SO_REUSEPORT and eBPF

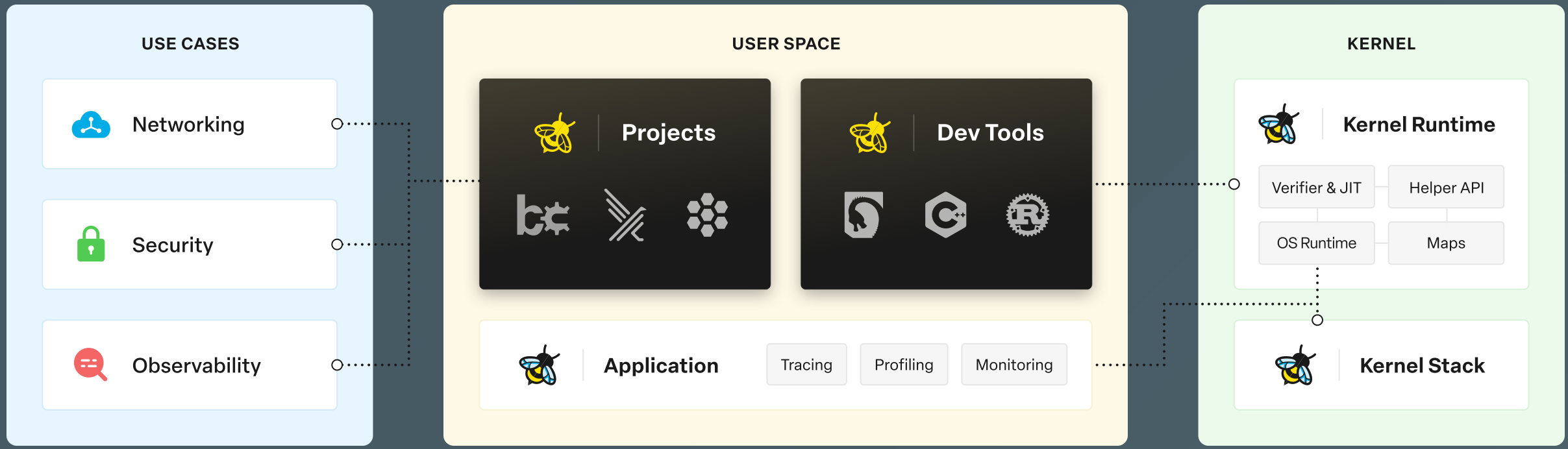
2024 ATS Spring Summit

Masaori Koshiba (masaori@apache.org)

Motivation

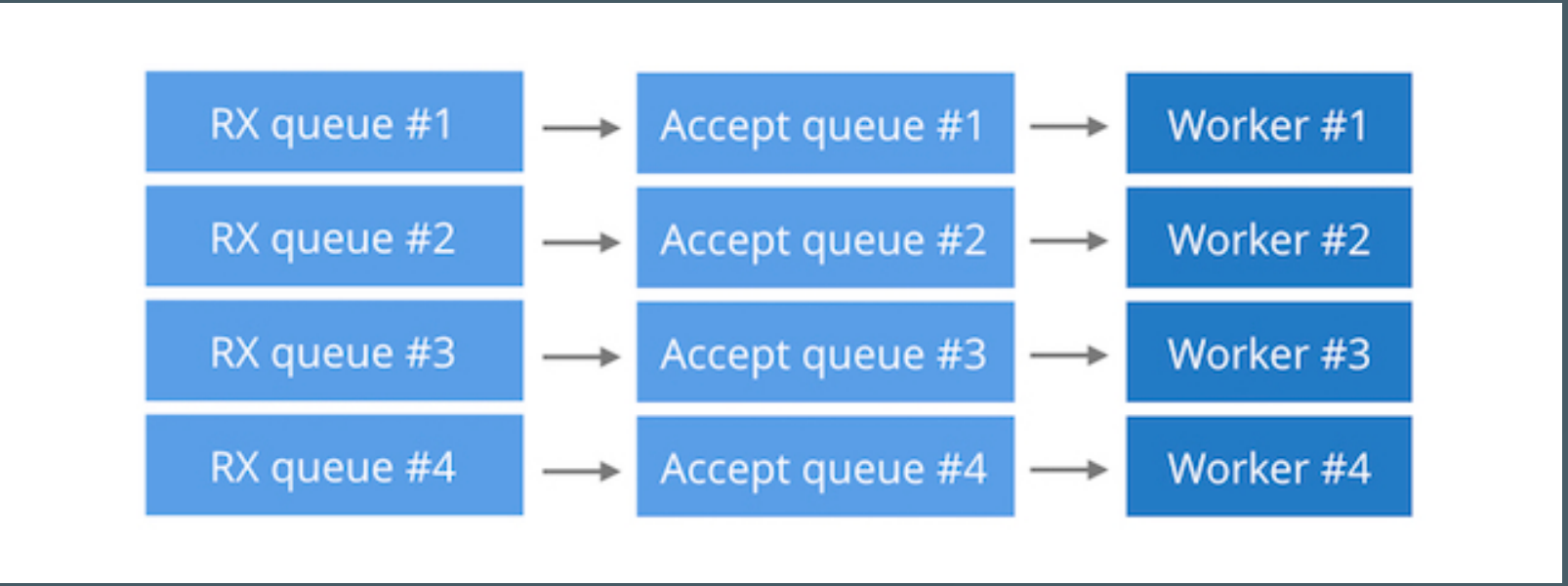
1. Perfect Locality → SO_INCOMING_CPU
2. Upgrade ATS without downtime → Load Balancing by eBPF

eBPF - Networking



1. Perfect Locality

- Aligning RX queue, Accept queue, and Application thread



- SO_INCOMING_CPU - it's bloken (v4.1 - v6.1). Fixed by v6.2.
- SO_ATTACH_REUSEPORT_CBPF / _EBPF

- e.g. SO_ATTACH_REUSEPORT_CBPF

```
int32_t mod = eventProcessor.num_threads(opt.etype);

struct sock_filter code[] = {
  // A = #cpu
  {BPF_LD | BPF_W | BPF_ABS, 0, 0, static_cast<uint32_t>(SKF_AD_OFF + SKF_AD_CPU)},
  // A = A % mod
  {BPF_ALU | BPF_MOD,        0, 0, mod                                             },
  // return A
  {BPF_RET | BPF_A,          0, 0, 0                                               },
};

struct sock_fprog p = {
  .len    = ARRAY_SIZE(code),
  .filter = code,
};

if (safe_setsockopt(fd, SOL_SOCKET, SO_ATTACH_REUSEPORT_CBPF, &p, sizeof(p)) < 0) {
  Error("SO_ATTACH_REUSEPORT_CBPF is configured but set is failed: %d", errno);
  return -errno;
}
```

Printing `sk_incoming_cpu`

Without CBPF program

```
» tcpaccept.py -P 8080 | grep "ET_NET 0"
PID     COMM         IP RADDR           RPORT LADDR           LPORT In-CPU
12598   [ET_NET 0]   4  127.0.0.1       44418 127.0.0.1       8080  49
12598   [ET_NET 0]   4  127.0.0.1       44496 127.0.0.1       8080  37
12598   [ET_NET 0]   4  127.0.0.1       44624 127.0.0.1       8080  60
12598   [ET_NET 0]   4  127.0.0.1       46246 127.0.0.1       8080  40
12598   [ET_NET 0]   4  127.0.0.1       46752 127.0.0.1       8080  26
12598   [ET_NET 0]   4  127.0.0.1       47650 127.0.0.1       8080  16
12598   [ET_NET 0]   4  127.0.0.1       47680 127.0.0.1       8080  10
...
```

Printing `sk_incoming_cpu`

With CBPF program

```
» tcpaccept.py -P 8080 | grep "ET_NET 0"
PID     COMM         IP RADDR           RPORT LADDR           LPORT In-CPU
10859   [ET_NET 0]   4  127.0.0.1       34802 127.0.0.1       8080  4
10859   [ET_NET 0]   4  127.0.0.1       35106 127.0.0.1       8080  4
10859   [ET_NET 0]   4  127.0.0.1       35890 127.0.0.1       8080  4
10859   [ET_NET 0]   4  127.0.0.1       37644 127.0.0.1       8080  4
10859   [ET_NET 0]   4  127.0.0.1       37878 127.0.0.1       8080  4
10859   [ET_NET 0]   4  127.0.0.1       38120 127.0.0.1       8080  4
10859   [ET_NET 0]   4  127.0.0.1       39116 127.0.0.1       8080  4
...
```

Benchmark (+ BRAVO PoC)

Without CBPF program

```
» taskset -c 0-63 wrk2 -c 512 -d 30 -t 64 -R 2000000 http://target/1KB
...
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     5.46s     4.45s   19.79s    58.42%
    Req/Sec    21.55k     1.99k   25.08k    64.58%
  40314358 requests in 29.99s, 13.44GB read
Requests/sec: 1344122.92
Transfer/sec:    458.90MB
```

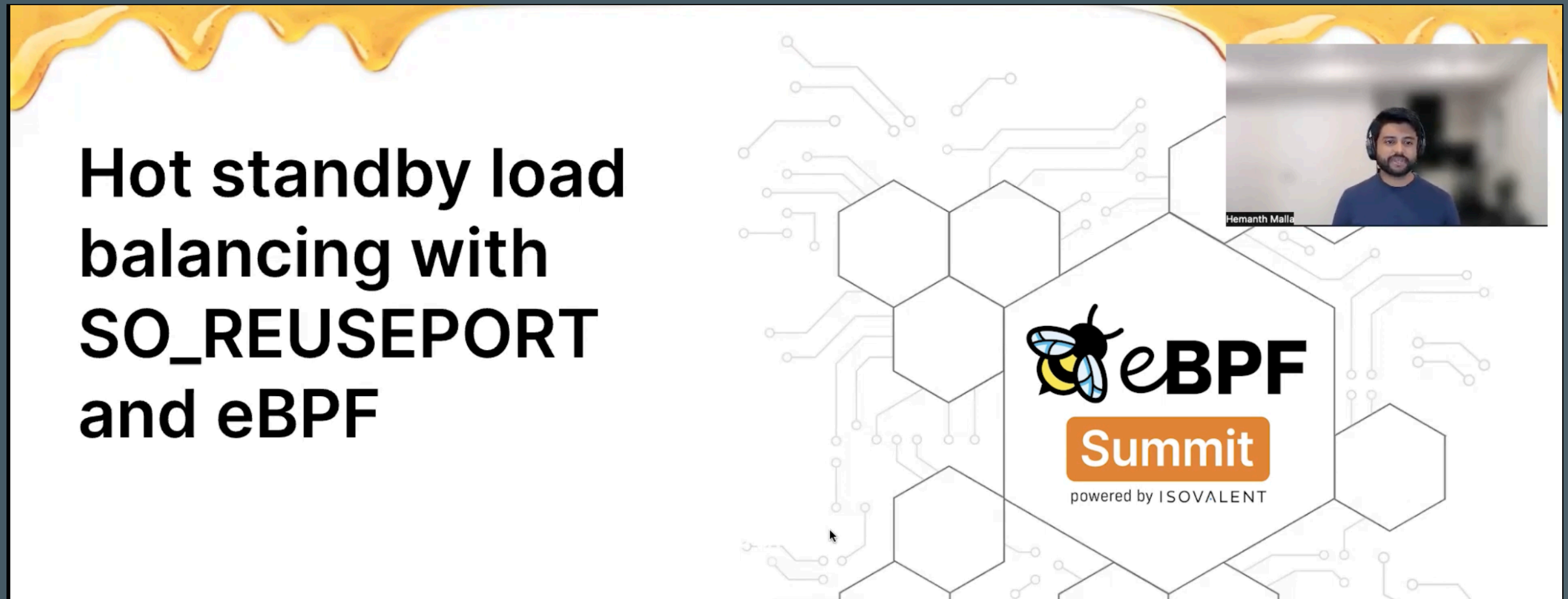
Benchmark (+ BRAVO PoC)

With CBPF program (5%+)

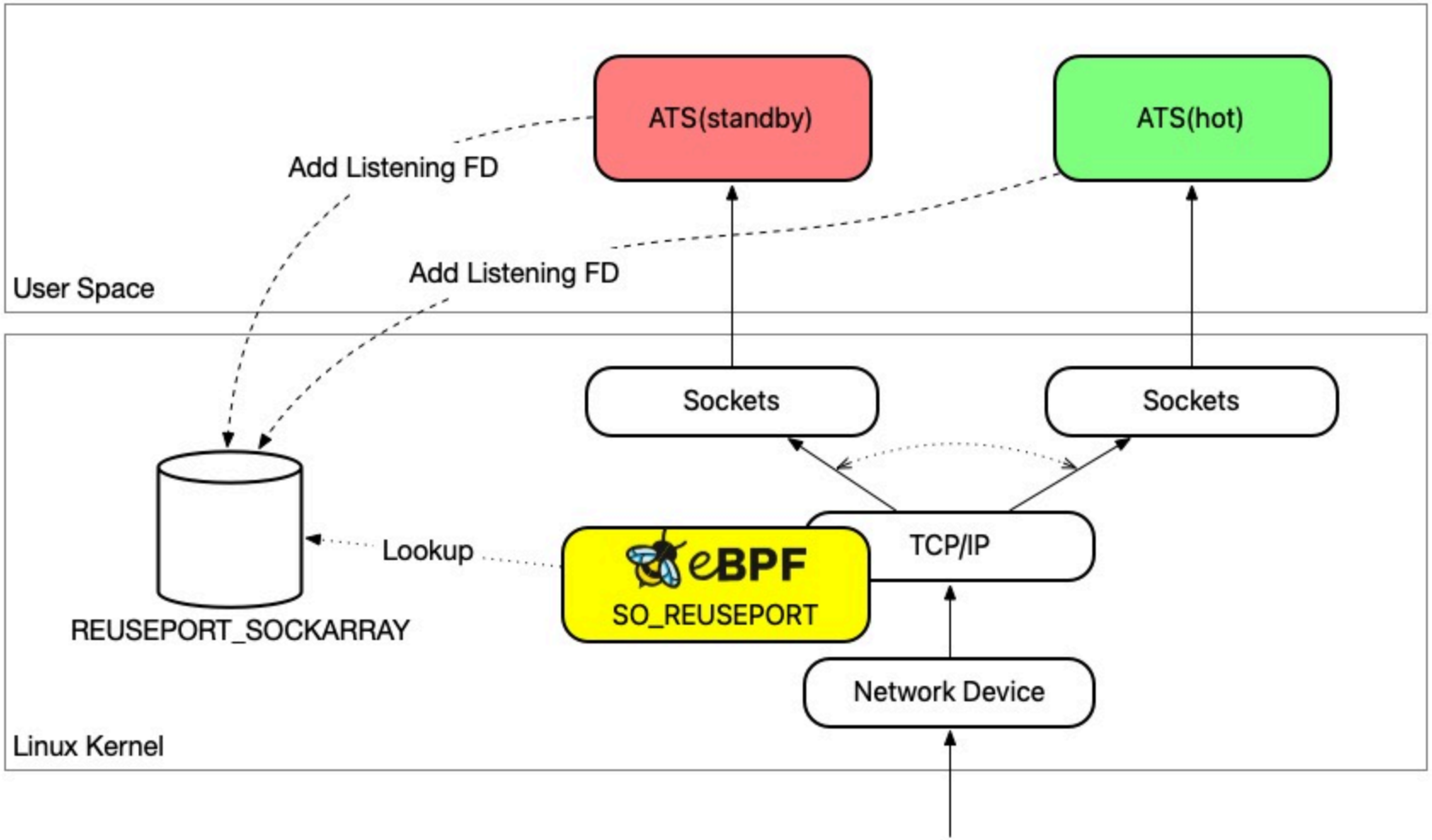
```
» taskset -c 0-63 wrk2 -c 512 -d 30 -t 64 -R 2000000 http://target/1KB
...
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     5.08s     3.69s   18.37s    64.45%
    Req/Sec    22.81k     1.75k   26.13k    65.06%
  42612692 requests in 29.99s, 14.21GB read
Requests/sec: 1420743.78
Transfer/sec:    485.06MB
```

2. Load Balancing by eBPF

eBPF Summit 2023



Load Balancing by eBPF



Demo - hot-standby

Hot

- /workspace/ats-1/bin/traffic_server

Standby

- /workspace/ats-2/bin/traffic_server

Ref.

- Perfect locality and three epic SystemTap scripts
- Custom loadbalancing for SO_REUSEPORT with eBPF
- BPF_PROG_TYPE_SK_REUSEPORT

SO_INCOMING_CPU

```
SO_INCOMING_CPU (gettable since Linux 3.19, settable since Linux 4.4)

  Sets or gets the CPU affinity of a socket.  Expects an integer flag.

    int cpu = 1;
    setsockopt(fd, SOL_SOCKET, SO_INCOMING_CPU, &cpu, sizeof(cpu));

  Because all of the packets for a single stream (i.e., all packets
  for the same 4-tuple) arrive on the single RX queue that is
  associated with a particular CPU, the typical use case is to employ
  one listening process per RX queue, with the incoming flow being
  handled by a listener on the same CPU that is handling the RX queue.
  This provides optimal NUMA behavior and keeps CPU caches hot.
```

SO_ATTACH_REUSEPORT_CBPF & _EBPF

```
SO_ATTACH_REUSEPORT_CBPF, SO_ATTACH_REUSEPORT_EBPF

  For use with the SO_REUSEPORT option, these options allow
  the user to set a classic BPF (SO_ATTACH_REUSEPORT_CBPF)
  or an extended BPF (SO_ATTACH_REUSEPORT_EBPF) program
  which defines how packets are assigned to the sockets in
  the reuseport group (that is, all sockets which have
  SO_REUSEPORT set and are using the same local address to
  receive packets).
```

tcp_migrate_req

tcp_migrate_req - BOOLEAN (5.14+)

The incoming connection is tied to a specific listening socket when the initial SYN packet is received during the three-way handshake. When a listener is closed, in-flight request sockets during the handshake and established sockets in the accept queue are aborted.

If the listener has SO_REUSEPORT enabled, other listeners on the same port should have been able to accept such connections. This option makes it possible to migrate such child sockets to another listener after close() or shutdown().

The BPF_SK_REUSEPORT_SELECT_OR_MIGRATE type of eBPF program should usually be used to define the policy to pick an alive listener. Otherwise, the kernel will randomly pick an alive listener only if this option is enabled.

Note that migration between listeners with different settings may crash applications. Let’s say migration happens from listener A to B, and only B has TCP_SAVE_SYN enabled. B cannot read SYN data from the requests migrated from A. To avoid such a situation, cancel migration by returning SK_DROP in the type of eBPF program, or disable this option.

Default: 0