Multipath TCP Upstreaming

Mat Martineau (Intel) and Matthieu Baerts (Tessares)

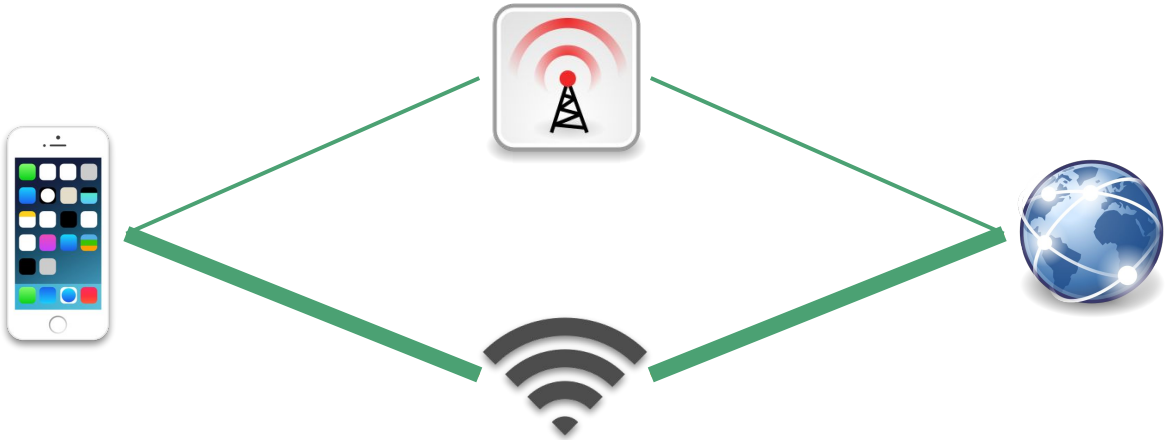
Plan

- Multipath TCP Overview
- First Patch Set Upstreaming Roadmap
- Advanced Features Roadmap
- Conclusion and links

What is MPTCP?

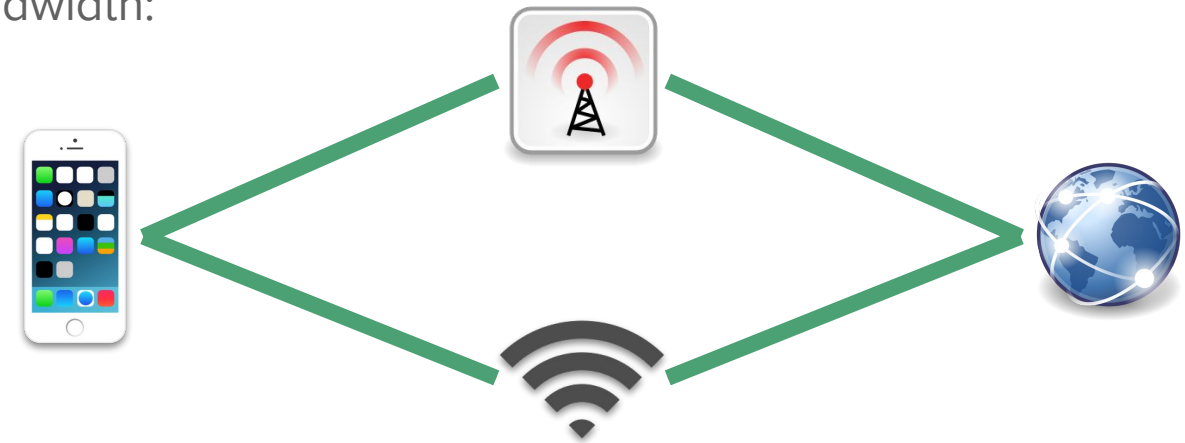
Multipath TCP (MPTCP)

- Exchange data for a single connection over different paths, simultaneously
- RFC-6824 and supported by IETF Multipath TCP (MPTCP) working group



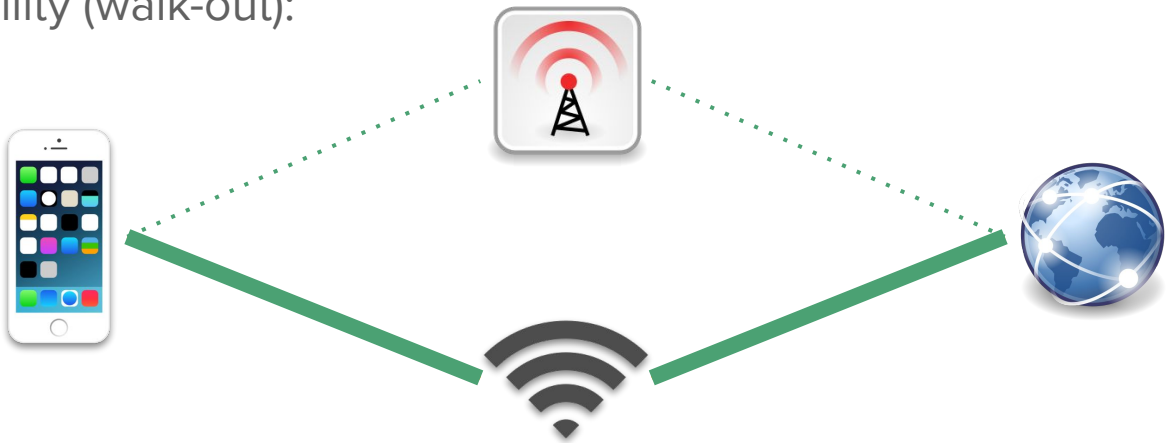
Multipath TCP (MPTCP)

- Exchange data for a single connection over different paths, simultaneously
- RFC-6824 and supported by IETF Multipath TCP (MPTCP) working group
- More bandwidth:



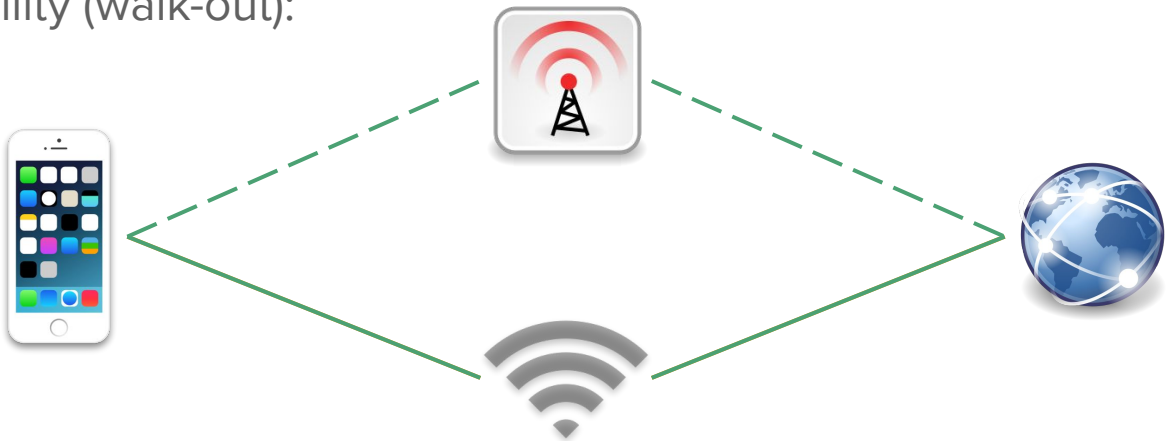
Multipath TCP (MPTCP)

- Exchange data for a single connection over different paths, simultaneously
- RFC-6824 and supported by IETF Multipath TCP (MPTCP) working group
- More mobility (walk-out):



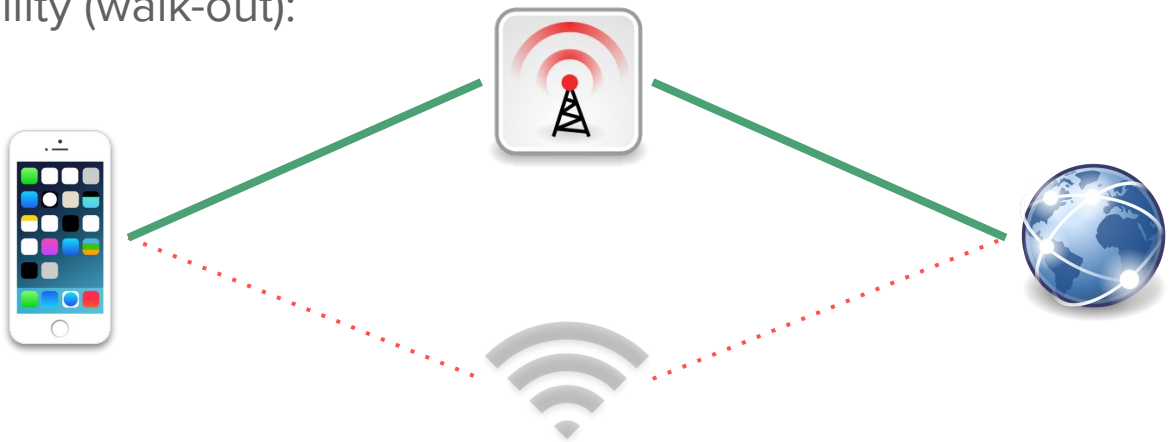
Multipath TCP (MPTCP)

- Exchange data for a single connection over different paths, simultaneously
- RFC-6824 and supported by IETF Multipath TCP (MPTCP) working group
- More mobility (walk-out):



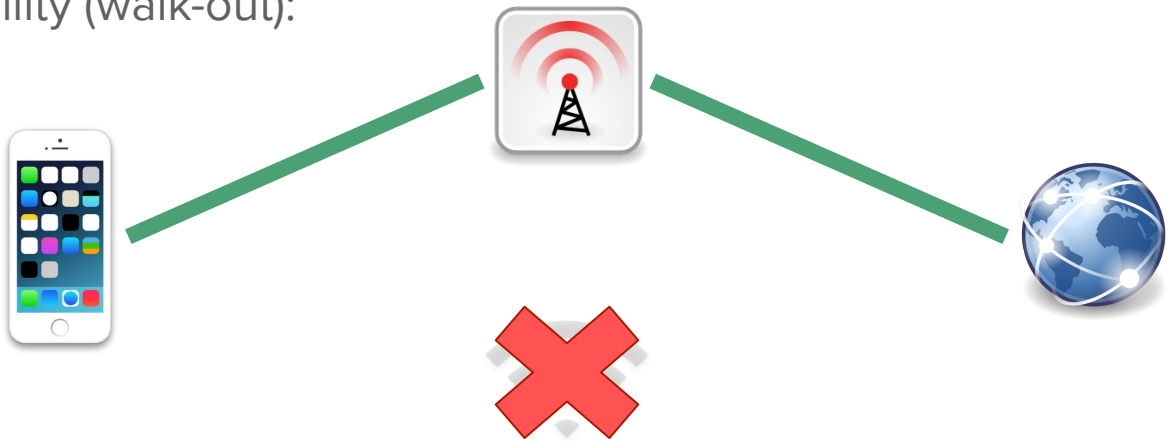
Multipath TCP (MPTCP)

- Exchange data for a single connection over different paths, simultaneously
- RFC-6824 and supported by IETF Multipath TCP (MPTCP) working group
- More mobility (walk-out):



Multipath TCP (MPTCP)

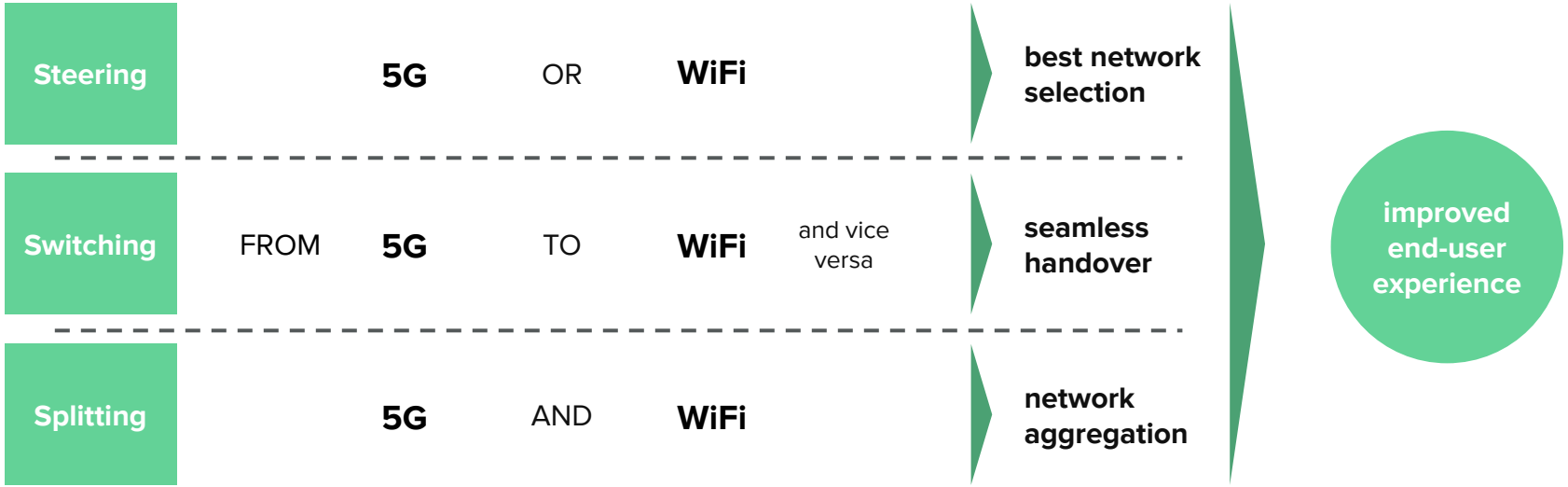
- Exchange data for a single connection over different paths, simultaneously
- RFC-6824 and supported by IETF Multipath TCP (MPTCP) working group
- More mobility (walk-out):



Multipath TCP Use Cases

- Smartphones (Apple, Samsung, LG, others)
  - Support failover / “walk-out” scenario.
  - More Bandwidth
- Residential Gateways (LTE + DSL, for example)
  - More Bandwidth
- Multipath TCP is part of 5G standardisation:
  - Access Traffic Steering, Switching and Splitting: ATSSS

Multipath TCP Use Cases: ATSSS



Defined in 3GPP Release 16, ATSSS is a core network function in 5G networks,
playing a key role in managing data traffic between 3GPP (5G, 4G) networks and non-3GPP (Wi-Fi) networks

3GPP A GLOBAL INITIATIVE

Existing Linux implementation

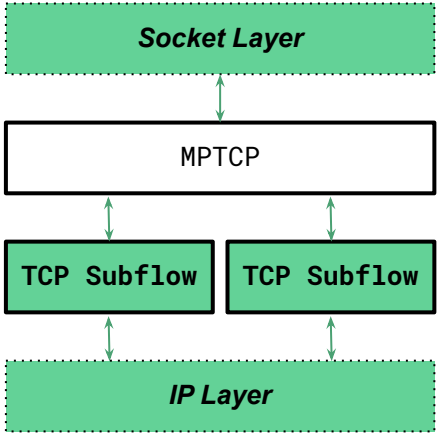
- First implementation for Linux kernel in March 2009
  - Latest MPTCP out-of-tree Linux kernel version is v0.95
  - Generally used as a client / server in current deployments, for millions of users
- But not upstreamable
  - Built to support experiments and rapid changes but not generic enough
  - Special purpose implementation of MPTCP

Guidelines for upstream

- New implementation cannot affect existing TCP stack:
  - Without performance regressions. No code size change if `CONFIG_MPTCP=n`
  - Maintainable and configurable
  - Can be used in a variety of deployments
- Multipath TCP will be "opt-in"
- Proceed in steps:
  - Minimal features set
  - Optimisations and advanced features for later

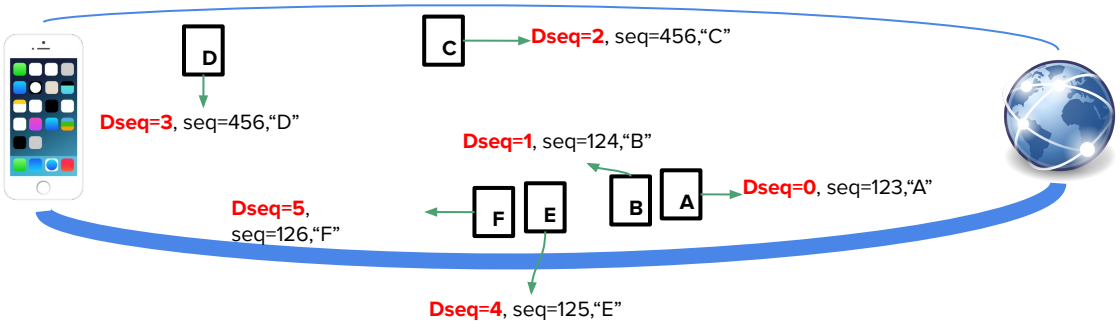
Protocol Overview: RFC 6824

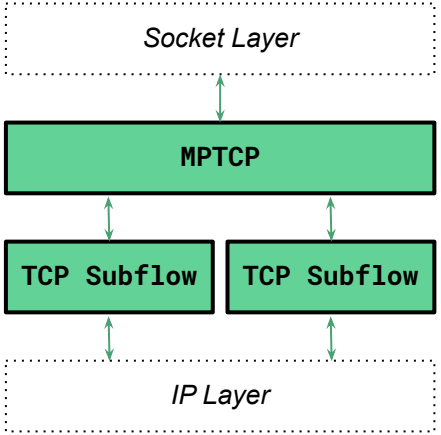
- **Looks like TCP on the wire, similar usage for apps**



Protocol Overview: RFC 6824

- Looks like TCP on the wire, similar usage for apps
- **Subflow mapping: Data Sequence Number**

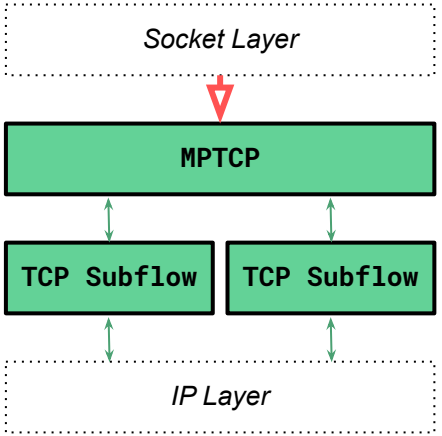




Smartphone icon by Blurred203 under CC-by-sa, others from Tango project, public domain

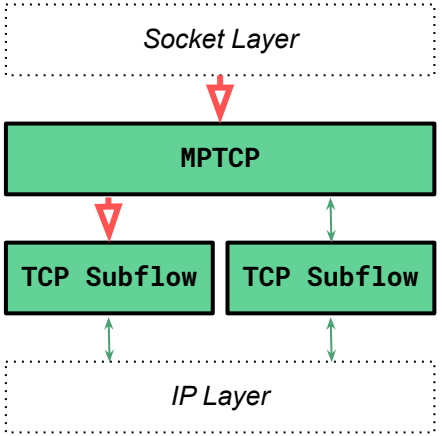
Protocol Overview: RFC 6824

- Looks like TCP on the wire, similar usage for apps
- **Subflow mapping: Data Sequence Number**



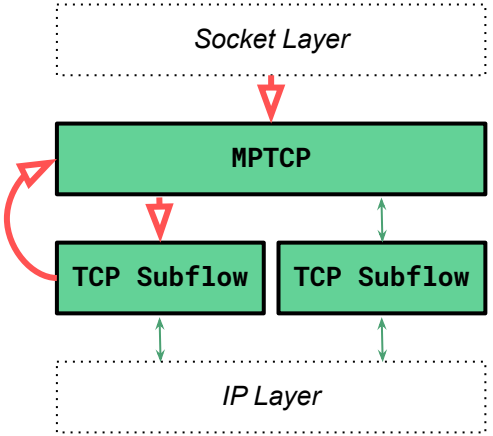
Protocol Overview: RFC 6824

- Looks like TCP on the wire, similar usage for apps
- **Subflow mapping: Data Sequence Number**



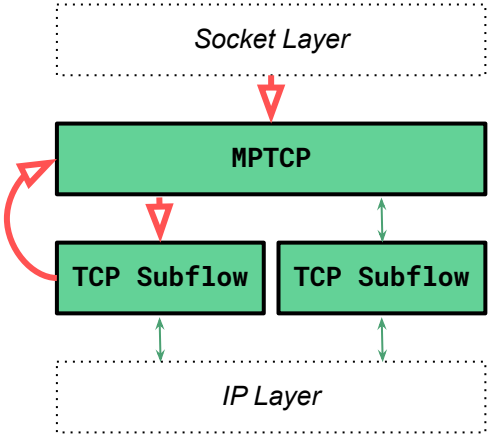
Protocol Overview: RFC 6824

- Looks like TCP on the wire, similar usage for apps
- **Subflow mapping: Data Sequence Number**



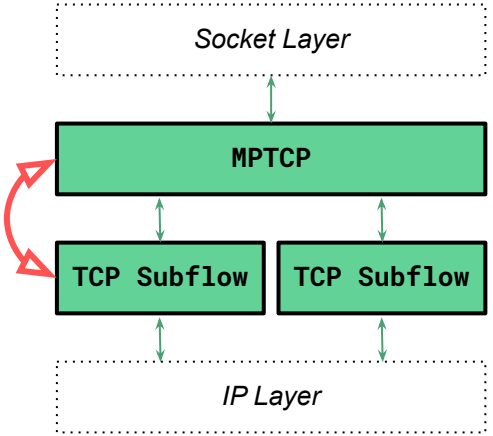
Protocol Overview: RFC 6824

- Looks like TCP on the wire, similar usage for apps
- **Subflow mapping: Data Sequence Number + ACK**



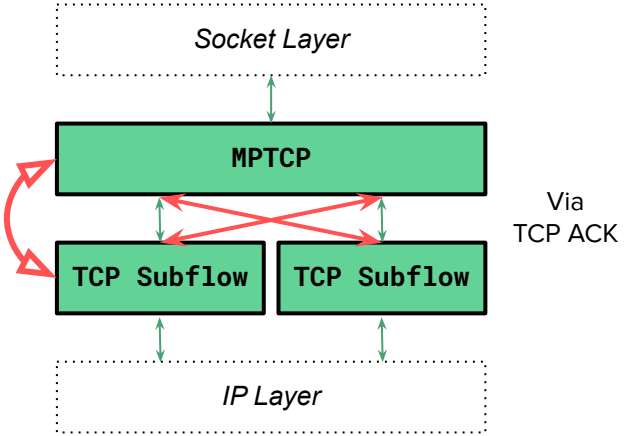
Protocol Overview: RFC 6824

- Looks like TCP on the wire, similar usage for apps
- Subflow mapping: Data Sequence Number + ACK
- **MP_CAPABLE, MP_JOIN, DATA_FIN**



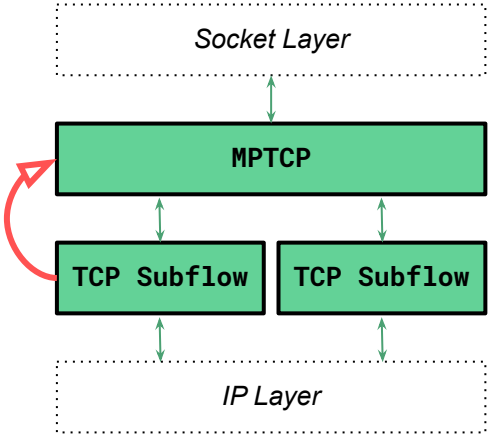
Protocol Overview: RFC 6824

- Looks like TCP on the wire, similar usage for apps
- Subflow mapping: Data Sequence Number + ACK
- MP_CAPABLE, MP_JOIN, DATA_FIN
- **Signaling: Add/Remove Addresses, Fast Close**



Protocol Overview: RFC 6824

- Looks like TCP on the wire, similar usage for apps
- Subflow mapping: Data Sequence Number + ACK
- MP_CAPABLE, MP_JOIN, DATA_FIN
- Signaling: Add/Remove Addresses, Fast Close
- **Coupled receive windows across TCP subflows**

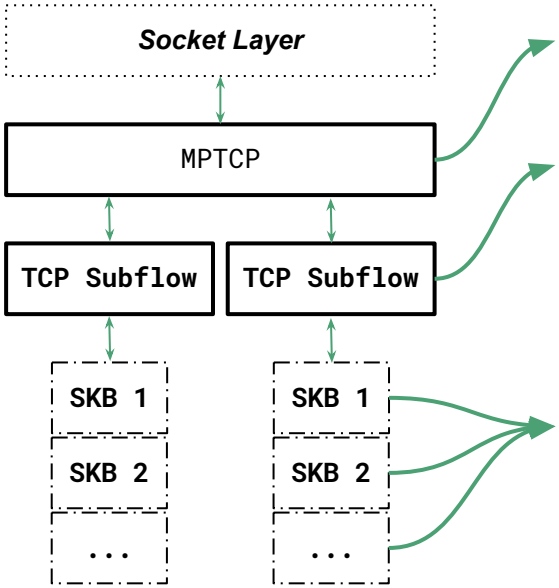


Multiple versions of MPTCP

- RFC 6824: *Experimental*
  - All known implementations support it, only this version
- RFC 6824 bis: *Standard*
  - Submitted to IESG for publication
  - Behavioral changes: MPTCP v0 ➡ MPTCP v1
  - Some parts easier to implement
  - Selected by 3GPP for 5G

First Patch Set Roadmap

MPTCP Socket architecture



IP Proto: `struct proto`

TCP ULP: `struct tcp_ulp_ops`
We start from: `tcp_request_sock_ops`

SKB extension: `struct mptcp_ext`
To store Data Sequence Signal (25 bytes)

Userspace API

- MPTCP selected when creating the socket:

```
socket(AF_INET(6), SOCK_STREAM, IPPROTO_MPTCP);
```

Userspace API

- MPTCP selected when creating the socket:

```
socket(AF_INET(6), SOCK_STREAM, IPPROTO_MPTCP);
```

  - `IPPROTO_MPTCP = IPPROTO_TCP | 0x100; /* = 262 */`

Userspace API

- MPTCP selected when creating the socket:

```
socket(AF_INET(6), SOCK_STREAM, IPPROTO_MPTCP);
```

  - `IPPROTO_MPTCP = IPPROTO_TCP | 0x100; /* = 262 */`

- `getsockopt()` / `setsockopt()` with MPTCP socket or its TCP subflows?

Userspace API

- MPTCP selected when creating the socket:

```
socket(AF_INET(6), SOCK_STREAM, IPPROTO_MPTCP);
```

  - `IPPROTO_MPTCP = IPPROTO_TCP | 0x100; /* = 262 */`

- `getsockopt()` / `setsockopt()` with MPTCP socket or its TCP subflows?

- Security: who can create MPTCP sockets?

  - Initial implementation will not be hardened by broad use yet (syzkaller, etc.)

  - `sysctl` per network namespace, MPTCP disabled by default: is it enough?

Diagnostics

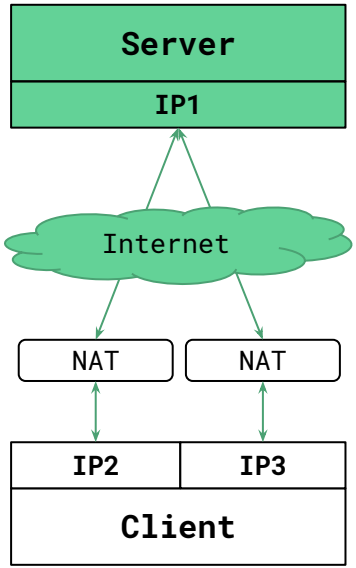
- MPTCP will have a collection of counters for diagnostic and debug purposes
- Per-socket data will be shared with userspace via `sock_diag(7)`
  - TCP ULP framework has been extended to enable diag
- Some TCP counters are also found in `/proc`
  - Should MPTCP add to these as well?

Tests

- Kernel Self Tests
  - Between multiple namespaces (`veth`)
  - MPTCP ⇔ MPTCP, MPTCP ⇔ TCP, TCP ⇔ MPTCP
  - Various conditions including packet loss, reordering, and variations in routing
- Packetdrill
  - Background project ongoing to add MPTCP support
  - Out-of-tree Packetdrill with MPTCP support but old and limited

Initial use case

- Server role is a good place to start
- Simpler path management
  - Client side handles multiple interfaces (like cellular + Wi-Fi)
- Common server configuration uses one public interface for clients
  - Advertising additional interfaces not required
- Client features all build on what's needed for servers



Code already merged upstream

- SKB extensions
  - Needed to carry MPTCP options that are tied to the data payload
  - Also used to remove `sp` (`sec_path`) and `nf_bridge` pointers from `struct sk_buff`
  - Suitable for data that can't fit in `sk_buff` and justifies memory overhead
- Add `inet_diag_ulp_info` to socket diag format and ULP `get_info` hook

Change in TCP Code

Git Stat:

```
include/linux/skbuff.h             |   11 ++
include/linux/tcp.h                |   51 ++++++++++
include/net/sock.h                 |    6 +-
include/net/tcp.h                  |   20 ++++
include/trace/events/sock.h        |    5 +-
include/uapi/linux/in.h            |    2 +
net/Kconfig                        |    1 +
net/Makefile                       |    1 +
net/ax25/af_ax25.c                 |    2 +-
net/core/skbuff.c                  |    7 ++
net/decnet/af_decnet.c             |    2 +-
net/ipv4/inet_connection_sock.c    |    2 +
net/ipv4/tcp.c                     |    8 +-
net/ipv4/tcp_input.c               |   29 +++++-
net/ipv4/tcp_ipv4.c                |    4 +-
net/ipv4/tcp_minisocks.c           |    6 ++
net/ipv4/tcp_output.c              |   62 ++++++++++++-
net/ipv4/tcp_ulp.c                 |   12 +++
```

Changes to TCP code

- `tcp_ulp_clone()`
- Export two low-level TCP functions and one struct
- SKBs with MPTCP extensions can’t be coalesced or collapsed
- MPTCP option parsing and writing
- `is_mptcp` flag in `tcp_sock` and `tcp_request_sock`

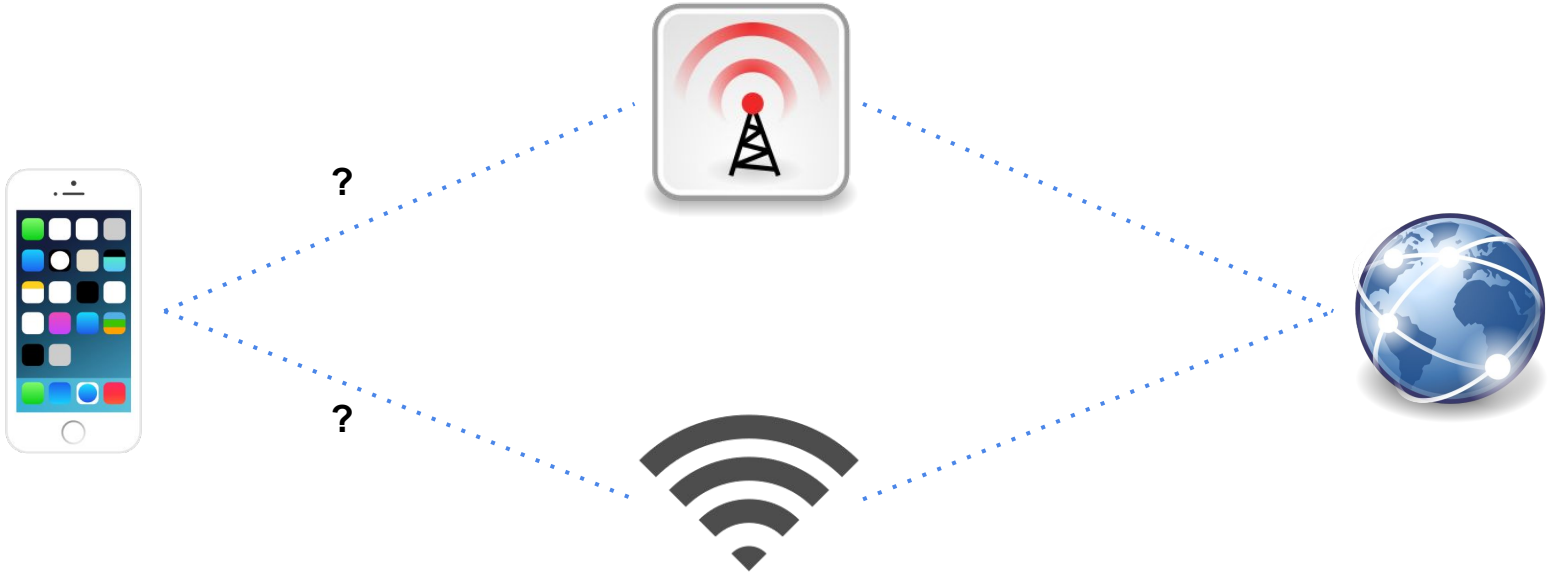
Changes to TCP code, continued

- One MPTCP-specific branch in TCP minisocks
- Call out to MPTCP from `tcp_data_queue` to add SKB extension and process ACKs
- Additional members in `struct tcp_options_received`
- Subflow receive window sharing will introduce changes too

Advanced Features Roadmap

Path Manager

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

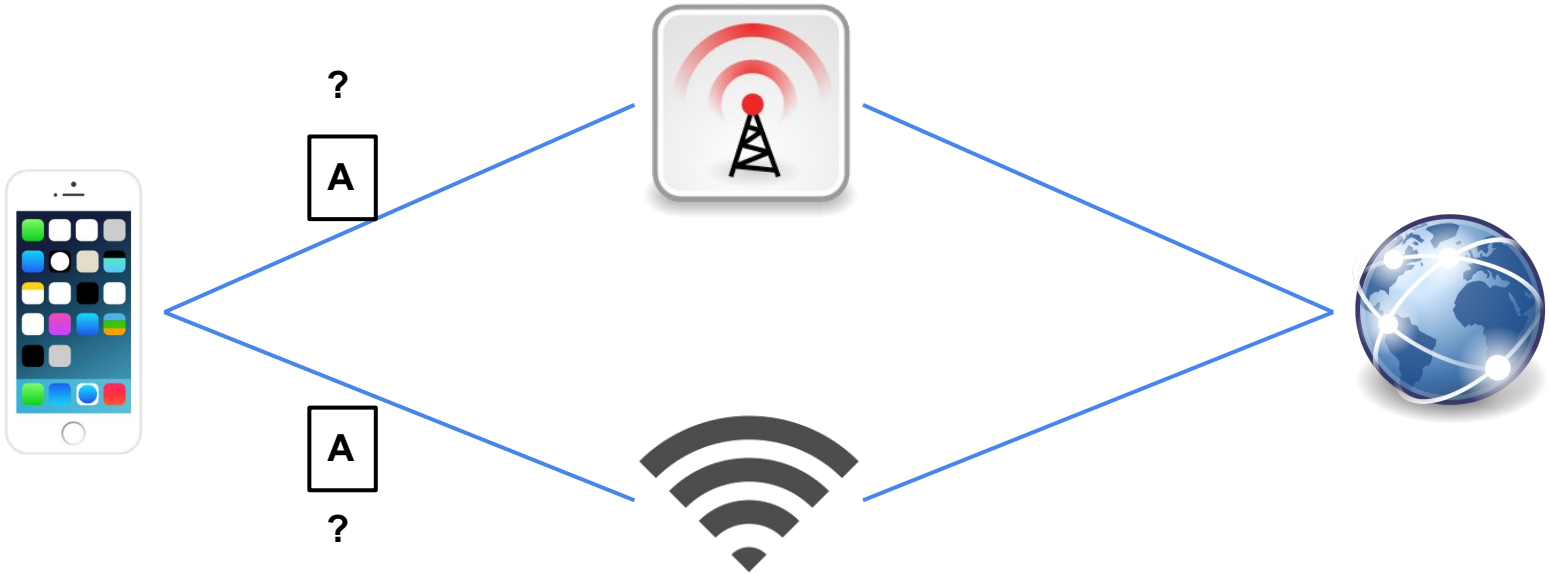


Userspace Path Manager

- Peers share `ADD_ADDR` and `REMOVE_ADDR` signals to advertise available addresses for each MPTCP connection
- Path manager runs in userspace and uses generic netlink to track address and local interface updates and request subflow changes
- Can be customized with different policies.
- Multipath TCP Daemon alpha release is available at github.com/intel/mptcpd

Packet Scheduler

On which available path packets will be sent? Reinject packets in another path?



Packet scheduling

- Different connections may optimize for throughput, latency, or redundancy.
- Peers can set a ‘backup’ flag on each subflow to limit transmission on that flow
- Include basic scheduler options in the kernel
- Consider eBPF to define custom schedulers, instead of kernel modules

Using MPTCP with unmodified binaries

- Some organizations want to take advantage of MPTCP without recompiling their userspace
- Can add BPF_CGROUP_SOCKET to attach an eBPF program that rewrites the protocol number passed to socket()
- Similar attachment points exist for bind() and connect()

MPTCP Performance optimizations

- Initial emphasis is on correctness and reasonable MPTCP performance
  - While not disrupting TCP's optimizations!
- Target performance optimizations based on data
- Protocol optimizations
  - Example: changing scheduler behavior for reinjection of data on different subflows
- TCP Fast Open support

Break-before-make

- MPTCP can keep a connection active even with zero subflows connected
    - Allows the session to continue by adding a subflow with MP_JOIN
- Can be useful to switch between access points
- Will add this capability if there’s demand for it

Subflow socket options

- One MPTCP socket manages a set of in-kernel subflow sockets
- Socket options that use TCP option space or change data flow could interfere
- The MPTCP socket can act as an intermediary for subflow options
- Will need to whitelist specific known-safe options
- Could expose file descriptors only good for getsockopt()/setsockopt()

Kernel TLS and MPTCP

- kTLS is built on top of TCP using ULP framework
- An MPTCP socket is not a TCP socket, so it doesn't have ULP
- TLS needs to operate on the MPTCP data stream, not subflow streams
  - TLS records could be split across subflows
  - MPTCP DSS mappings are specific to TCP sequence numbers
- `TLS_SW` appears feasible but would need work to integrate with an MPTCP socket type

Conclusion

Conclusion

- Build around TCP as much as we can.
- We are close to having an initial patch set ready.

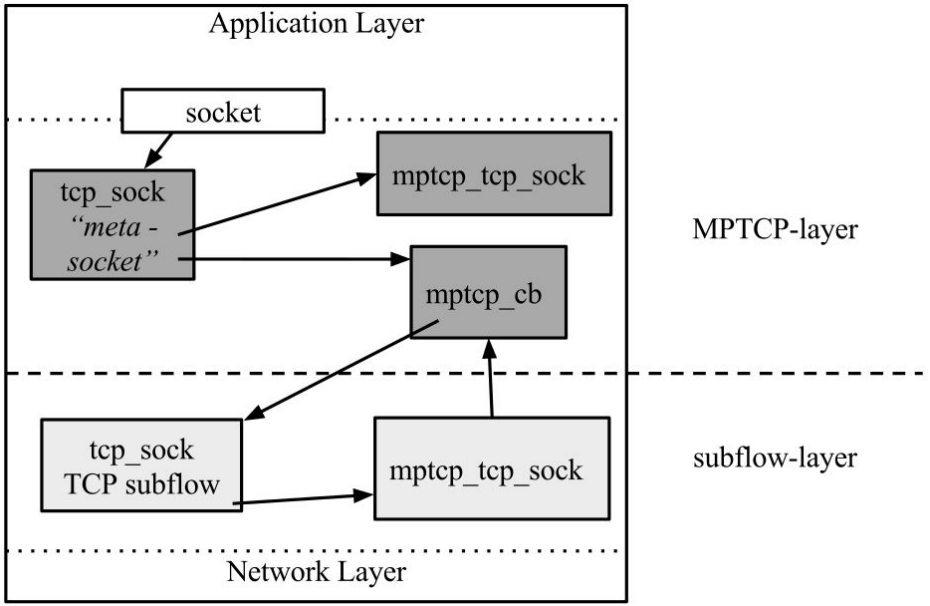
This project is open to everybody.

- Wiki: https://is.gd/mptcp_upstream
- Mailing list: https://lists.01.org/mailman/listinfo/mptcp
- Git repository: https://github.com/multipath-tcp/mptcp_net-next
- Paper: https://linuxplumbersconf.org/event/4/contributions/435/
- mathew.j.martineau@linux.intel.com
- matthieu.baerts@tessares.net

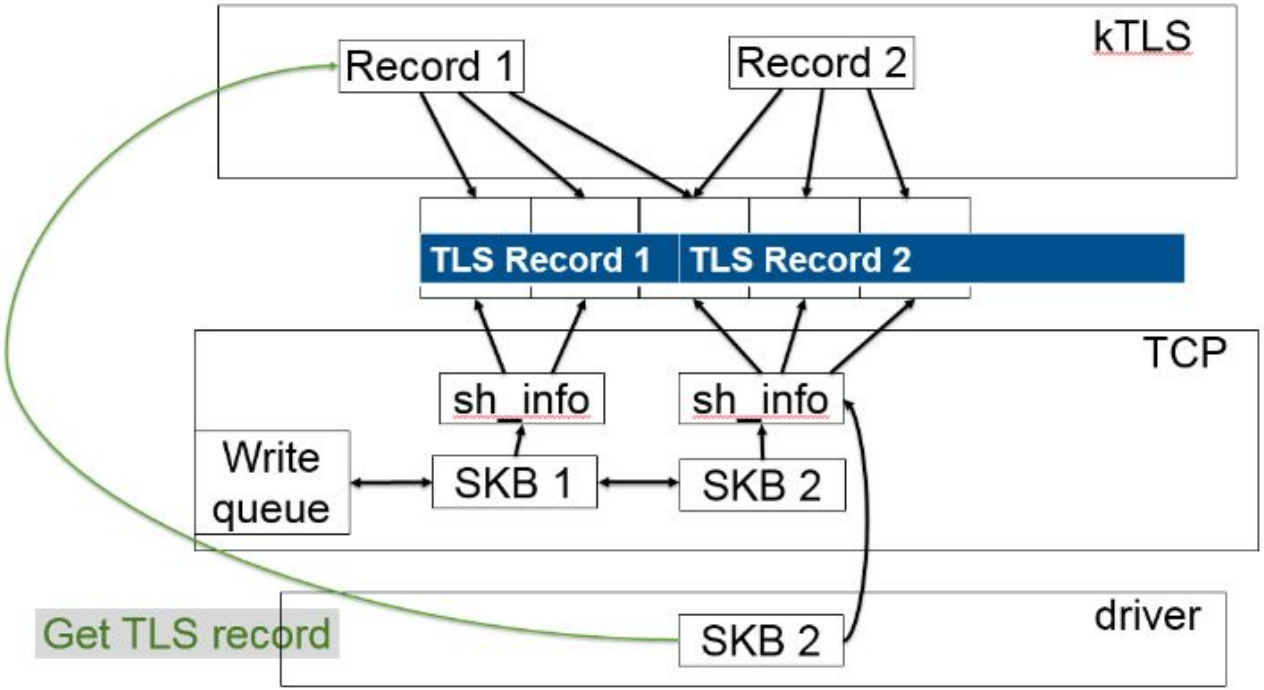
Backup slides

Protocol challenges

Relations between structures



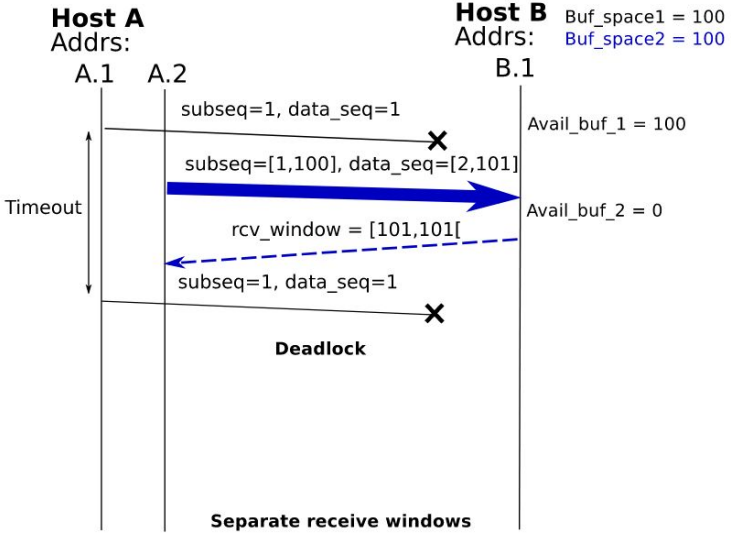
kTLS record

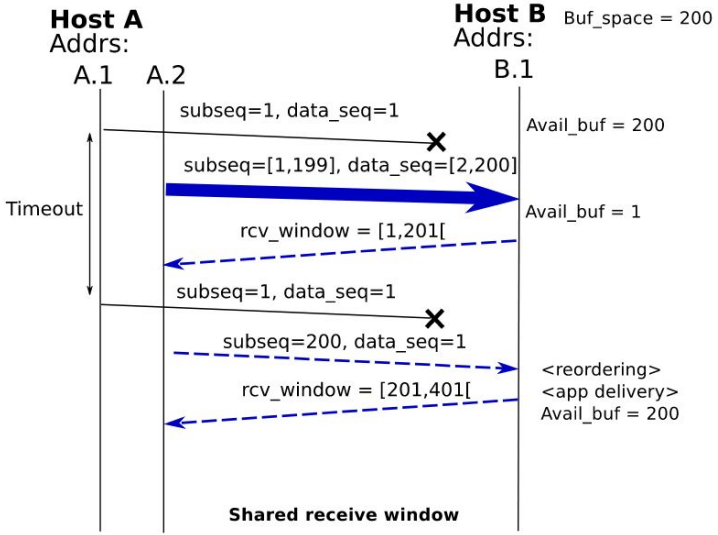


From: https://netdevconf.org/1.2/papers/netdevconf-TLS.pdf

Protocol challenges

Coupled receive windows across TCP subflows

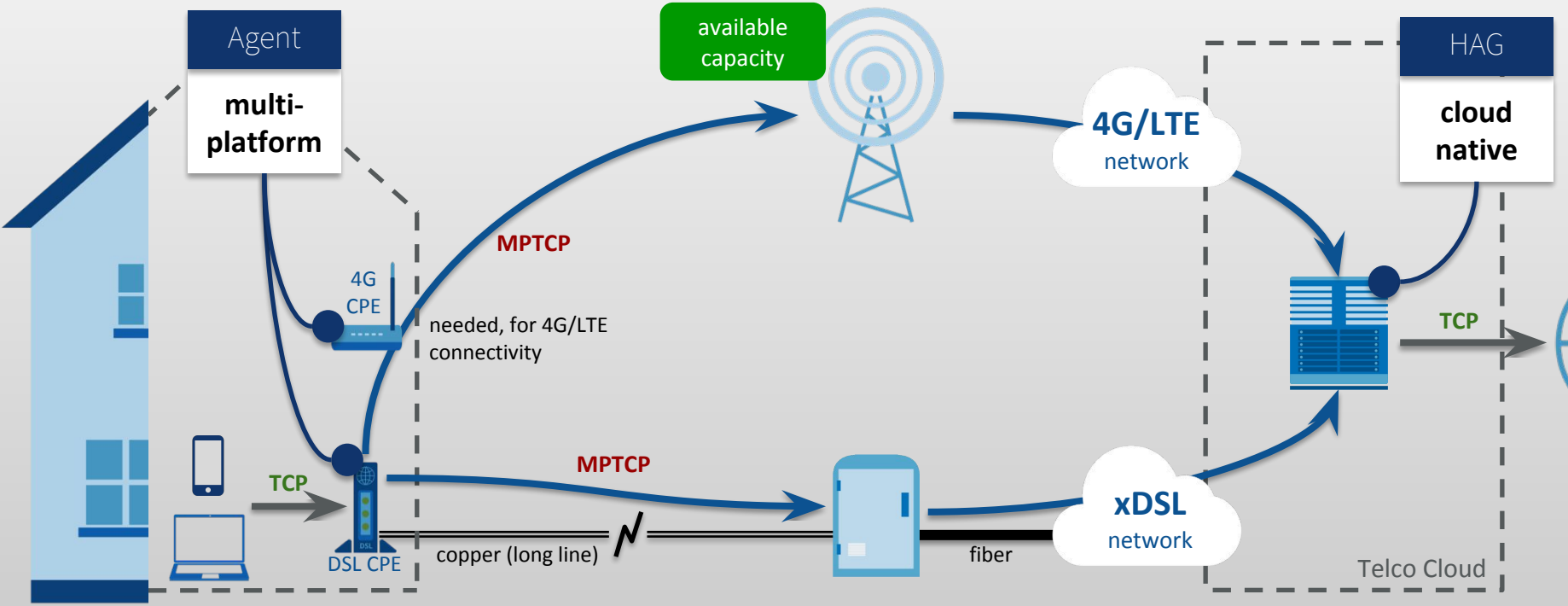




Used with Sébastien Barré's permission

Multipath TCP (MPTCP)

Hybrid access network use-case (BBF TR-348 by Tessares - SwissCom - OVH)

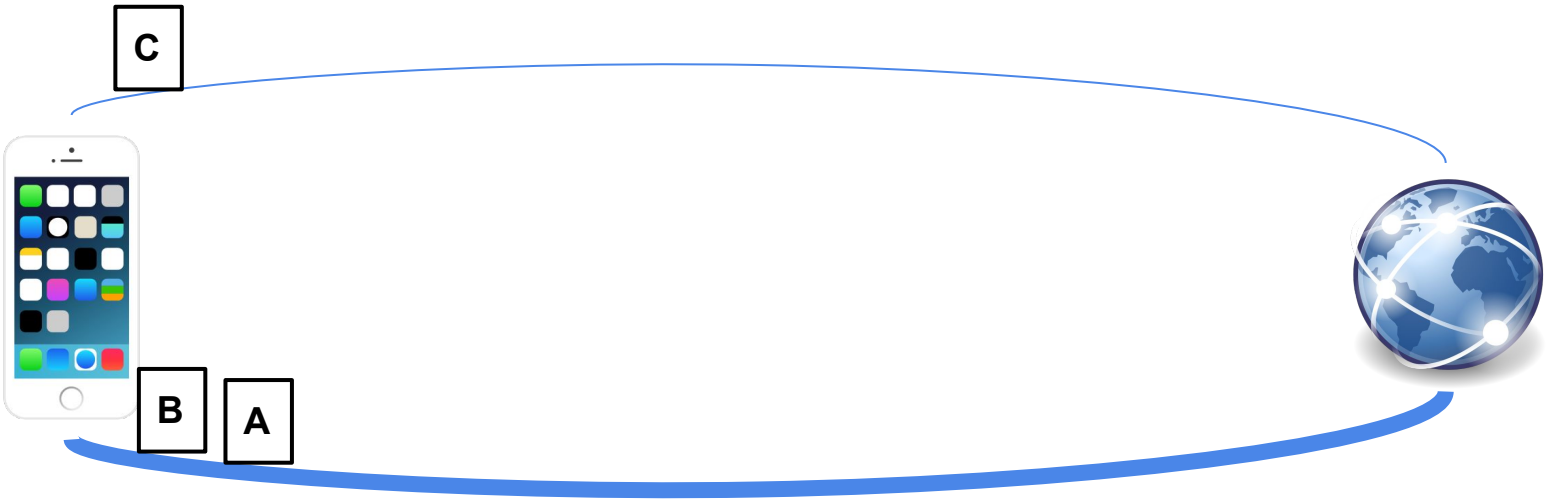


Images from Tessares

Protocol challenges

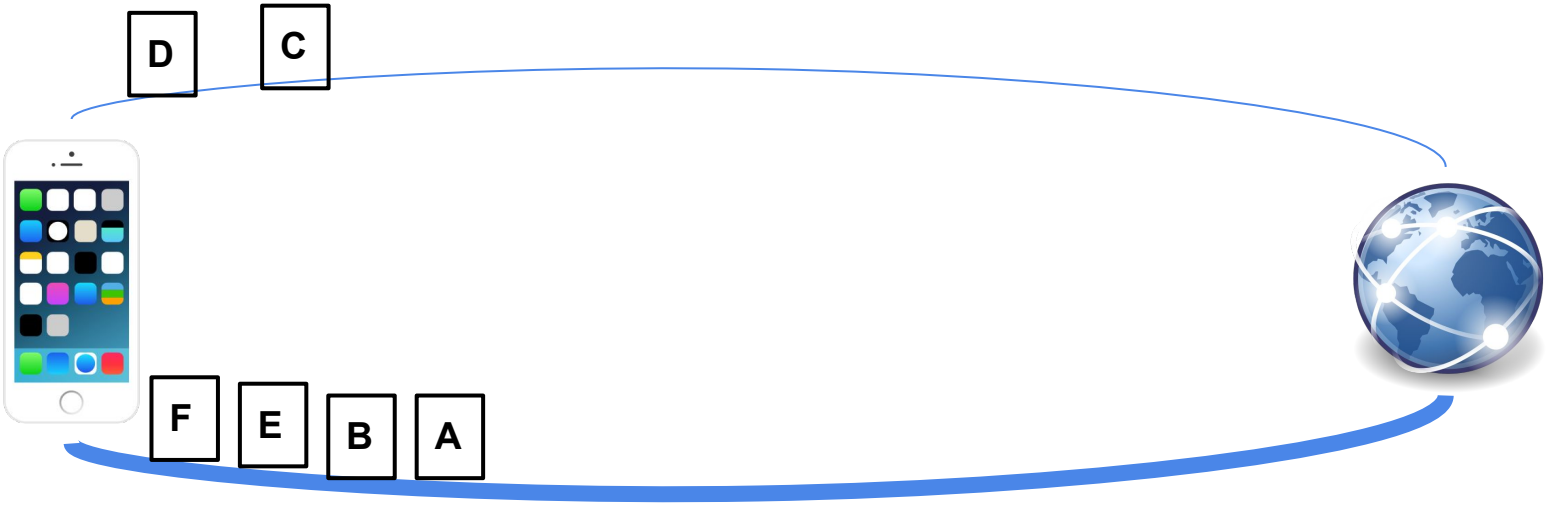
Protocol challenges

Data sequence numbers and mappings



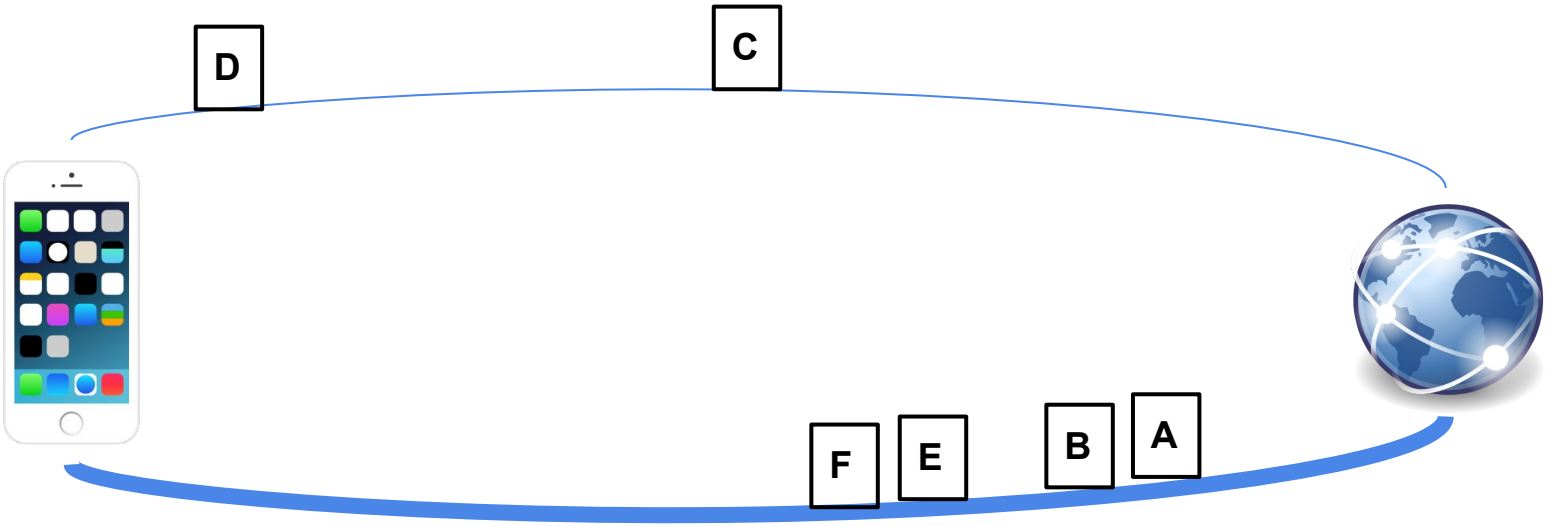
Protocol challenges

Data sequence numbers and mappings



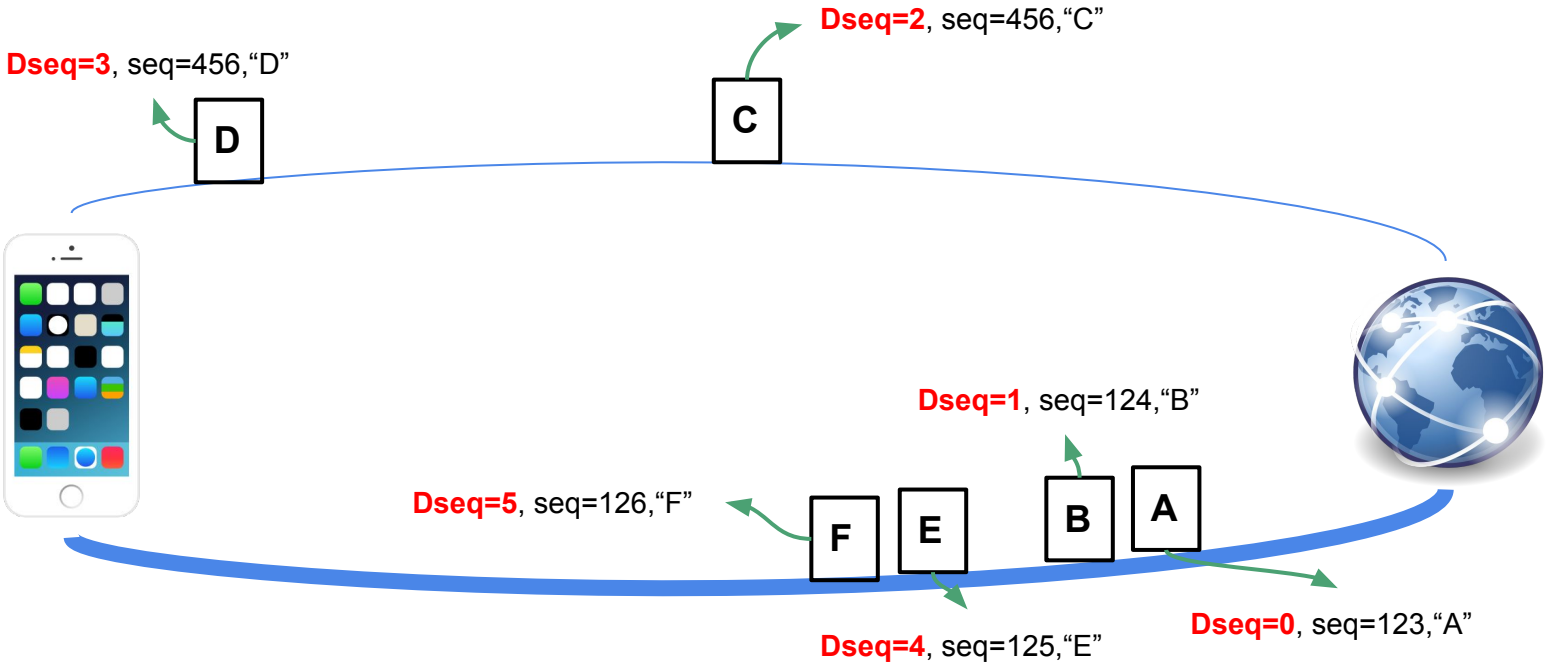
Protocol challenges

Data sequence numbers and mappings



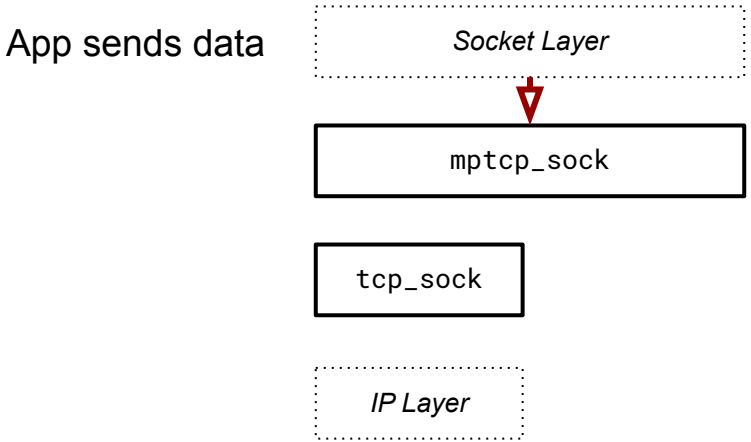
Protocol challenges

Data sequence numbers and mappings



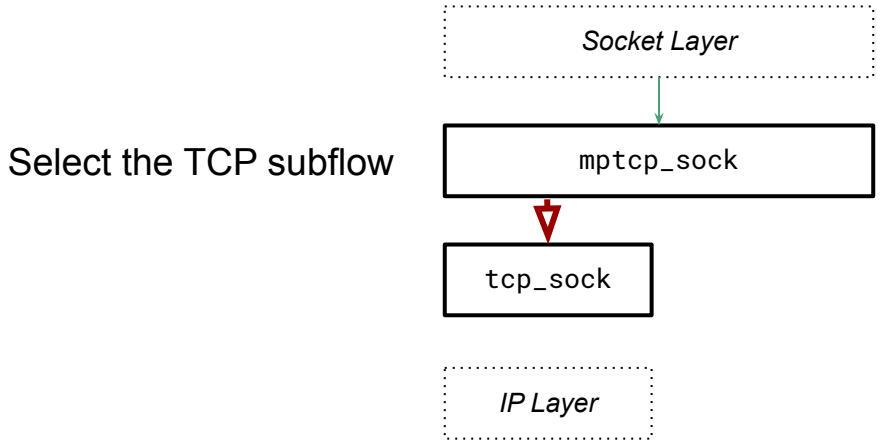
Protocol challenges

Data sequence numbers and mappings



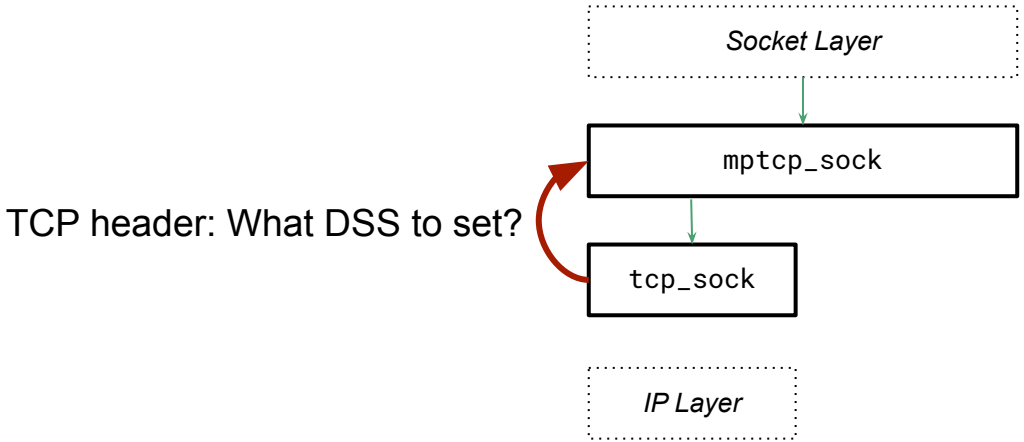
Protocol challenges

Data sequence numbers and mappings



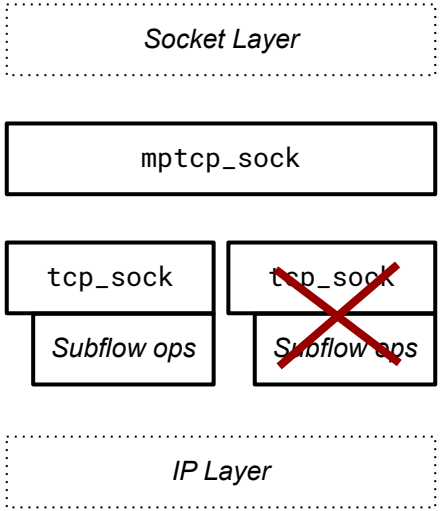
Protocol challenges

Data sequence numbers and mappings



Protocol challenges

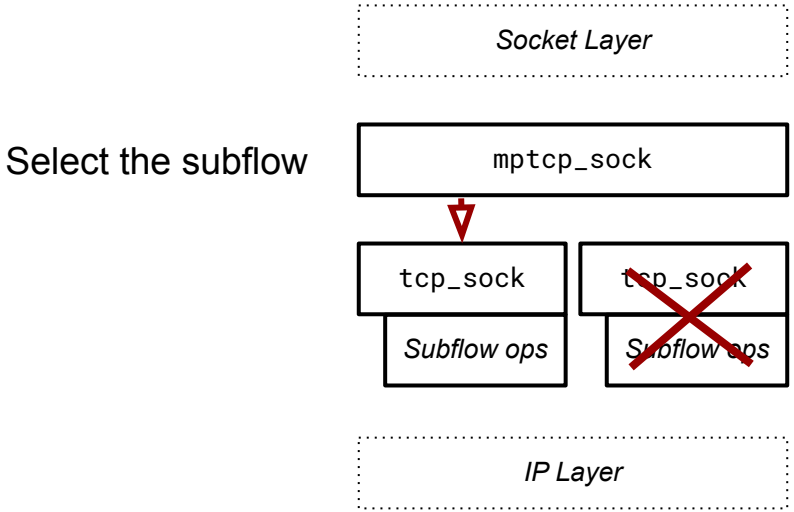
Sending of ACKs to signal options, e.g. REMOVE_ADDR in a TCP ACK



Notification: one iface is down

Protocol challenges

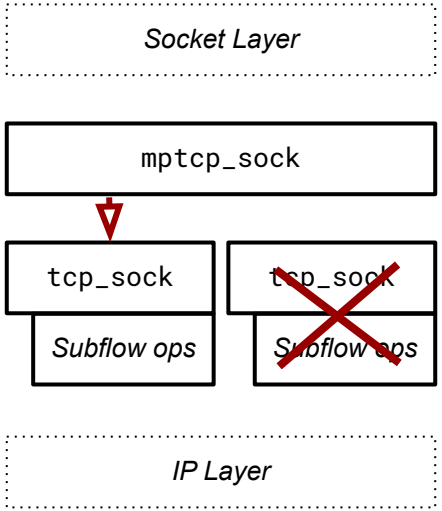
Sending of ACKs to signal options, e.g. REMOVE_ADDR in a TCP ACK



Protocol challenges

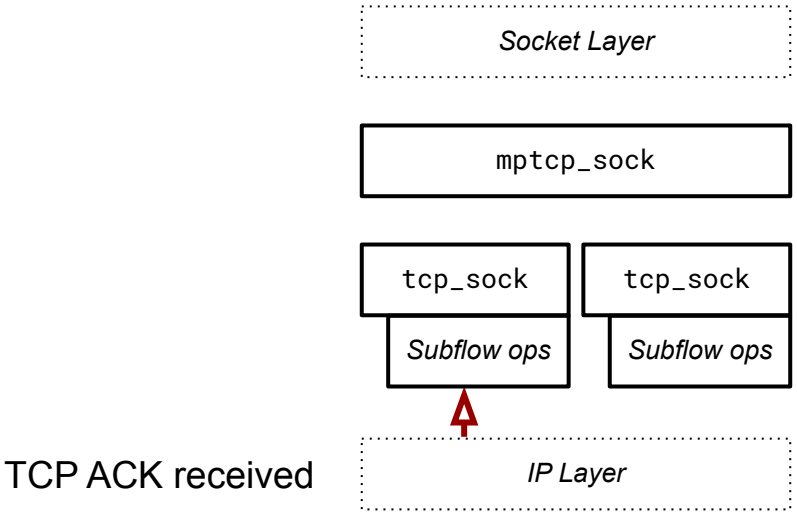
Sending of ACKs to signal options, e.g. REMOVE_ADDR in a TCP ACK

Sending a ACK not from TCP stack



Protocol challenges

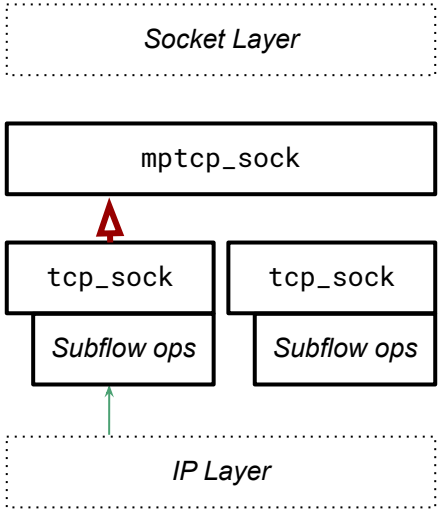
Reception of ACKs with signaling options, e.g. REMOVE_ADDR in a TCP ACK



Protocol challenges

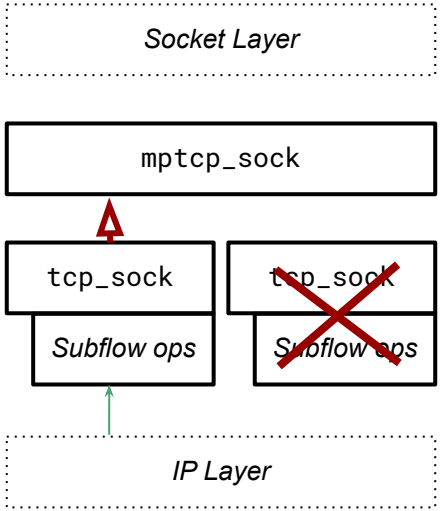
Reception of ACKs with signaling options, e.g. REMOVE_ADDR in a TCP ACK

TCP ACK is not dropped



Protocol challenges

Reception of ACKs with signaling options, e.g. REMOVE_ADDR in a TCP ACK



Protocol challenges

Signaling with MPTCP:

● MP_CAPABLE	SYN
● MP_JOIN	SYN
● DSEQ / DACK	ALL
● FAST_CLOSE	ACK followed by RST
● ADD_ADDR	ACK
● REMOVE_ADDR	ACK