MultiPath TCP : Linux Kernel implementation

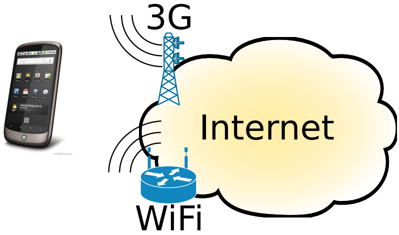
Presenter: Christoph Paasch
IP Networking Lab
UCLouvain, Belgium

August 28, 2012

http://multipath-tcp.org

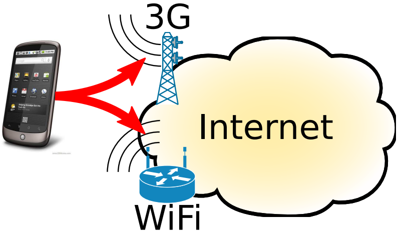
Networks are becoming Multipath

Mobile devices can connect to the Internet via different interfaces



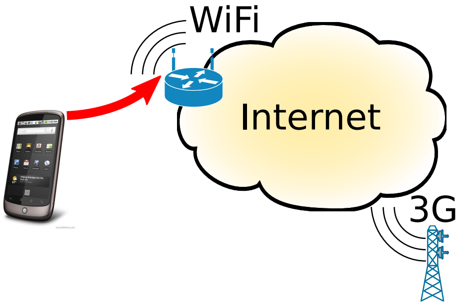
Networks are becoming Multipath

Mobile devices can connect to the Internet via different interfaces



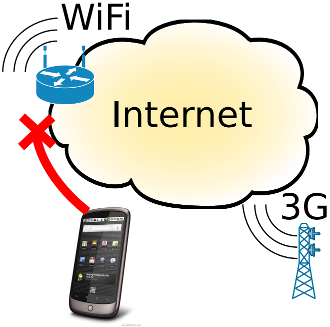
However, TCP does not support this

Smartphones have to restart their data-transfer when moving away from the WiFi access-point.



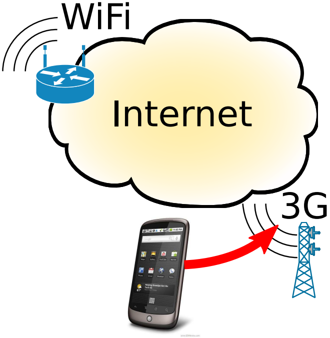
However, TCP does not support this

Smartphones have to restart their data-transfer when moving away from the WiFi access-point.



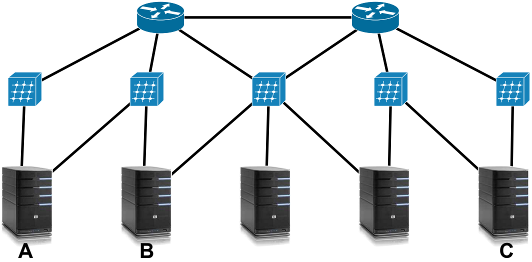
However, TCP does not support this

Smartphones have to restart their data-transfer when moving away from the WiFi access-point.



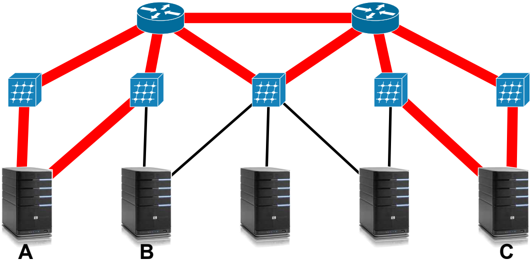
Networks are becoming Multipath

Data-centers have a large redundant infrastructure



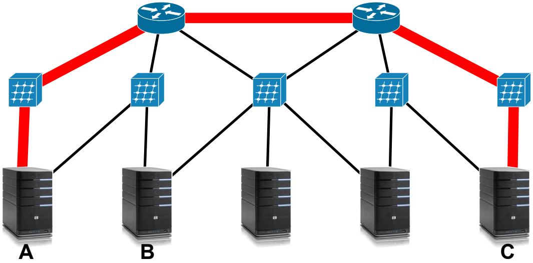
Networks are becoming Multipath

Data-centers have a large redundant infrastructure



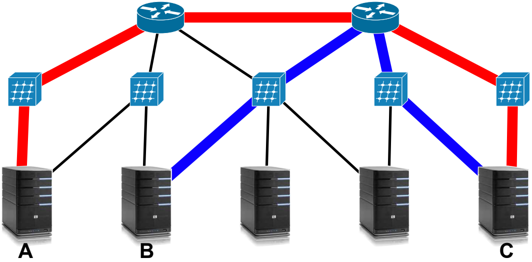
However, TCP is suboptimal

Collisions in data-center reduce the bandwidth and result in suboptimal load-balancing



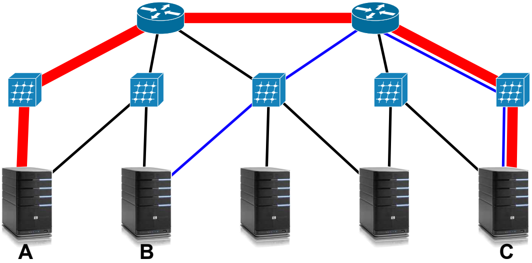
However, TCP is suboptimal

Collisions in data-center reduce the bandwidth and result in suboptimal load-balancing



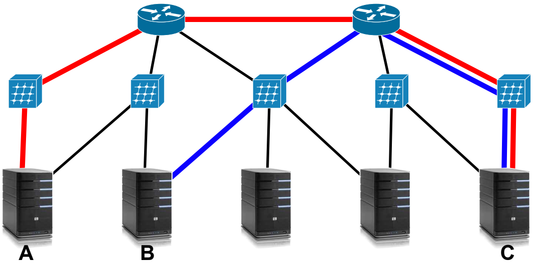
However, TCP is suboptimal

Collisions in data-center reduce the bandwidth and result in suboptimal load-balancing



However, TCP is suboptimal

Collisions in data-center reduce the bandwidth and result in suboptimal load-balancing

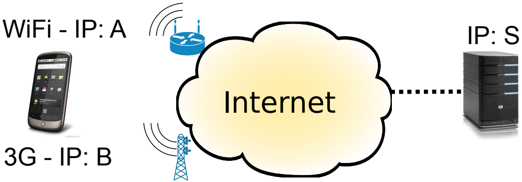


Mismatch between the **multipath network** and the **single-path transport** protocol.

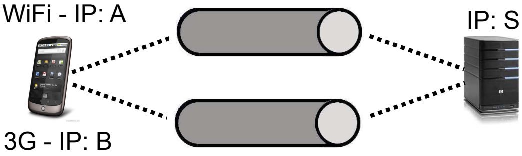
MultiPath TCP

- Runs with **unmodified applications**
- Works over **today's Internet**
- **IPv4/IPv6** are both supported (even simultaneously)

MultiPath TCP

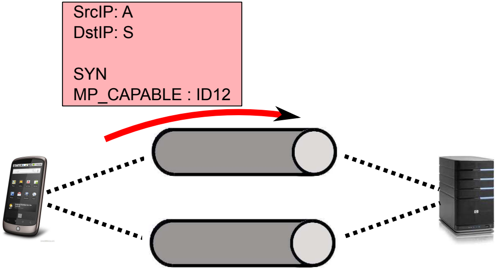


MultiPath TCP



MultiPath TCP

Is the server MPTCP-capable?

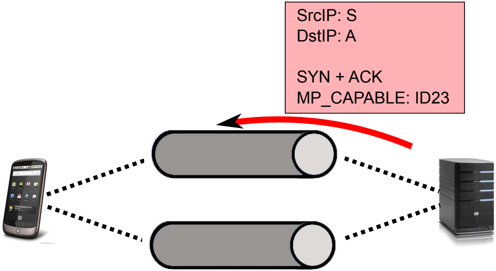


SrcIP: A
DstIP: S

SYN
MP_CAPABLE : ID12

MultiPath TCP

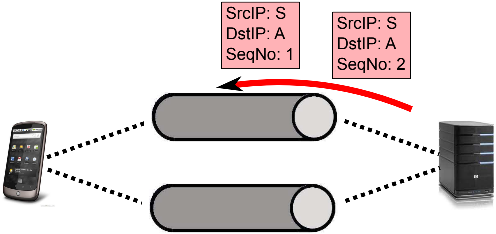
Is the server MPTCP-capable?



SrcIP: S
DstIP: A

SYN + ACK
MP_CAPABLE: ID23

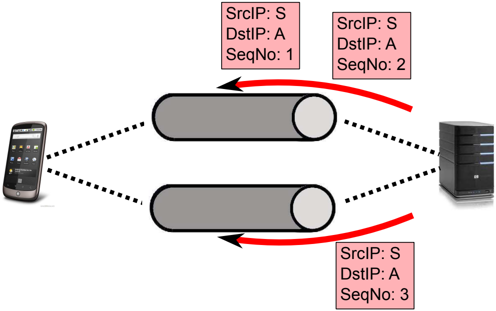
Sending Data - Naiv approach



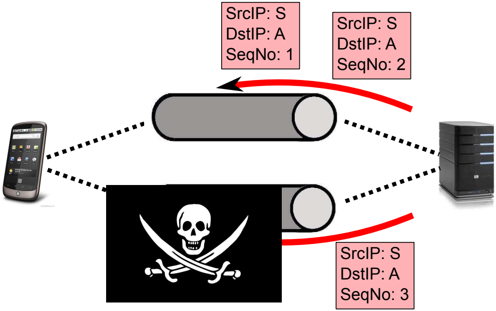
SrcIP: S
DstIP: A
SeqNo: 1

SrcIP: S
DstIP: A
SeqNo: 2

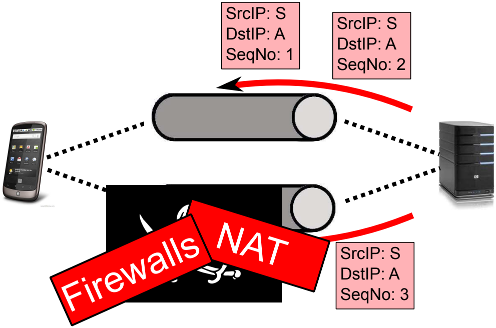
Sending Data - Naiv approach



Sending Data - Naiv approach



Sending Data - Naiv approach

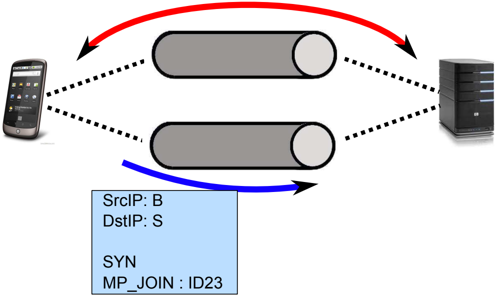


SrcIP: S
DstIP: A
SeqNo: 1

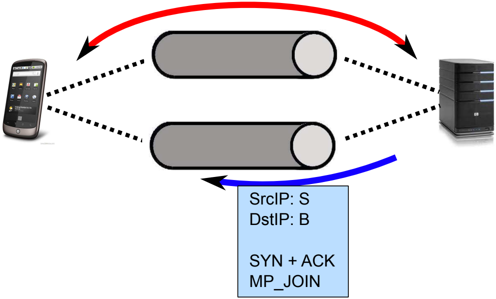
SrcIP: S
DstIP: A
SeqNo: 2

SrcIP: S
DstIP: A
SeqNo: 3

Establish separate subflows

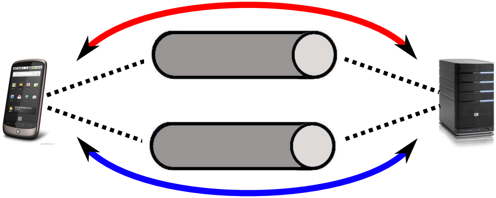


Establish separate subflows



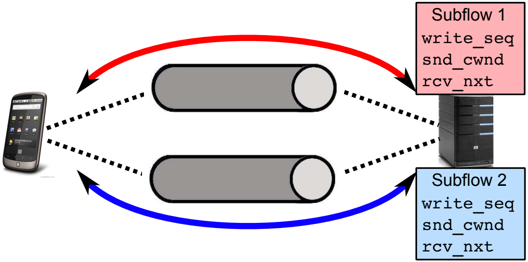
MultiPath TCP

Establish separate subflows

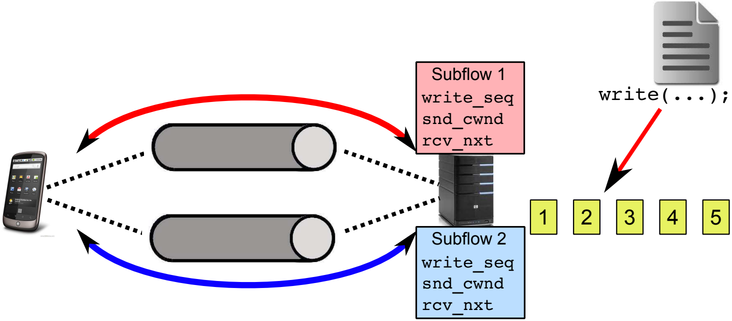


MultiPath TCP

Establish separate subflows

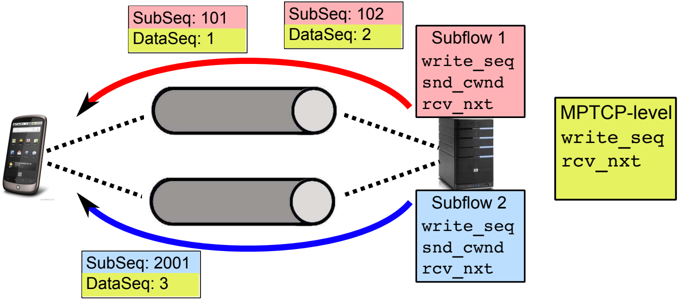


Sending Data



MultiPath TCP

Sending Data



MultiPath TCP

- Both subflows can be used simultaneously.
- Addresses are advertised with `ADD_ADDR` and removed by `REMOVE_ADDR`.
- Subflows can be dynamically added and removed during the lifetime of the connection.

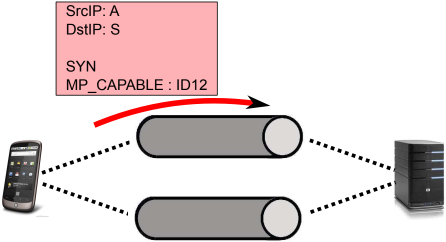
MultiPath TCP

Linux Kernel Implementation

Available at http://multipath-tcp.org

Establishing first subflow

Exchanged Messages

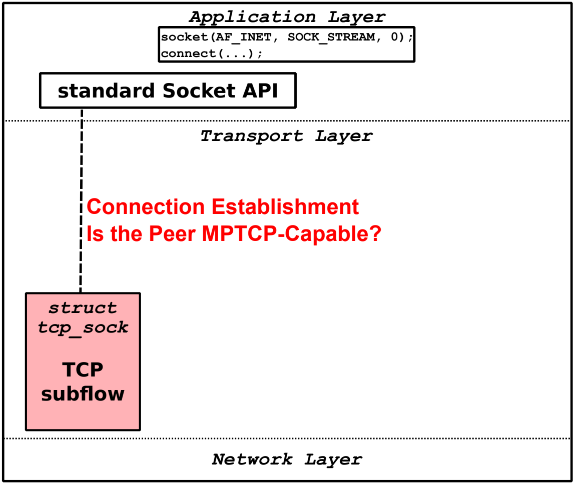


SrcIP: A
DstIP: S

SYN
MP_CAPABLE : ID12

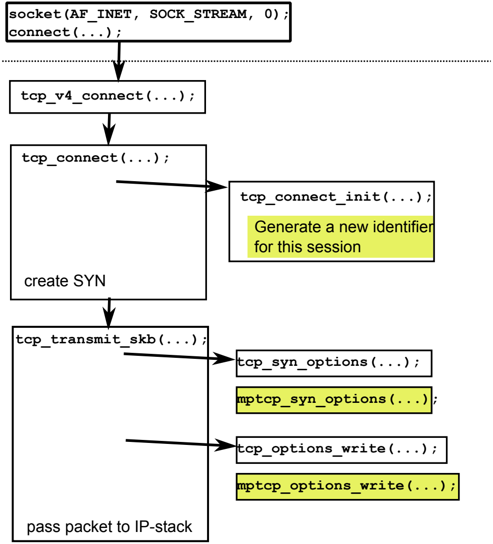
Establishing first subflow

High-Level Kernel design - Client Side



Establishing first subflow

In-depth call-stack - Client Side



```
socket(AF_INET, SOCK_STREAM, 0);
connect(...);
```

```
tcp_v4_connect(...);
```

```
tcp_connect(...);
```

create SYN

```
tcp_connect_init(...);
```

Generate a new identifier for this session

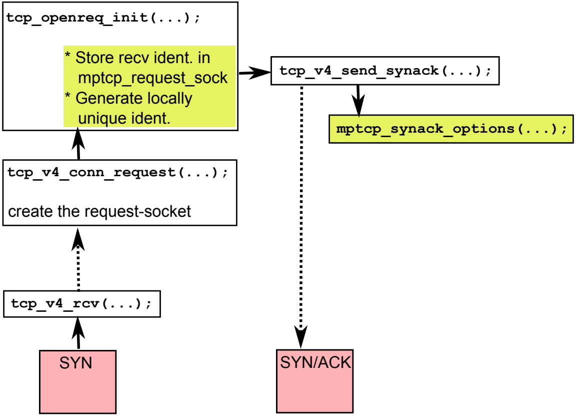
```
tcp_transmit_skb(...);
```

pass packet to IP-stack

```
tcp_syn_options(...);
mptcp_syn_options(...);
tcp_options_write(...);
mptcp_options_write(...);
```

Establishing first subflow

In-depth call-stack - Server Side



tcp_openreq_init(...);

* Store recv ident. in mptcp_request_sock
* Generate locally unique ident.

tcp_v4_send_synack(...);

mptcp_synack_options(...);

tcp_v4_conn_request(...);

create the request-socket

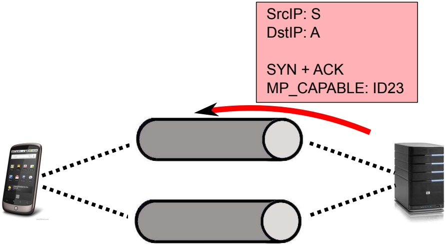
tcp_v4_rcv(...);

SYN

SYN/ACK

Establishing first subflow

Exchanged Messages

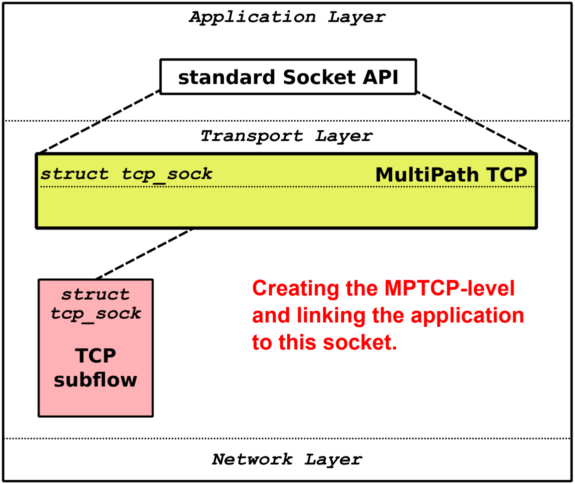


SrcIP: S
DstIP: A

SYN + ACK
MP_CAPABLE: ID23

Establishing first subflow

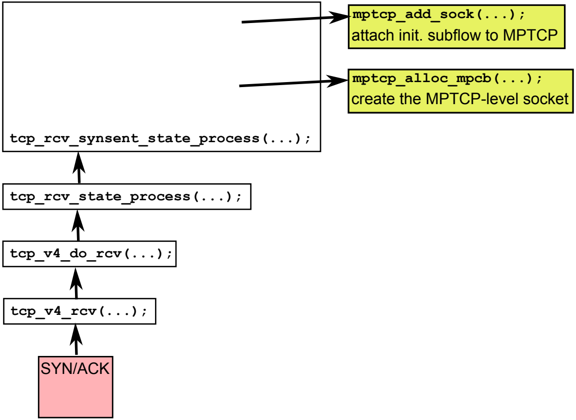
High-Level Kernel design - Client Side



Creating the MPTCP-level and linking the application to this socket.

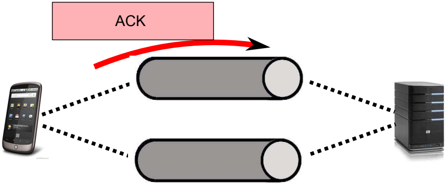
Establishing first subflow

In-depth call-stack - Client Side


mptcp_add_sock(...);
attach init. subflow to MPTCP
mptcp_alloc_mpcb(...);
create the MPTCP-level socket
tcp_rcv_synsent_state_process(...);
tcp_rcv_state_process(...);
tcp_v4_do_rcv(...);
tcp_v4_rcv(...);
SYN/ACK

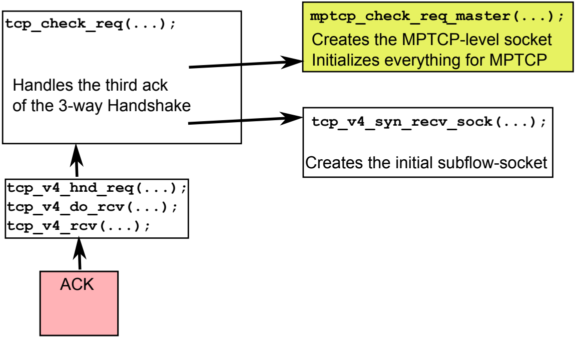
Establishing first subflow

Exchanged Messages



Establishing first subflow

In-depth call-stack - Server Side



tcp_check_req(...);

Handles the third ack of the 3-way Handshake

mptcp_check_req_master(...);

Creates the MPTCP-level socket

Initializes everything for MPTCP

tcp_v4_syn_recv_sock(...);

Creates the initial subflow-socket

tcp_v4_hnd_req(...);

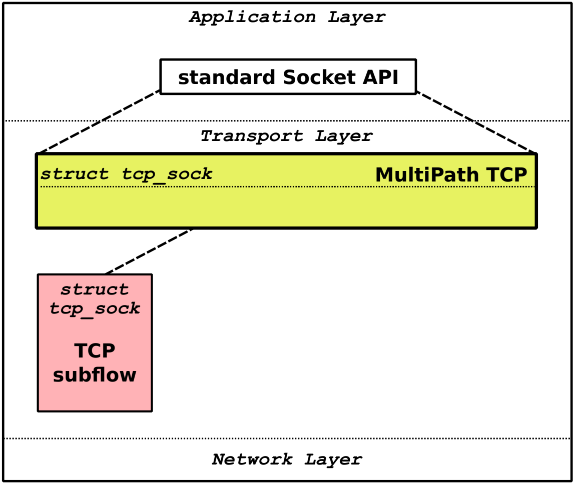
tcp_v4_do_rcv(...);

tcp_v4_rcv(...);

ACK

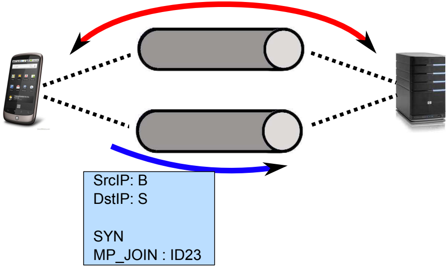
Establishing first subflow

High-Level Kernel design



Establishing additional subflows

Exchanged Messages

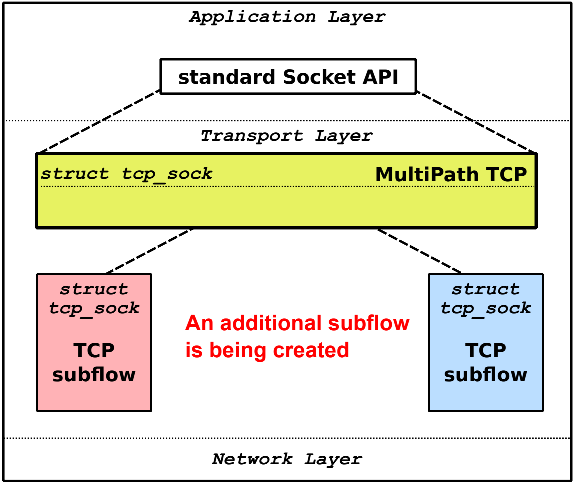


SrcIP: B
DstIP: S

SYN
MP_JOIN : ID23

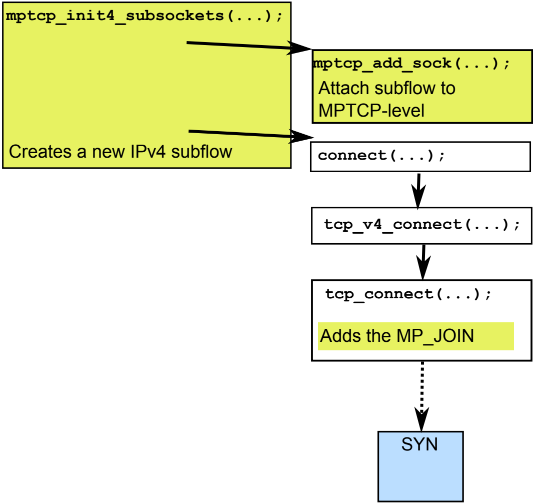
Establishing additional subflows

High-Level Kernel design - Client Side



Establishing additional subflows

In-depth call-stack - Client Side



mptcp_init4_subsockets(...);

Creates a new IPv4 subflow

mptcp_add_sock(...);

Attach subflow to MPTCP-level

connect(...);

tcp_v4_connect(...);

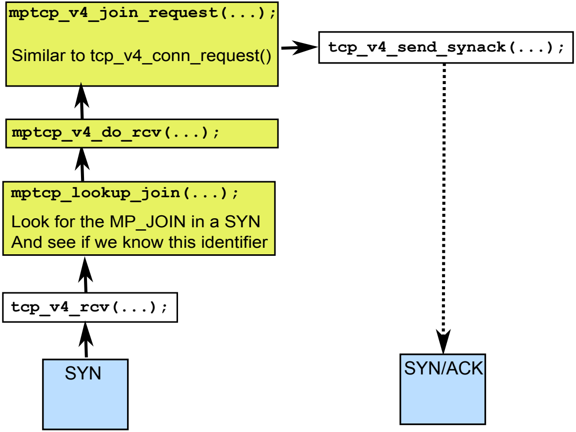
tcp_connect(...);

Adds the MP_JOIN

SYN

Establishing additional subflows

In-depth call-stack - Server Side



mptcp_v4_join_request(...);

Similar to tcp_v4_conn_request()

tcp_v4_send_synack(...);

mptcp_v4_do_rcv(...);

mptcp_lookup_join(...);

Look for the MP_JOIN in a SYN
And see if we know this identifier

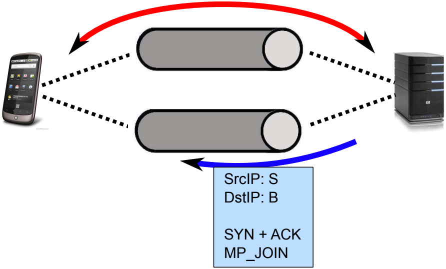
tcp_v4_rcv(...);

SYN

SYN/ACK

Establishing additional subflows

Exchanged Messages

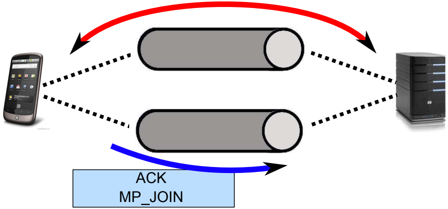


SrcIP: S
DstIP: B

SYN + ACK
MP_JOIN

Establishing additional subflows

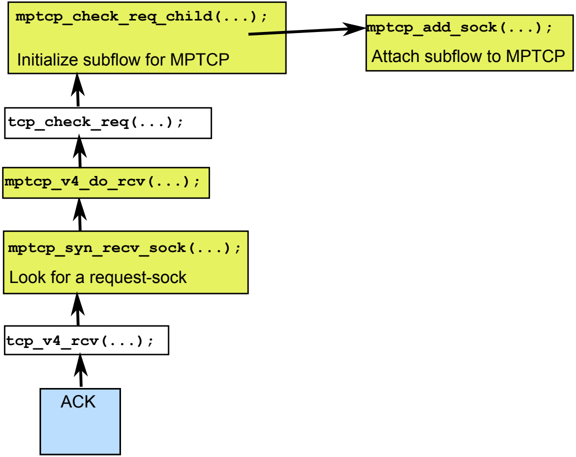
Exchanged Messages



ACK
MP_JOIN

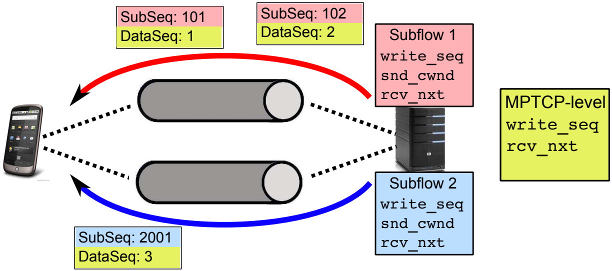
Establishing additional subflows

In-depth call-stack - Server Side



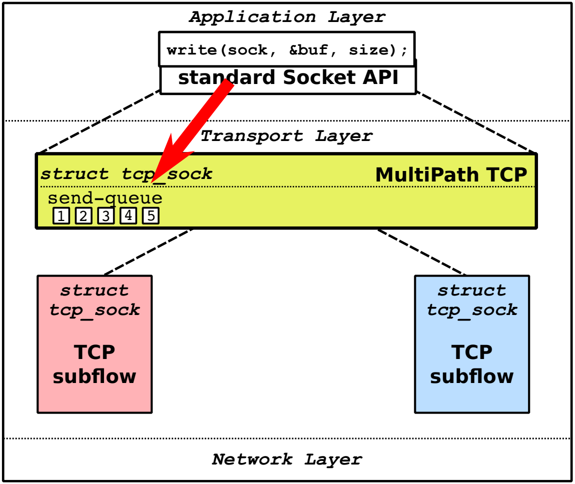
Sending Data

Exchanged Messages



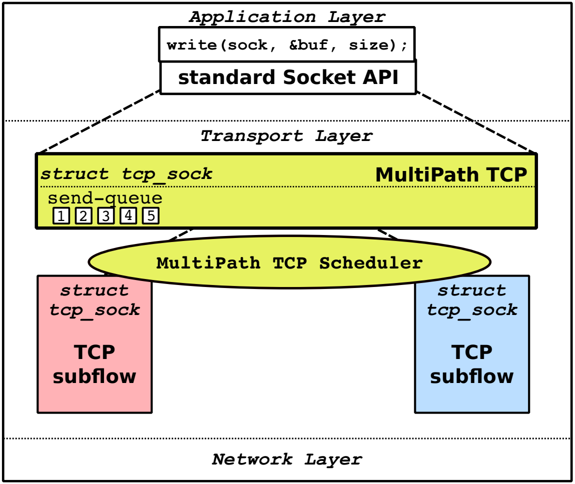
Sending Data

High-Level Kernel design

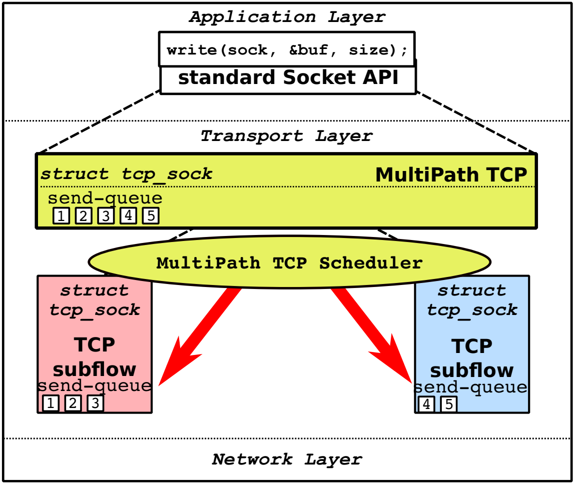


Sending Data

High-Level Kernel design

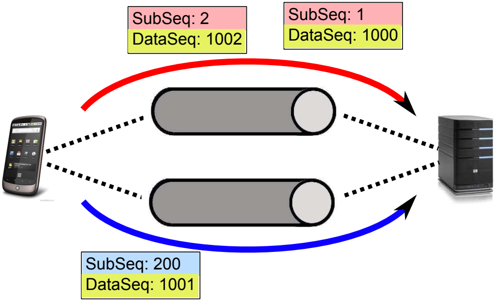


High-Level Kernel design



Receiving Data

Packets can be reordered at the data-level due to delay-differences.



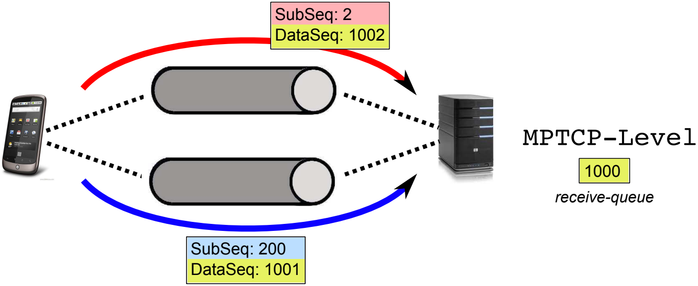
SubSeq: 2
DataSeq: 1002

SubSeq: 1
DataSeq: 1000

SubSeq: 200
DataSeq: 1001

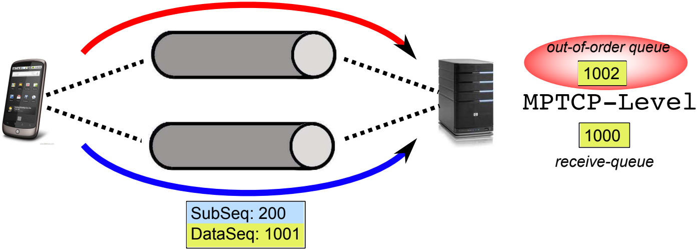
Receiving Data

Packets can be reordered at the data-level due to delay-differences.



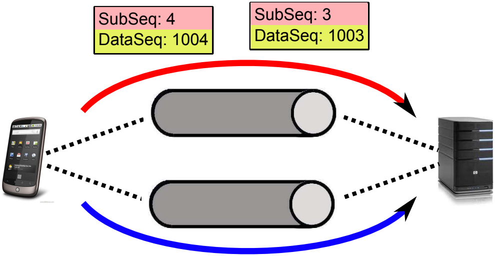
Receiving Data

Packets can be reordered at the data-level due to delay-differences.



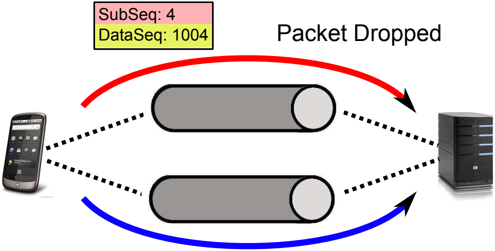
Receiving Data

A loss at the subflow-level (or network-reordering) can also cause reordering at the subflow-level



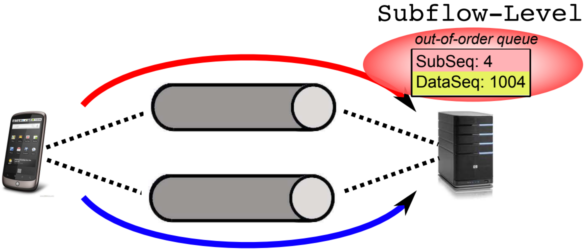
Receiving Data

A loss at the subflow-level (or network-reordering) can also cause reordering at the subflow-level



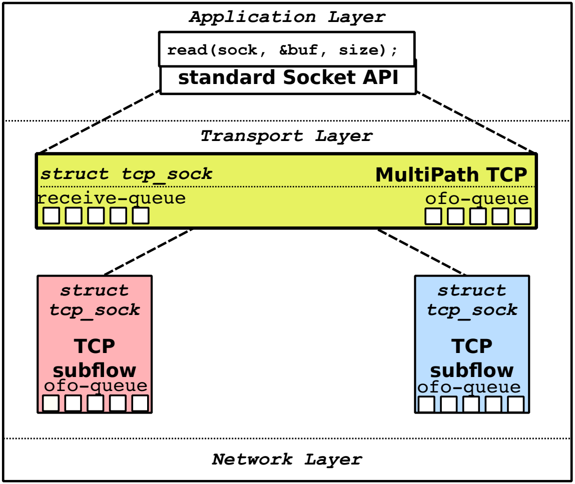
Receiving Data

Subflow-level out-of-order queues are necessary to handle the retransmission at the subflow-level



Receiving Data

High-Level Kernel design



MultiPath TCP

Design Challenges

Writing the Data-sequence number in the TCP-options

Naive approach

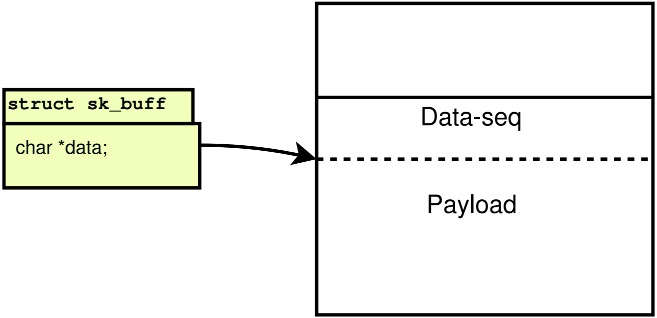
```
struct tcp_skb_cb {
        __u32            seq; /* Starting sequence number */
        [...]
#ifdef CONFIG_MPTCP
        __u32            data_seq;
        __u32            end_data_seq;
        __u32            data_ack;
       [...]
#endif
}
```

- Writing data-seq in *tcp_options_write*
- Increased *tcp_skb_cb* by 24 bytes.

Writing the Data-sequence number in the TCP-options

Our solution

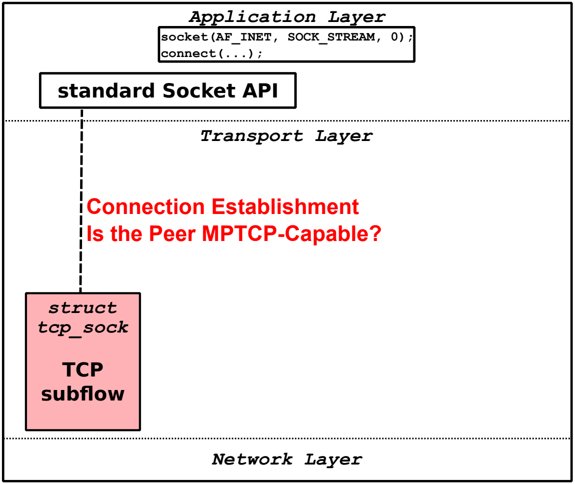
Inside the MPTCP-scheduler write the data-seq on top of the payload, **before** calling *tcp_transmit_skb*.



No more increase of *tcp_skb_cb*.

Creating the MPTCP-level on the client-side.

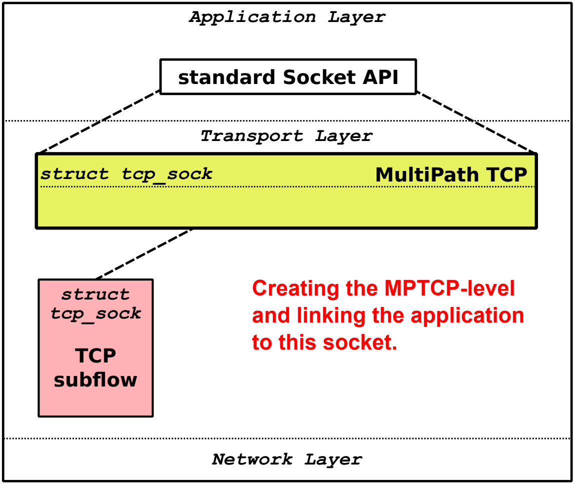
Current Design



```
socket(AF_INET, SOCK_STREAM, 0);
connect(...);
```

Creating the MPTCP-level on the client-side.

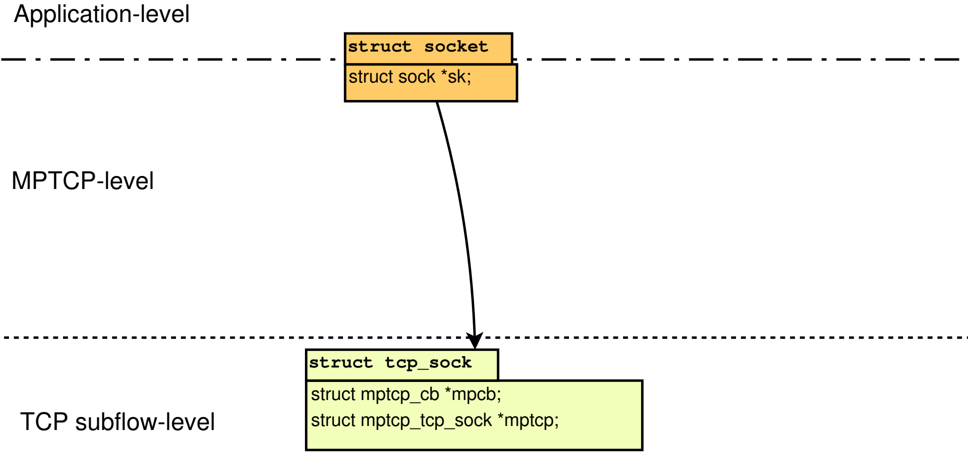
Current Design



Creating the MPTCP-level and linking the application to this socket.

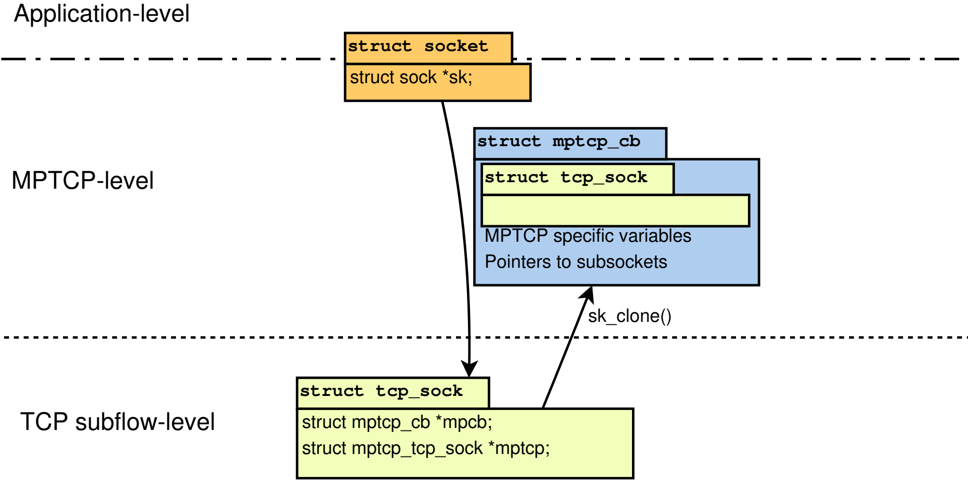
Creating the MPTCP-level on the client-side.

Current Design



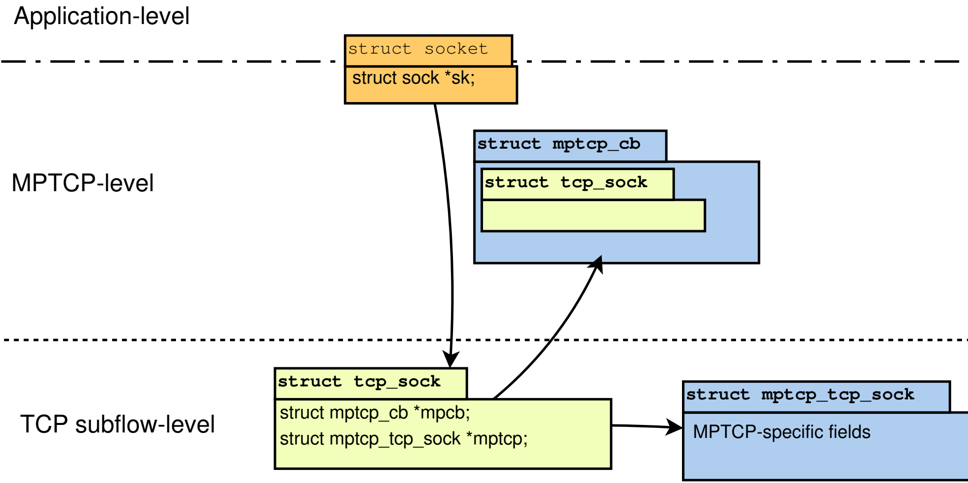
Creating the MPTCP-level on the client-side.

Current Design



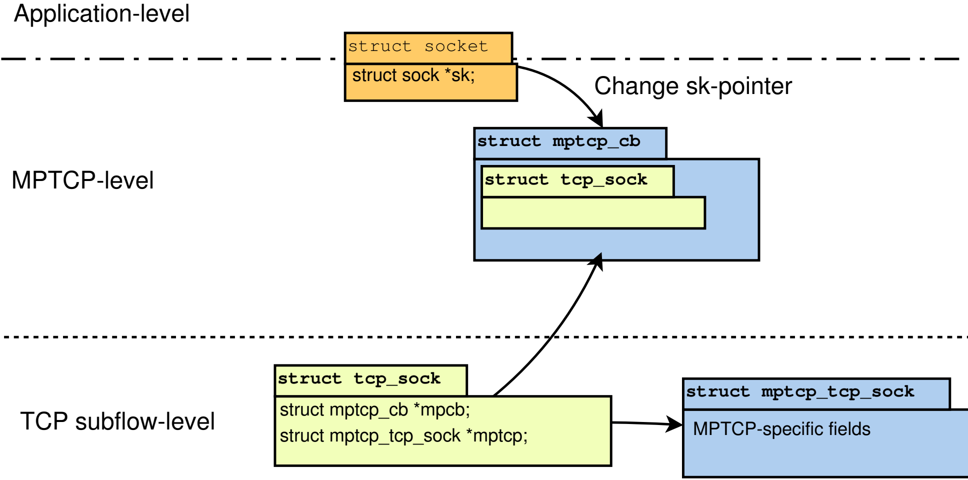
Creating the MPTCP-level on the client-side.

Current Design



Creating the MPTCP-level on the client-side.

Current Design



Creating the MPTCP-level on the client-side.

Current Design

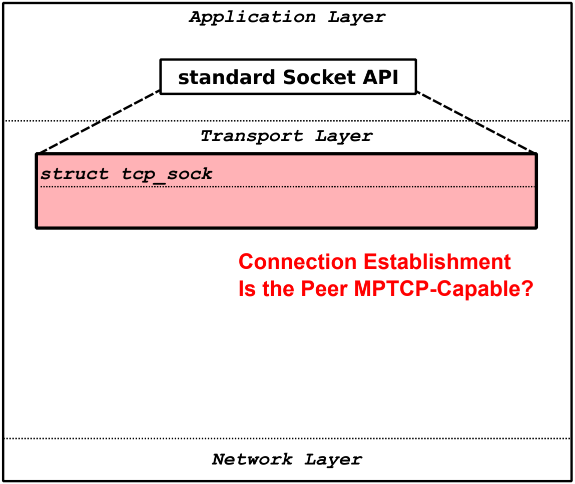
- Problems, if the application does a system-call on the socket, **before** the reception of the SYN+ACK
- Fix: Wait for the SYN+ACK. E.g., `tcp_sendmsg`:

```
/* Wait for a connection to finish. */
if ((1 << sk->sk_state) & ~(TCPF_ESTABLISHED |
    TCPF_CLOSE_WAIT))
        if ((err = sk_stream_wait_connect(sk, &timeo))
            != 0)
                goto do_error;
```

- We need to do this in **all** functions that take a lock on the socket! `tcp_recvmsg`, `tcp_splice_read`, `ip_setsockopt`, `ip_get_sockopt`, `tcp_ioctl`, ...and many more

Creating the MPTCP-level on the client-side.

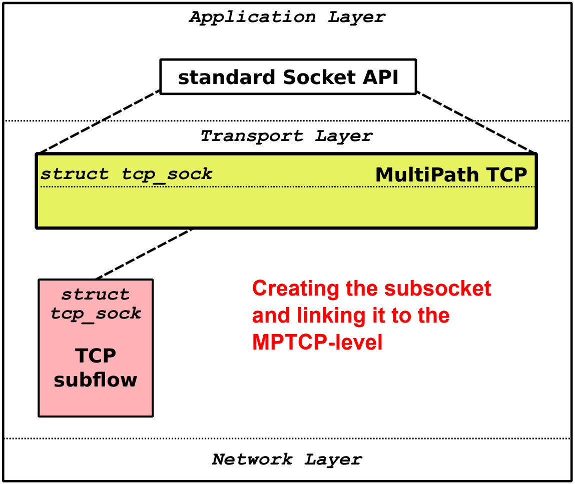
Upcoming Design



Connection Establishment
Is the Peer MPTCP-Capable?

Creating the MPTCP-level on the client-side.

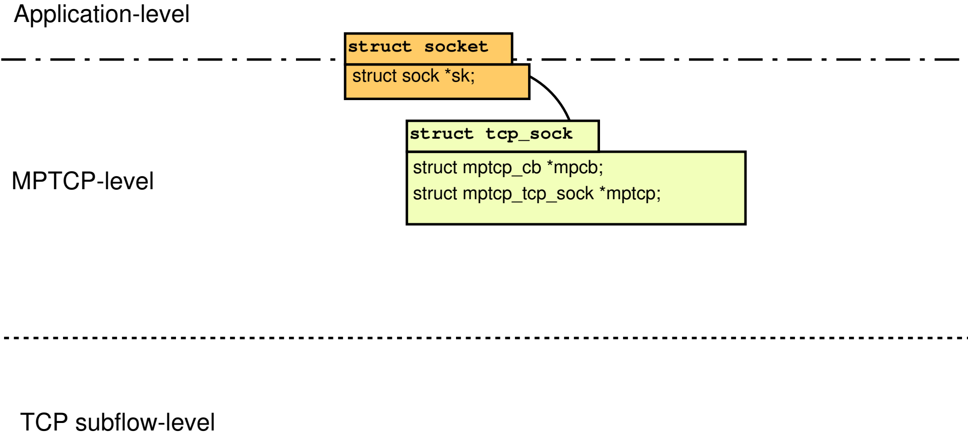
Upcoming Design



Creating the subsocket and linking it to the MPTCP-level

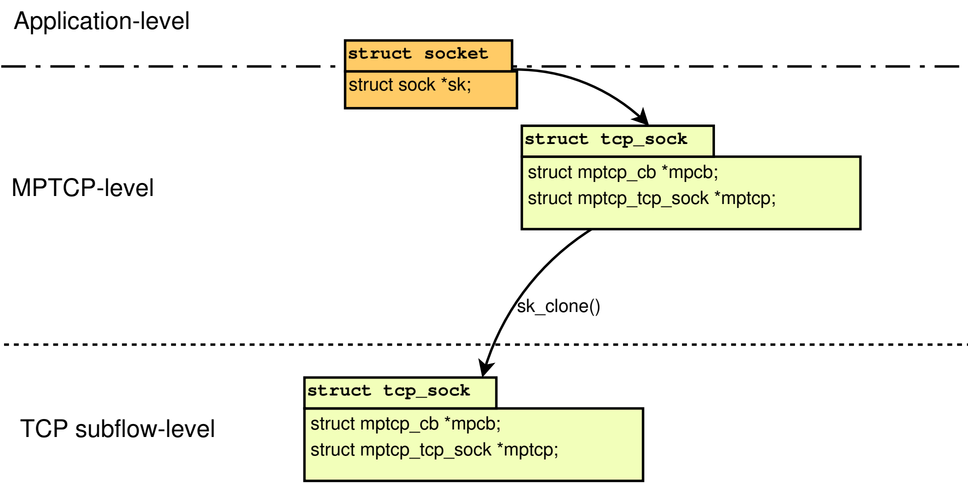
Creating the MPTCP-level on the client-side.

Upcoming Design

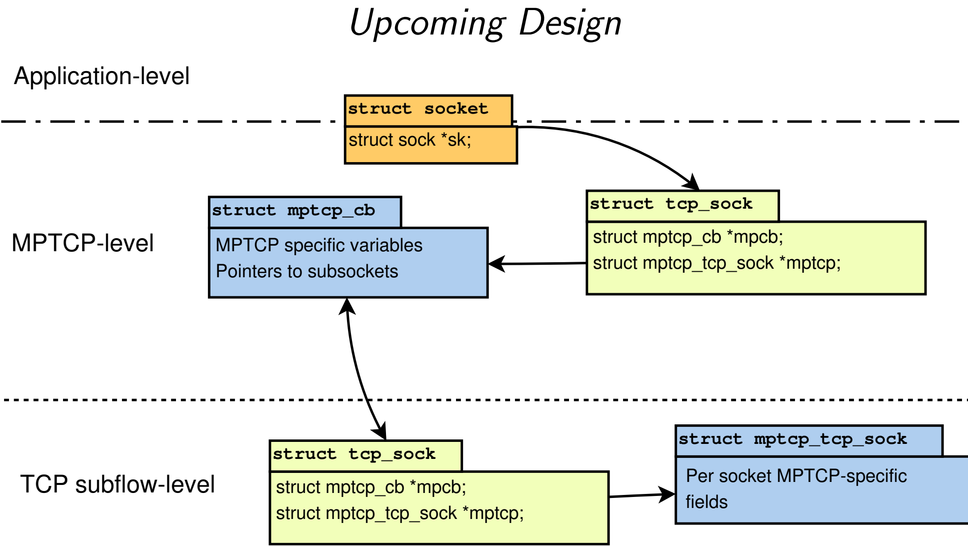


Creating the MPTCP-level on the client-side.

Upcoming Design



Creating the MPTCP-level on the client-side.



Upcoming Design

Application-level

MPTCP-level

TCP subflow-level

struct socket

struct sock *sk;

struct tcp_sock

struct mptcp_cb *mpcb;
struct mptcp_tcp_sock *mptcp;

struct mptcp_cb

MPTCP specific variables
Pointers to subsockets

struct tcp_sock

struct mptcp_cb *mpcb;
struct mptcp_tcp_sock *mptcp;

struct mptcp_tcp_sock

Per socket MPTCP-specific fields

Handling socket options

Questions

- Lots of socket options in the TCP/IP stack
- Some are for the MPTCP-level (*SO_SNDBUF*), some should get passed onto all other subflows (*IP_TTL*)
- This requires a lot of changes in TCP unrelated functions (e.g., `do_ip_setsockopt`)

How could we handle this?

Still lots of changes to the TCP-stack

Questions

We have a lot of:

```
if (tcp_sk(sk)->mpc) {
        DO_SOME_MPTCP_STUFF
} else {
        DO_USUAL_TCP_STUFF
}
```

Submitting MPTCP upstream???

- $\sim$ 10000 lines of code
- Tightly integrated in the TCP-stack
- More work to do:
  - Cleanup - better separate MPTCP from TCP
  - Some missing features
  - Support TSO
  - Support NET_DMA
  - ...
- How to split the patch in small pieces?

Conclusion

Freely available at **http://multipath-tcp.org**
Download it, try it out, contribute!

UCLouvain MPTCP-Team:
Sébastien Barré
Christoph Paasch
Gregory Detal
Fabien Duchene

Prof. Olivier Bonaventure

Thanks to our previous and present partners/contributors:

trilogy Google NOKIA Connecting People intel CHANGE