MPTCP : Linux Kernel implementation status

Presenter : Christoph Paasch
IP Networking Lab
Université catholique de Louvain

28 mars 2012

http://mptcp.info.ucl.ac.be
Outline

1. Implementation status
2. Implementation challenges
3. Performance results
Implementation status

Compared to last IETF-80 presentation (cfr. Sébastien Barré at IETF 80 - Prague March 2011)

- MPTCP security (draft v07)
- IPv6
- Fully support all kind of middleboxes (segment-splitting/coalescing, payload-modifying, pro-actively acking middleboxes, . . .)
- Support reception of 64-bit data-sequence-numbers
- Mobility supported with REMOVE_ADDR
- Forced closure supported with MP_FASTCLOSE
- Support for api-draft is ongoing
- SMP is supported (new locking architecture)
- MPTCP is on Linux kernel version 3.0 (soon 3.2)
Linux MPTCP community

Total contributions from all people (ordered by number of commits):

- Sébastien Barré (UCLouvain - now Thelis)
- Christoph Paasch (UCLouvain)
- Jaakko Korkeaniemi (Aalto)
- Gregory Detal (UCLouvain)
- Fabien Duchêne (UCLouvain)
- Andreas Seelinger (RWTH-Aachen)
- Andreas Ripke (Neclab)
- Vlad Dogaru (Intel)
- Lavkesh Lahngir (Kanpur University)
- John Ronan (TSSG)
- Brandon Heller (Stanford University)
Outline

1. Implementation status
2. Implementation challenges
3. Performance results
Applications are able to avoid TIME-WAIT (e.g., apache2, apachebenchmark,...)

On the data-level this works.

But not at the subflow-level ...

Many subflows are lingering around in TIME-WAIT state although the application tried to avoid it.
Applications poll the socket to do passive closing

Host-A

\[ \text{poll()} \]

Meta-Socket

State: Established

Sub-Socket

State: Established

Host-B

\[ \text{close()} \]

Meta-Socket

State: Established

Sub-Socket

State: Established
Applications poll the socket to do passive closing

Host-A
\[ \text{poll()} \rightarrow \text{SEND\_SHUTDOWN} \]

Meta-Socket
State: CLOSE-WAIT

Sub-Socket
State: Established

Host-B

Meta-Socket
State: FIN-WAIT-2

Sub-Socket
State: Established
After the *DATA FIN* the application does a passive close.

**Host-A - passive closer**
- `close()`/`shutdown()`
- Meta-Socket: State: CLOSE
- Sub-Socket: State: Established

**Host-B**
- Meta-Socket: State: Time-Wait
- Sub-Socket: State: Established

DATA FIN →

DATA ACK ←
MPTCP - Avoiding TIME-WAIT

However, subflow close does not respect the passive close

Host-A - passive closer

Host-B

Meta-Socket
State: CLOSE

subflow-close()

Sub-Socket
State: Fin-Wait-1

Meta-Socket
State: Time-Wait

subflow-close()

Sub-Socket
State: Fin-Wait-1

subflow-FIN
However, subflow close does not respect the passive close

Host-A - passive closer

Host-B

Meta-Socket

State: CLOSE

Meta-Socket

State: Time-Wait

Sub-Socket

State: Time-Wait

Sub-Socket

State: Time-Wait

subflow-ACK
MPTCP - Avoiding TIME-WAIT

How to continue after closing the meta-sockets?

Host-A - passive closer

Host-B

close() / shutdown()

Meta-Socket

State: CLOSE

Sub-Socket

State: Established

Meta-Socket

State: Time-Wait

Sub-Socket

State: Established

DATA_FIN

DATA_ACK
Don’t close the subflow, wait for the subflow-fin!

Host-A - passive closer

Wait for subflow-fin
(max-timeout = RTO)

Host-B

Meta-Socket
State: CLOSE

Sub-Socket
State: Established

Data FIN

Data ACK
MPTCP - Avoiding TIME-WAIT

Don’t close the subflow, wait for the subflow-fin!

Host-A - passive closer

Host-B

Meta-Socket
State: CLOSE

Sub-Socket
State: Close-Wait

Meta-Socket
State: Time-Wait

Sub-Socket
State: Fin-Wait-2

subflow-FIN

subflow-ACK
MPTCP - Avoiding TIME-WAIT

Enforced passive-close on the subflow

Host-A - passive closer

Meta-Socket
State: CLOSE

Sub-Socket
State: Close-Wait

↓passive subflow-close()

subflow-FIN

Host-B

Meta-Socket
State: Time-Wait

Sub-Socket
State: Fin-Wait-2

↓subflow-close()
MPTCP - Avoiding TIME-WAIT

Enforced passive-close on the subflow

Host-A - passive closer

Meta-Socket
State: CLOSE

Sub-Socket
State: CLOSE

subflow-close()

Host-B

Meta-Socket
State: Time-Wait

Sub-Socket
State: Time-Wait

subflow-FIN

subflow-ACK
Outline

1. Implementation status
2. Implementation challenges
3. Performance results
Apache benchmarking micro-flows

100 simultaneous requests, for a total of 100,000 requests of varying size

Apache benchmarking micro-flows

100 simultaneous requests, for a total of 100,000 requests of varying size

Apache benchmarking micro-flows

100 simultaneous requests, for a total of 100000 requests of varying size


Presenter: Christoph Paasch - IP Networking Lab  http://mptcp.info.ucl.ac.be
Flow-to-core affinity

- Individual TCP-flows are steered to the same CPU-core to avoid reordering inside the receive-code.
- MPTCP has lots of L1/L2 cache-misses because the individual subflows are steered on different CPU-cores.
- MPTCP-aware Receive-Flow-Steering sends all subflows on the same CPU-core.
Flow-to-core affinity - 10 Gbps interfaces

![Graph showing average goodput in Gbps versus number of iperf-sessions]

- **TCP**

**Presenter**: Christoph Paasch - IP Networking Lab

[http://mptcp.info.ucl.ac.be]
Flow-to-core affinity - 10 Gbps interfaces

Average Goodput in Gbps

Number of iperf-sessions
Flow-to-core affinity - 10 Gbps interfaces

![Graph showing average goodput in Gbps vs number of iperf-sessions for TCP, MPTCP, and MPTCP-aware RFS.](http://mptcp.info.ucl.ac.be)
Vertical Handover with MPTCP

Skype-call from MPTCP-enabled host via MPTCP-Proxy to regular TCP. Vertical Handover from WiFi to 3G during the Skype-call.
What remains to be done before proposing something to netdev?

- Minor missing pieces (e.g., sending 64-bit DSN, ...)
- A cleaner separation between layers to avoid increasing the size of `struct sk_buff`
- Support of TCP SYN-Cookies
- Support of NET-DMA
- Support of TSO
- More cleanup, ...
Readings:


http://mptcp.info.ucl.ac.be

Install MPTCP and use it !!! :)