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)
Is the server MPTCP-capable?

SrcIP: A
DstIP: S
SYN
MP_CAPABLE : ID12
Is the server MPTCP-capable?

SrcIP: S
DstIP: A
SYN + ACK
MP_CAPABLE: ID23
Sending Data - Naïve approach
Sending Data - Naive approach

- Source IP: S
- Destination IP: A
- Sequence Number: 1
- Source IP: S
- Destination IP: A
- Sequence Number: 2
- Source IP: S
- Destination IP: A
- Sequence Number: 3
Sending Data - Naiv approach
Sending Data - Naïve approach

![Diagram showing data transmission through different paths with NAT and firewalls]

- SrcIP: S, DstIP: A, SeqNo: 1
- SrcIP: S, DstIP: A, SeqNo: 2
- SrcIP: S, DstIP: A, SeqNo: 3
Establish separate subflows
Establish separate subflows
Establish separate subflows
Establish separate subflows
Sending Data

Subflow 1
- write_seq
- snd_cwnd
- rcv_nxt

Subflow 2
- write_seq
- snd_cwnd
- rcv_nxt

write(...);
MultiPath TCP

Sending Data

SubSeq: 101
DataSeq: 1

SubSeq: 102
DataSeq: 2

Subflow 1
write_seq
snd_cwnd
rcv_nxt

Subflow 2
write_seq
snd_cwnd
rcv_nxt

Subflow 2
write_seq
snd_cwnd
rcv_nxt

MPTCP-level
write_seq
rcv_nxt

SubSeq: 2001
DataSeq: 3
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

- **Application Layer**
  - `socket(AF_INET, SOCK_STREAM, 0);`
  - `connect(...);`

- **standard Socket API**

- **Transport Layer**

- **Connection Establishment**
  - Is the Peer MPTCP-Capable?

- **Network Layer**

**struct tcp_sock**

**TCP subflow**

IP Networking Lab  [http://multipath-tcp.org](http://multipath-tcp.org)
Establishing first subflow

In-depth call-stack - Client Side

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

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

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

Generate a new identifier for this session

```
create SYN
```

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

pass packet to IP-stack
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_conn_request(...);`
  - create the request-socket

- `tcp_v4_rcv(...);`

- `tcp_v4_send_synack(...);`

- `mptcp_synack_options(...);`

**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

- **Application Layer**
  - Standard Socket API

- **Transport Layer**
  - `struct tcp_sock`
  - MultiPath TCP

- **Network Layer**
  - TCP subflow

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

In-depth call-stack - Client Side

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

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

- `tcp_v4_hnd_req(...);
  tcp_v4_do_rcv(...);
  tcp_v4_rcv(...);`

- `ACK`

- `mptcp_check_req_master(...);`
  - Creates the MPTCP-level socket
  - Initializes everything for MPTCP

- `tcp_v4_syn_recv_sock(...);`
  - Creates the initial subflow-socket
Establishing first subflow

High-Level Kernel design

Application Layer

standard Socket API

Transport Layer

struct tcp_sock

MultiPath TCP

Network Layer

struct tcp_sock

TCP subflow

IP Networking Lab  http://multipath-tcp.org
Establishing additional subflows

Exchanged Messages

SrcIP: B
DstIP: S
SYN
MP_JOIN : ID23
Establishing additional subflows

High-Level Kernel design - Client Side

Application Layer

standard Socket API

Transport Layer

struct tcp_sock MultiPath TCP

struct tcp_sock

TCP subflow

An additional subflow is being created

Network Layer
Establishing additional subflows

In-depth call-stack - Client Side

- `mptcp_init4_subsockets(...);`
- `connect(...);`
- `mptcp_add_sock(...);`  
  Attach subflow to MPTCP-level
- `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()

mptcp_v4_do_rcv(...);

mptcp_lookup_join(...);
Look for the MP_JOIN in a SYN
And see if we know this identifier

tcp_v4_send_synack(...);

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

- mptcp_check_req_child(...);
  Initialize subflow for MPTCP

- mptcp_v4_do_rcv(...);

- mptcp_syn_recv_sock(...);
  Look for a request-sock

- tcp_v4_rcv(...);

- ACK

- mptcp_add_sock(...);
  Attach subflow to MPTCP
Sending Data

Exchanged Messages

Subflow 1
- write_seq
- snd_cwnd
- rcv_nxt

Subflow 2
- write_seq
- snd_cwnd
- rcv_nxt

MPTCP-level
- write_seq
- rcv_nxt

SubSeq: 101
DataSeq: 1

SubSeq: 102
DataSeq: 2

SubSeq: 2001
DataSeq: 3
Sending Data

High-Level Kernel design

```
write(sock, &buf, size);
```

```
struct tcp_sock
```

MultiPath TCP

TCP subflow

TCP subflow

Network Layer

Transport Layer

Application Layer
High-Level Kernel design

- **Application Layer**
  - `write(sock, &buf, size);`
  - standard Socket API

- **Transport Layer**
  - `struct tcp_sock`
  - send-queue
  - MultiPath TCP Scheduler
  - TCP subflow
  - `struct tcp_sock`
  - TCP subflow

- **Network Layer**
Sending Data

High-Level Kernel design

Application Layer

write(sock, &buf, size);

standard Socket API

Transport Layer

struct tcp_sock
send-queue

1 2 3 4 5

MultiPath TCP

Network Layer

TCP subflow

send-queue

1 2 3

MultiPath TCP Scheduler

TCP subflow

send-queue

4 5

http://multipath-tcp.org
Packets can be reordered at the data-level due to delay-differences.
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.
A loss at the subflow-level (or network-reordering) can also cause reordering at the subflow-level.
Subflow-level out-of-order queues are necessary to handle the retransmission at the subflow-level.
Receiving Data

High-Level Kernel design

Application Layer

standard Socket API

Transport Layer

struct tcp_sock
receive-queue

MultiPath TCP

struct tcp_sock
receive-queue

Network Layer

TCP
subflow

TCP
subflow
MultiPath TCP

Design Challenges
Writing the Data-sequence number in the TCP-options

**Naive approach**

```c
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

- **Application Layer**
  ```c
  socket(AF_INET, SOCK_STREAM, 0);
  connect(...);
  ```
- **Transport Layer**
- **Network Layer**

Connection Establishment
Is the Peer MPTCP-Capable?

```c
struct tcp_sock

TCP subflow
```
Creating the MPTCP-level on the client-side.

**Current Design**

```
Creating the MPTCP-level
and linking the application
and linking the application
to this socket.
```
Creating the MPTCP-level on the client-side.

Current Design

Application-level

MPTCP-level

TCP subflow-level

```c
struct socket
struct sock *sk;
```

```c
struct tcp_sock
struct mptcp_cb *mpcb;
struct mptcp_tcp_sock *mptcp;
```
Creating the MPTCP-level on the client-side.

Current Design

Application-level

MPTCP-level

TCP subflow-level

struct socket
struct sock *sk;

struct mptcp_cb

struct tcp_sock

MPTCP specific variables
Pointers to subsockets

sk_clone()

struct tcp_sock
struct mptcp_cb *mpcb;
struct mptcp_tcp_sock *mptcp;
Creating the MPTCP-level on the client-side.

Current Design

Application-level

struct socket
struct sock *sk;

MPTCP-level

struct mptcp_cb
struct tcp_sock

TCP subflow-level

struct tcp_sock
struct mptcp_cb *mpcb;
struct mptcp_tcp_sock *mptcp;

MPTCP-specific fields
Creating the MPTCP-level on the client-side.

Current Design

Application-level

MPTCP-level

TCP subflow-level

struct socket
struct sock *sk;

struct mptcp_cb
struct tcp_sock

struct tcp_sock
struct mptcp_cb *mpcb;
struct mptcp_tcp_sock *mptcp;

MPTCP-specific fields

Change sk-pointer
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`:

```c
/* 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_getsockopt`, `tcp_ioctl`, ... and many more.

IP Networking Lab  http://multipath-tcp.org  24 / 29
Creating the MPTCP-level on the client-side.

**Upcoming Design**

[Diagram showing layers of the stack with labels: Application Layer, Transport Layer, Network Layer. Key points include:
- `struct tcp_sock`
- Connection Establishment
- Is the Peer MPTCP-Capable?
- Standard Socket API]

IP Networking Lab  http://multipath-tcp.org
Creating the MPTCP-level on the client-side.

Upcoming Design

Application Layer

standard Socket API

Transport Layer

struct tcp_sock

MultiPath TCP

Network Layer

struct tcp_sock

TCP subflow

Creating the subsocket and linking it to the MPTCP-level
Creating the MPTCP-level on the client-side.

Upcoming Design

Application-level

```
struct socket
struct sock *sk;
```

MPTCP-level

```
struct tcp_sock
struct mptcp_cb *mpcb;
struct mptcp_tcp_sock *mptcp;
```

TCP subflow-level
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;

sk_clone()

struct tcp_sock
struct mptcp_cb *mpcb;
struct mptcp_tcp_sock *mptcp;
Creating the MPTCP-level on the client-side.

Upcoming Design

Application-level

MPTCP-level

TCP subflow-level

```
struct socket
struct sock *sk;
```

```
struct mptcp_cb
MPTCP specific variables
Pointers to subsockets
```

```
struct tcp_sock
struct mptcp_cb *mpcb;
struct mptcp_tcp_sock *mptcp;
```

```
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:

```c
if (tcp_sk(sk)->mpc) {
    DO_SOME_MPTCP_STUFF
} else {
    DO_USUAL_TCP_STUFF
}
```
Submitting MPTCP upstream???

- ~ 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: