Shaping the Linux kernel MPTCP implementation towards upstream acceptance

Doru-Cristian Gucea
Octavian Purdila

Agenda

MPTCP in a nutshell

Use-cases

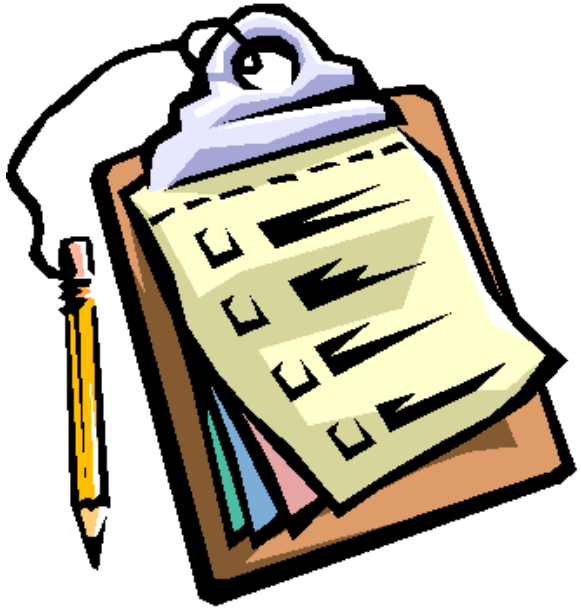
Basics

Initial Linux kernel implementation

Implementation alternatives

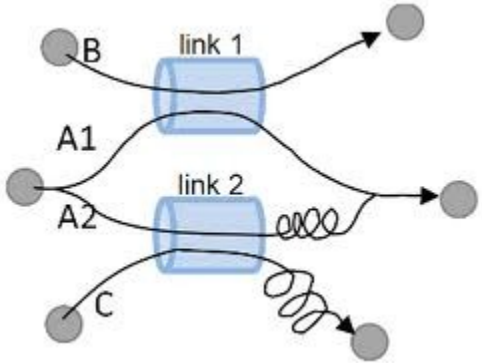
Towards upstream submission

Questions

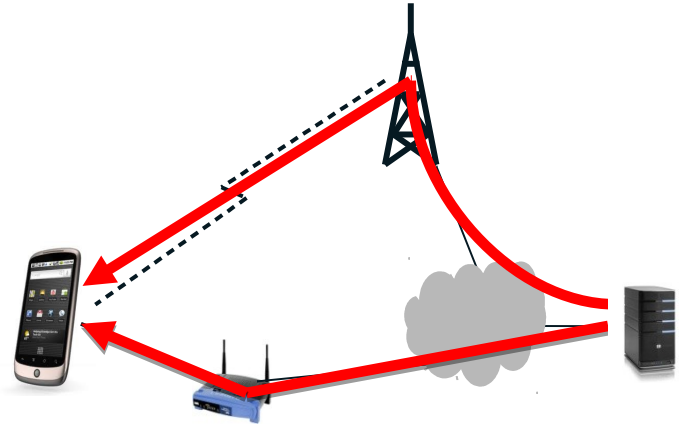


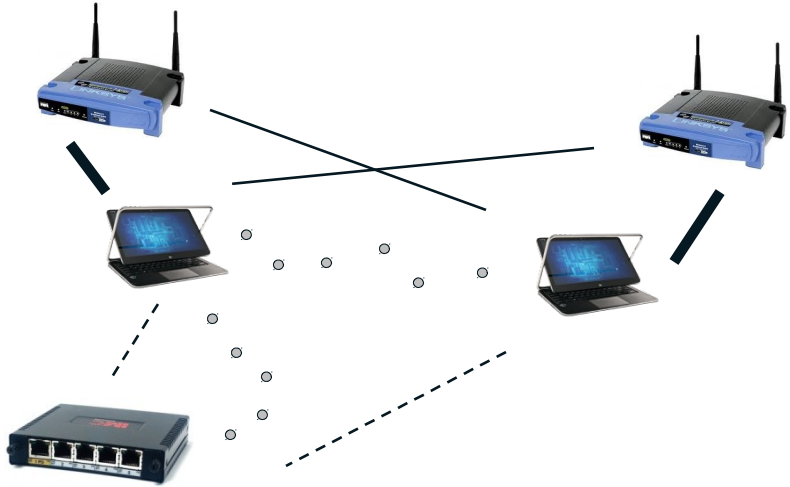
MPTCP in nutshell

- Transport level multi-path solution
- Unmodified applications and network
- Works at least as well as regular TCP
- Works when a regular TCP would work
- Falls back to regular TCP if needed
- Fair with TCP, moves traffic away from congestion

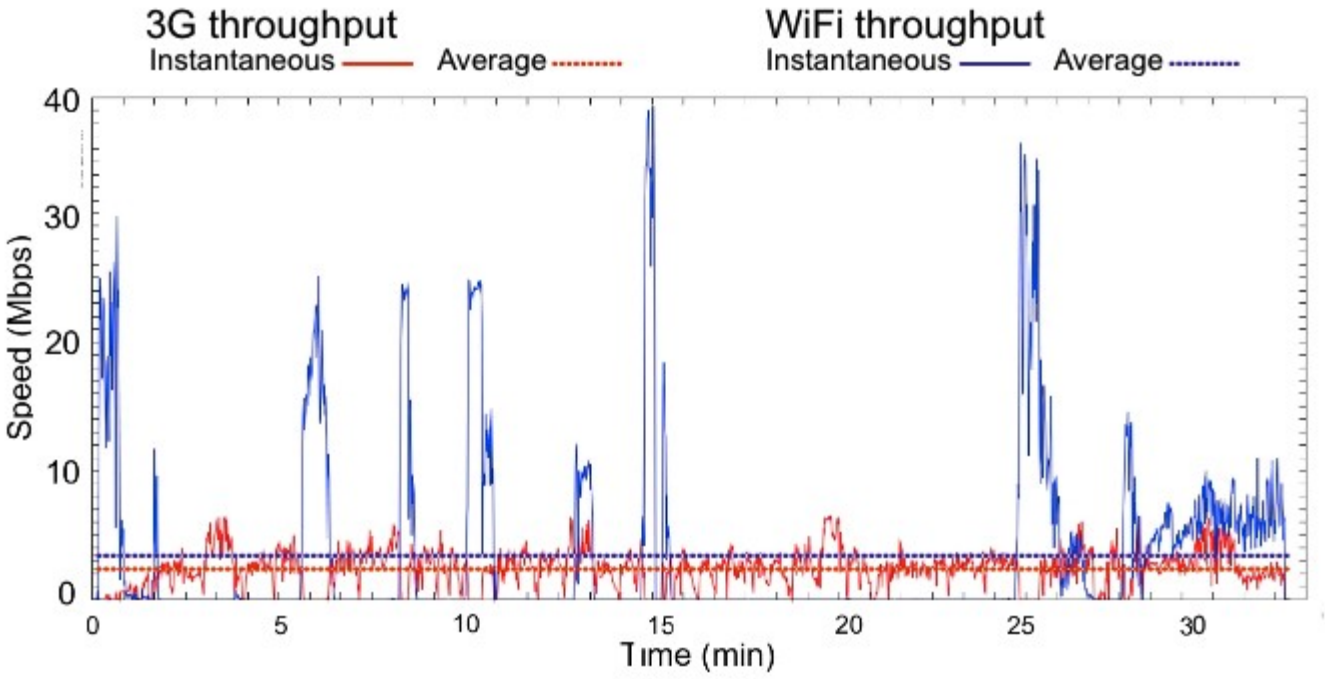


Improved mobility with MPTCP





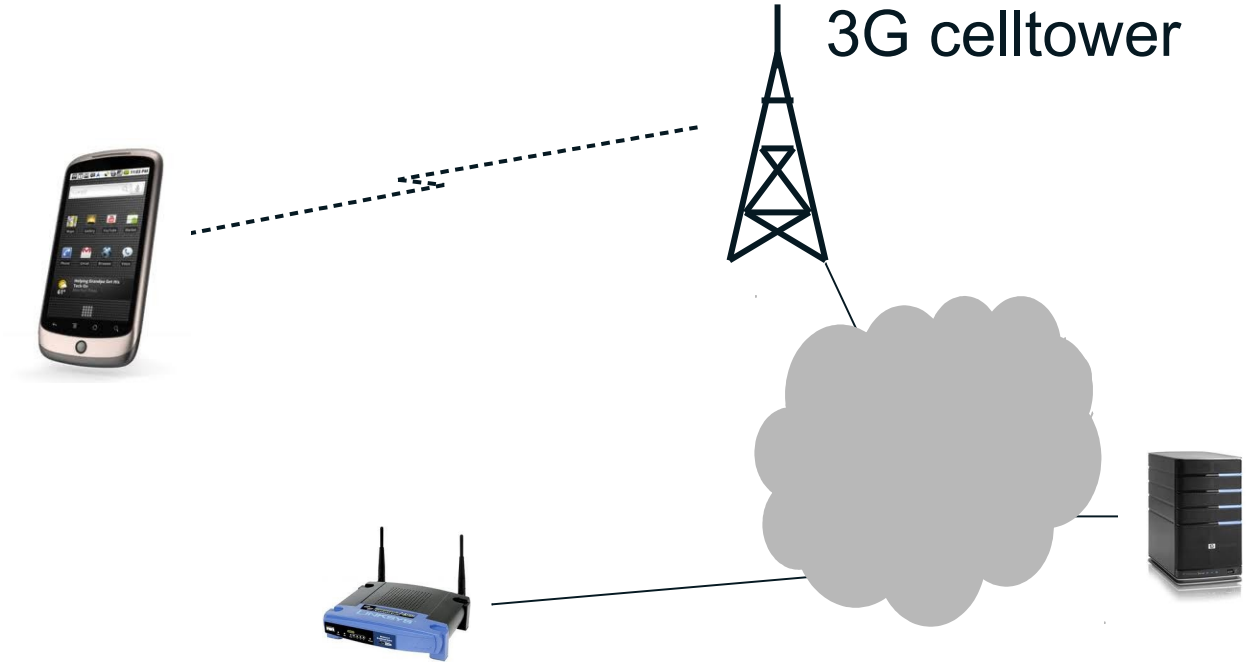
Throughput during a subway trip



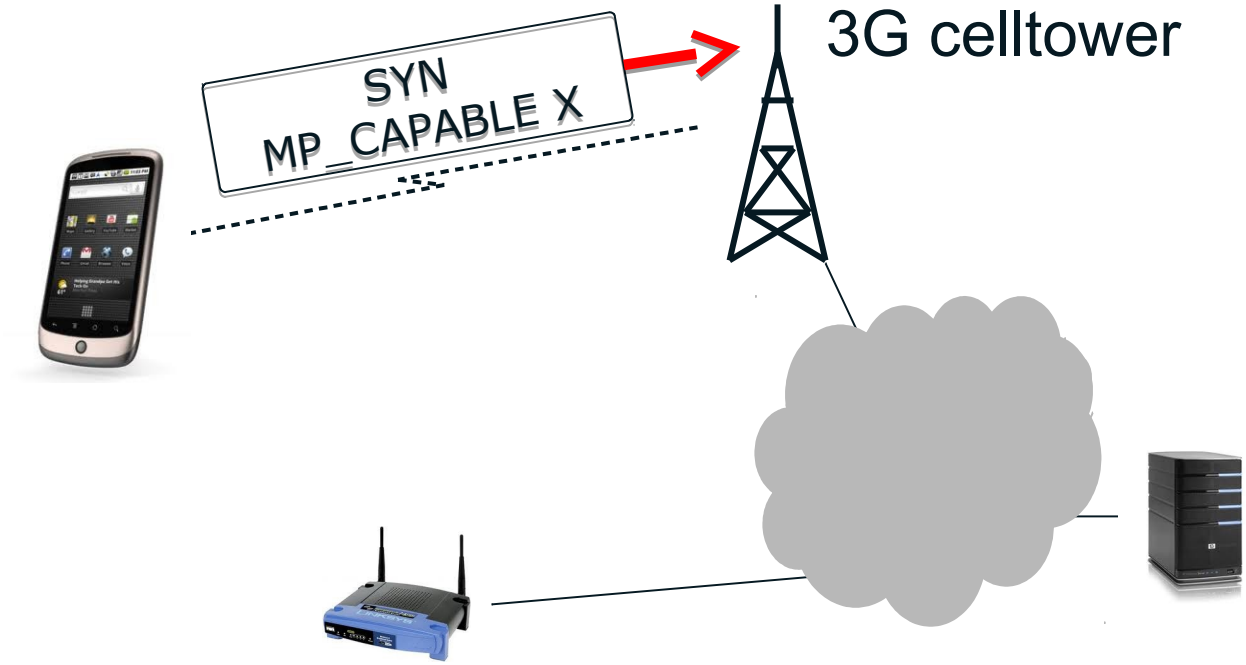
Other MPTCP use-cases

- Power improvements via race to idle
- VM migration across different network domains
- Multi-WiFi: take advantage of multiple APs
- Improved throughput and reliability in the data-center

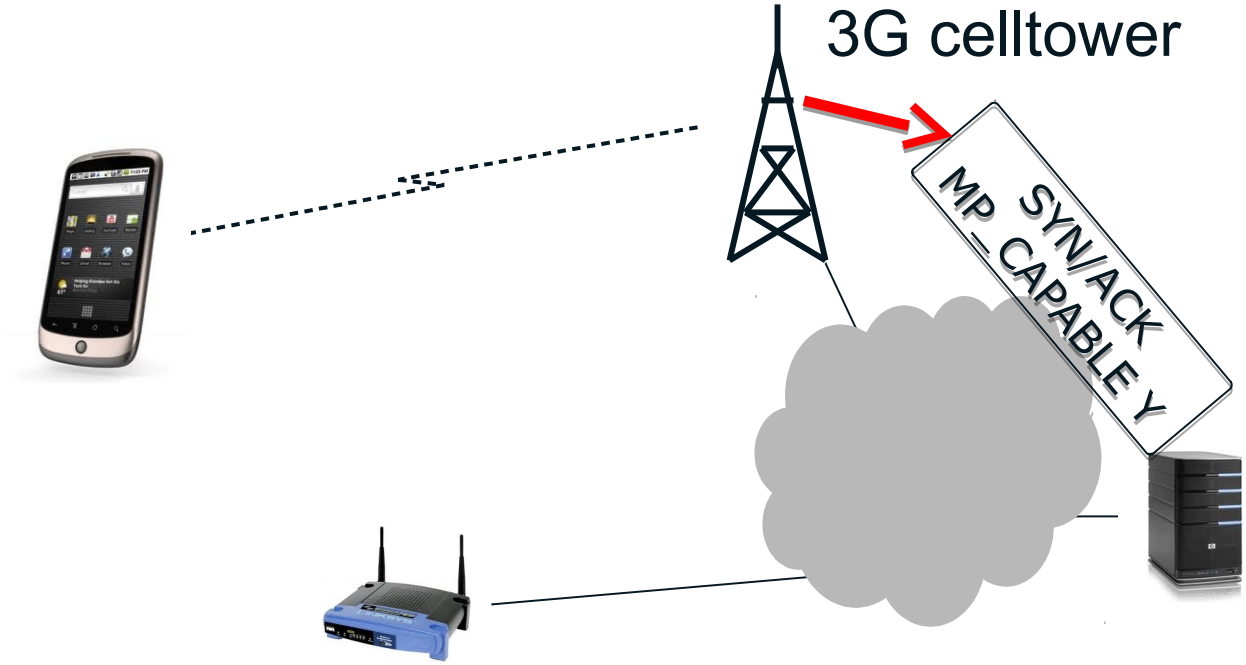
MPTCP basics



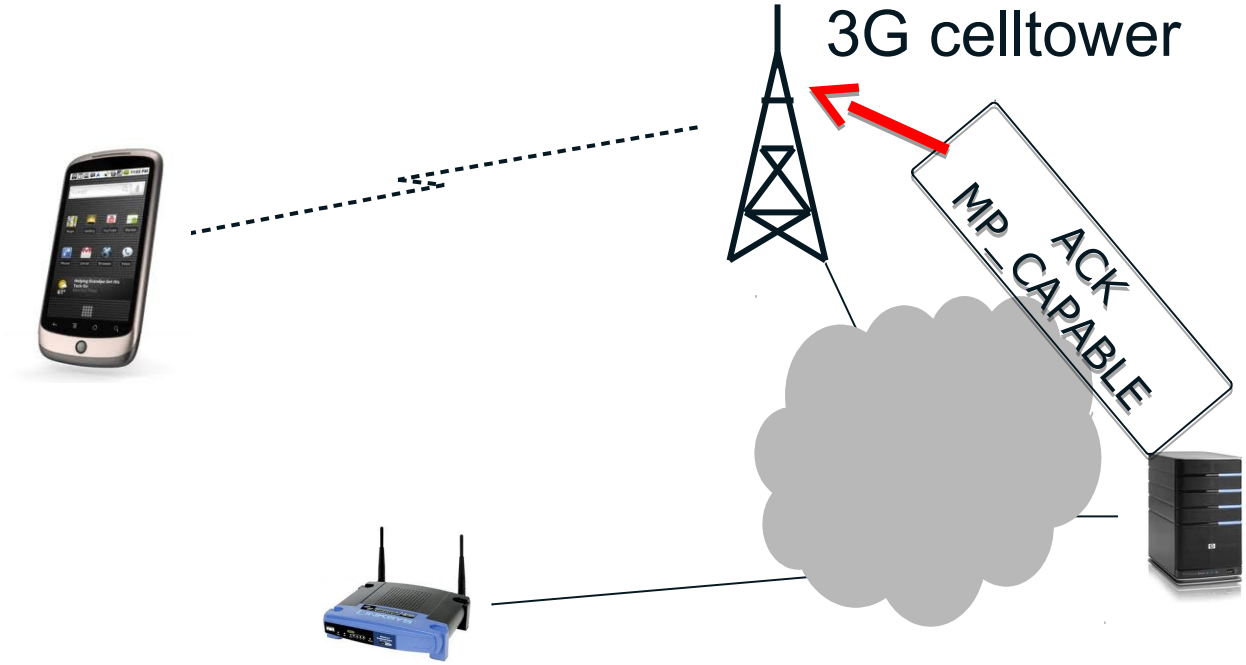
MPTCP basics



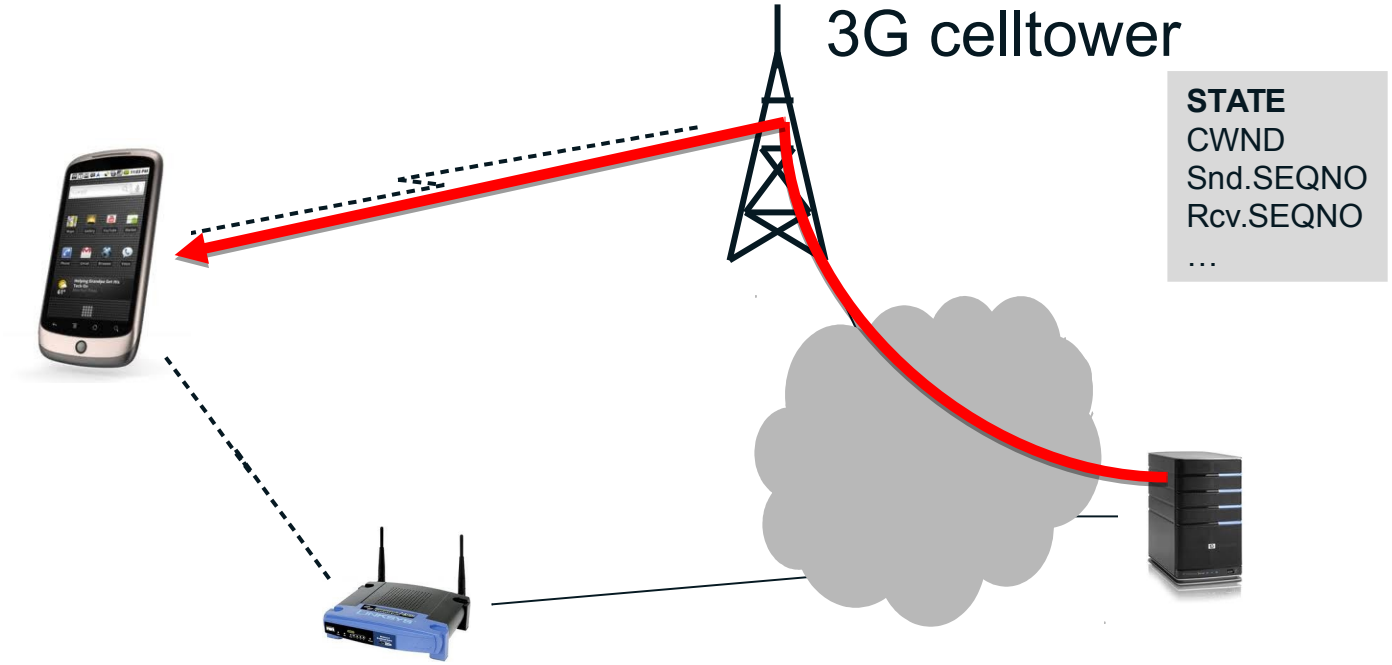
MPTCP basics



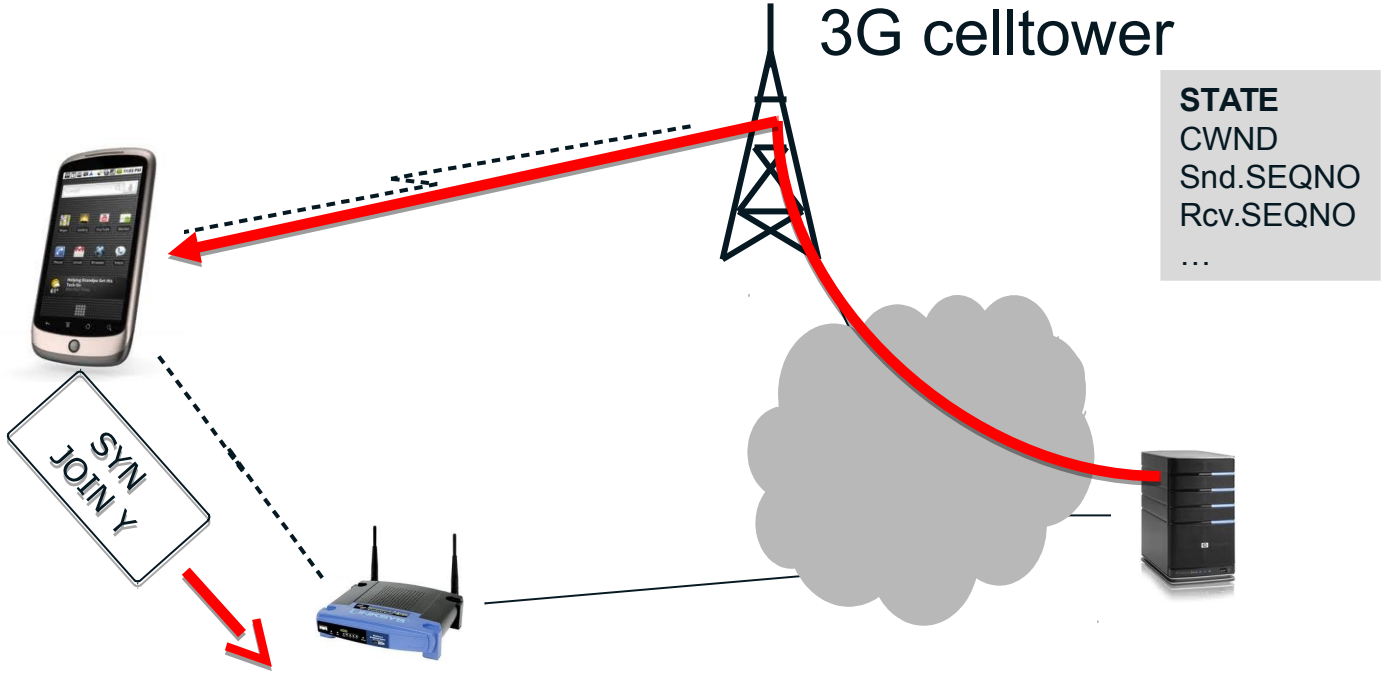
MPTCP basics



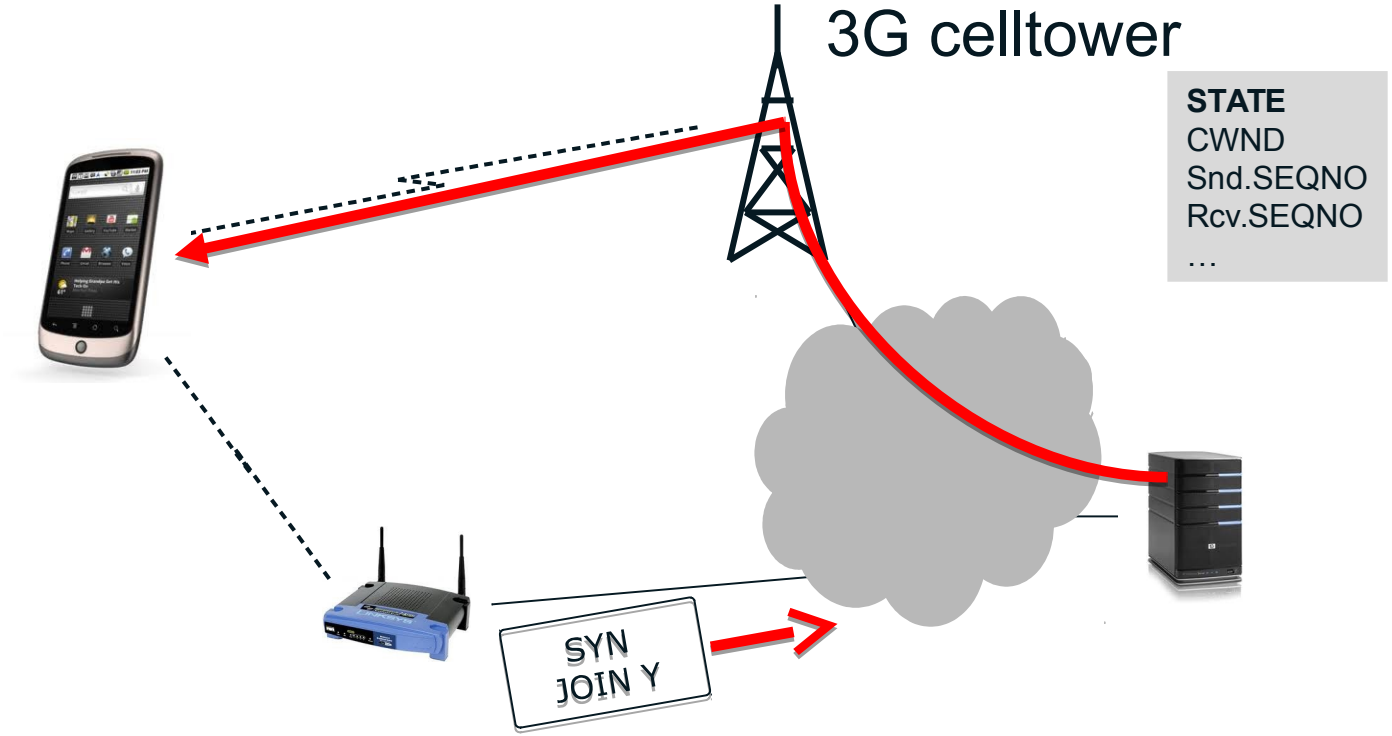
MPTCP basics



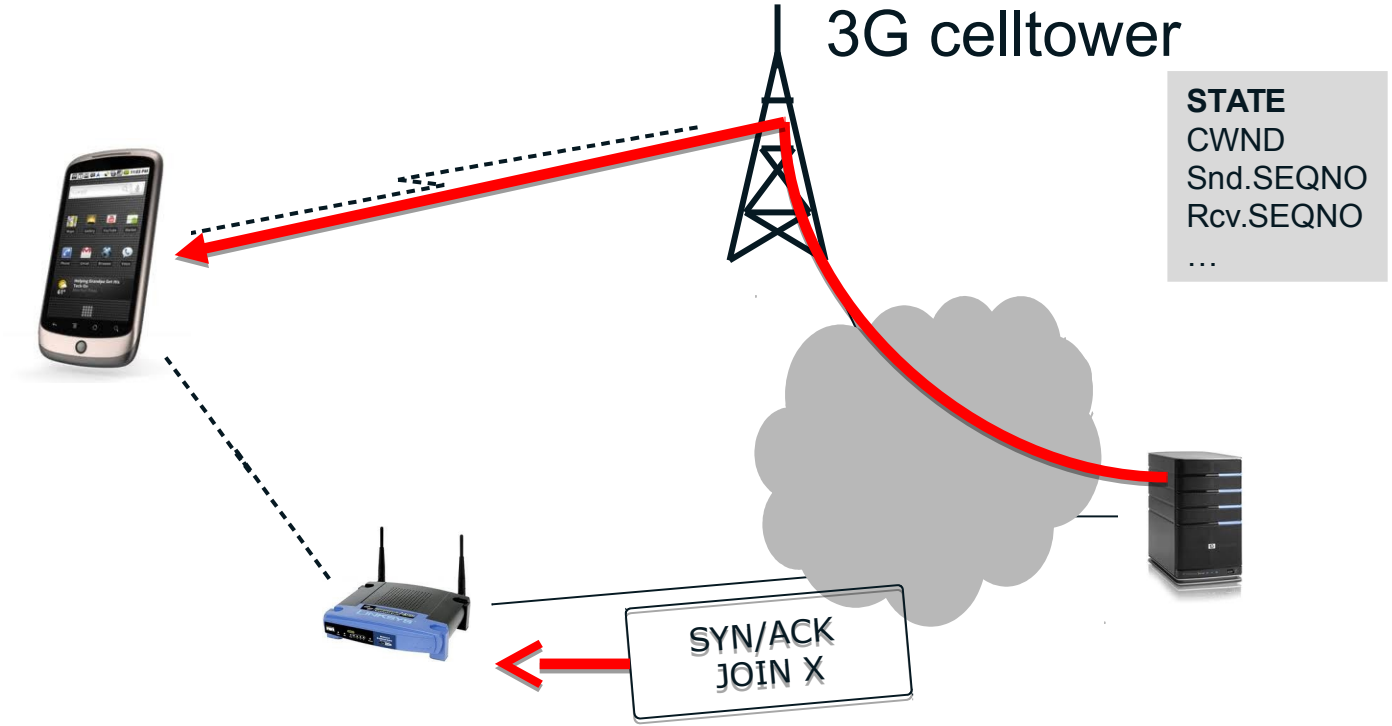
MPTCP basics



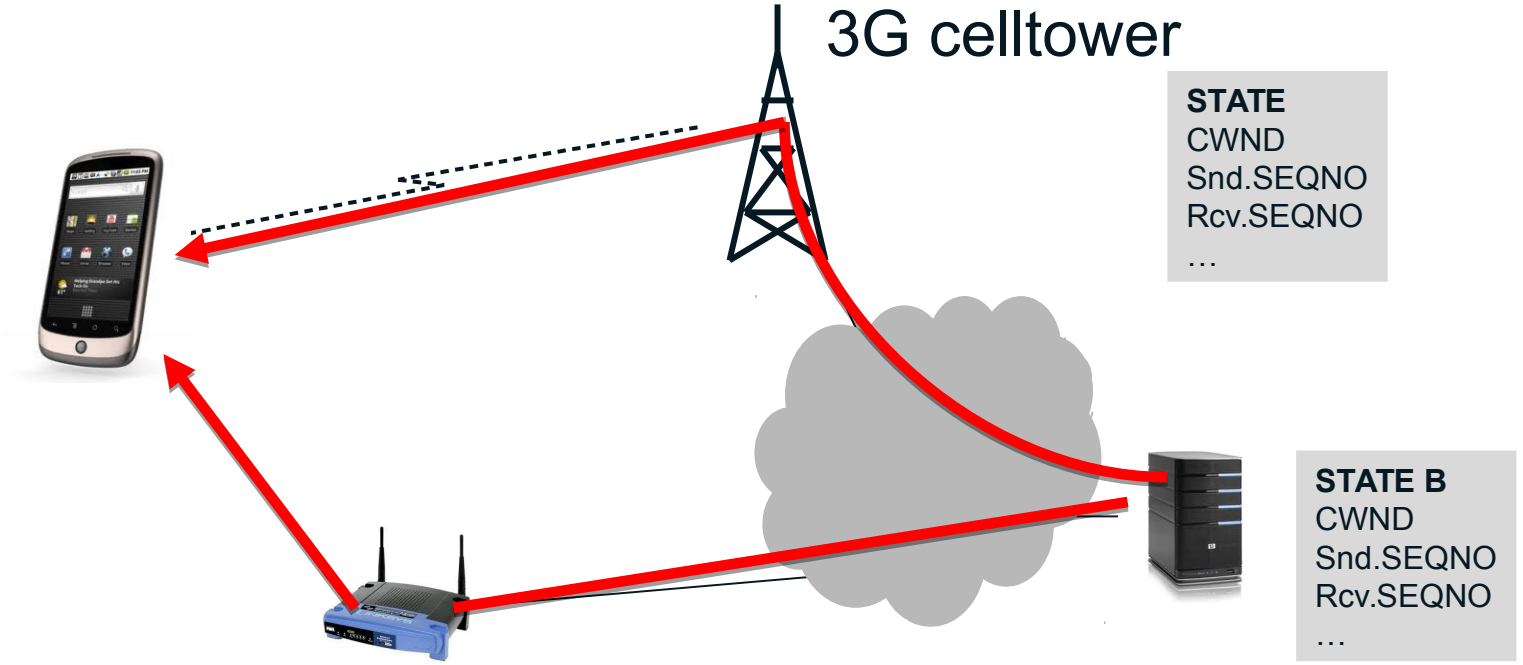
MPTCP basics



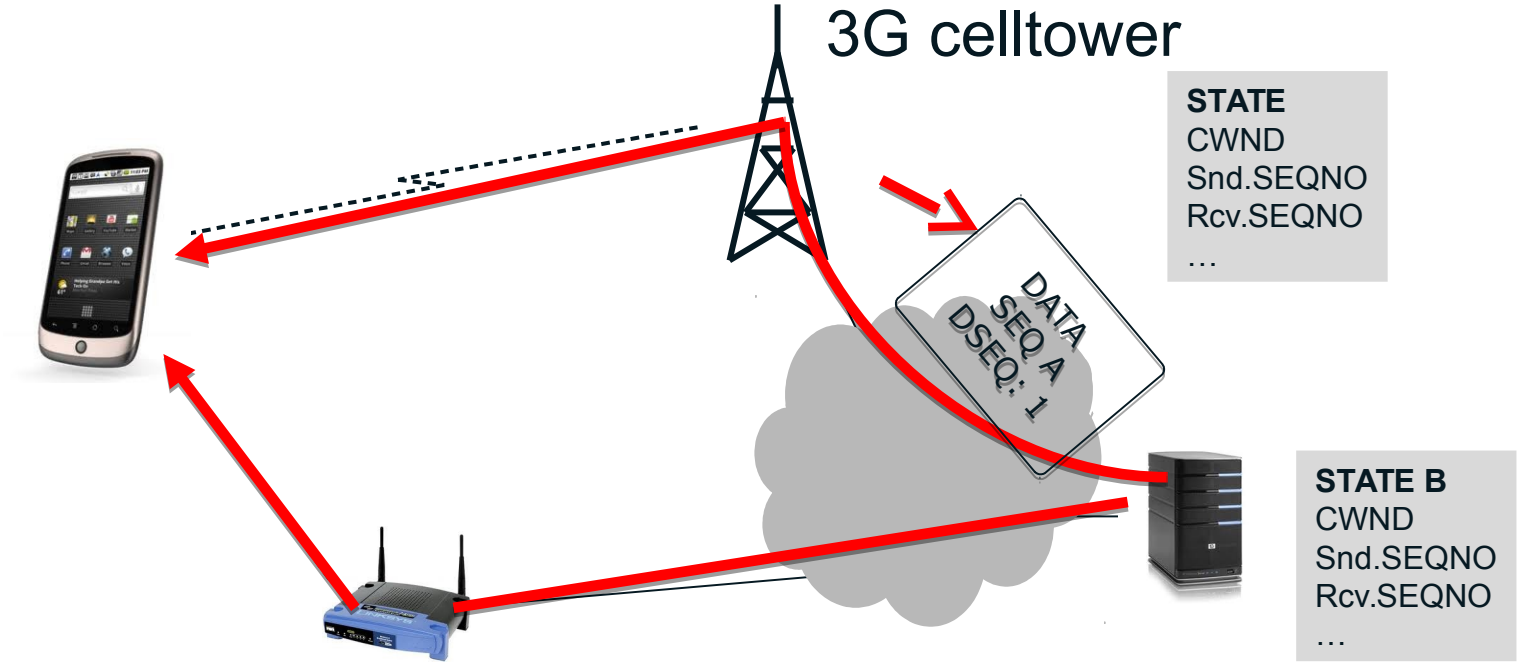
MPTCP basics



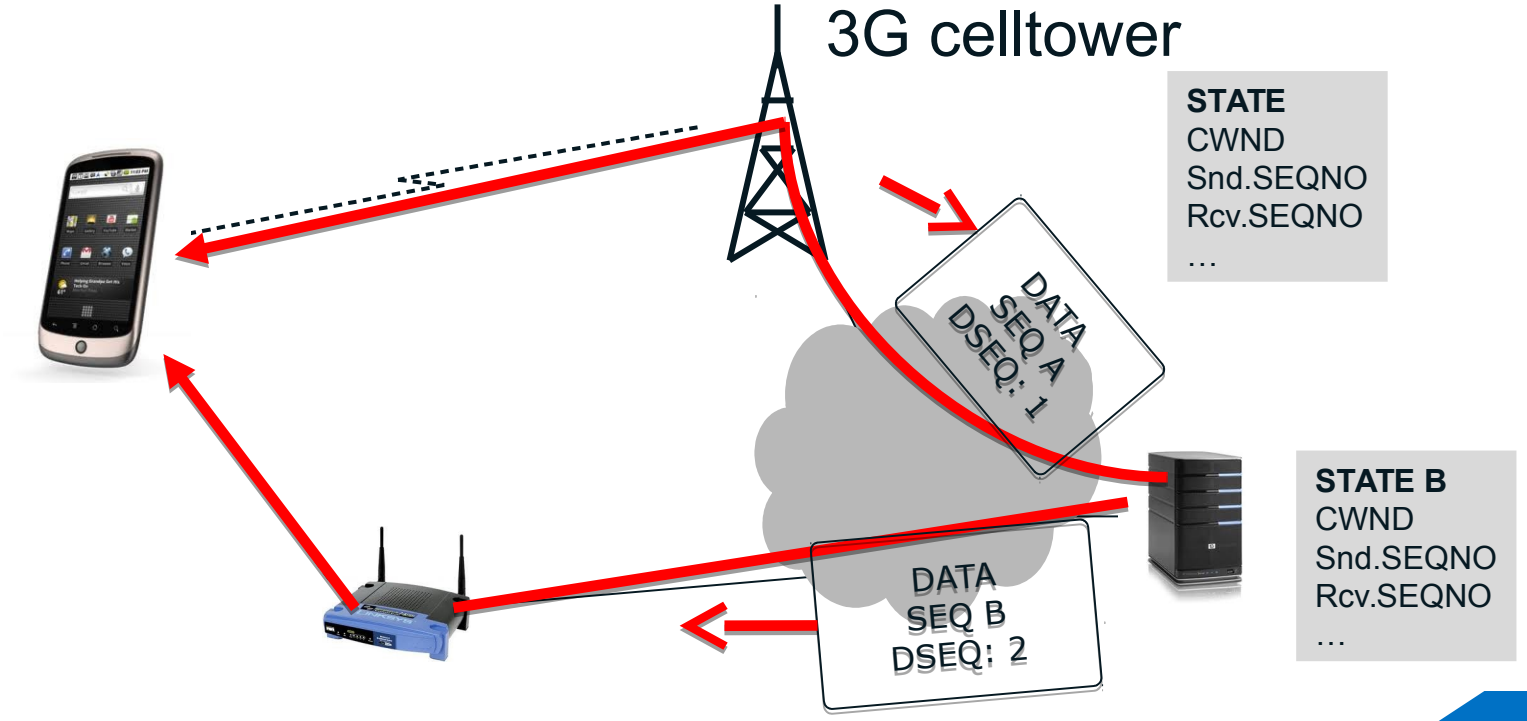
MPTCP basics



MPTCP basics



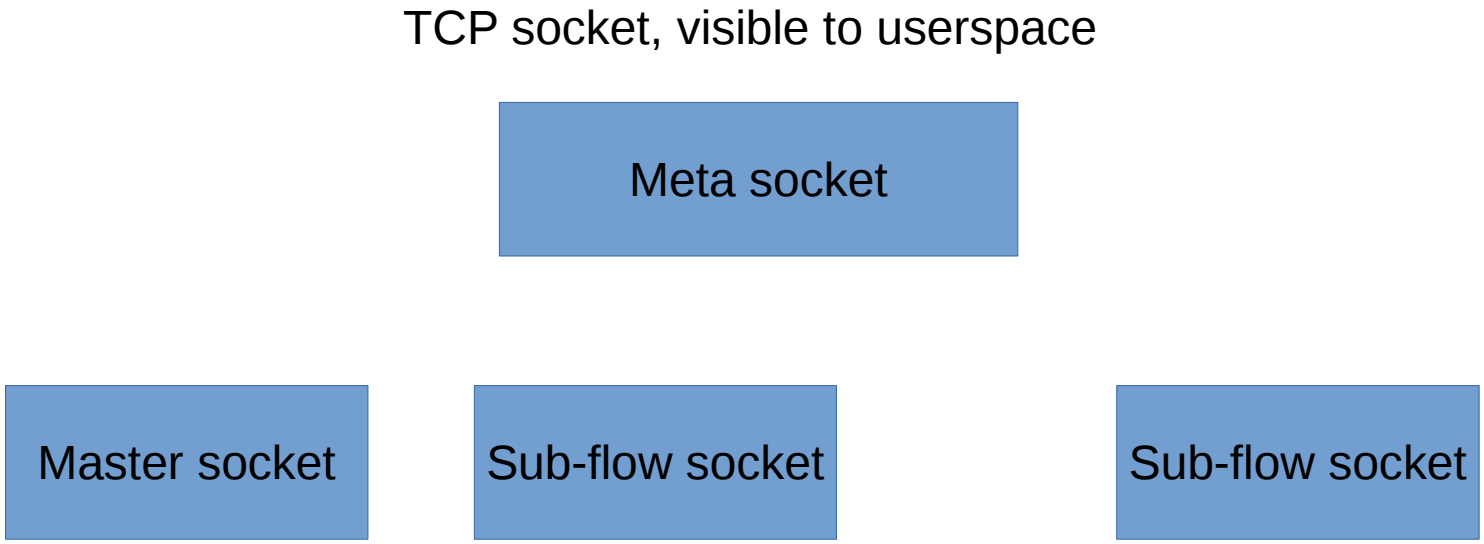
MPTCP basics



Initial Linux kernel implementation

- Implementation done directly at the TCP level
- Good performance
- Low overhead when falling back to TCP
- Intrusive changes that increases the complexity of the TCP stack

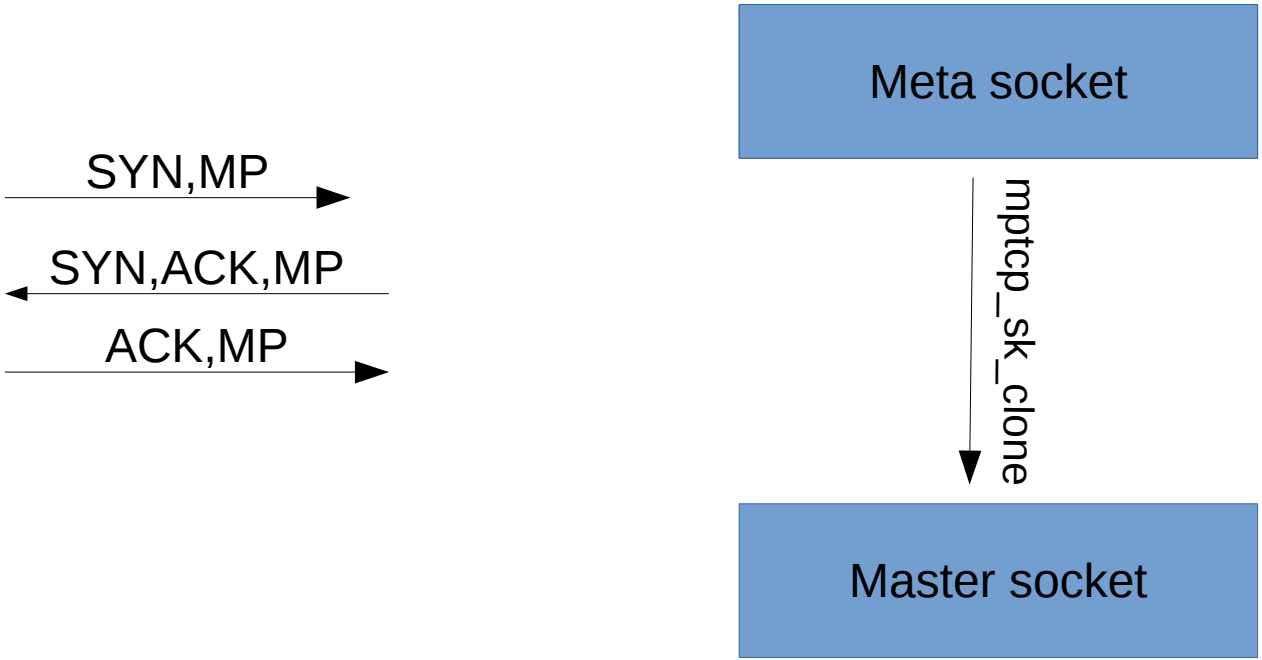
Key MPTCP structures



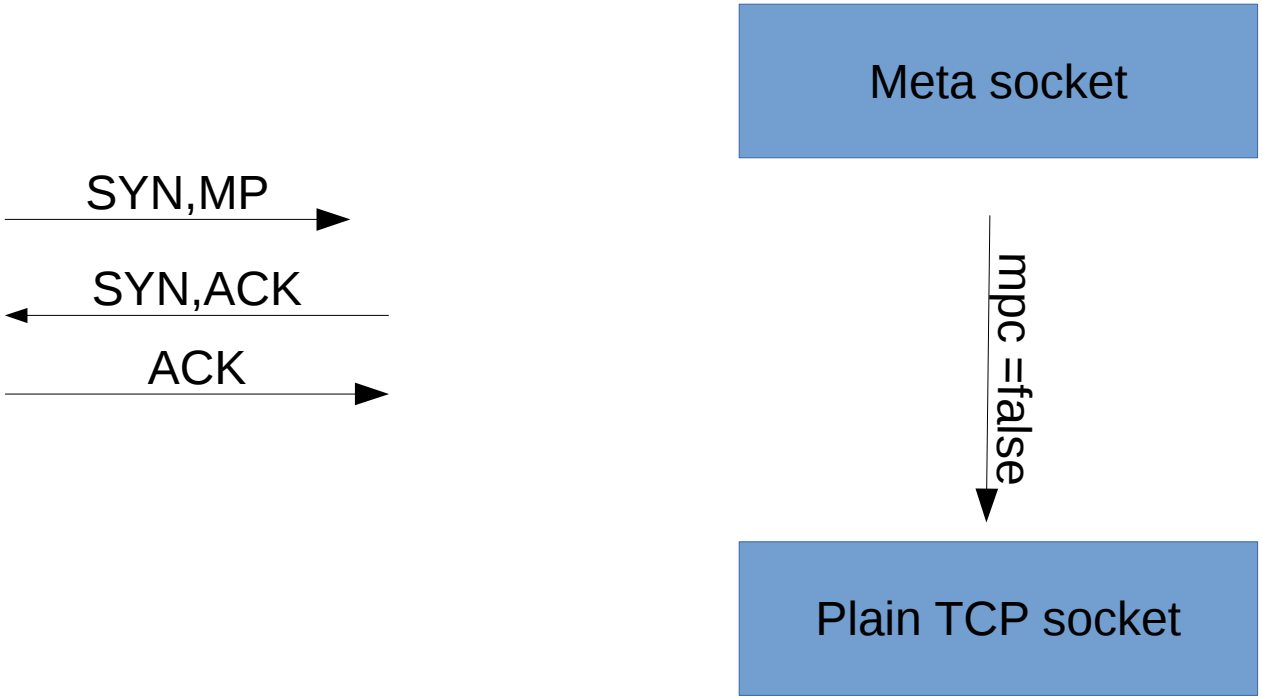
Pros/cons of using TCP sockets

- Re-use TCP code that transfers data to/from userspace (tcp_sendmsg, tcp_recvmsg)
- Makes MPTCP transparent to the application (including fallback)
- Some TCP functions now must deal with 3 cases
  - TCP socket
  - MPTCP sub-flow socket
  - MPTCP meta socket

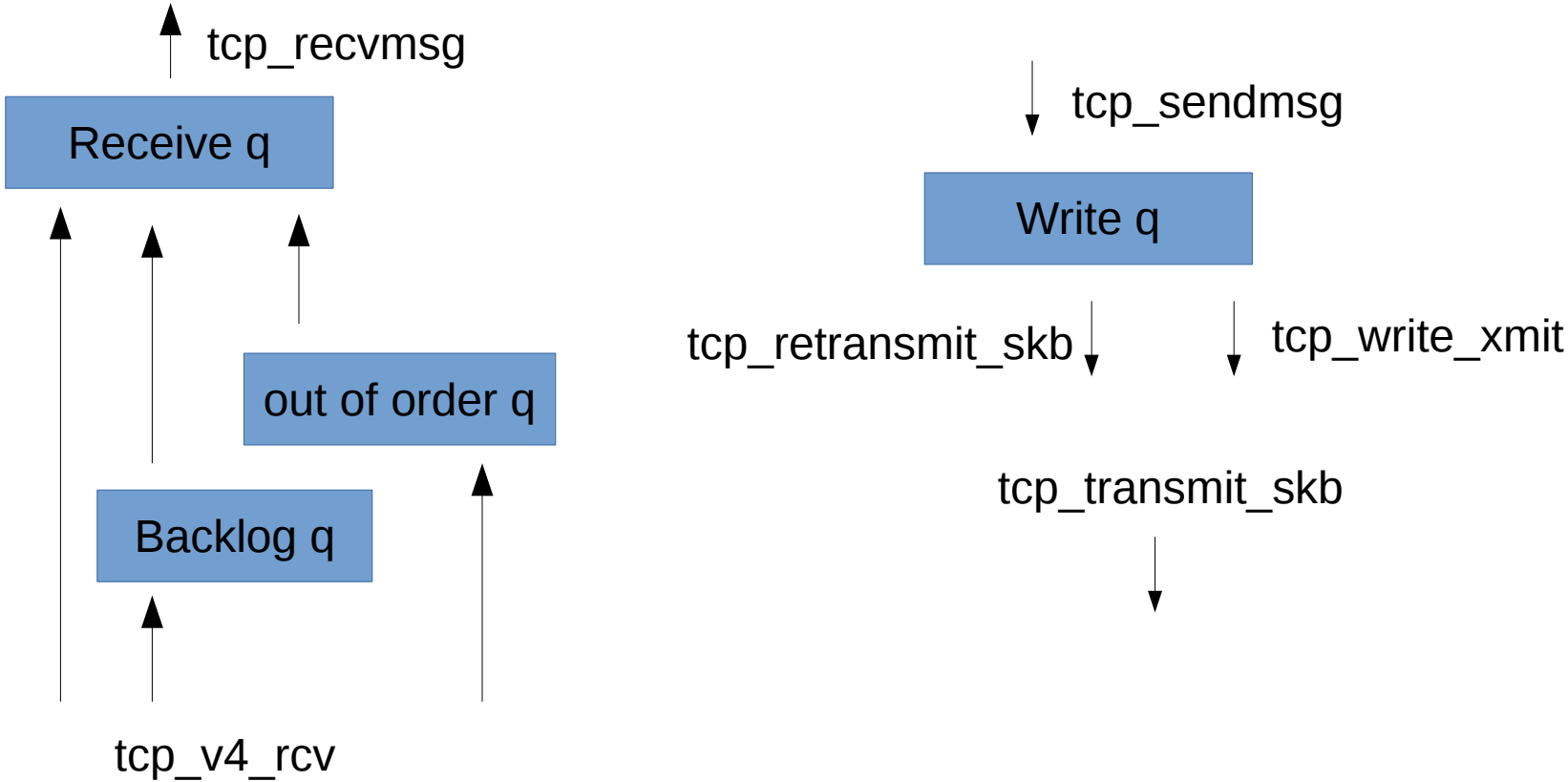
Creating the master socket



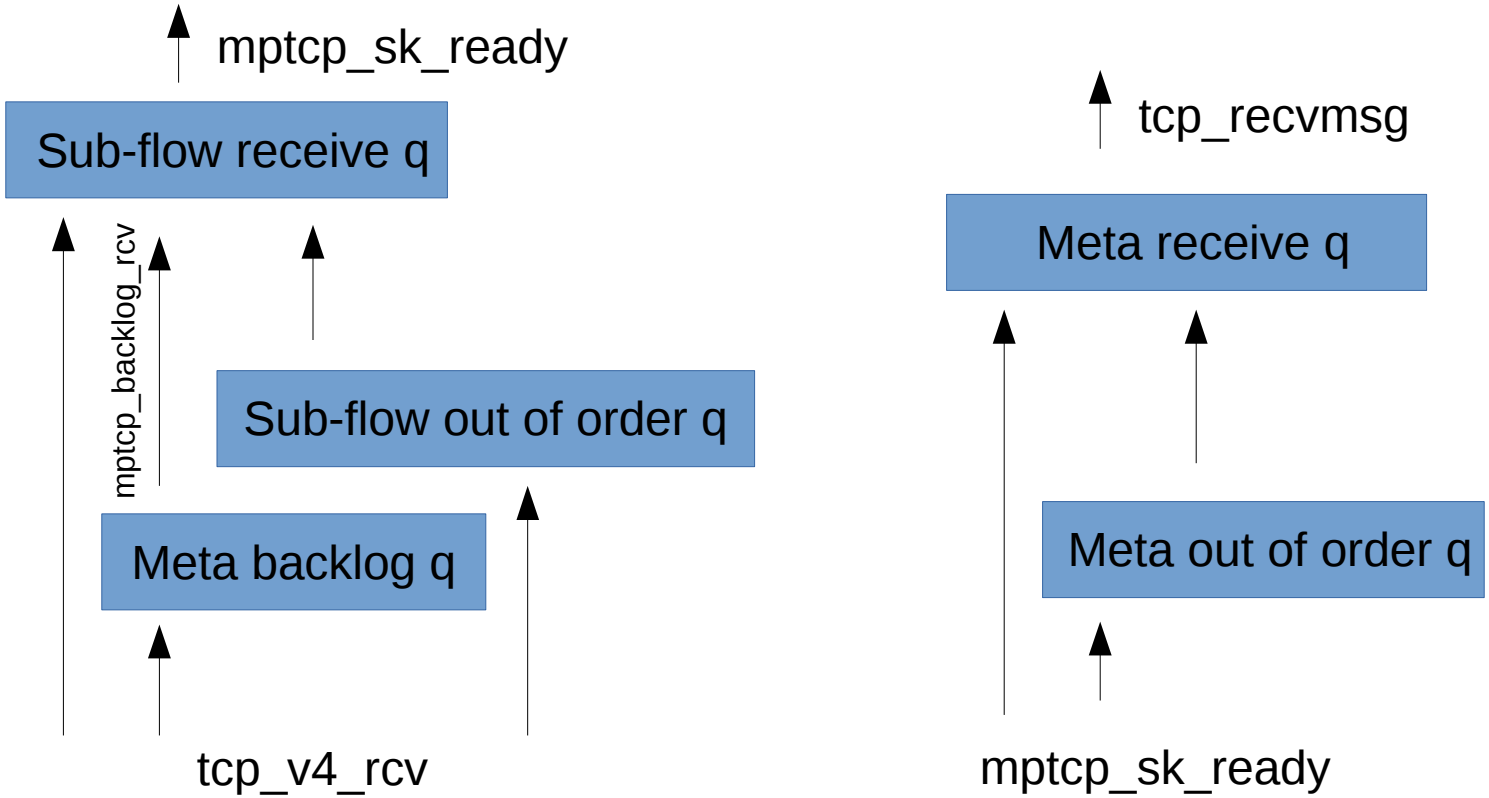
Fallback to TCP



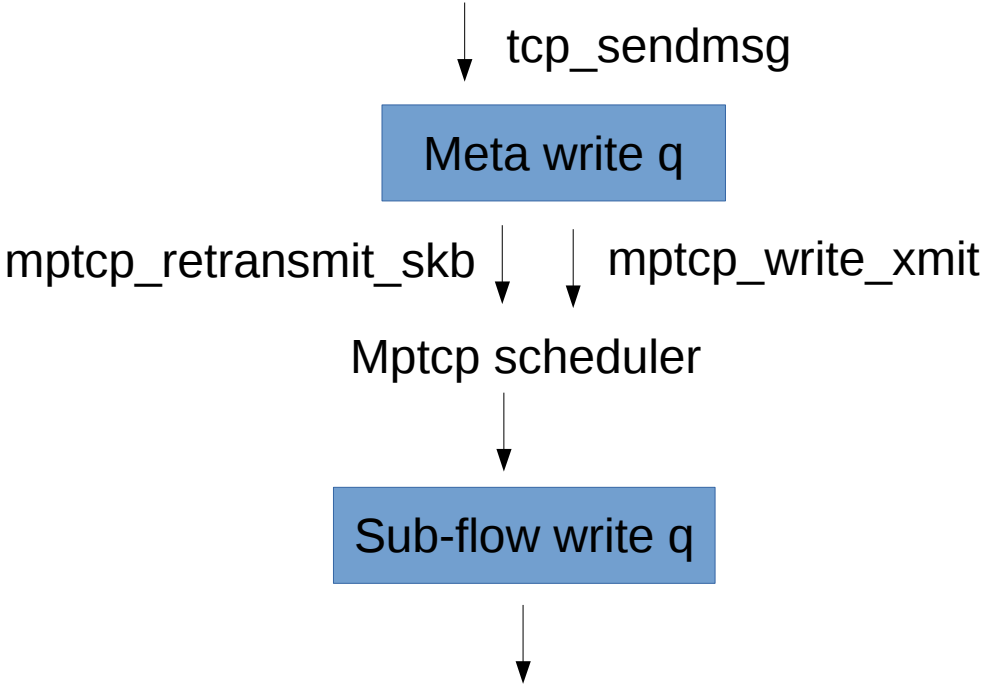
TCP queues (simplification)



MPTCP receive path

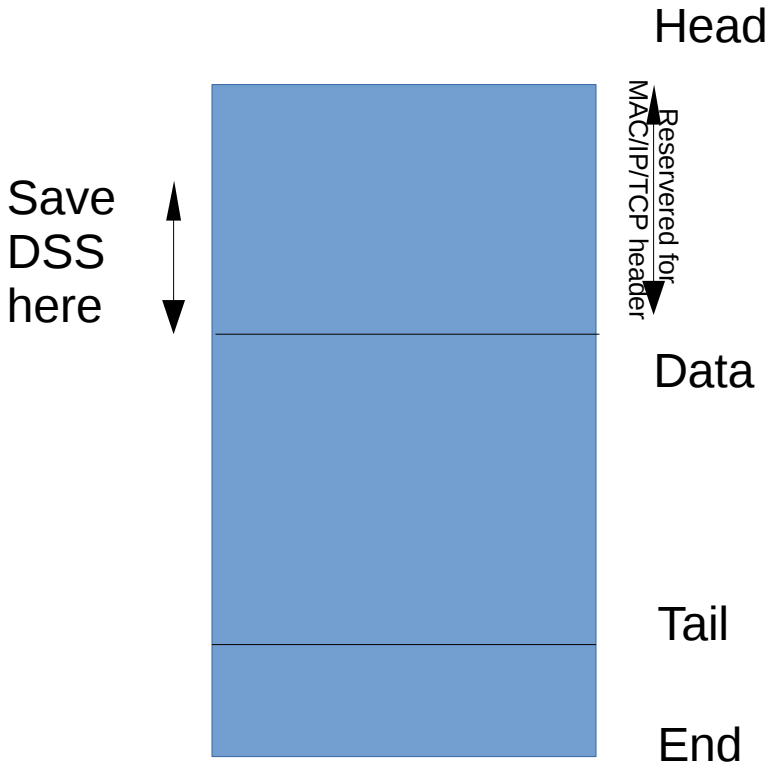


MPTCP send path



MPTCP DSS options

- Maps meta seq to sub-flow seq
- 20 bytes
- Does not fit into skb->cb
- Save them in the skb data in the space reserved for the TCP header
- Everytime pskb_copy() is called from the TCP stack we need to copy DSS manually



Alternative approaches

- Userspace implementation
  - Requires infrastructure to pass / receive MPTCP options from userspace
    - Use control messages for sendmsg/recvmsg
    - Needs new userspace ABIs to handle the connection handshake
  - Complete rewrite
- Create a separate layer on top of TCP (similar with how NFS, CIFS uses TCP)

Towards a separate MPTCP layer

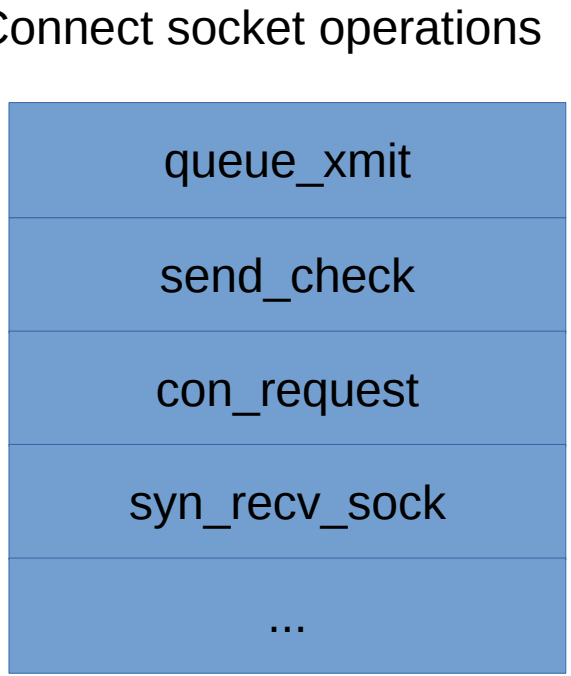
- Eliminate the branches in the TCP code that deals with MPTCP sub-flow and meta sockets
  - Isolate the MPTCP sub-flow functionality
  - MPTCP meta sockets are not TCP sockets – create a new protocol to deal with them
- Eliminate code duplication between TCP and MPTCP

MPTCP sub-flow specific code in TCP

- MPTCP connection hand-shake
  - Client side
  - Server side
- MPTCP receive window
- MPTCP send and receive path hooks
- MPTCP coupled congestion code

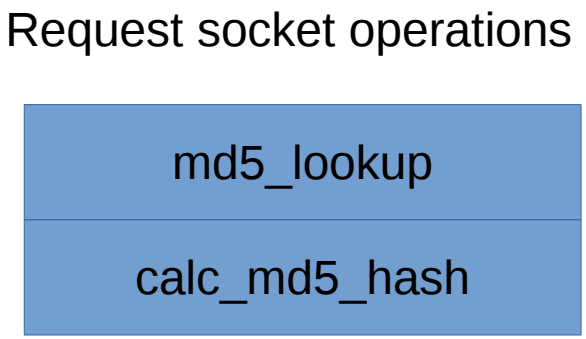
Isolate MPTCP connection handshake – TX

- Connection socket operations are used to abstract and isolate IPv4 and IPv6
- By defining MPTCP specific connection socket operation we isolated the TX part of MPTCP sub-flow handshake



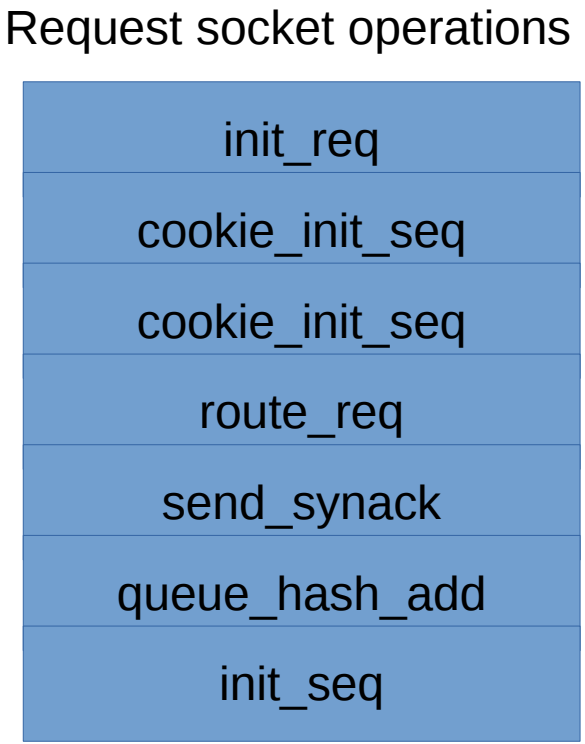
Isolate MPTCP connection handshake – RX

- On the receive side request socket operations are used to abstract MD5 code
- Unfortunately these operations are not enough to isolate MPTCP code but...
- We noticed significant code duplication between the IPv4 and IPv6 paths



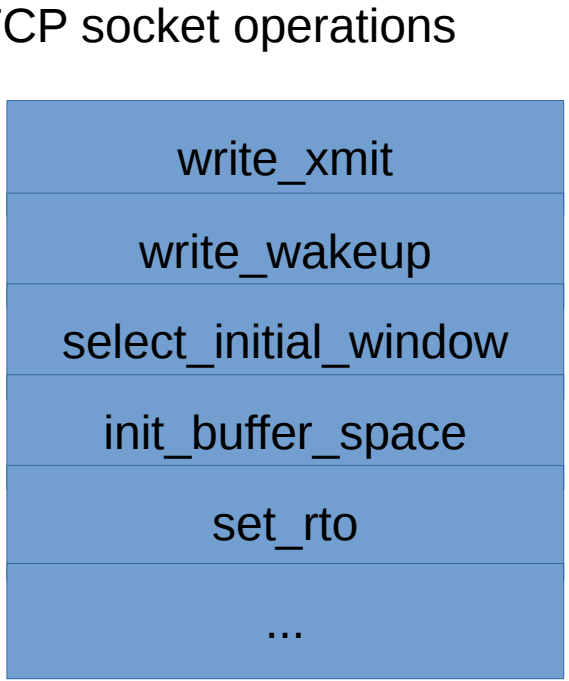
Isolate MPTCP connection handshake – RX

- Added new operations to abstract and isolate the IPv4 and IPv6 paths
- With that we also isolated the RX part of MPTCP sub-flow handshake



Isolate MPTCP receive window and send path

- Introduce a new structure to abstract some TCP socket operations
- Specific operations for the meta socket, sub-flow socket and regular TCP



git diff v3.18..mptcp_trunk --stat (>10)

```
include/linux/tcp.h                |  85 +-
include/net/sock.h                 |  11 +
include/net/tcp.h                  | 194 ++-
net/core/sock.c                    |  35 +-
net/ipv4/af_inet.c                 |  27 +-
net/ipv4/inet_connection_sock.c    |  21 +-
net/ipv4/tcp.c                     | 182 ++-
net/ipv4/tcp_fastopen.c            |  28 +-
net/ipv4/tcp_input.c               | 320 +++-
net/ipv4/tcp_ipv4.c                | 202 ++-
net/ipv4/tcp_minisocks.c           |  95 +-
net/ipv4/tcp_output.c              | 254 ++--
net/ipv4/tcp_timer.c               |  81 +-
net/ipv6/tcp_ipv6.c                | 274 +++-
```

Separate MPTCP meta layer

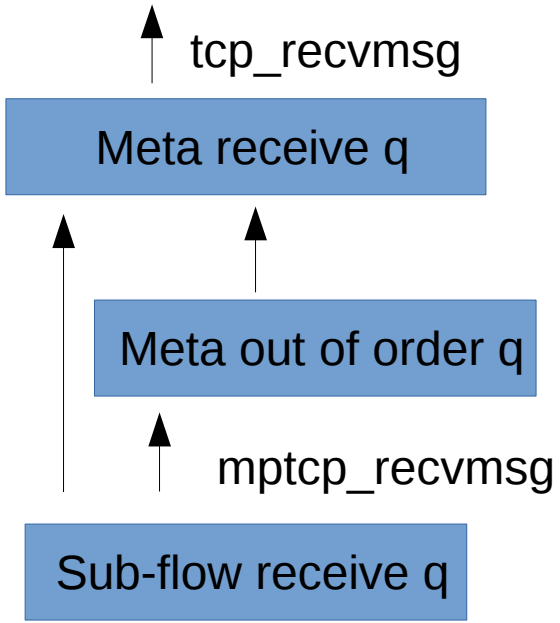
- Rationale: the meta socket is not a TCP socket
- Create a new IP protocol level socket for MPTCP

```
socket(AF_INET, SOCK_STREAM, IPROTO_TCP|TCPEXT_MPTCP)
```

WIP: early allocation of the master socket

- Allocate the master socket as soon as the meta socket is created
- Falling back to TCP adds overhead as we now go through the meta socket
- MPTCP is not transparent at the application level*
- Simplifies the connect path:
  - Connect of meta socket translates to connect on master socket
  - Avoids cloning the meta socket and changes in the inet connection layer

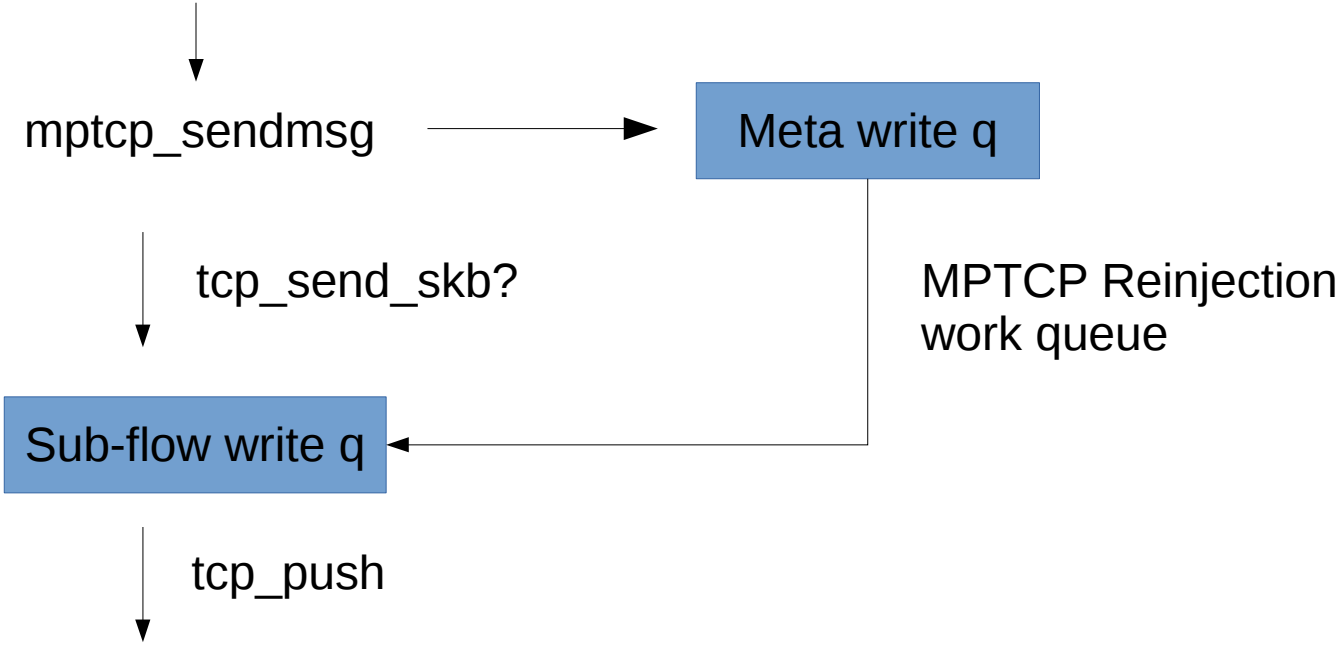
WIP: receive path




```
sk_data_ready (on sub-flow sockets)
    scan the rx queue update DSS mapping
    mark sub-flow and wake-up meta socket

mptcp_recvmsg (on meta socket)
    wait_event(wq, ready_sub-flows)
    for all ready sub-flows
        lock sock(sub-flow)
        tcp_read_sock(sub-flow) - recv actor
            clone SKB and add to meta socket
        clear sub-flow ready bit
        release_sock(sub-flow)
    tcp_recvmsg (meta socket, O_NONBLOCK)
```

WIP: send path



Conclusions

- MPTCP has interesting use-cases and it is used commercially
- The initial Linux kernel implementation had large TCP stack changes
- We have been steadily reducing changes to the TCP stack
- We believe a separate MPTCP layer should help us reduce TCP changes even more and help us manage the complexity

Thank you!