops *skops)
{
        /* (1) Look for a particular option kind == 0xDA (Delay Ack) */
        syn_opt_in.kind = 0xDA;
        err = bpf_load_hdr_opt(skops, &syn_opt_in, sizeof(syn_opt_in),
                               BPF_LOAD_HDR_OPT_TCP_SYN);

        /* (2) Client does not support 0xDA option.  Write nothing in SYNACK. */
        if (err == -ENOMSG) return CG_OK;

        /* (3) Ask client to resend the option later if server is in syncookie */
        if (skops->args[0] == BPF_WRITE_HDR_TCP_SYNACK_COOKIE)
                synack_opt_out.data[0] |= OPTION_F_RESEND;

        /* (4) Write the server max delay ack in synack */
        synack_opt_out.data[1] = 10; /* 10ms max delay ack */
        bpf_store_hdr_opt(skops, &synack_opt_out, sizeof(synack_opt_out), 0);
}
```

BPF TCP Header Option

Passive Side Established

```
static int handle_passive_estab(struct bpf_sock_ops *skops)
{
      /* (1) Look for a particular option “0xDA” in SYN */
      syn_opt_in.kind = 0xDA;
      err = bpf_load_hdr_opt(skops, &syn_opt_in, sizeof(syn_opt_in),
                             BPF_LOAD_HDR_OPT_TCP_SYN);

      /* (2) Client does not have 0xDA option */
      if (err == -ENOMSG) return CG_OK;

      /* (3) Use a lower RTO to match the delay ack of the client */
      min_rto_us = syn_opt_in.data[1] * 1000;
      bpf_setsockopt(skops, SOL_TCP, TCP_BPF_RTO_MIN, &min_rto_us,
                     sizeof(min_rto_us));
}
```

sk storage for BPF Program

- It is very common that a bpf program wants to associate some data to a specific sk
- For example, a new TCP CC may want to store a few more data points of a connection
- hashtab way:
  - Define a bpf hashmap with the 4-tuple as the key and the data as the value.
    - Expensive: cpu for the lookup.
    - Maintenance nightmare: when to remove this key from the map?
- bpf_sk_storage way
  - Store the data directly at the sk itself and the data will go away with the sk
  - bpf_sk_storage_get(smap, sk, …)
  - Benchmark shows >50% lookup time improvement
  - Being re-purposed to other kernel objects (e.g. bpf_inode_storage)

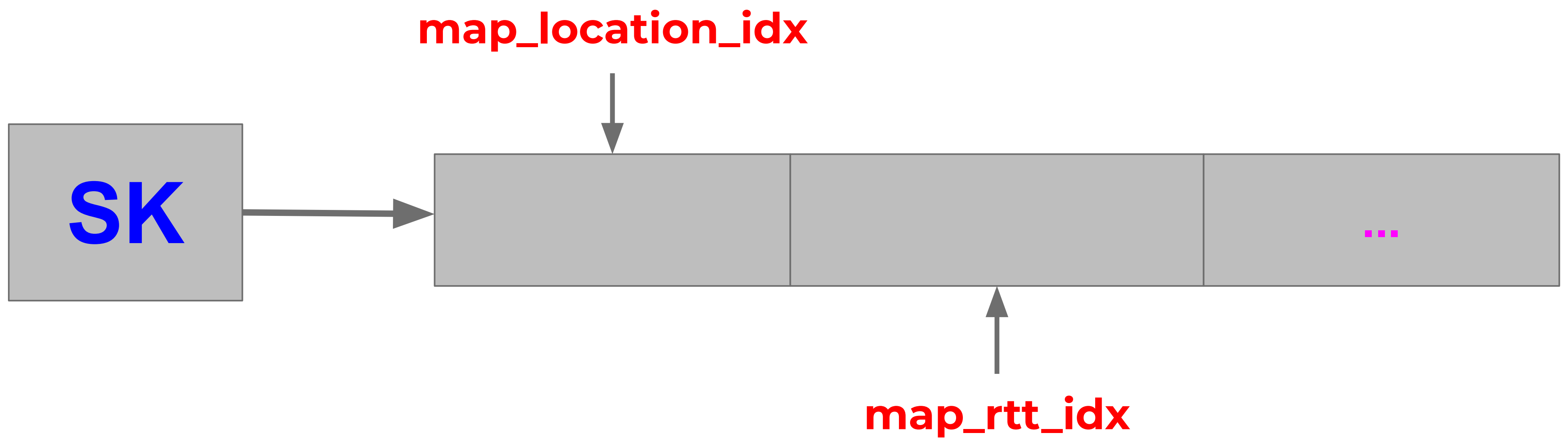
sk storage for BPF Program

BPF_MAP_TYPE_SK_STORAGE

- Define BPF_MAP_TYPE_SK_STORAGE map.
  - Key must be a socket fd
  - Value is whatever to be stored in the sk
- For example, two SK_STORAGE map defined:
  - **map_rtt** to store RTT data of a sk
  - **map_location** to store location data of the remote side (East/West coast, APAC, EUR...etc)

sk storage for BPF Program

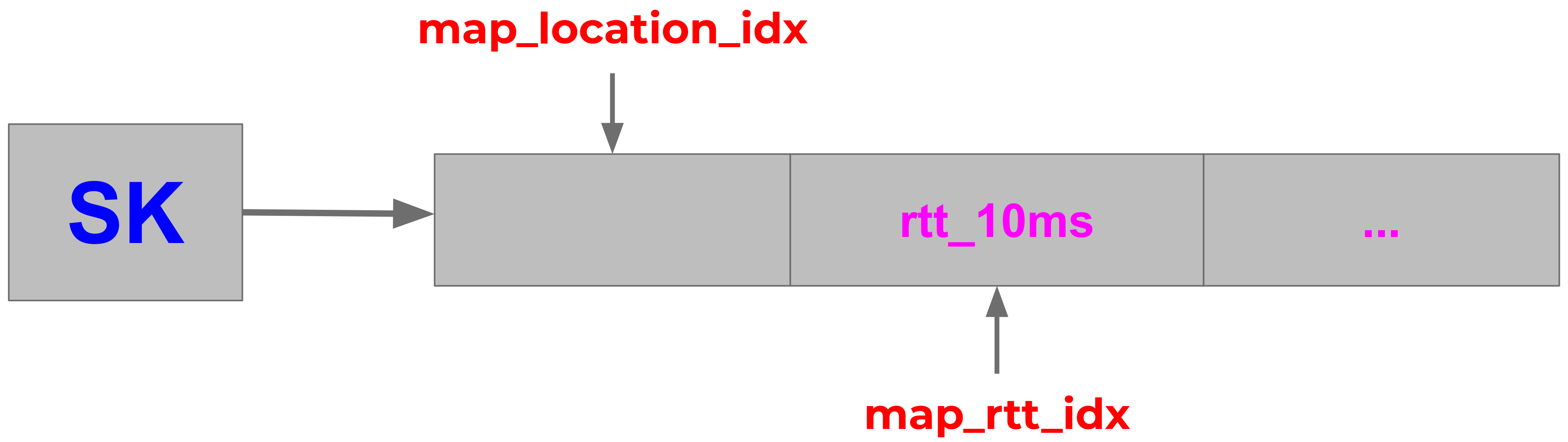
Access from BPF program



sk storage for BPF Program

Access from BPF program

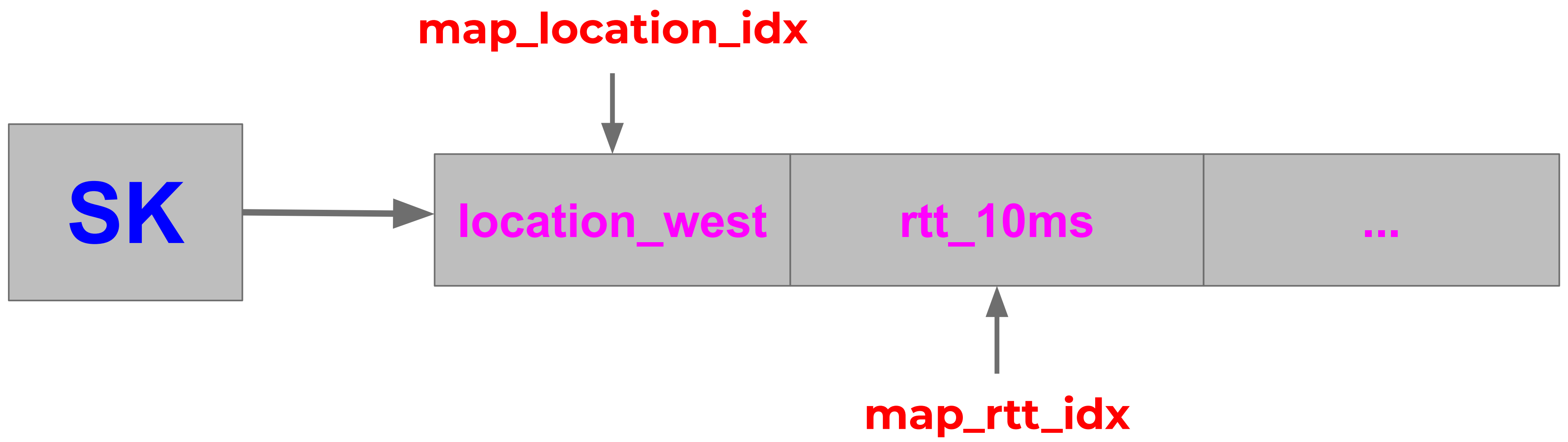
bpf_sk_storage_get(&**map_rtt**, **sk**, &**rtt_10ms**, BPF_SK_STORAGE_GET_F_CREATE);



sk storage for BPF Program

Access from BPF program

```
bpf_sk_storage_get(&map_rtt, sk, &rtt_10ms, BPF_SK_STORAGE_GET_F_CREATE);

bpf_sk_storage_get(&map_location, sk, &location_west, BPF_SK_STORAGE_GET_F_CREATE);
```

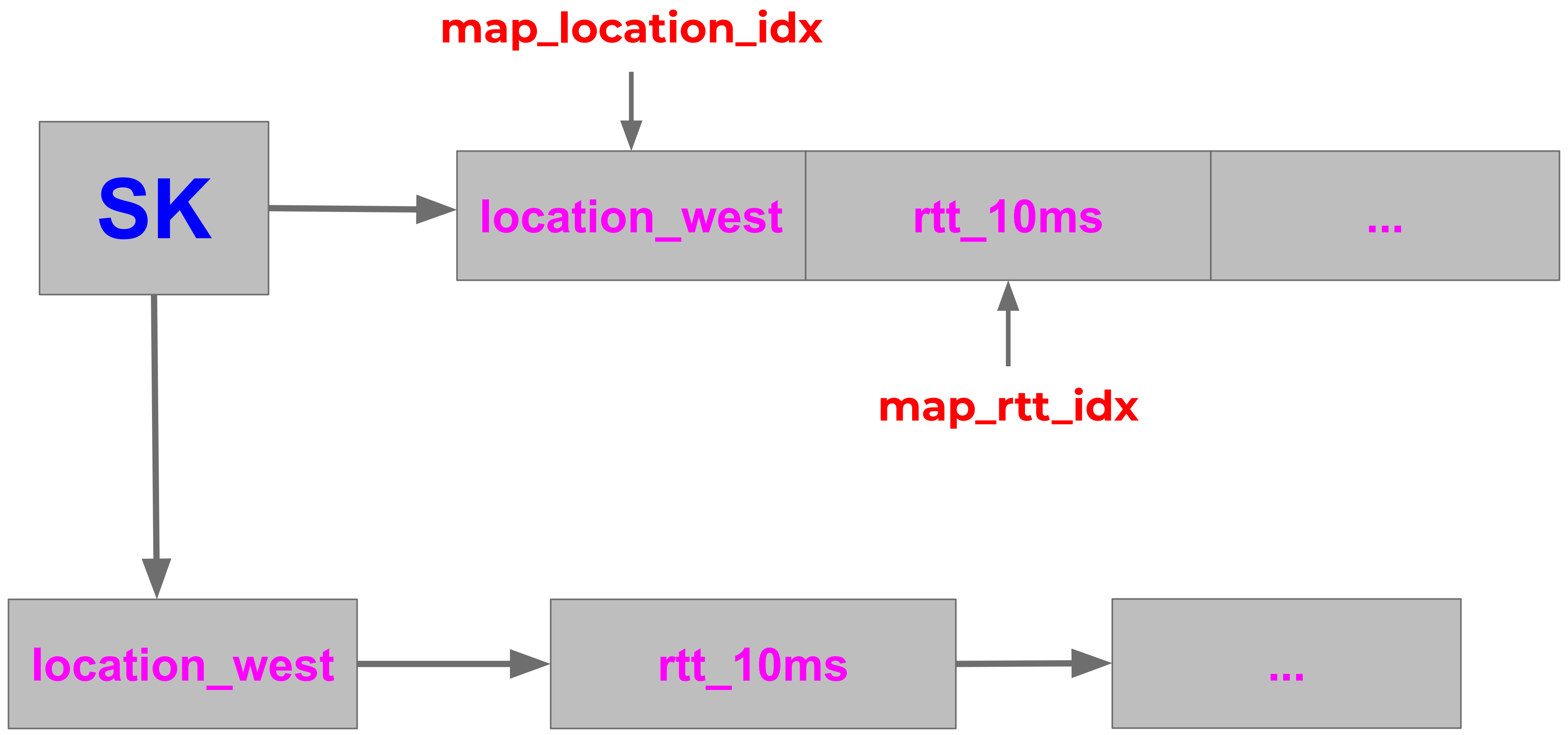


sk storage for BPF Program

Access from BPF program

bpf_sk_storage_get(&**map_rtt**, **sk**, &**rtt_10ms**, BPF_SK_STORAGE_GET_F_CREATE);

bpf_sk_storage_get(&**map_location**, **sk**, &**location_west**, BPF_SK_STORAGE_GET_F_CREATE);



sk storage for BPF Program

Access from userspace

- Access BPF_MAP_TYPE_SK_STORAGE map through regular map API
  - bpf_map_update_elem(map_location_fd, &**sk_fd**, &location_east, 0)
- It must hold a socket fd
- For a shared map, other processes may not have a hold on the fd ☹
- Other maps have a similar situation (as a value), e.g. sockmap, reuseport_array…etc.
- An ID for each sk: there is already sk cookie
- A generic way to do sk cookie => fd?

What Next?

Q&A

- What else do you want to de-ossify in BPF?