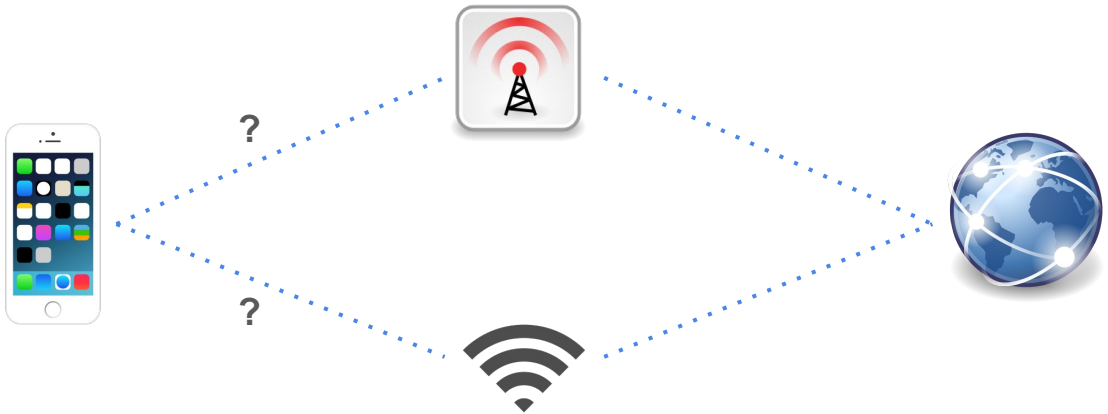
Generic Path-Manager Support with eBPF

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

Path Manager

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

→ Should be controlled by applications / users



Slide from Netdev0x12.

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 about 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: new helper function

Based on TCP-BPF (in mainstream Linux) by Lawrence Brakmo:

- Hooks at different phases of a TCP connection
  or when connection state changes
- Read & write to many fields of tcp_sock

Code status

Kernel changes: ~300 LoCs

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

Conclusion

eBPF makes it easier to extend Linux MPTCP - with good performance

More details in our Netdev 0x13 paper (Section 4):

"Making the Linux TCP stack more extensible with eBPF"

Git repository: https://github.com/hoang-tranviet/tcp-options-bpf,
on branch bpf-pm-v2.2-netdev

Backup slides

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