Towards Smart Multipath TCP-enabled Applications

Benjamin Hesmans*, Gregory Detal⁺, Sébastien Barré⁺, Raphaël Bauduin*, Olivier Bonaventure*

* firstname.lastname@uclouvain.be
⁺ firstname.lastname@tessares.net
Regular TCP
Regular TCP
Multipath TCP
Multipath TCP

Regular TCP extension that allows packets belonging to one connection to be sent over different paths.

RFC 6824
Multipath TCP

- Socket interface unchanged
- MPTCP is hidden to the application
Multipath TCP

Design objective: Support unmodified applications

- Socket interface unchanged
- MPTCP is hidden to the application
Multipath TCP : Various clients
Multipath TCP: Various clients
Multipath TCP: Various clients
Multipath TCP: Various servers
Multipath TCP: Various interfaces types

- ADSL, VDSL, Cable...
- Satellite
- Interfaces types and technologies
- 3G, 4G...
Multipath TCP: Various black boxes on the paths

Network is not a dumb pipe anymore
Multipath TCP: Various applications needs
Multipath TCP: Various applications needs
Multipath TCP: Overview

ADSL, VDSL, Cable ...

Satellite

$ $

3G, 4G ...

17
Multipath TCP: Overview

MPTCP should be adapted based on the context and application needs.
Agenda

1. Introduction to MPTCP in Linux
2. Solution Description
3. Use cases
4. Socket Implementation
Introduction to MPTCP in Linux

MPTCP is modular
Multipath TCP: Path manager

1. Open the master subflow
2. SYN, MP_CAPABLE (Key A)
3. SYN + ACK, MP_CAPABLE (Key B)
4. ACK, MP_CAPABLE (Key A, Key B)

Token calculation:
- tokenA = H(keyA)
- tokenB = H(keyB)
Multipath TCP: Path manager

- SYN, **MP_JOIN** (TokenB, NonceA)
- SYN + ACK, **MP_JOIN** (TokenA, NonceB, HMAC)
- ACK, **MP_JOIN** (HMAC)

Should I open subflow on other available Interfaces?

Path manager

tokenA = H(keyA)
tokenB = H(keyB)
Multipath TCP: Path manager

Should I define it as back up?

Path manager

tokenA = H(keyA)
tokenB = H(keyB)
Multipath TCP: Path manager

Subflow is broken. What should I do?
Path manager
Multipath TCP : Scheduler

Which packet should be the next one to send? **Scheduler**
Multipath TCP: Scheduler

Which path should I use to send the next packet?

Scheduler
Multipath TCP: Scheduler

MP_DSS(seq, ack)
Multipath TCP : Modularity

1. Scheduler
   ○ On which subflow should I send this packet?
   ○ Which packet should be the next one to be sent?

2. Path Manager
   ○ What should I do with this new remote address?
     i. remote / local
     ii. back up?
   ○ Events
     i. Connection established
     ii. Subflow established
     iii. Subflow deleted (and why)
Multipath TCP : Path Manager in User space

- Modularity available since 2013.
- ...Still the number of schedulers or path managers is low!
- **Move the Path Manger to user space**
  - More flexibility
  - Ease the development
- **Expose advanced API to applications**
Agenda

1. Introduction to MPTCP in Linux
2. Solution Description
   a. Kernel space solution
   b. User space implementation
3. Use cases
4. Socket Implementation
Kernel

- Plug on the already defined interface
Kernel

- Plug on the already defined interface
- Does not take any decisions, but retransmits all the events to the user space through netlink (1, 2)
Kernel

- Plug on the already defined interface
- Does not take any decisions, but retransmits all the events to the user space through netlink (1, 2)
Kernel

- Plug on the already defined interface
- Does not take any decisions, but retransmits all the events to the user space through netlink (1, 2)
- Listens to commands from user space via netlink and translates them to actions (3, 4)
Kernel

- Plug on the already defined interface
- Does not take any decisions, but retransmits all the events to the user space through netlink (1, 2)
- Listens to commands from user space via netlink and translates them to actions (3, 4)
User space

- Offers API to send commands to the MPTCP connections through netlink (5, 1)
User space

- Offers API to send commands to the MPTCP connections through netlink (5, 1)
User space

- Offers API to send commands to the MPTCP connections through netlink (5, 1)
- Receives events through netlink (4, 6) and generates call backs
User space

- Offers API to send commands to the MPTCP connections through netlink (5, 1)
- Receives events through netlink (4, 6) and generates call backs
1. Introduction to MPTCP
2. Solution Description
3. Use cases
   a. Redefined back up
   b. Refresh subflows
4. Socket Implementation

Agenda
Today’s implementation

WIFI Default interface

Mobile Back up interface
Today’s implementation

WIFI Default interface

Mobile Back up interface
Today’s implementation
Today’s implementation
Today’s implementation

- Moves away from the wifi access point
- Wifi subflow is slow but not dead
- “Slow” is somehow subjective and context dependant
Today’s implementation

- Moves away from the wifi access point
- Wifi subflow is slow but not dead
  - “Slow” is somehow subjective and context dependant

Back up subflow is there but not used as long as the other subflow is present
Today’s implementation

- Early establishment of back up subflow even if it’s not used
- Back up subflow is not used if the other subflow is still open
Back Up Redefinition

WIFI Default interface

Mobile Back up interface
Back Up Redefinition
Back Up Redefinition

No additional subflow is established over back up interface
Today’s implementation

Moves away from the wifi access point

WIFI Default interface

Slow subflow is detected

Mobile Back up interface
Today’s implementation

Moves away from the wifi access point

Slow subflow is detected

WIFI Default interface

New subflow is established on the back up interface and does NOT have a low priority
Today’s implementation

- Back up subflow is not established if it’s not needed
- Back up subflow is immediately used when it’s opened
Today VS Back Up Redefinition

Today (Kernel)

- Establish connection between all pairs of address
- Never use subflow established over a back up interface unless it’s the only one left

Back Up Redefinition (User space)

- Establish connection between all pairs of not back-up address
- Do not establish subflow over a back up interface unless the RTO on the other interface is higher than R
Today PM VS Back Up Redefinition - Experiment
Today PM VS Back Up Redefinition - Experiment

- Exchange a small file
Today PM VS Back Up Redefinition - Experiment

- Exchange a small file
- After 1 second, 30% loss on the default interface
Today PM VS Back Up Redefinition - Experiment

- Exchange a small file
- After 1 second, 30% loss on the default interface
- Compare
  - Today’s
  - Backup Redefinition ($R=1$)
Today’s implementation - Experiment
Today’s implementation - Experiment

Link have 30% losses

TCP retransmissions, subflow is still there but slow
Today’s implementation - Experiment

Link have 30% losses

TCP retransmissions, subflow is still there but slow

Long time to finish
Today’s implementation - Experiment

TCP retransmissions, subflow is still there but slow

Long time to finish

Link have 30% losses

Back up subflow is not used even if it’s established

Relative sequence Number (10^5 Bytes)
Back up Redefinition - Experiment

Link become have 30 % losses
Back up Redefinition - Experiment

Link become have 30% losses

RTO is growing
Back up Redefinition - Experiment

- Link becomes have 30% losses
- RTO is growing

Establishing a new subflow over the back up interface
Back up Redefinition - Experiment

- Link become have 30% losses
- RTO is growing
- Establishing a new subflow over the back up interface
- Completion time is reduced
Today VS Back Up Redefinition

![Graphs showing relative sequence number vs relative time for Master and Back up datasets.](image)

- Master
- Back up

Relative Time (s) vs Relative Sequence Number (10^5 Bytes)
Today VS Back up Redefinition

- Saving energy if no subflow is established over the back up interface
- Saving money if no subflow is established over the back up interface
- Faster recovery if the default interface is not usable anymore
Agenda

1. Introduction to MPTCP
2. Solution Description
3. Use cases
   a. Redefined back up
   b. Refresh subflows
4. Socket Implementation
MPTCP in data centers: ndiffports

MPTCP in data centers: ndiffports¹

MPTCP in data centers: ndiffports
MPTCP in data centers : ndiffports
MPTCP in data centers : ndiffports
MPTCP in data centers: ndiffports
ndiffports - Experiment

- Exchange a file over the ECMP topology
ndiffports - Experiment

- Exchange a file over the ECMP topology
- Run the experiments 150 times
  - with Nddiffports (N=5)
- Observe the completion time
ndiffports - Experiment
ndiffports
ndiffports
ndiffports
ndiffports
ndiffports

All subflows could use the same path
ndiffports - Experiment

Found 2 of the 4 paths
ndiffports - Experiment

Found 2 of the 4 paths
Found 3 of the 4 paths
ndiffports - Experiment

Found 2 of the 4 paths

Found 3 of the 4 paths

Found 4 of the 4 paths
Revisited ndiffports
Revisited ndiffports
Ndiffports VS Revisited ndiffports

Ndiffports (Kernel)

When connection is established:

open $N - 1$ new subflows

Revisited ndiffports (User space)

When connection is established:

open $N - 1$ new subflows

Every $T$ seconds:

Ask the pacing rate for all the subflows
Close the subflow with the lowest pacing rate
Establish a new subflow
ndiffports VS Revisited ndiffports - Experiment

- Exchange a file over the ECMP topology
- Run the experiments 150 times
  - with Ndiffports (N=5)
  - with Refresh Subflows (N=5, T=2.5 seconds)
- Compare the completion time
ndiffports VS Revisited ndiffports - Experiment
Refresh subflows VS ndiffports - Experiment
Agenda

1. Introduction to MPTCP
2. Solution Description
3. Use cases
4. **Socket Implementations**
Netlink (Daemon-like)

Applications

PM A  PM B  PM C  PM D

Library

PM Module

1

2

3

4

Kernelspace

Userspace

Socket options

Applications

1

2

MPTCP

Kernelspace

Userspace
Socket API example: open a new subflow

```c
unsigned int optlen;
struct mptcp_sub_tuple *sub_tuple;
struct sockaddr_in *addr;
int error;

optlen = sizeof(struct mptcp_sub_tuple) + 2 * sizeof(struct sockaddr_in);
sub_tuple = malloc(optlen);

addr = (struct sockaddr_in*) &sub_tuple->addrs[0];
addr->sin_family = AF_INET;
addr->sin_port = htons(12345);
inet_pton(AF_INET, "10.0.0.1", &addr->sin_addr);

addr++;
addr->sin_family = AF_INET;
addr->sin_port = htons(1234);
inet_pton(AF_INET, "10.1.0.1", &addr->sin_addr);

error = getsockopt(sockfd, IPPROTO_TCP, MPTCP_OPEN_SUB_TUPLE, sub_tuple, &optlen);
```
# Socket API description

<table>
<thead>
<tr>
<th>Name</th>
<th>Input</th>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPTCP_GET_SUB_IDS</td>
<td>-</td>
<td>subflow list</td>
<td>Get the current list of subflows viewed by the kernel</td>
</tr>
<tr>
<td>MPTCP_GET_SUB_TUPLE</td>
<td>id</td>
<td>sub tuple</td>
<td>Get the pair ip and ports used by the subflow identified by id</td>
</tr>
<tr>
<td>MPTCP_OPEN_SUB_TUPLE</td>
<td>tuple</td>
<td>-</td>
<td>Request a new subflow with pair of ip and ports</td>
</tr>
<tr>
<td>MPTCP_CLOSE_SUB_ID</td>
<td>id</td>
<td>-</td>
<td>Close the subflow identified by id</td>
</tr>
<tr>
<td>MPTCP_SUB_GETSOCKOPT</td>
<td>id, sock opt</td>
<td>sock ret</td>
<td>Redirects the <code>getsockopt</code> given in input to the subflow identified by id and return the value returned by the operation</td>
</tr>
<tr>
<td>MPTCP_SUB_SETSOCKOPT</td>
<td>id, sock opt</td>
<td>-</td>
<td>Redirects the <code>setsockopt</code> given in input to the subflow identified by id.</td>
</tr>
</tbody>
</table>

Table 1: Implemented MPTCP socket options
Conclusion

- New flexible API proposal for MPTCP path management
- Proof-of-concept for different use-cases
- Per use case context improvement
- Enable new flavour of MPTCP
- MPTCP available at www.multipath-tcp.org
- Socket API patches available and VM available on www.multipath-tcp.org to try.
Towards Smart Multipath TCP-enabled Applications

Benjamin Hesmans*, Gregory Detal†, Sébastien Barré†, Raphaël Bauduin*, Olivier Bonaventure*

* firstname.lastname@uclouvain.be
† firstname.lastname@tessares.net
Back Up Slides
Socket API : example : Refresh subflow

rotate_tresh = X
rotate_count = 1
tot_bytes    = 0

while (n = read()) != 0 do
    tot_bytes += n
    if tot_bytes > rotate_count * rotate_tresh then
        refresh_sub()
        rotate_count ++
Smart streaming
Fullmesh VS Smart streaming

Fullmesh (Kernel)

- Establish connection between all pairs of address
- Never use subflow established over a backup interface unless it’s the only one left

Smart streaming (User space)

- Establish connection between all pairs of not back-up address
- Do not establish subflow over a back-up interface unless we did not send $F^{-1}$ of the data $d$ after $T^{-1}$ of the time $t$. 
Fullmesh VS Smart stream - Experiment

- Send blocks of 64 kB of data every second
- Compare
  - Fullmesh
  - Smart Stream \((T^{-1}=0.5, F^{-1}=0.5, t=1, d=64kB)\)
- With different loss rates on the default interface

Note: full description of the topology in the paper