Safe sharing of the network with eBPF

Presenters:

Balasubramanian Madhavan <bmadhavan@meta.com>

Prankur Gupta <prankgup@meta.com>

Linux Plumbers Conference

Richmond, USA Nov 2023

This talk

- Network health
  - Problems in Datacenter networking space (aka. small RTT flows)
  - Receiver Window Tuning
  - Various tuning opportunities

- Custom congestion control algo with eBPF

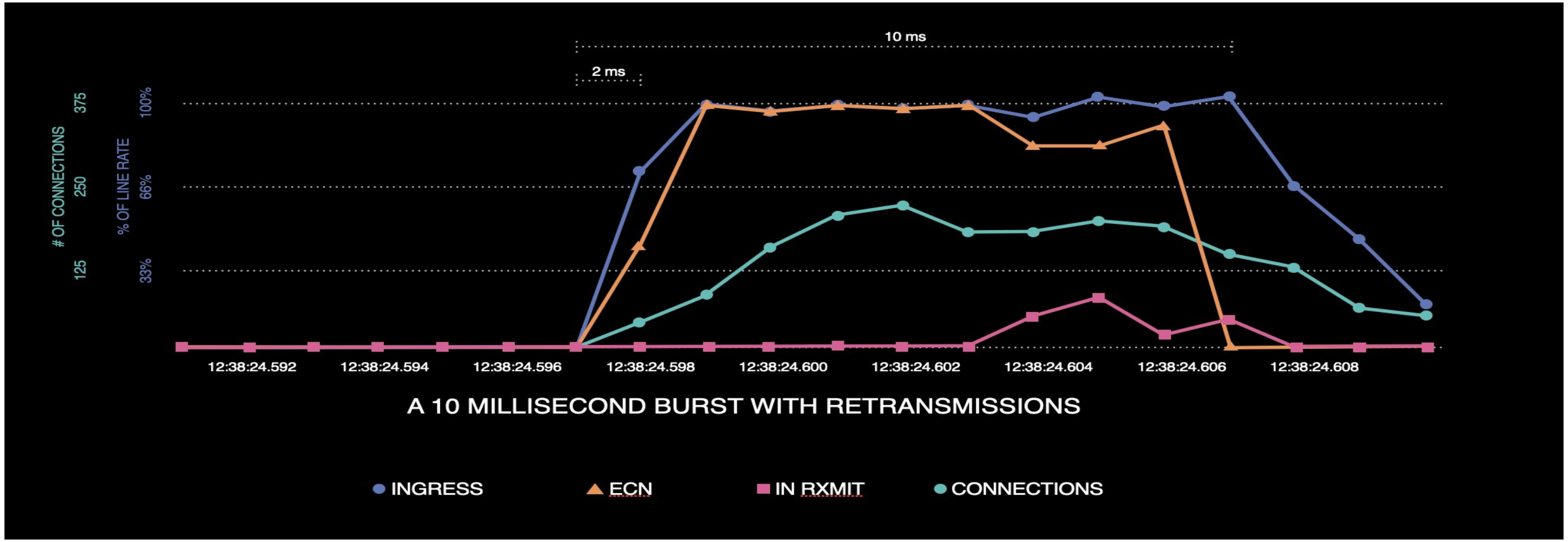
- Control Plane for BPF Management
  - Managing multiple features (10+) utilizing 7 different attach points (sockops, TC etc)
  - Challenges and Enhancements

Who we are

Combined & continuous work by transport teams at Meta

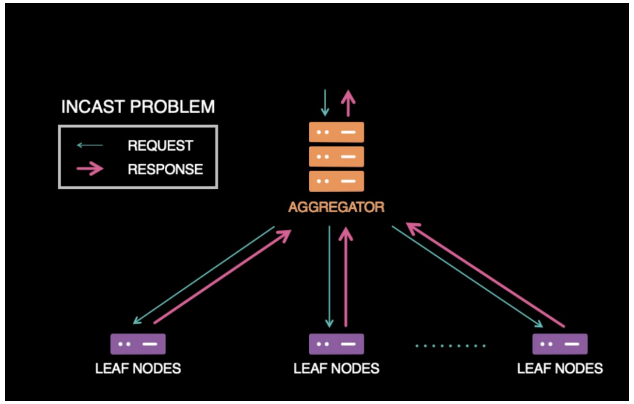
- Prashanth Kannan (Host Networking)
- Balasubramanian Madhavan (Host Networking)
- Prankur (Host Networking)
- Neil Spring (Network Analytics)
- Srikanth Sundaresan (Network Analytics)
- Martin Lau (Kernel)
- Lawrence Brakmo
- Abhishek Dhamija (Host Networking)
- Jie Ming (Host Networking)
- Miao Xu (Host Networking)
- Chris Canel (Host Networking)
- Jakub Kicinski (Host Networking)
- Tanuja Ingale

Microbursts



Receiver Window Tuning

- Using "flow control" knob in TCP to tune flows within DC
- A TC program implemented using eBPF to intercept packets at the egress and rewrites the advertised windows.
- Complements congestion control from the "receiver side" to limit windows to match BDP.
- Helps bursty services from taking over entire shared rack buffers.
- Improvements:
  - Reduced packet discards at all shared devices like host NICs, rack switches.
  - ~50% reduction in switch buffer utilization.
  - ~30% improvement in sRTT.



More details on this tuning @ NetworkingAtScale22 talk.

Need for BPF ? Why not just use setsockopt()

Indeed the `window clamp` is a socket option, changeable with setsockopt().

1) Works only for “passive open”, but not for active open cases.

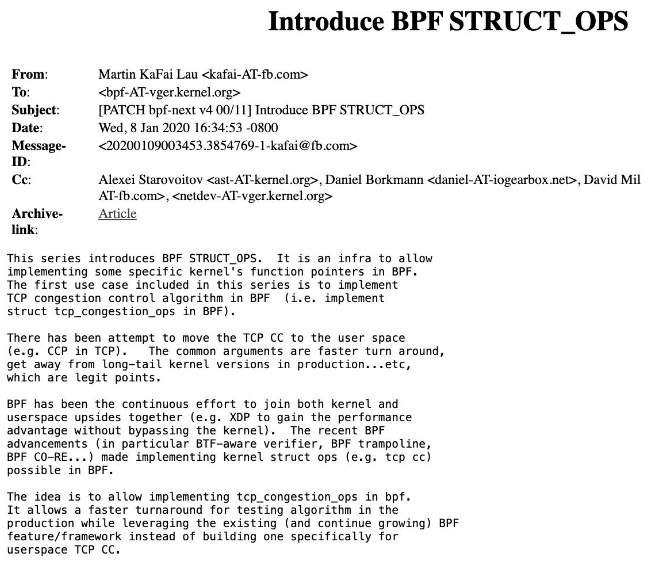
2) Can be set only during start. Setting in the middle of a connection did not work. Why ?
   Kernel bug: `rcv_ssthresh` was not set when `window_clamp` was changed.
   Fixed in Patch: **tcp: enable mid stream window clamp**

3) Setting the clamp didn’t stick.
   Why ? “ Kernel’s adaptive receive buffer tuning “
   What needs to be done: Disable `net.ipv4.tcp_moderate_rcvbuf and set` SO_RCVBUF
   NO. But we don’t need to tune the receive buffer.

Using BPF, provides an independent way to manage just the window.

Other tunings

Problem space	**Scope**	**Tuning Type**	**Improvements**	**eBPF program type**
Burst sizes in large BDP flows	Long RTT	Pacing & Max Pacing rate	Reduction in packet loss at hosts by ~50% Improvement in service P99 latency by 2x	sockops
Customize TCP's defaults	Short RTT	Connection timeout	Improve reliability during timeouts.	sockops
	long RTT	Init congestion window	Reduction in x-region avg. query latency by 20%	sockops
CC selector	Short RTT Long RTT Best effort	Pick a CC	Ease of Operation (ease of experimentation, upgrades, iterations, bug fixes)	sockops



Introduce BPF STRUCT_OPS

From:	Martin KaFai Lau <kafai-AT-fb.com>
To:	<bpf-AT-vger.kernel.org>
Subject:	[PATCH bpf-next v4 00/11] Introduce BPF STRUCT_OPS
Date:	Wed, 8 Jan 2020 16:34:53 -0800
Message-ID:	<20200109003453.3854769-1-kafai@fb.com>
Cc:	Alexei Starovoitov <ast-AT-kernel.org>, Daniel Borkmann <daniel-AT-iogearbox.net>, David Mil AT-fb.com>, <netdev-AT-vger.kernel.org>
Archive-link:	Article

```
This series introduces BPF STRUCT_OPS.  It is an infra to allow
implementing some specific kernel's function pointers in BPF.
The first use case included in this series is to implement
TCP congestion control algorithm in BPF  (i.e. implement
struct tcp_congestion_ops in BPF).

There has been attempt to move the TCP CC to the user space
(e.g. CCP in TCP).   The common arguments are faster turn around,
get away from long-tail kernel versions in production...etc,
which are legit points.

BPF has been the continuous effort to join both kernel and
userspace upsides together (e.g. XDP to gain the performance
advantage without bypassing the kernel).  The recent BPF
advancements (in particular BTF-aware verifier, BPF trampoline,
BPF CO-RE...) made implementing kernel struct ops (e.g. tcp cc)
possible in BPF.

The idea is to allow implementing tcp_congestion_ops in bpf.
It allows a faster turnaround for testing algorithm in the
production while leveraging the existing (and continue growing) BPF
feature/framework instead of building one specifically for
userspace TCP CC.
```

https://lwn.net/Articles/809092/

Why customize congestion control ?

- Ability to add new congestion signals (ECN, different delay signals)
- Improve existing algorithm (add transient burst handling, ability to differentiate host & fabric level congestion)
- Add monitoring within CC for better debuggability
- Perform advanced operations like parse custom headers

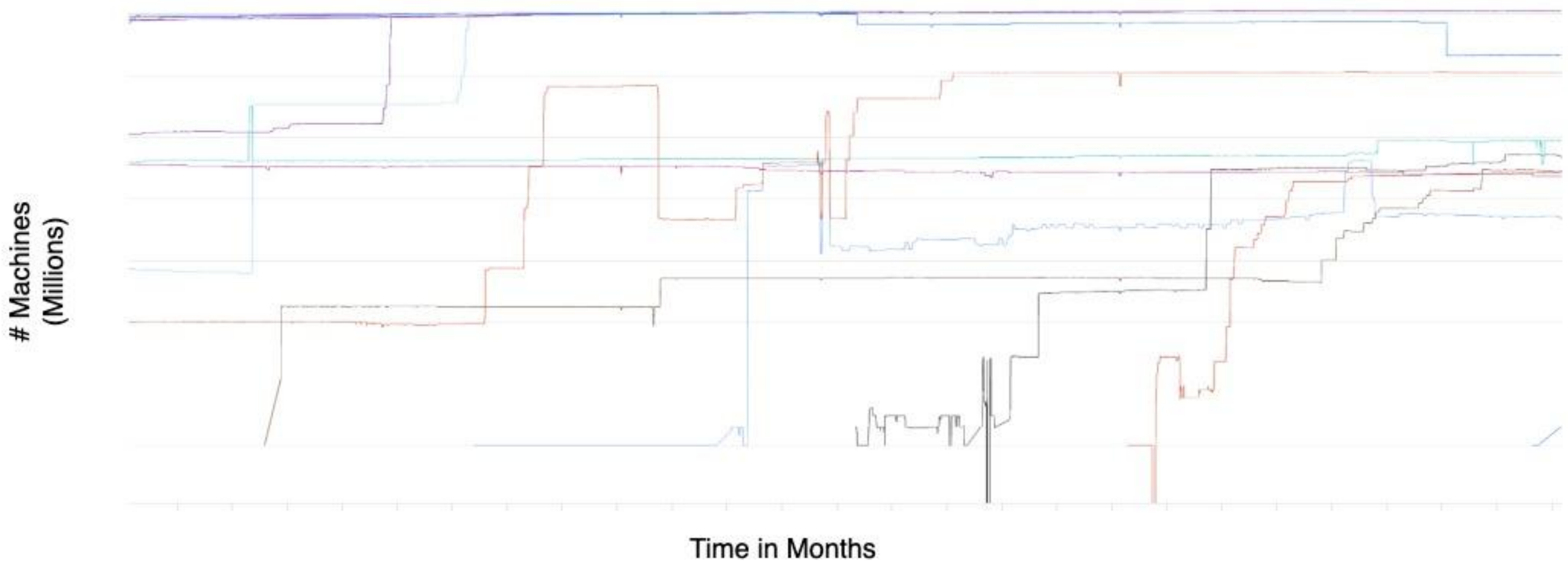
A delay based CC within BPF

- Aim: Target delay i.e. Expected delay without congestion
- Signals (1)
  - RTT
  - One way Queueing Delay
  - Backward compatibility with ECN
- Custom loss & congestion handling (3)

```
BPF_STRUCT_NAME(CC_PROGRAM_NAME) = {
………….
.pkts_acked = (void*)pkts_acked_cc_impl,    ------> (1) Collect delay samples & determine the the action (CWND inc or dec)
.cong_avoid = (void*)cong_avoid_cc_impl,    ------> React based on the observed delay signal.
.cwnd_event = (void*)cwnd_event_cc_impl,    ------> (3)
.set_state = (void*)set_state_cc_impl,      ------> (3)
.ssthresh = (void*)ssthresh_cc_impl,
………
};
```

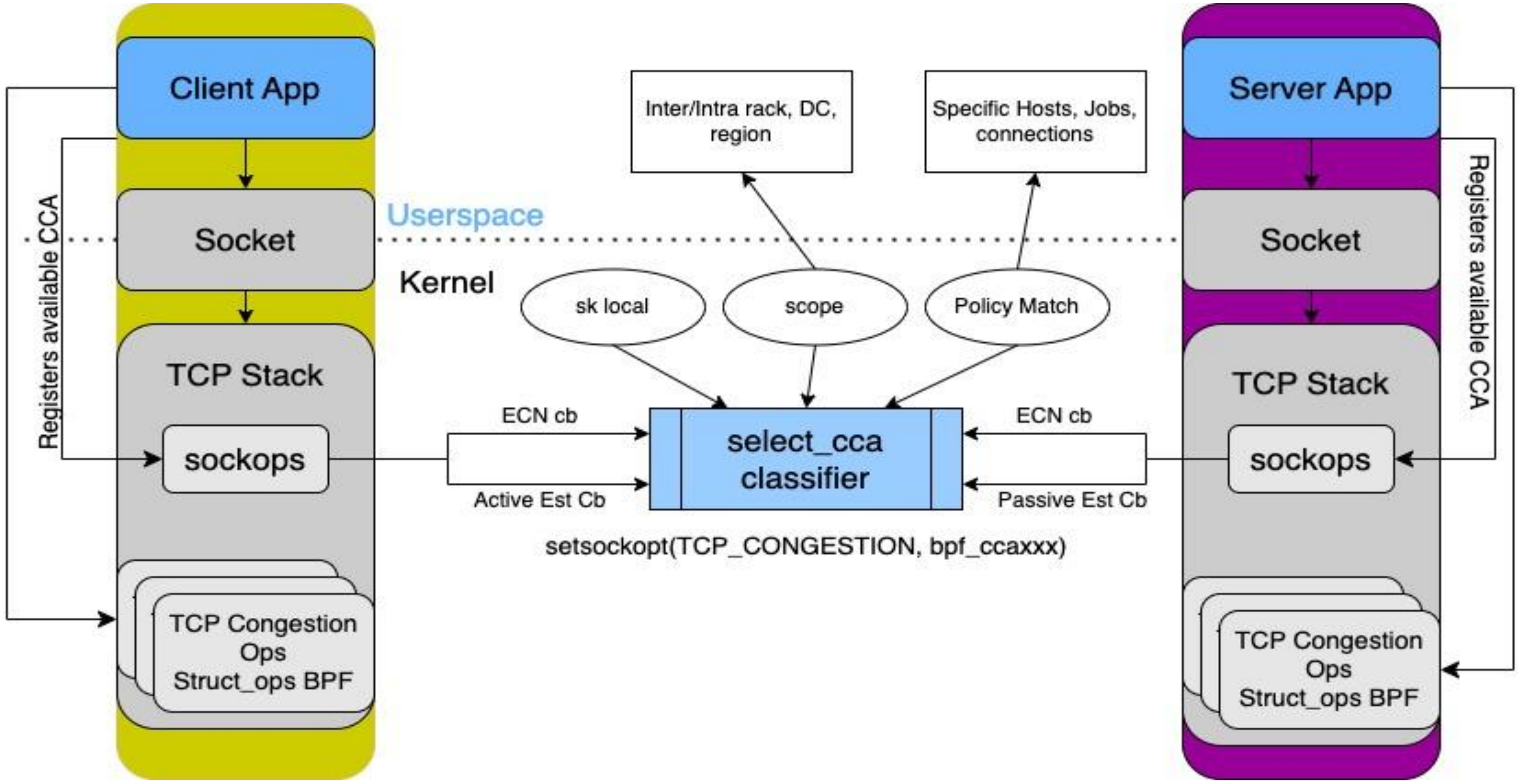
NetEdit: network eBPF control plane

10+ features that tune transport stack with varying degree of rollout that are loosely coupled with services
Need common observability and error detection infrastructure across features.



Control plane for BPF Management

- Why?
  - Fast A/B tests
  - Allows for granular configuration of features
  - Facilitates rollout and rollback in just minutes
  - Developer velocity: hide BPF attach-point + kernel version dependencies
  - Efficiency: process once, re-use across BPF programs with socket cache
  - Reliability: split user space/kernel lifetime management with always on-BPF dataplane
  - Easy to adopt latest BPF features such as btf, skel, and sklocal


Client App
Socket
TCP Stack
sockops
TCP Congestion Ops Struct_ops BPF
Registers available CCA
Userspace
Kernel
Inter/Intra rack, DC, region
Specific Hosts, Jobs, connections
sk local
scope
Policy Match
select_cca classifier
ECN cb
Active Est Cb
ECN cb
Passive Est Cb
setsockopt(TCP_CONGESTION, bpf_ccaxxx)
Server App
Socket
TCP Stack
sockops
TCP Congestion Ops Struct_ops BPF
Registers available CCA

Observations in Prod

- Long Lived Connections
- Control Plane reboot/downtime
- Sharing of pinned objects

Long lived Connections

- Tunings and A-B tests need control over when connections are migrated
- Reduce variability - How to test a network change for every connection without forcing all the services to restart? can different versions of CCA co-exist ?
- Sometimes connections lifetime can be in order of days to weeks
- tcp/iter to rescue - migrate all connections to one version of CCA

Control plane reboot/downtime

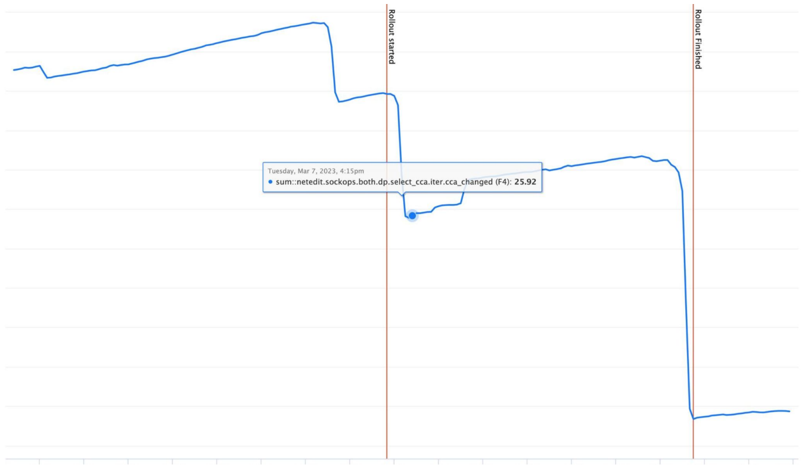
- Services will continue to create new connections while control plane is restarting

- Network tunings will be skipped

- tc/iter can help but not always
  Ex: Settings set during start of the connection like ECN negotiation

- Solution: Always on dataplane*

- Pinning bpf programs/maps/links



6% new conns when Control plane was down conformed to NetEdit features

* Although without control plane policies will be stale but for a small window which is fine.

Sharing/Upgrading Pinned Objects

- Updating pinned programs and maps without any downtime
  Atomic upgrades not applicable for all attach points
- Migrating state from old maps to new maps
- What if the map or program structure changed in new version
- Access control (multiple writer/reader) for shared maps across processes
- Solution: BPF versioning manager