How hard can it be?

Adding Multipath TCP to the upstream Linux kernel

Mat Martineau (Intel), Matthieu Baerts (Tessares)
Christoph Paasch (Apple), Peter Krystad (Intel)

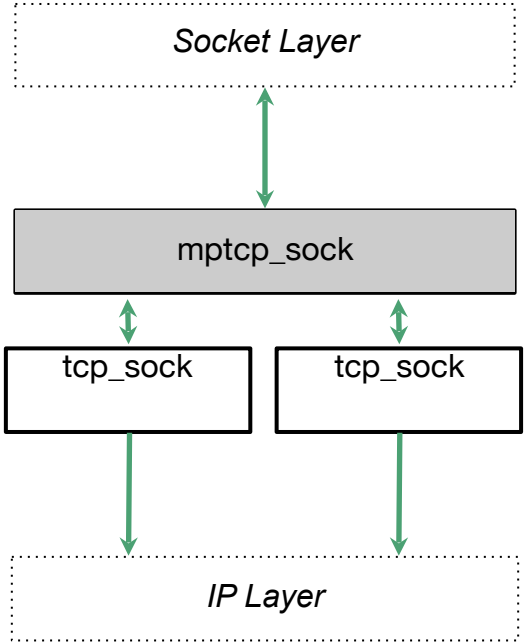
Why upstreaming MPTCP is complicated?

Why upstreaming MPTCP is so complicated?

- Linux TCP is highly optimized
- Cannot take github.com/multipath-tcp/mptcp
  - Built to support experiments and rapid changes but not generic enough
  - Special purpose implementation of MPTCP
- New implementation cannot affect existing TCP stack:
  - Without performance regressions
  - Maintainable and configurable
  - Can be used in a variety of deployments

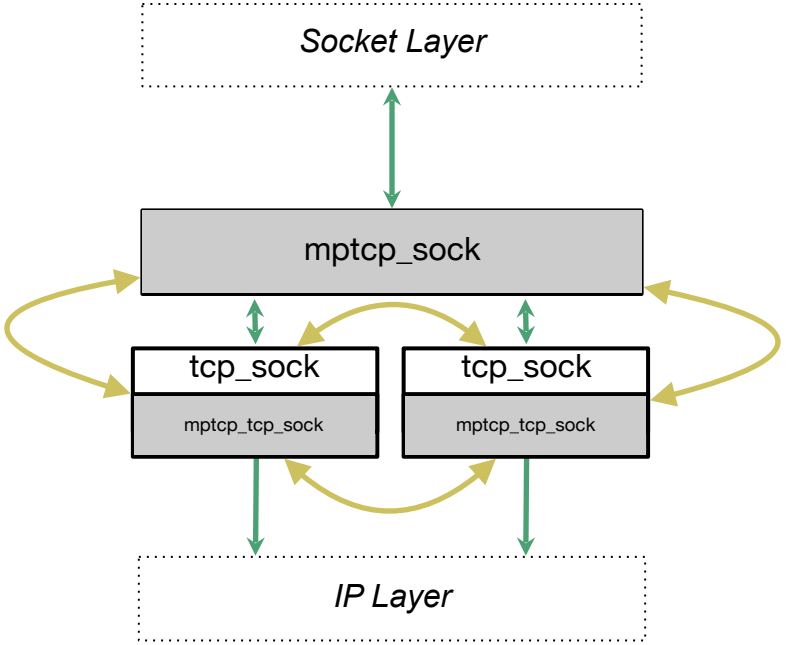
Why upstreaming MPTCP is so complicated?

An ideal MPTCP implementation:



Why upstreaming MPTCP is so complicated?

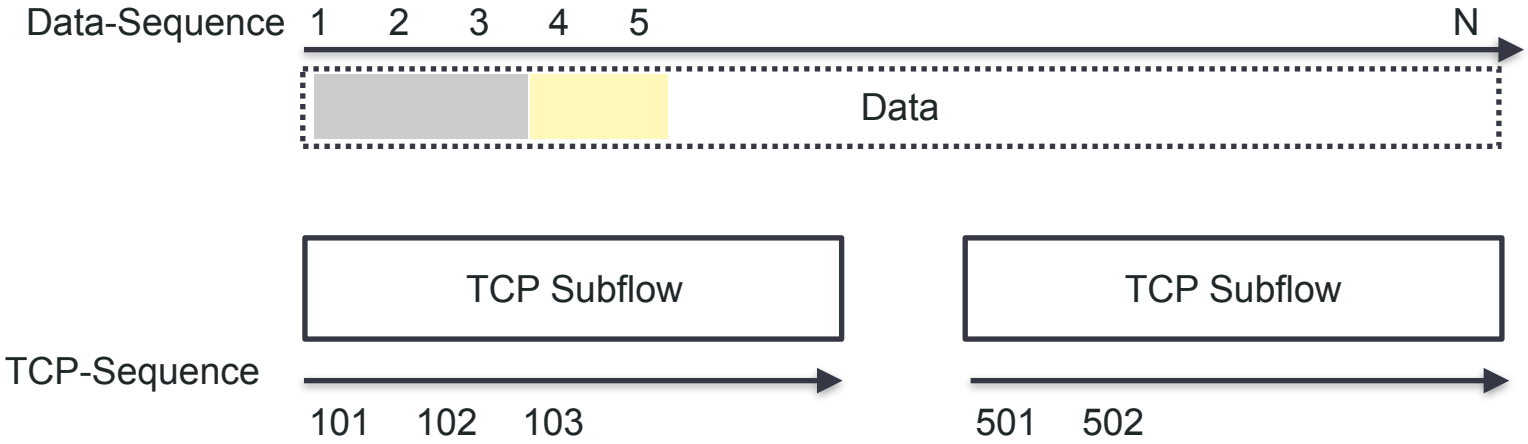
A ***real*** MPTCP implementation:



Protocol challenges

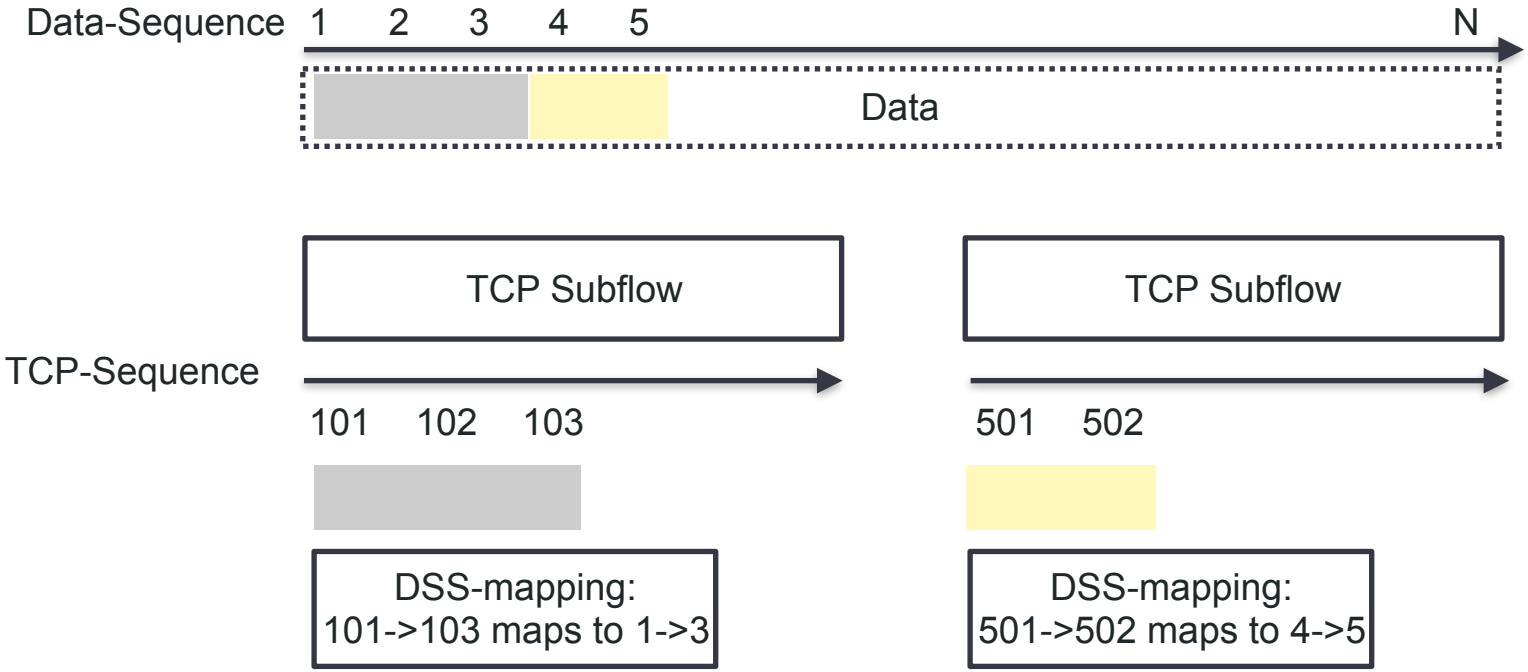
Protocol challenges

Data sequence numbers and mappings



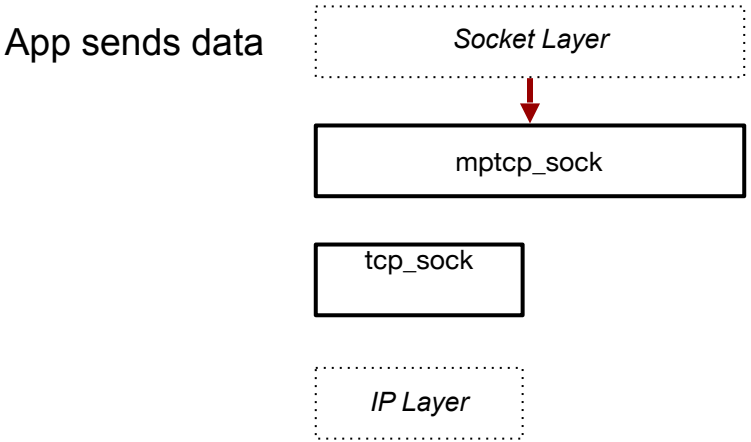
Protocol challenges

Data sequence numbers and mappings



Protocol challenges

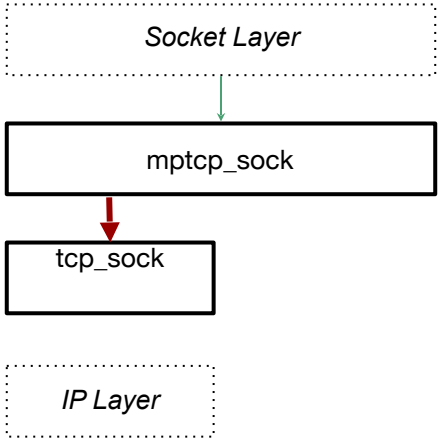
Data sequence numbers and mappings



Protocol challenges

Data sequence numbers and mappings

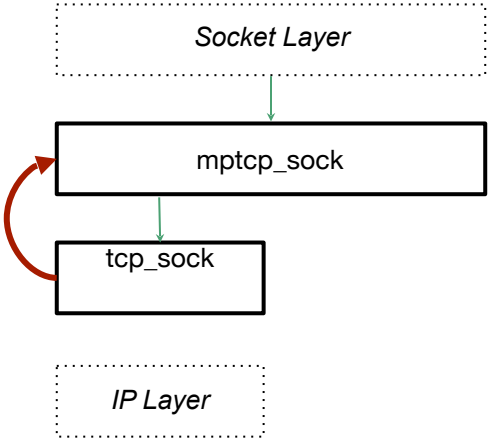
Select the TCP subflow



Protocol challenges

Data sequence numbers and mappings

TCP header: What DSS to set?



Protocol challenges

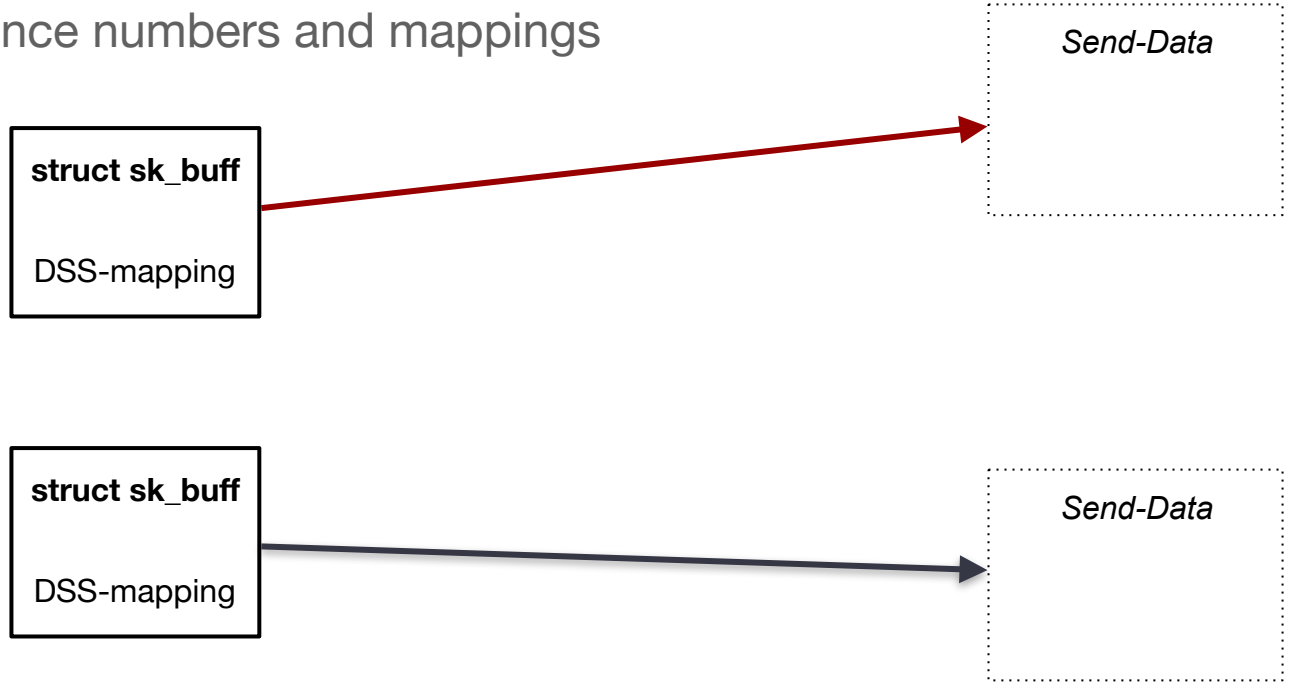
Data sequence numbers and mappings

Protocol challenges

Data sequence numbers and mappings

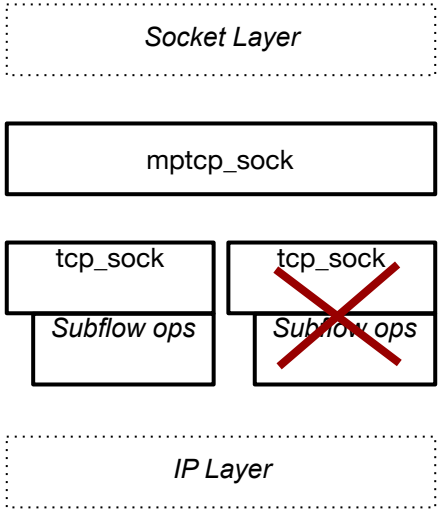
Protocol challenges

Data sequence numbers and mappings



Protocol challenges: Cross Layer Signaling

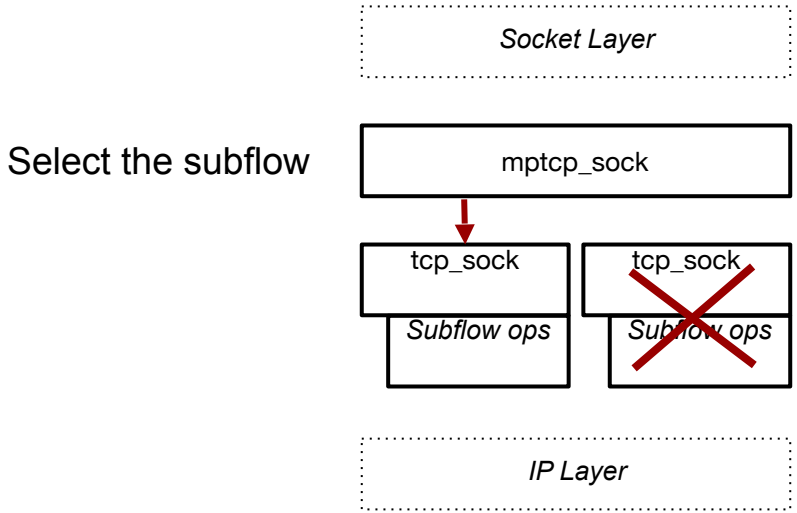
Sending of ACKs to signal options, e.g. REMOVE_ADDR in a TCP ACK



Notification: one iface is down

Protocol challenges: Cross Layer Signaling

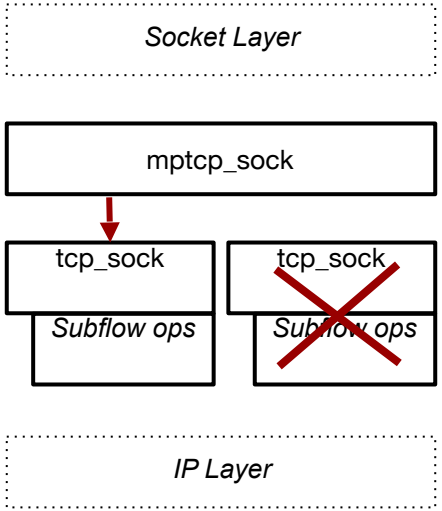
Sending of ACKs to signal options, e.g. REMOVE_ADDR in a TCP ACK



Protocol challenges: Cross Layer Signaling

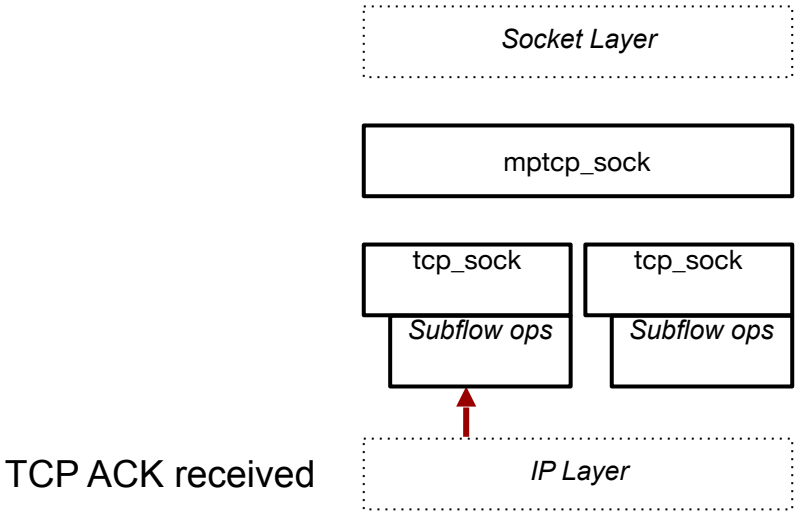
Sending of ACKs to signal options, e.g. REMOVE_ADDR in a TCP ACK

Sending a ACK not from TCP stack



Protocol challenges: Cross Layer Signaling

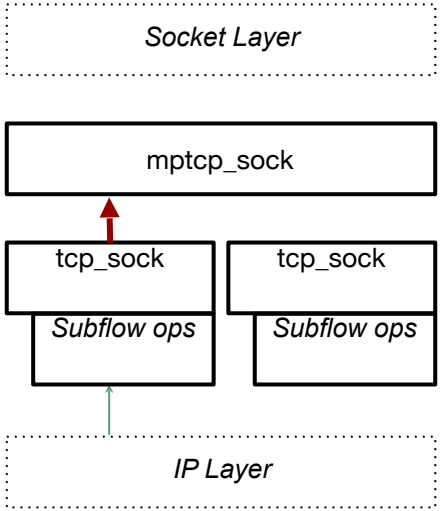
Reception of ACKs with signaling options, e.g. REMOVE_ADDR in a TCP ACK



Protocol challenges: Cross Layer Signaling

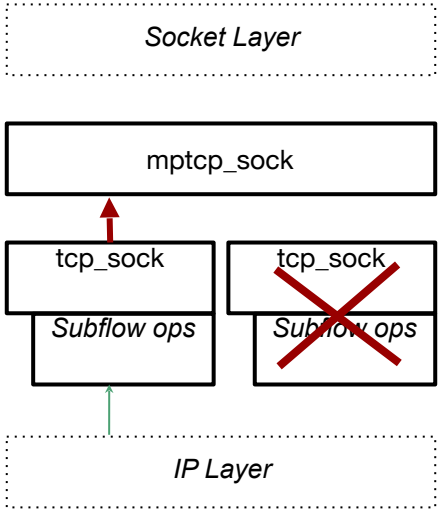
Reception of ACKs with signaling options, e.g. REMOVE_ADDR in a TCP ACK

TCP ACK is not dropped



Protocol challenges: Cross Layer Signaling

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



Protocol challenges: Cross Layer Signaling

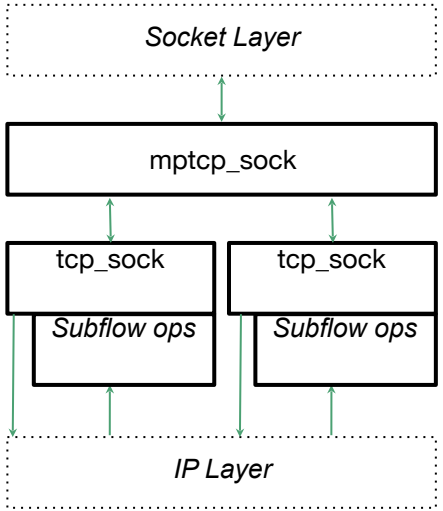
Signaling with MPTCP:

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

Fitting in to net/

What fits well

- New MPTCP socket type
  - struct proto / struct inet_protosw
  - socket(AF_INET, SOCK_STREAM, IPPROTO_MPTCP);
- In-kernel TCP interfaces
  - do_tcp_sendpages()
  - tcp_read_sock()
- IP networking core
  - struct inet_connection_sock_af_ops



What is a challenge in the networking subsystem

- Indirect call inefficiency
- struct sk_buff non-extensibility
  - Data size is variable but metadata is constant
  - These structures need to shrink, not grow

```
struct sk_buff {
  /* size:    232,  cachelines:    4 */
  /* members: 74,   sum members:   229 */
  /* holes:   2,    sum holes:     3 */
  /* bit holes: 1,  sum bit holes: 1 bits */
  /* last cacheline: 40 bytes */
};
```

```
struct skb_shared_info {
  /* size:    320,  cachelines:  5 */
  /* members: 13,   sum members: 316 */
  /* holes:   1,    sum holes:   4 */
};
```

Next Steps

Next Steps

- Received supportive feedback from netdev community
- Reduce MPTCP to minimal viable implementation
- Consolidate “up-calls” to reduce impact
- Extend struct sk_buff in cache-line agnostic way

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

Lessons Learned

Lessons Learned

- Protocol design directly impacts implementations

- TCP options are best used for ***“unreliable”*** signals
- Cross-layer interactions should be ***asynchronous***

- Prototyping during standardization is very different from deployment