Making the Linux TCP stack more extensible with eBPF

Viet-Hoang Tran, Olivier Bonaventure
(INL, UCLouvain)

Supporting new TCP option

The standard way to extend TCP

But implementation?

requires kernel changes

Supporting new TCP option is hard

True for just experiment

More with deployment: upstreaming patches?

Stand on the shoulders of giants...

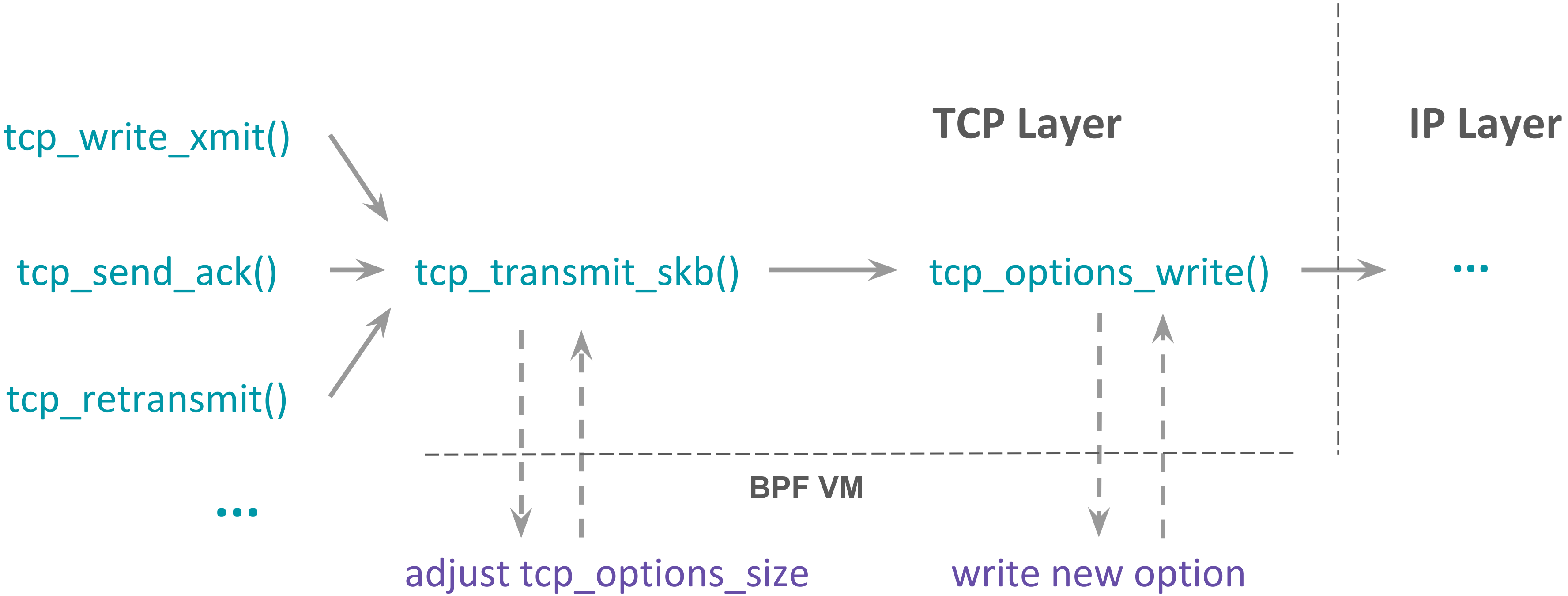
Based on TCP-BPF by Lawrence Brakmo

TCP-BPF (since 4.13) already has:

- Hooks at different phases of a TCP connection

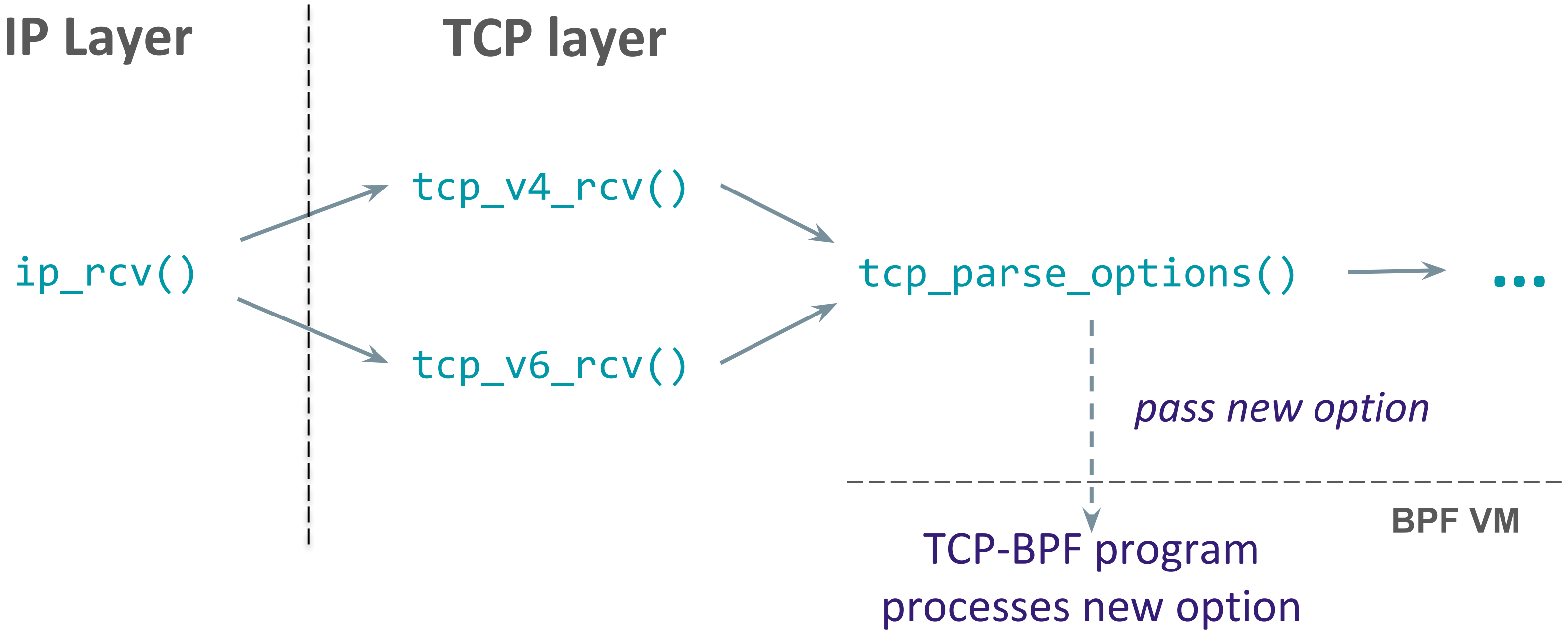
  or when connection state changes
- Read & write to many fields of tcp_sock
- Indirect access with bpf_getsockopt, bpf_setsockopt
- ...

Add new option: 2 steps



One more thing: update current MSS

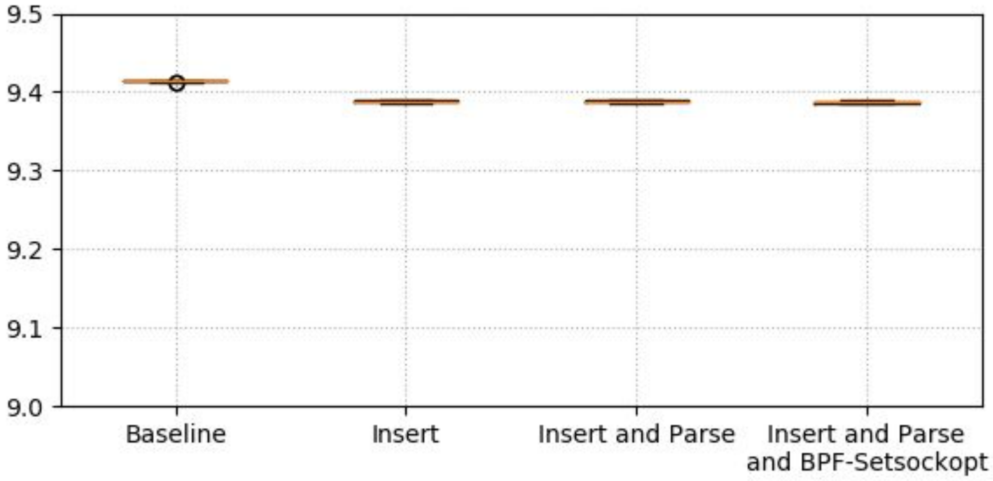
Parse new option



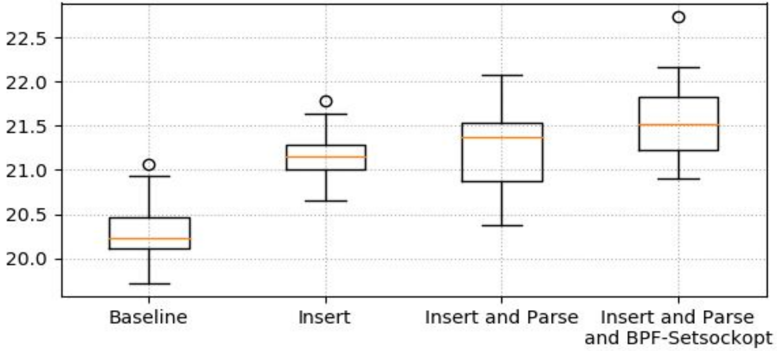
Overhead

Disable hooks by default

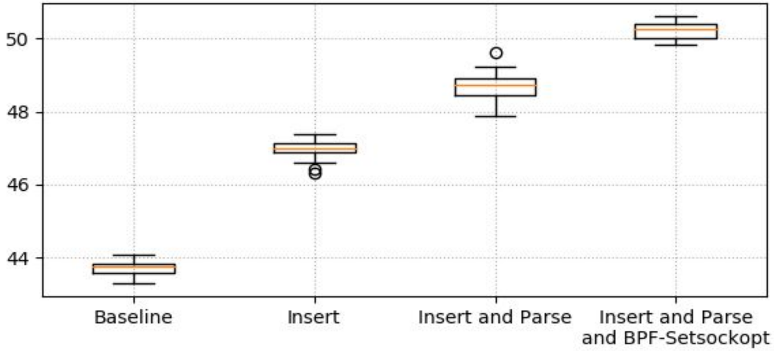
- iperf3 transfer over 10 Gbps link
- trigger on every packet



Average Throughput (Gbps)



Sender's CPU usage (%)

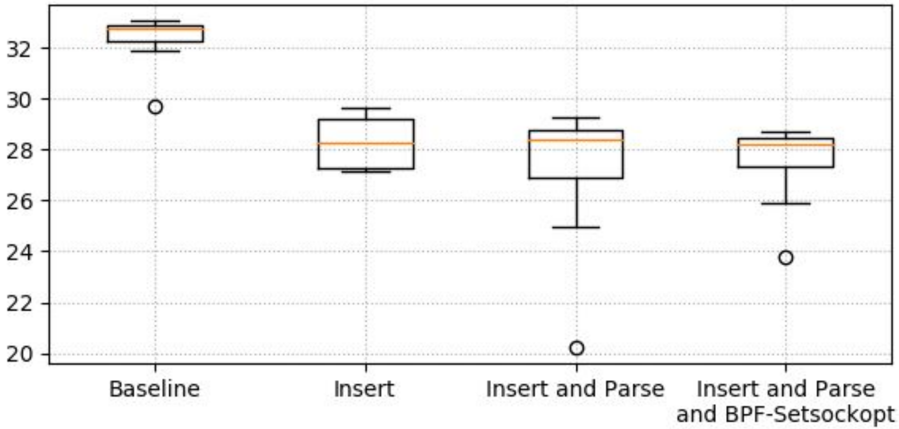


Receiver's CPU usage (%)

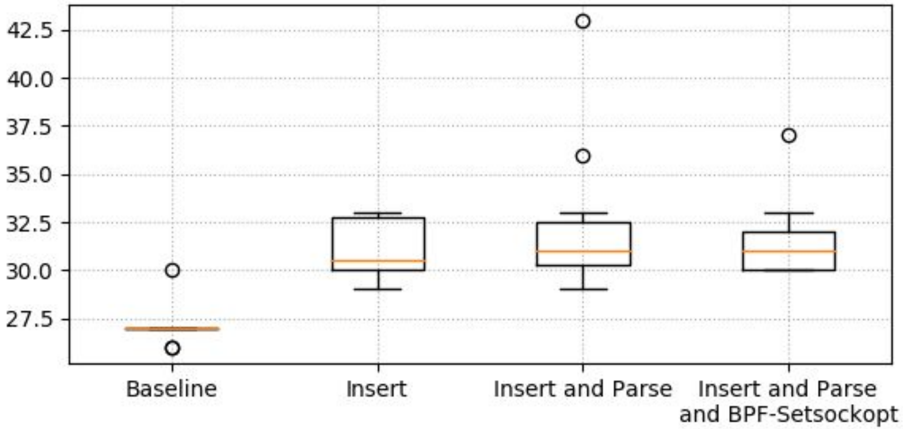
Extreme (and unrealistic) benchmark

over loopback interface

trigger on every packet



Average Throughput (Gbps)



RTT (usecs)

Use cases

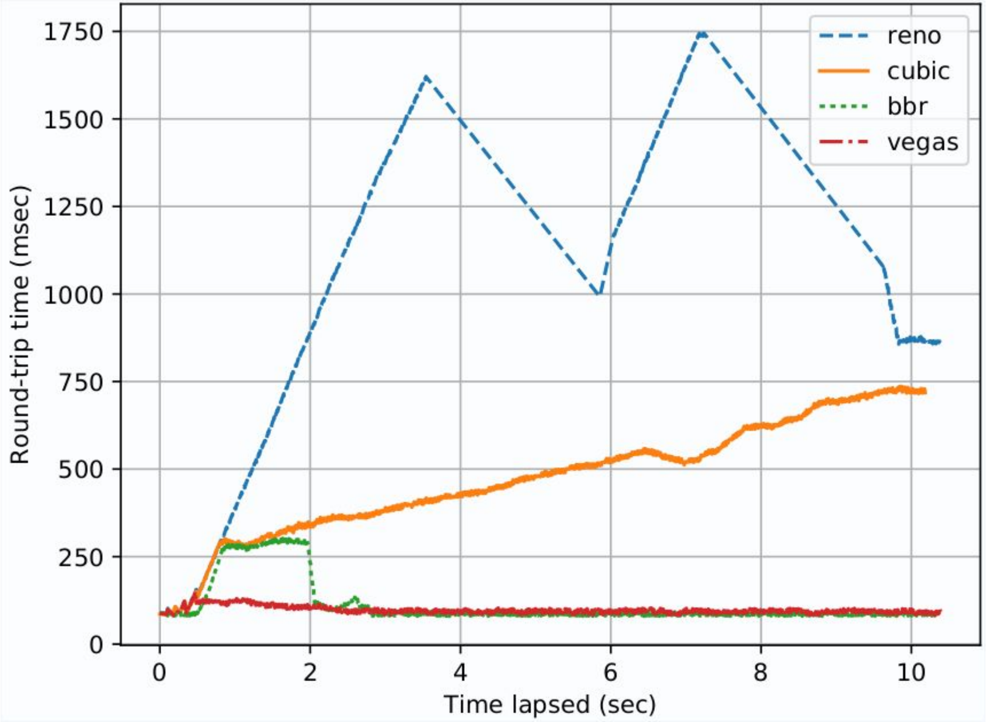
User Timeout Option

TCP User Timeout (UTO):

max time waiting for the ACK of transmitted data
before resetting the connection

RFC 5482: TCP option to announce/request this value

Congestion Control Request Option

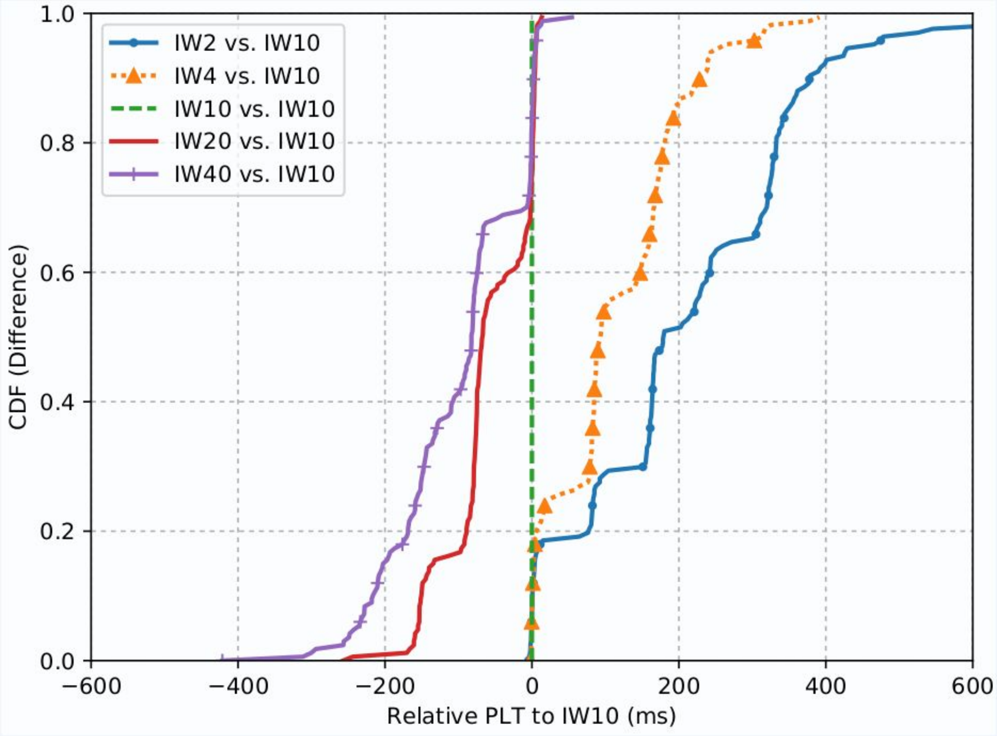


Receiver requests the sender to use a desired CC algorithm for the connection

E.g. Clients prefer low latency over throughput

Two sides shared the list of CC beforehand

Initial CWND option



When the receivers know more about the network bottleneck.

Delayed ACK Option

Motivation: Too many ACKs or too few ACKs is not good.

→ The need to know remote’s ACK delay strategy
… or to request the desired configuration

This option carries two values:

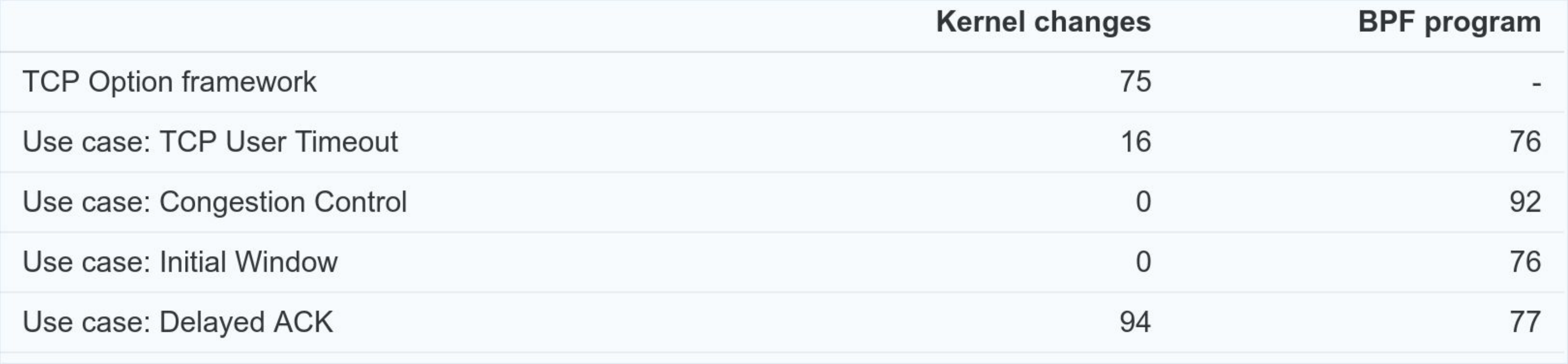
Delack timeout: relatively as a fraction of RTT
Segs count: Number of received segs before sending an ACK

What about the middleboxes?

RFC 6994: “Shared Use of Experimental TCP Options”
(PROPOSED STANDARD)

Network operators “should” support (or fix it otherwise)

Code Status



	Kernel changes	BPF program
TCP Option framework	75	-
Use case: TCP User Timeout	16	76
Use case: Congestion Control	0	92
Use case: Initial Window	0	76
Use case: Delayed ACK	94	77

Caveats

- Option size <= 4 Bytes, extensible to 16 Bytes
- Decouple from cgroup-v2?

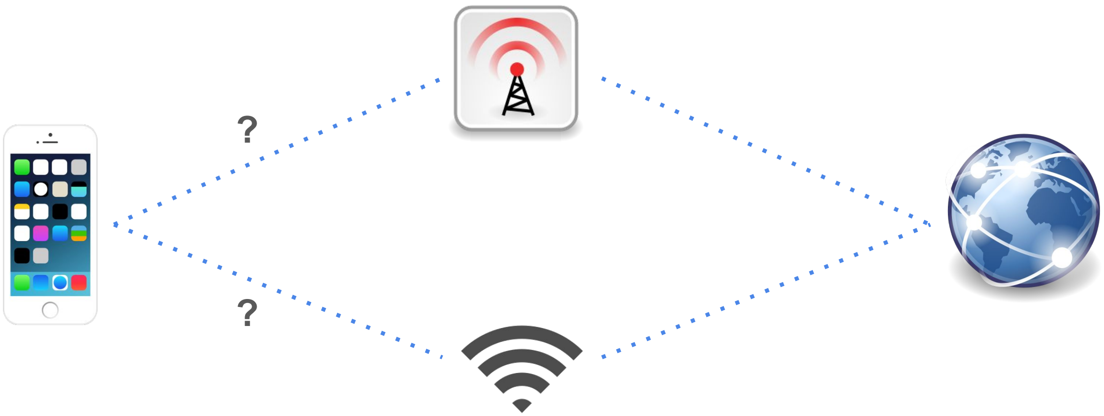
Making the Linux TCP stack more extensible with eBPF

Making the Linux MPTCP stack more extensible with eBPF

Path Manager

Which path to create/remove? Which address to announce?

→ Should be controlled by application / user



Slide from Netdev0x12.
Smartphone and WiFi icons by Blurred203 and Antü Plasma under CC-by-sa, others from Tango project, public domain

Supporting user-defined Path Managers (PM)

Netlink-based PM framework

- + Available in mptcp-trunk branch (out-of-tree)
- + Control plane in uspace
- + Clean layering

Issues:

- Under high load, netlink messages may be lost
- Need separated facilities to support:
  - set/getsockopt (e.g. access subflow-level info)
  - TCP state change notification
  - policy to refuse the establishment of a subflow

What if eBPF-based approach

+ Performance
+ Built-in support for TCP state tracking
+ Easy to apply custom policy on subflow establishment
- Restricted by current eBPF limits
- Less layering separation?
- BPF program can be called from different contexts → Locking is trickier

Our prototype

To track events:	New TCP-BPF callbacks
To store local/remote addresses and subflows:	BPF maps
To open a subflow:	helper function

New TCP-BPF callbacks to track events

No more than 3 arguments

- MPTCP Session created
- MPTCP Session established
- MPTCP Session closed (e.g. fallback to regular TCP)
- Subflow established
- Subflow closed
- Remote IP address added/removed

Extend TCP-BPF context

Extend struct `bpf_sock_ops` with mirrored fields from struct sock:

mptcp_loc_token

mptcp_rem_token

mptcp_loc_key

mptcp_rem_key

mptcp_flags

Open subflows

via helper function: `mptcp_open_subflow`()

- (bpf_sock, srcIP+port, dstIP+port) as input
- if a field of tuple is unset: use existing or kernel-assigned IP/port
- extract meta_sk and other mptcp info from bpf_sock

But usually, we are in softirq context: cannot open subflow directly

→ Schedule into workqueue instead

→ subflow is actually opened later

Examples

Two minimal PMs were implemented as BPF programs:

ndiffports PM: ~20 LoCs

fullmesh PM: ~200 LoCs

Open issues

Handle events of local IP address changed:
Need to send events to each BPF program in each cgroup

Remove subflows: (already done automatically in kernel when receiving a REMOVE_ADDR option)

Store the subflows? or query on-demand?

Dual-stack support: would be similar to bpf_bind()?

Multiple PMs? e.g. each PM per netns

Wrap up

More details in our paper

Git repository: https://github.com/hoang-tranviet/tcp-options-bpf

hoang.tran[.at.]uclouvain.be

Backup slides