TCP-BPF

PROGRAMMATICALLY TUNING TCP BEHAVIOR THROUGH BPF

Lawrence Brakmo

Facebook

brakmo@fb.com

INTRODUCTION

- Framework for optimizing TCP parameters programmatically
- Use flow information when setting parameters
  - Ex: tune for intra-DC flows
    - small buffers
    - small SYN RTO
    - clamp cwnd)
  - Ex: tune for WAN flows
    - larger buffers
    - larger INIT_CWND
    - larger RWND

INTRODUCTION (2)

- Can use application supplied BPF maps to make decisions
  - BPF prefix maps
- Can make rules that apply to the whole organization as opposed to per-DC (like ip-route)
- No need to modify application or libraries
- Easier to change policies

ALTERNATIVES

- Setsockopt
  - Need to modify applications or libraries
  - Policy tied to application/library
- Sysctl
  - Global or per network namespace
  - More difficult to optimize and fine tune
- Ip-route
  - Per flow, but rules are more restrictive
  - Harder to implement global rules

USES

- Optimizing per-flow TCP parameters
  - Within organization (enable internally available features)
  - External traffic (INIT_CWND, etc.)
  - Intra-DC traffic
  - WAN traffic
- Experimenting
  - Test INIT_CWND values per IP prefix

OVERVIEW

- TCP-BPF is a new BPF type program
- Unlike most other BPF programs, it is called from many places. Uses *op* field to specify
  - Desired value
    - BPF_SOCK_OPS_TIMEOUT_INIT
    - BPF_SOCK_OPS_RWND_INIT
    - BPF_SOCK_OPS_NEEDS_ECN
    - BPF_SOCK_OPS_BASE_RTT

OVERVIEW (2)

- *op* field can also specify
  - Connection state or code place
    - BPF_SOCK_OPS_TCP_CONNECT_CB
      - Set snd and rcv buffer sizes
    - BPF_SOCK_OPS_ACTIVE_ESTABLISHED_CB
      - Set cwnd clamp, congestion algorithm,
    - BPF_SOCK_OPS_PASSIVE_ESTABLISHED_CB
      - Set snd and rcv buffer sizes, cwnd clamp, congestion algorithmn

PROGRAM VIEW

```
struct bpf_sock_ops {
    __u32 op;
    union {
        __u32 reply;
        __u32 replylong[4];
    };
    __u32 family;
    __u32 remote_ip4;       /* Stored in NBO */
    __u32 local_ip4;        /* Stored in NBO */
    __u32 remote_ip6[4];    /* Stored in NBO */
    __u32 local_ip6[4];     /* Stored in NBO */
    __u32 remote_port;      /* Stored in NBO */
    __u32 local_port;       /* stored in HBO */
/* where NBO = Network Byte Order
   and   HBO = Host Byte Order
 */
};
```

KERNEL VIEW

```
struct bpf_sock_ops_kern {
    struct   sock *sk;
       u32     op;
       union {
              u32 reply;
              u32 replylong[4];
       };
};
```

BPF_SETSOCKOPT

- SO_RCVBUF
- SO_SNDBUF
- SO_MAX_PACING_RATE
- SO_PRIORITY
- SO_RCVLOWAT
- SO_MARK
- TCP_CONGESTION
- TCP_BPF_IW*
- TCP_BPF_SNDCWND_CLAMP*

BPF_GETSOCKOPT

- TCP_CONGESTION

USAGE

- Creating a cgroupv2 and putting shell into it

```
mkdir -p /tmp/cgroupv2
mount -t cgroup2 none /tmp/cgroupv2
mkdir -p /tmp/cgroupv2/foo
bash
echo $$ >> /tmp/cgroupv2/foo/cgroup.procs
```

USAGE (2)

- To load a TCP-BPF program
  - `load_sock_ops [-l] <cgroupv2> <tcp-bpf program>`

- Example:
  - `load_sock_ops –l /tmp/cgroupv2/foo tcp_iw_kern.o`

- **`–l`** flag keeps program running and printing bpf buffer

- To remove/unload: `load_sock_ops -r <cgroupv2>`

EXAMPLE: TUNING FOR INTRA-DC

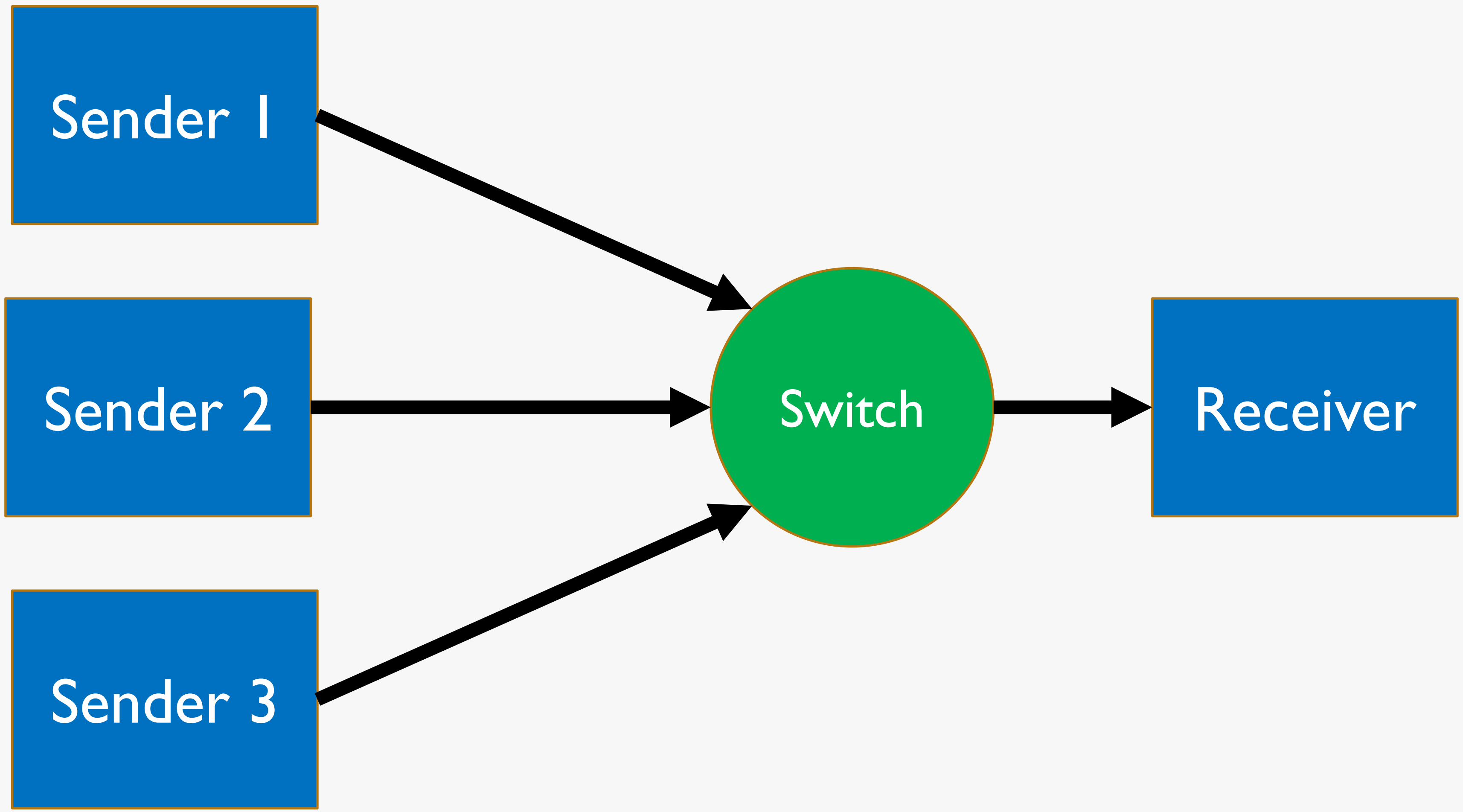
```
SEC("sockops")
int bpf_clamp(struct bpf_sock_ops *skops
{
    int bufsize = 150000;
    int to_init = 10;
    int clamp = 100;
    int rv = 0;
    int op;

    /* Check that both hosts are within same datacenter. For this example
     * it is the case when the first 5.5 bytes of their IPv6 addresses are
     * the same.
     */
    if (skops->family == AF_INET6 && skops->local_ip6[0] == skops->remote_ip6[0] &&
        (bpf_ntohl(skops->local_ip6[1]) & 0xfff00000) ==
        (bpf_ntohl(skops->remote_ip6[1]) & 0xfff00000)) {
```

EXAMPLE (2)

```
        switch (op) {
        case BPF_SOCK_OPS_TIMEOUT_INIT:
            rv = to_init;
            break;
        case BPF_SOCK_OPS_TCP_CONNECT_CB:
            /* Set sndbuf and rcvbuf of active connections */
            rv = bpf_setsockopt(skops, SOL_SOCKET, SO_SNDBUF, &bufsize,
                                sizeof(bufsize));
            rv = rv + bpf_setsockopt(skops, SOL_SOCKET, SO_RCVBUF, &bufsize,
                                         sizeof(bufsize));
            break;
        case BPF_SOCK_OPS_ACTIVE_ESTABLISHED_CB:
            rv = bpf_setsockopt(skops, SOL_TCP, TCP_BPF_SNDCWND_CLAMP, &clamp,
                                sizeof(clamp));
            break;
        case BPF_SOCK_OPS_PASSIVE_ESTABLISHED_CB:
            /* Set cwnd clamp and sndbuf, rcvbuf of passive connections */
            …
        default:
            rv = -1;
    } else { rv = -1; }
    skops->reply = rv;
    return 1;
}
```

DATACENTER EXPERIMENT

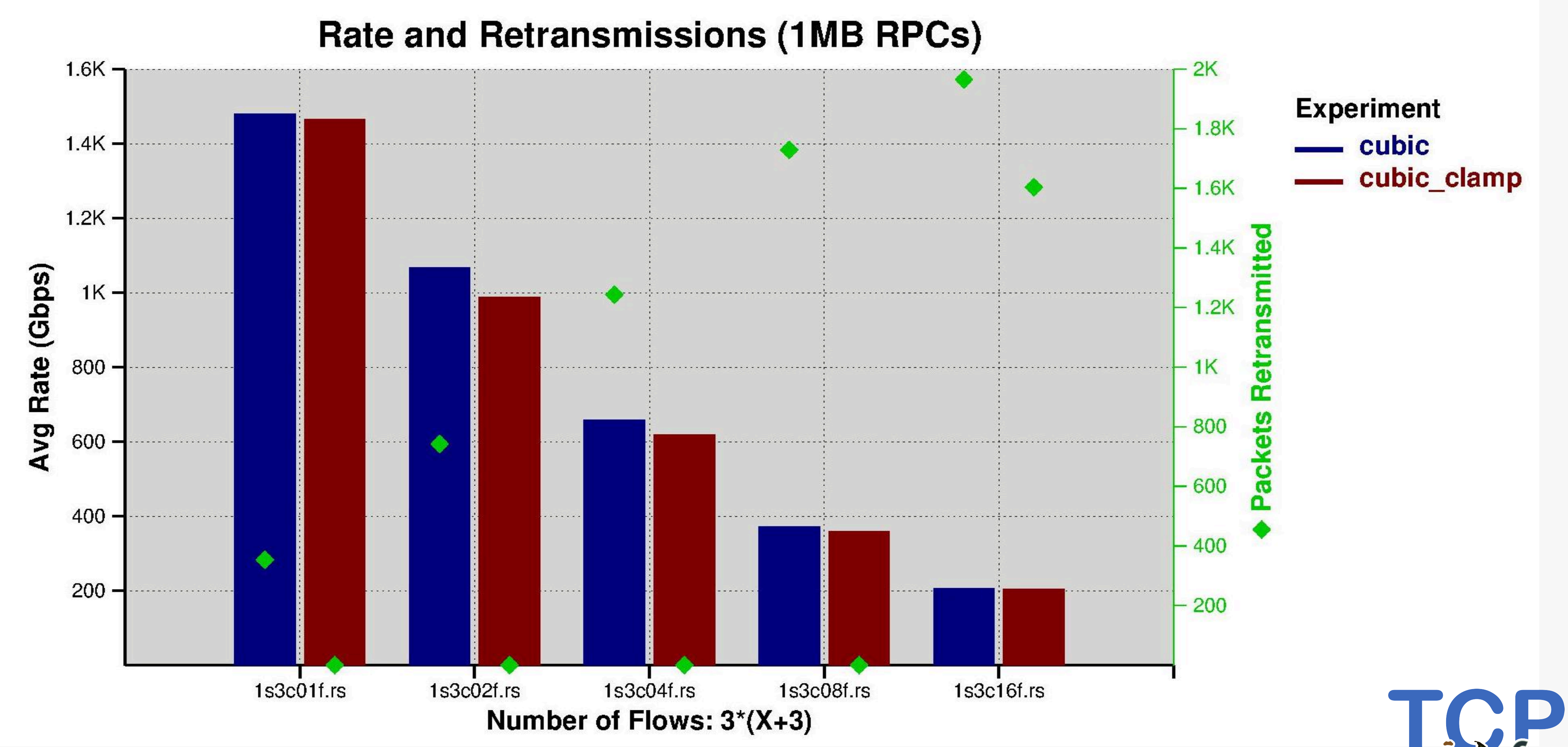


Flows per sender:

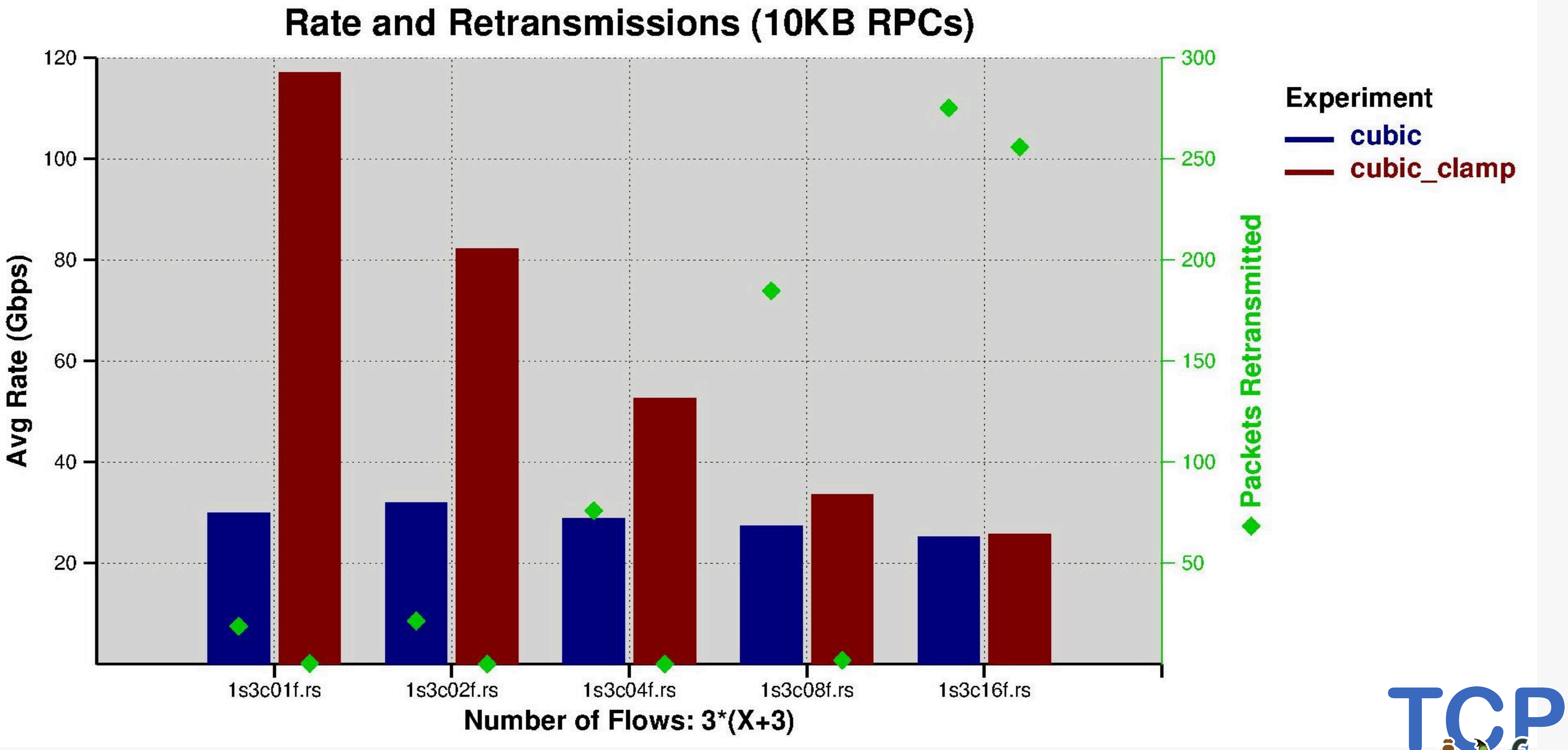
X 1MB RPCs
2 10KB RPCs
1 Stream

X in {1, 2, 4, 8, 16}

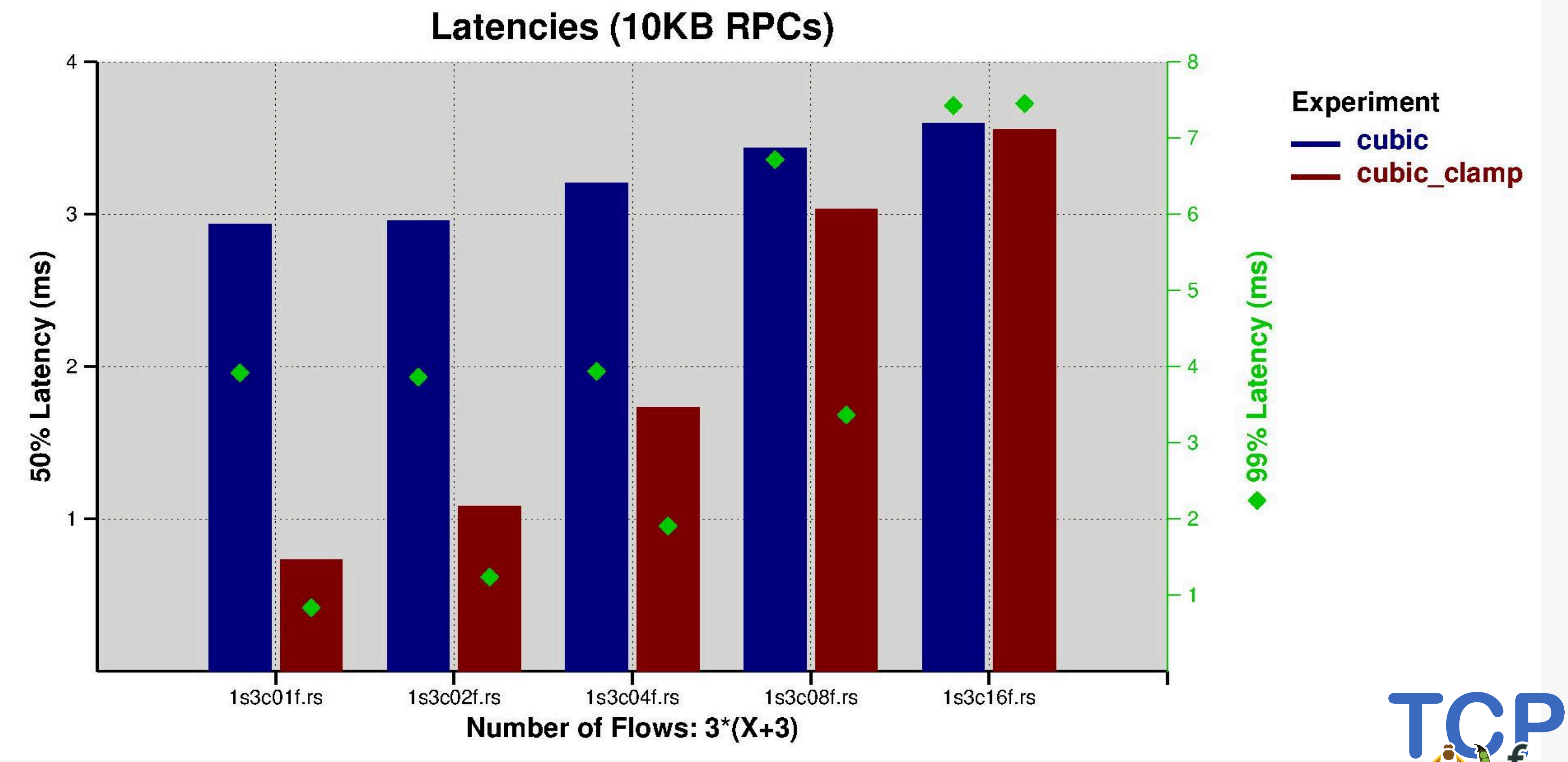
1MB RPC: RATE AND RETRANSMISSIONS



10KB RPC: RATE AND RETRANSMISSIONS



10KB: LATENCIES



EXPERIMENT (3)

- Most of the benefit from clamping cwnd
- Can do other ways (ip-route)
- But not as convenient
- Will be able to do a lot more

NEXT STEPS

MAKE MORE STATE AVAILABLE TO TCP-BPF PROGRAM

- (R or R/W) either directly of using get/set sockops
- TOS / TCLASS
  - Direct DCTCP or NV to separate switch queue
  - Mark higher priority traffic
- Flowlabels
  - More interesting algorithms for changing due to retrans
- Cwnd, ssthresh, RTT, …
  - Using values to make decisions
  - Collecting info for experiments (init cwnd per subnet)

ADD MORE ENTRY/CALL POINTS

- RTO, retransmits – lower rate of calls (unless using BBR)
- New minRTT
- Packets received or sent – high rate
- Use bitmap to enable/disable calling TCP-BPF program per potentially expensive call point
- Normally disabled, can be enabled when connection is established
  - Based on flow information (port, IP addresses), statistically (0.01% of flows), etc.
  - For experiments

ADD SUPPORT FOR NEW TCP HEADER OPTIONS

- Support "local" options
- No issues with patches we cannot upstream
  - Assumes we can upstream mechanisms
- Support sharing experimental options

BPF BASED CONGESTION ALGORITHM

- Support testing and development
  - Easier to test changes, only need new enough kernel
  - No need to worry with modules for each kernel version
- New CA that calls BPF program
- Probably not TCP-BPF based

FINAL THOUGHTS

- TCP-BPF is available starting at kernel version 4.13
- Thanks to
  - Alexei Starovoitov
  - Daniel Borkmann

TCP