Traffic Control the 🐇Rabbit(MQ) with Rust🦀 using RedBPF

eBPF Summit

Xun Lou

October 28, 2020

In This Talk…

- Different “types” of BPF programs
- Write BPF programs in Rust
- Add new feature in RedBPF
- Use BPF maps to make stateful decisions
- Load the program and protect the Rabbit(MQ)!

About Me

- Software Engineer @ CCP Games
- @aquarhead on GitHub, Twitter…
- Rust (and Elixir)
- Disclaimer: new to BPF & kernel networking, pardon my mistake and welcome corrections!

Sad Rabbit Has No Memory

- A faulty client spammed “AMQP consumers”
- RabbitMQ cluster runs out of memory
- Need a way to limit the number of consumers
- But adding such a feature in RabbitMQ could be a long process…

Build a Limiter in BPF

- Let’s use BPF to get a quick win!
- Track how many “AMQP consumers” have been declared for each connection
- Drop further consumer declare packets once the limit is hit

RedBPF

- Most frameworks require C for BPF programs
- RedBPF uses Rust for *both* in-kernel and user-space programs - benefits from LLVM integration
- Rust: expressive type system, modern toolchain - but most importantly, I love Rust!
- For networking, RedBPF supports XDP and SocketFilter programs, however…

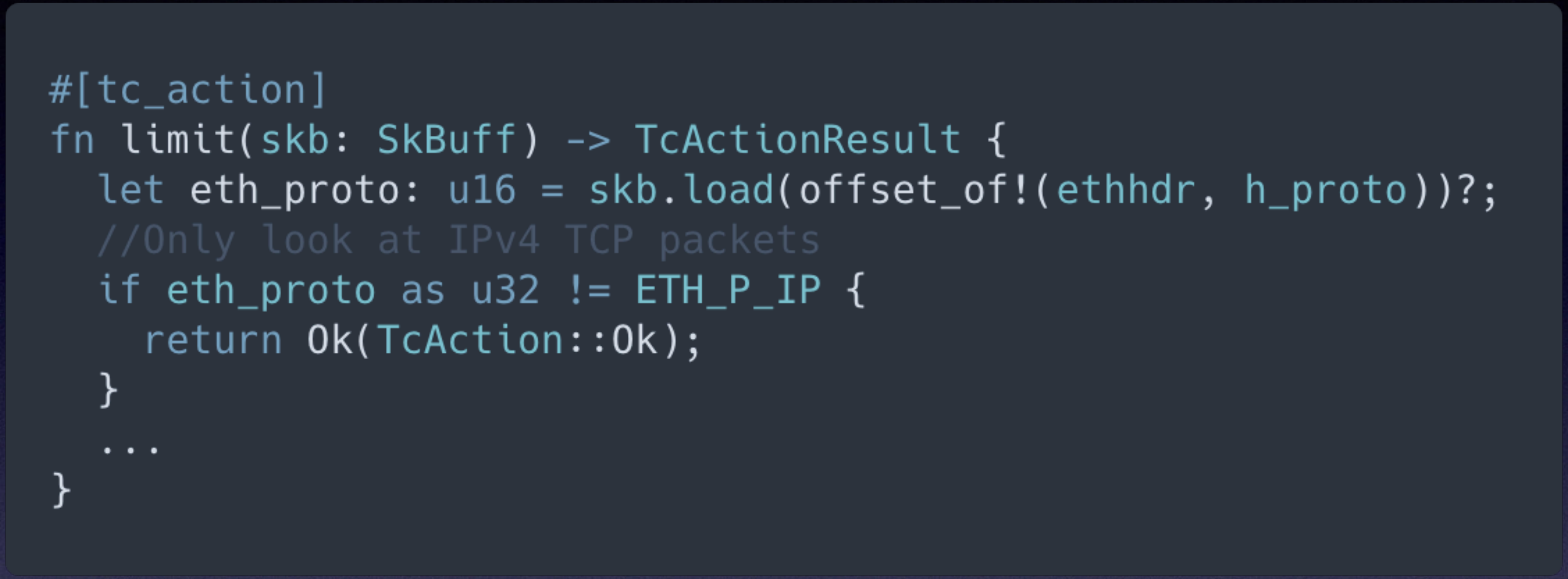
Traffic Control for Real

- XDP doesn't seem would work (full TCP packet hasn't been constructed yet - I could be wrong)
- SocketFilter is not useful: it only **duplicates** filtered traffic to a user-space program (e.g. for analyzing), does not affect original packets
- \`tc\` can actually control packets! And use BPF!
- Let's add support for it in RedBPF

`tc` Support in RedBPF

- BPF programs are all the “same”
- “Type” really depends on the input and how the kernel interprets the output
- `tc` programs also take `sk_buff` - steal from SocketFilter
- Use Enum to wrap potential return codes
- Done in https://github.com/redsift/redbpf/pull/97

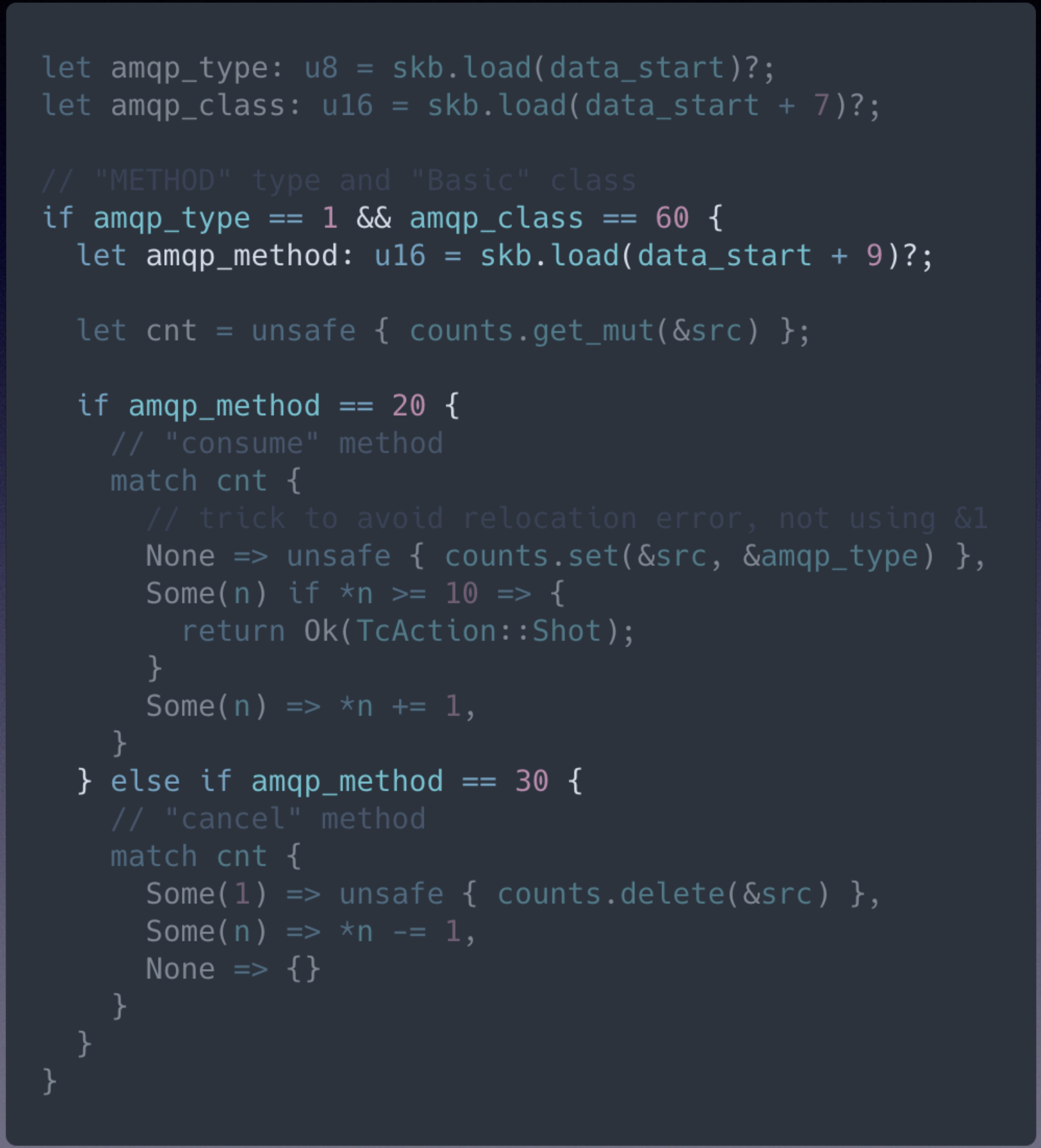
Write BPF in Rust



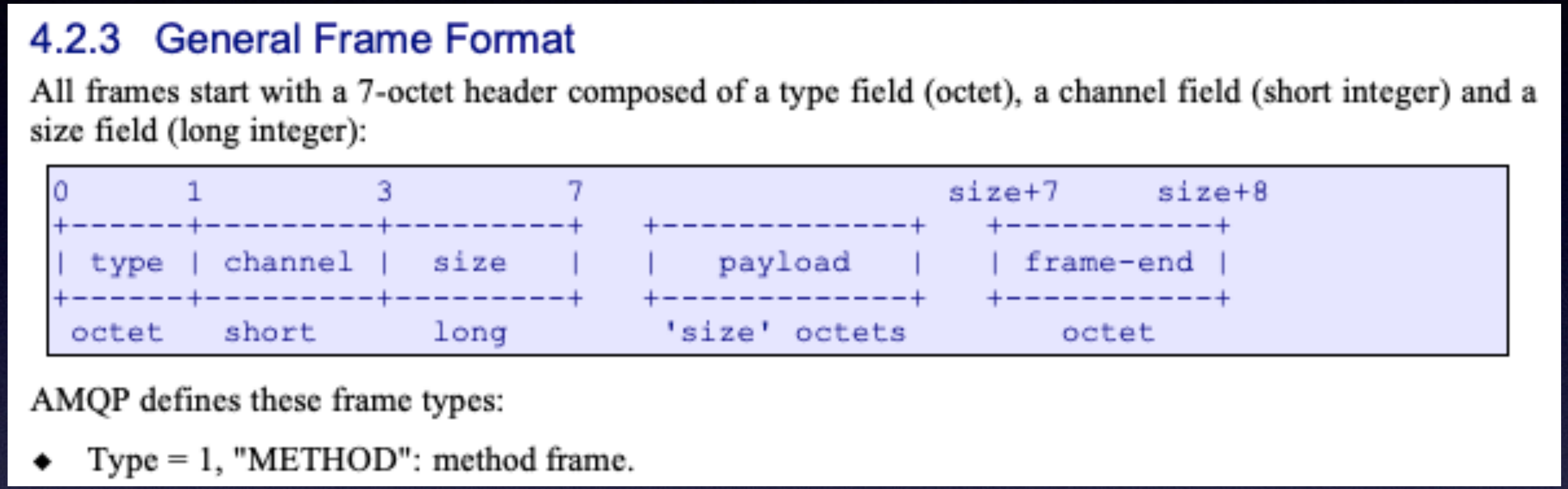
```
#[tc_action]
fn limit(skb: SkBuff) -> TcActionResult {
  let eth_proto: u16 = skb.load(offset_of!(ethhdr, h_proto))?;
  //Only look at IPv4 TCP packets
  if eth_proto as u32 != ETH_P_IP {
    return Ok(TcAction::Ok);
  }
  ...
}
```

- Ethernet frame, IP header, TCP header
  - Only look at IPv4, TCP packet to AMQP port
  - Extract source IP & port as BPF map key

Extract AMQP Methods



```
let amqp_type: u8 = skb.load(data_start)?;
let amqp_class: u16 = skb.load(data_start + 7)?;

// "METHOD" type and "Basic" class
if amqp_type == 1 && amqp_class == 60 {
  let amqp_method: u16 = skb.load(data_start + 9)?;

  let cnt = unsafe { counts.get_mut(&src) };

  if amqp_method == 20 {
    // "consume" method
    match cnt {
      // trick to avoid relocation error, not using &1
      None => unsafe { counts.set(&src, &amqp_type) },
      Some(n) if *n >= 10 => {
        return Ok(TcAction::Shot);
      }
      Some(n) => *n += 1,
    }
  } else if amqp_method == 30 {
    // "cancel" method
    match cnt {
      Some(1) => unsafe { counts.delete(&src) },
      Some(n) => *n -= 1,
      None => {}
    }
  }
}
```



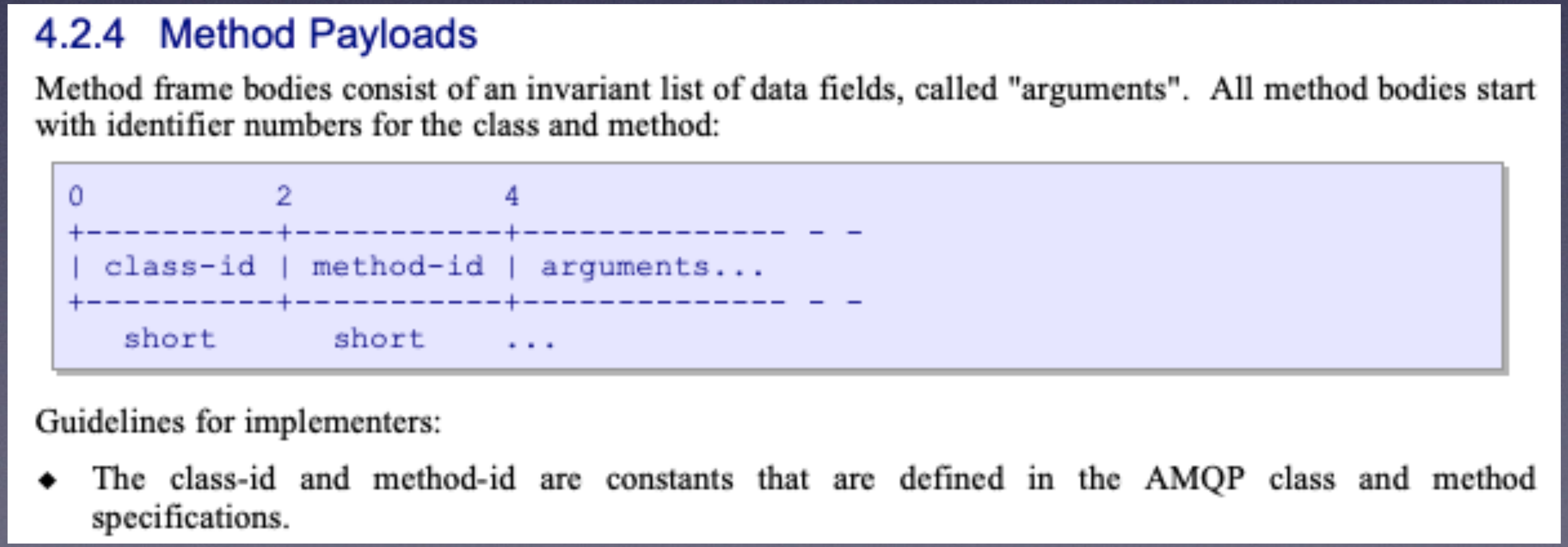
4.2.3 General Frame Format

All frames start with a 7-octet header composed of a type field (octet), a channel field (short integer) and a size field (long integer):

```
0      1         3         7                  size+7      size+8
+------+---------+---------+  +-------------+  +-----------+
| type | channel |  size   |  |   payload   |  | frame-end |
+------+---------+---------+  +-------------+  +-----------+
 octet   short      long       'size' octets       octet
```

AMQP defines these frame types:

- Type = 1, "METHOD": method frame.



4.2.4 Method Payloads

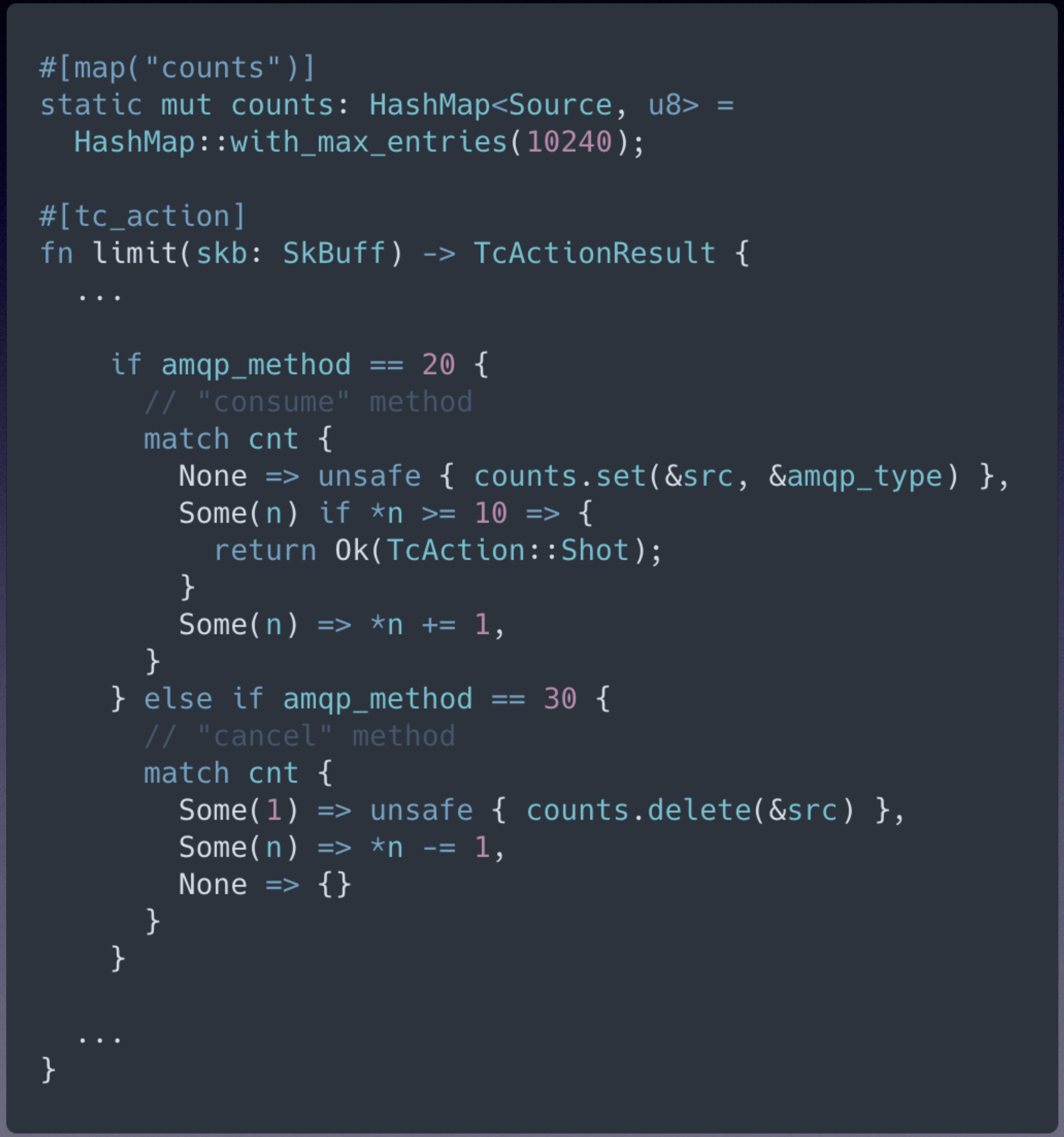
Method frame bodies consist of an invariant list of data fields, called "arguments". All method bodies start with identifier numbers for the class and method:

```
0          2           4
+----------+-----------+-------------- - -
| class-id | method-id | arguments...
+----------+-----------+-------------- - -
   short       short     ...
```

Guidelines for implementers:

- The class-id and method-id are constants that are defined in the AMQP class and method specifications.

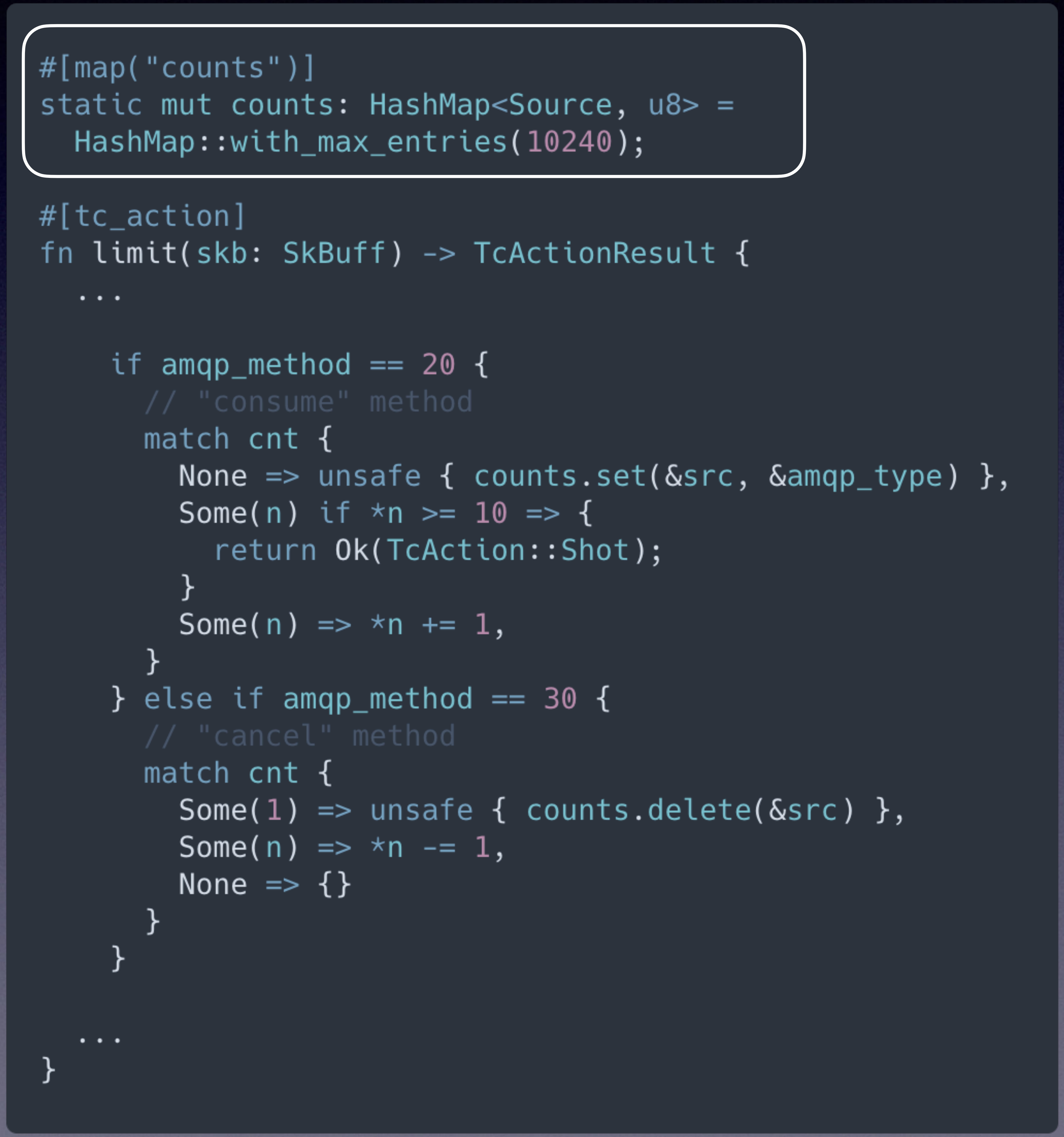
Use BPF Maps



```
#[map("counts")]
static mut counts: HashMap<Source, u8> =
  HashMap::with_max_entries(10240);

#[tc_action]
fn limit(skb: SkBuff) -> TcActionResult {
  ...

    if amqp_method == 20 {
      // "consume" method
      match cnt {
        None => unsafe { counts.set(&src, &amqp_type) },
        Some(n) if *n >= 10 => {
          return Ok(TcAction::Shot);
        }
        Some(n) => *n += 1,
      }
    } else if amqp_method == 30 {
      // "cancel" method
      match cnt {
        Some(1) => unsafe { counts.delete(&src) },
        Some(n) => *n -= 1,
        None => {}
      }
    }

  ...
}
```

Use BPF Maps

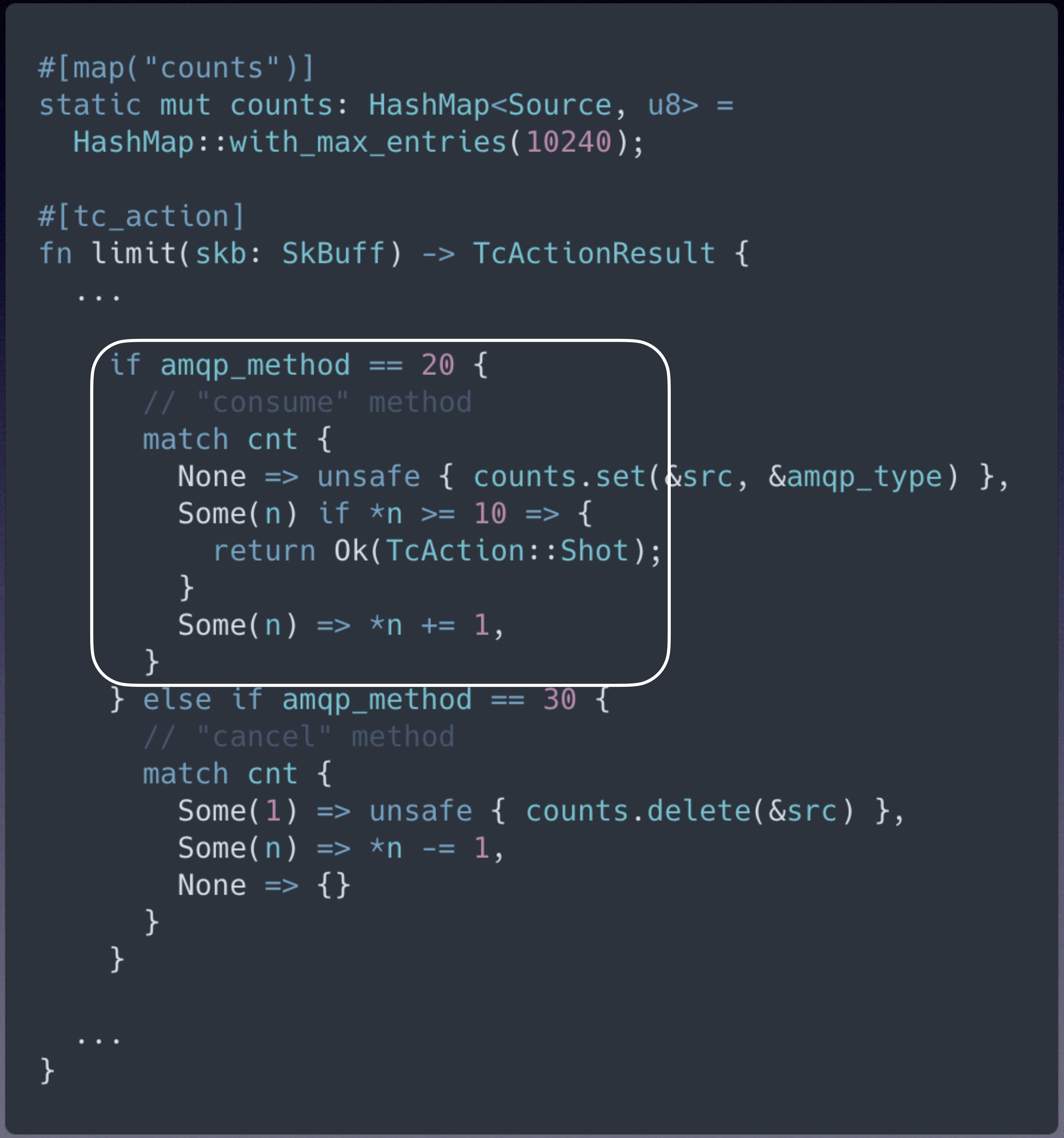
- Using the source IP & port as map key
- Map is a counter for consumers per connection



```
#[map("counts")]
static mut counts: HashMap<Source, u8> =
  HashMap::with_max_entries(10240);

#[tc_action]
fn limit(skb: SkBuff) -> TcActionResult {
  ...

    if amqp_method == 20 {
      // "consume" method
      match cnt {
        None => unsafe { counts.set(&src, &amqp_type) },
        Some(n) if *n >= 10 => {
          return Ok(TcAction::Shot);
        }
        Some(n) => *n += 1,
      }
    } else if amqp_method == 30 {
      // "cancel" method
      match cnt {
        Some(1) => unsafe { counts.delete(&src) },
        Some(n) => *n -= 1,
        None => {}
      }
    }

  ...
}
```

Use BPF Maps

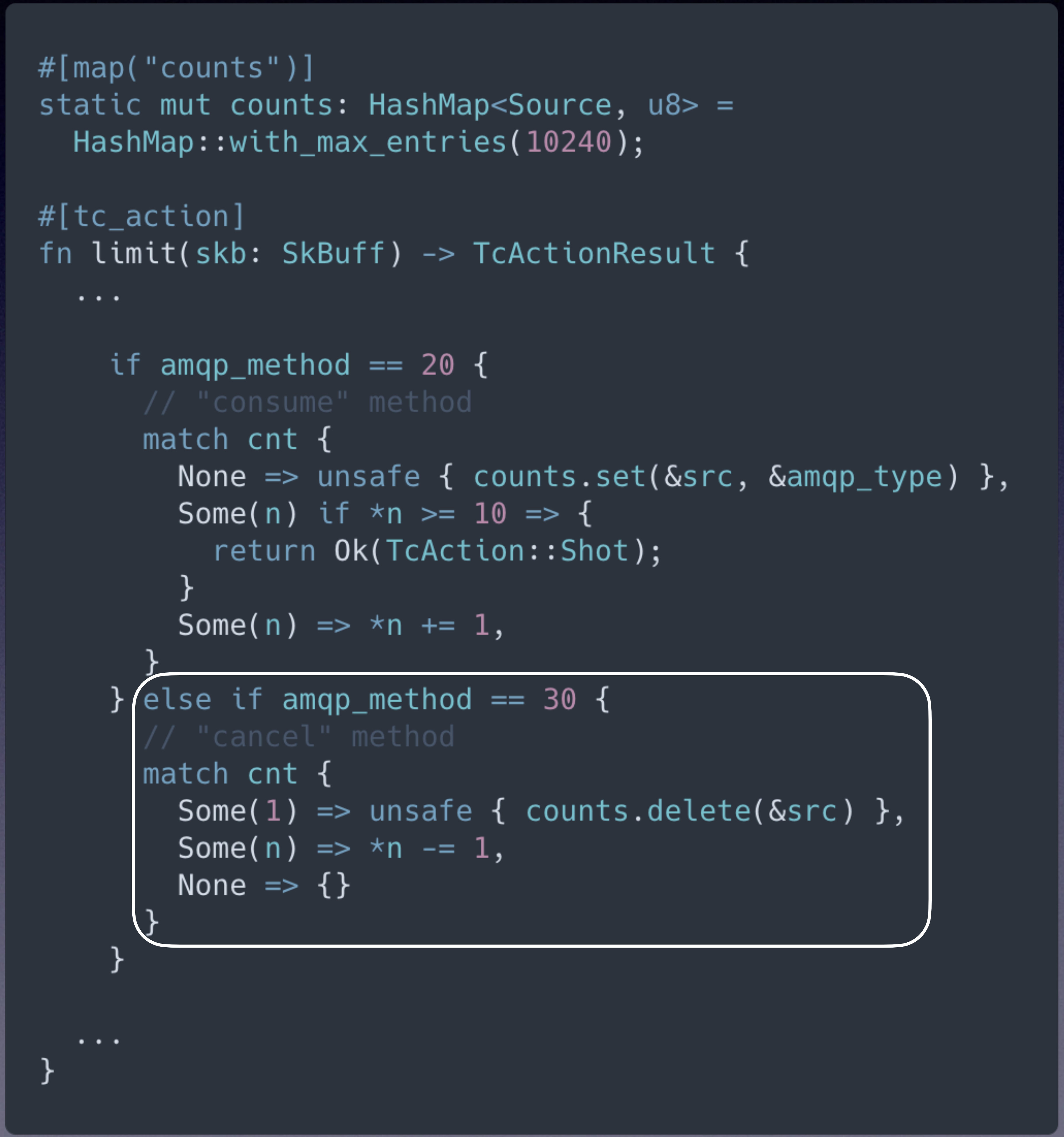
- Using the source IP & port as map key
- Map is a counter for consumers per connection
- Increase when declare



```
#[map("counts")]
static mut counts: HashMap<Source, u8> =
  HashMap::with_max_entries(10240);

#[tc_action]
fn limit(skb: SkBuff) -> TcActionResult {
  ...

    if amqp_method == 20 {
      // "consume" method
      match cnt {
        None => unsafe { counts.set(&src, &amqp_type) },
        Some(n) if *n >= 10 => {
          return Ok(TcAction::Shot);
        }
        Some(n) => *n += 1,
      }
    } else if amqp_method == 30 {
      // "cancel" method
      match cnt {
        Some(1) => unsafe { counts.delete(&src) },
        Some(n) => *n -= 1,
        None => {}
      }
    }

  ...
}
```

Use BPF Maps

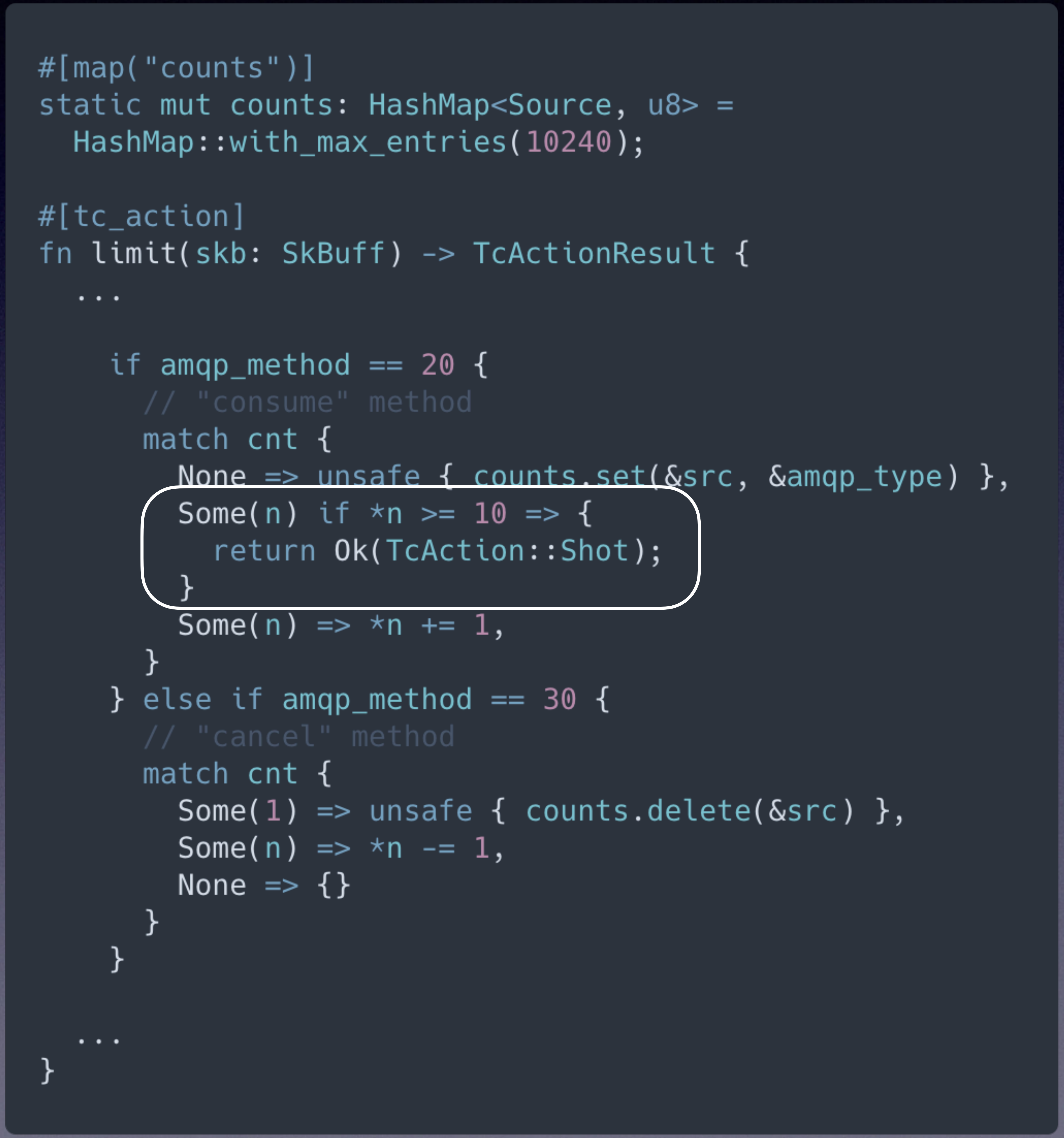
- Using the source IP & port as map key
- Map is a counter for consumers per connection
- Increase when declare
- Decrease when cancel



```
#[map("counts")]
static mut counts: HashMap<Source, u8> =
  HashMap::with_max_entries(10240);

#[tc_action]
fn limit(skb: SkBuff) -> TcActionResult {
  ...

    if amqp_method == 20 {
      // "consume" method
      match cnt {
        None => unsafe { counts.set(&src, &amqp_type) },
        Some(n) if *n >= 10 => {
          return Ok(TcAction::Shot);
        }
        Some(n) => *n += 1,
      }
    } else if amqp_method == 30 {
      // "cancel" method
      match cnt {
        Some(1) => unsafe { counts.delete(&src) },
        Some(n) => *n -= 1,
        None => {}
      }
    }

  ...
}
```

Use BPF Maps

- Using the source IP & port as map key
- Map is a counter for consumers per connection
- Increase when declare
- Decrease when cancel
- Drop (Shot) the declare packet if count is 10

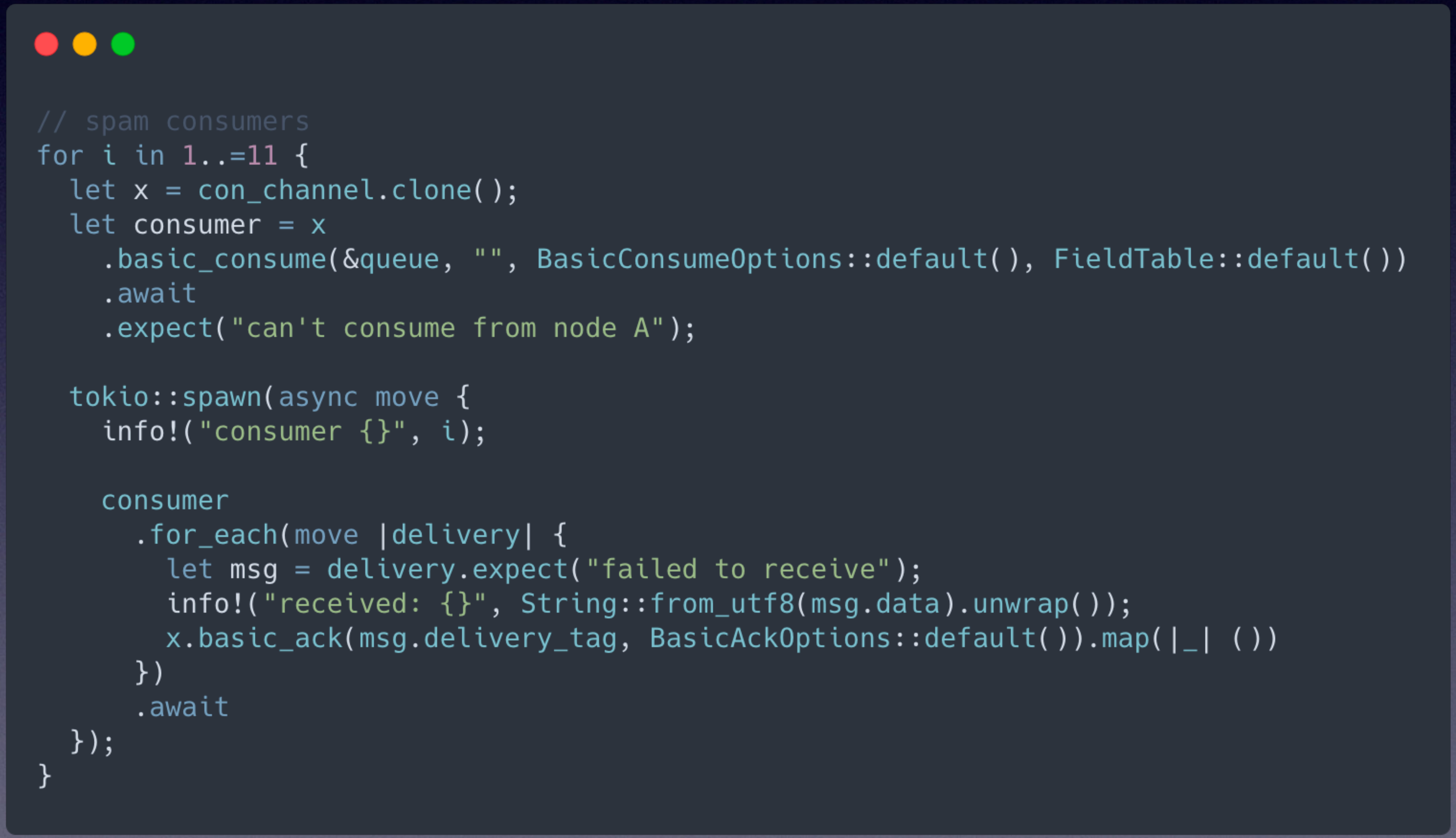


```rust
#[map("counts")]
static mut counts: HashMap<Source, u8> =
  HashMap::with_max_entries(10240);

#[tc_action]
fn limit(skb: SkBuff) -> TcActionResult {
  ...

    if amqp_method == 20 {
      // "consume" method
      match cnt {
        None => unsafe { counts.set(&src, &amqp_type) },
        Some(n) if *n >= 10 => {
          return Ok(TcAction::Shot);
        }
        Some(n) => *n += 1,
      }
    } else if amqp_method == 30 {
      // "cancel" method
      match cnt {
        Some(1) => unsafe { counts.delete(&src) },
        Some(n) => *n -= 1,
        None => {}
      }
    }

  ...
}
```

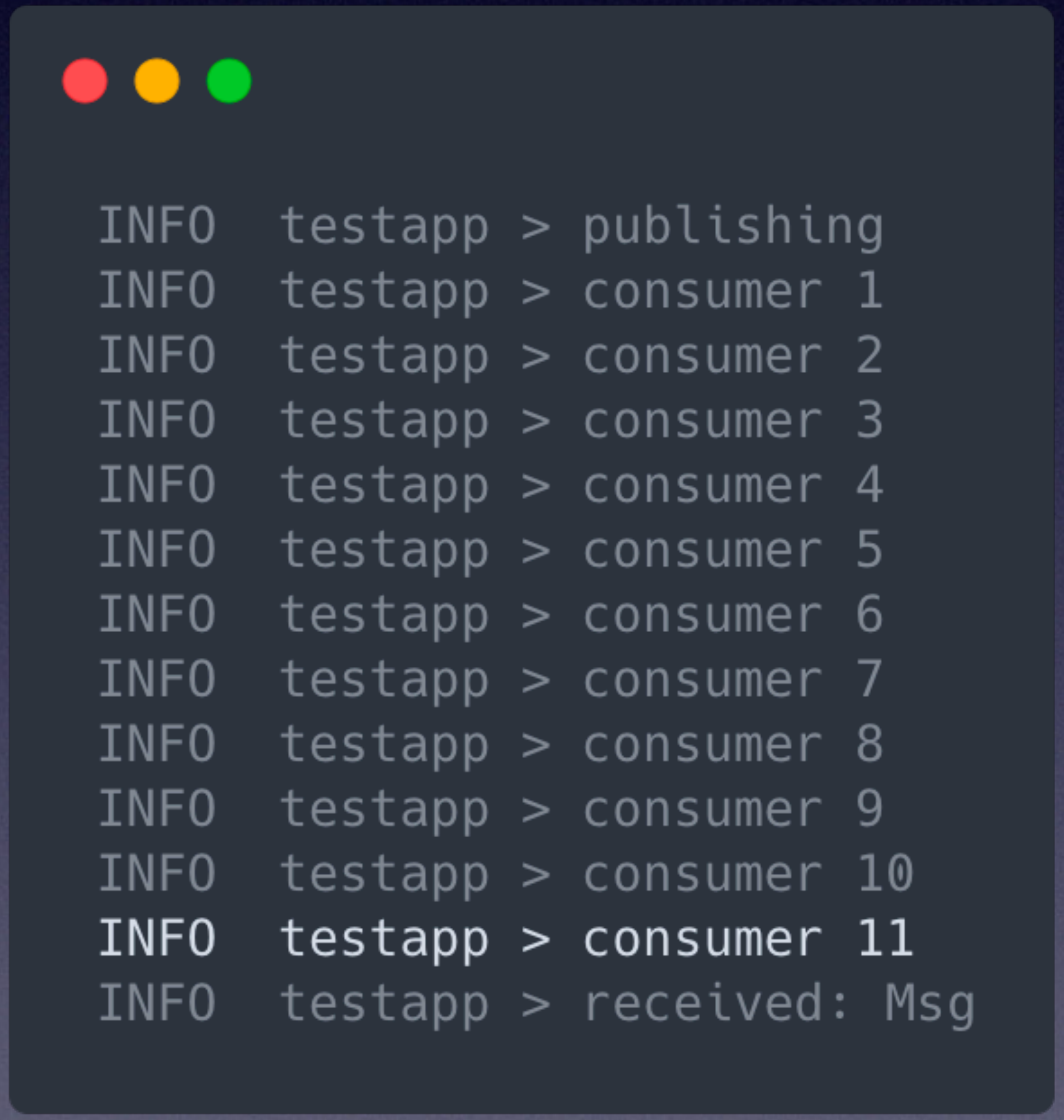
See it in Action!

Can we protect the Rabbit?

Without Limiter



```
// spam consumers
for i in 1..=11 {
  let x = con_channel.clone();
  let consumer = x
    .basic_consume(&queue, "", BasicConsumeOptions::default(), FieldTable::default())
    .await
    .expect("can't consume from node A");

  tokio::spawn(async move {
    info!("consumer {}", i);

    consumer
      .for_each(move |delivery| {
        let msg = delivery.expect("failed to receive");
        info!("received: {}", String::from_utf8(msg.data).unwrap());
        x.basic_ack(msg.delivery_tag, BasicAckOptions::default()).map(|_| ())
      })
      .await
  });
}
```

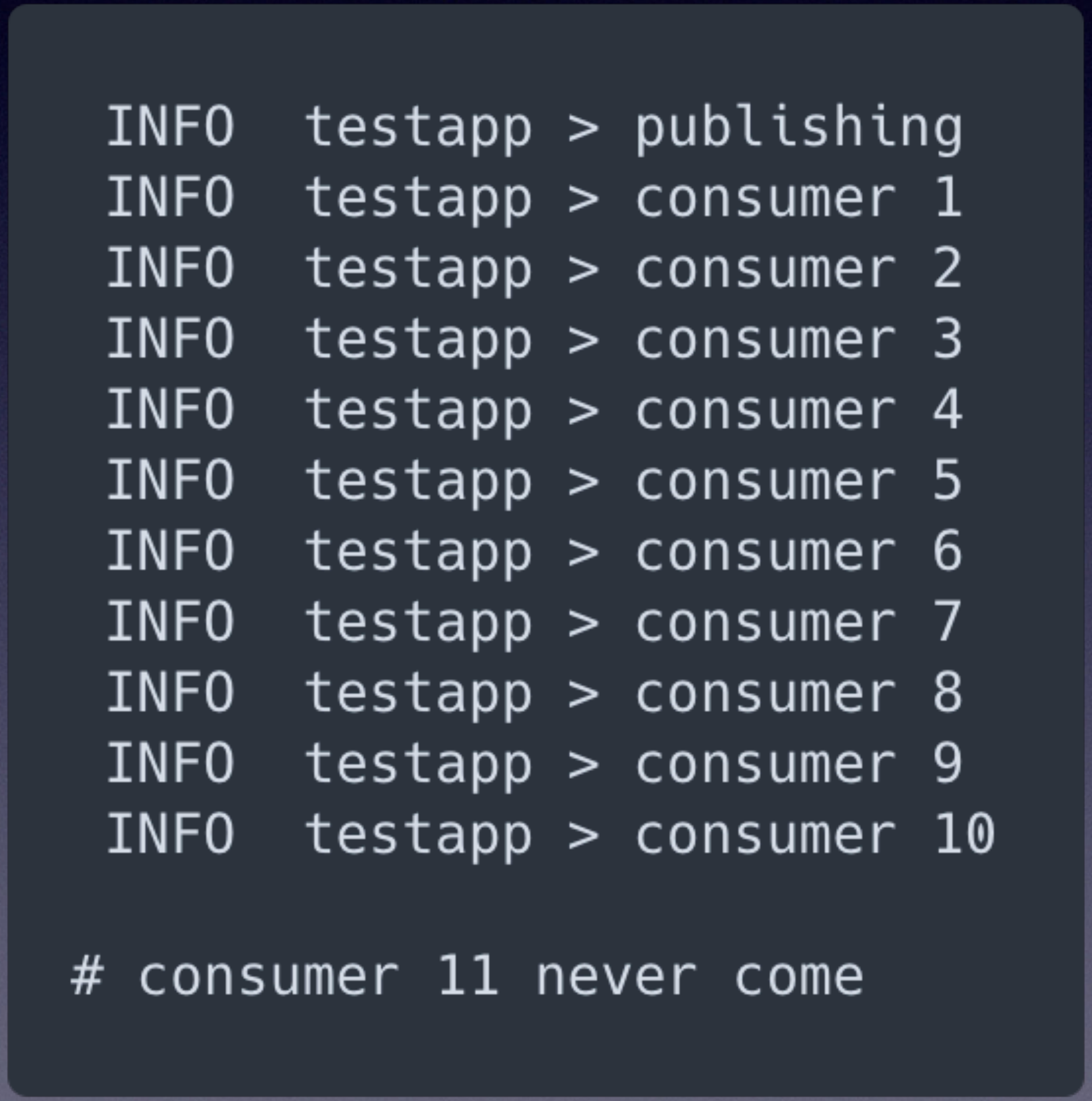


```
INFO  testapp > publishing
INFO  testapp > consumer 1
INFO  testapp > consumer 2
INFO  testapp > consumer 3
INFO  testapp > consumer 4
INFO  testapp > consumer 5
INFO  testapp > consumer 6
INFO  testapp > consumer 7
INFO  testapp > consumer 8
INFO  testapp > consumer 9
INFO  testapp > consumer 10
INFO  testapp > consumer 11
INFO  testapp > received: Msg
```

Attach `tc` Program

```
$ cargo make release

$ sudo tc qdisc add dev [device name] clsact

$ sudo tc filter add dev [device name] ingress \
    bpf da obj target/bpf/programs/limit/limit.elf \
    sec tc_action/limit
```

Rabbit(MQ) Protected



```
 INFO  testapp > publishing
 INFO  testapp > consumer 1
 INFO  testapp > consumer 2
 INFO  testapp > consumer 3
 INFO  testapp > consumer 4
 INFO  testapp > consumer 5
 INFO  testapp > consumer 6
 INFO  testapp > consumer 7
 INFO  testapp > consumer 8
 INFO  testapp > consumer 9
 INFO  testapp > consumer 10

# consumer 11 never come
```

BPF (Kernel) vs. Application

- BPF programs can be developed and deployed very quickly, and with great confidence due to kernel verifier

- Extra effort to track deeper state in applications (e.g. channel/connection relationship)

- BPF can cause unintended behavior (e.g. broken connection), but still a worthy tradeoff, especially in preventing misuse

More on RedBPF

- Plan to make RedBPF support more (all) program types - make it a generic compiler (BCC)
- Add utility functions to help dealing with network headers etc…
- Improve the compile output - ensure it works with other loader, size etc…
- Give RedBPF a try! Contributions welcome!

Takk!

Code: https://github.com/aquarhead/protect-the-rabbit
Talk to me: aquarhead@gmail.com / @aquarhead
https://aqd.is