MPTCP is coming

NetConf, Boston 2019

Outline

- Why MPTCP ?
    - Why does it take so long?
- Current status and implementation
    - Skb extensions, ULP, inet/tcp diag
- Roadmap, current work
    - MP_JOIN, upstream by the end of the year?
    - Future work

Why MPTCP?

Two main use cases

- Keep logical connection established when endpoints address changes
- Use multiple links, e.g. cable and 5G, at same time for same connection

The latter usually comes with MPTCP Proxy scheme, terminated at local provider/DC

Focus on a limited feature set for initial upstreaming:

- Server use case
- Performance optimizations are deferred
- Path manager and scheduler customization are deferred

Current Status

Highlevel overview

- “Mptcp-next” kernel
    - Complete rewrite, not related to the out of tree implementation from multipath-tcp.org
    - targets upstream inclusion
    - Growing development team: Intel, Tessares, Redhat
- bool CONFIG_MPTCP switch
    - No code change when MPTCP=n
- Can create mptcp connections via SOCK_STREAM and IPPROTO_MPTCP
- MP_JOIN support WIP, i.e. only single flow (with DSS mapping)
- Small mptcp selftest script that is being extended as we cover more use cases
- Parallel effort to “de-feature” existing out-of-tree implementation

Current status

Implementation: Architectural overview

- MPTCP meta socket
  - Gets created on behalf of userspace via socket(), accept() etc.
  - Contains the mptcp relevant parts (e.g. logical sequence numbers, subflows (tcp sockets, and so on)
- Subflows (tcp sockets) are stored in a list via meta socket
- ULP plumbs tcp sockets to the mptcp (parent) socket (back pointer)
- Userspace does not interact with the subflow connections directly
- RFC patches with current snapshot were sent to net-next last monday
  - Verify that the current direction is sane
- By the end of the year:
  - DATA_FIN, MP_JOIN/active-backup support (server side)
  - Official net-next submission

diffstat

```
net/core/skbuff.c                  |    7
net/ipv4/inet_connection_sock.c    |    2
net/ipv4/tcp.c                     |    4 +-
net/ipv4/tcp_input.c               |   25 +-
net/ipv4/tcp_ipv4.c                |    4 +-
net/ipv4/tcp_output.c              |   62 +-
net/ipv4/tcp_ulp.c                 |   12 +
net/mptcp/crypto.c                 |  206 ++++
net/mptcp/options.c                |  621 ++++++++++
net/mptcp/pm.c                     |   66 ++
net/mptcp/protocol.c               | 1043 ++++++++++++++++++++
net/mptcp/protocol.h               |  229 ++++
net/mptcp/subflow.c                |  344 ++++++
net/mptcp/token.c                  |  373 ++++
32 files changed, 3955 insertions(+), 8 deletions(-)
```

Main data structures (1)

Mptcp socket structure

```
struct mptcp_sock {
        /* inet_connection_sock must be the first member */
        struct inet_connection_sock sk;
        u64        local_key, remote_key, write_seq, ack_seq;
        struct list_head conn_list;
        struct socket   *subflow; // outgoingconnect/listener/!mp_capable
};
```

This is what gets created on “socket(... , IPPROTO_MPTCP)” (IPPROTO_MPTCP == 262)

Used to store keys (to authenticate subflows), sequence numbers, and a list of active subflows.

Main data structures (2)

subflow/mptcp socket plumbing

```
struct subflow_context {
        struct  list_head node;/* conn_list of subflows */
        [..]
        u16     request_mptcp : 1,  /* send MP_CAPABLE */
        request_cksum : 1,          [..]
         struct  socket *tcp_sock;  /* underlying tcp_sock */
         struct  sock *conn;       /* parent mptcp_sock */
          void    (*tcp_sk_data_ready)(struct sock *sk);
};
```

This gets stored in tcp sockets icsk->icsk_ulp_data for all tcp connections that are created on behalf of mptcp.

MPTCP -> subflow association

Responder/server side

- socket(STREAM, MPTCP), then listen/bind
  - mptcp_bind() creates an internal tcp socket and adds ULP::

    sock_create_kern(net, .., IPPROTO_TCP, &sock)

    tcp_set_ulp(sock->sk, "mptcp")

ULP initialization sets icsk->icsk_af_ops to a subflow specific variant and calls inet_bind on the tcp socket.

- accept():
  - mptcp_accept() calls kernel_accept() on the underlying tcp listening socket
  - can then associate the tcp flow with existing mptcp socket (wip), or create a new mptcp socket (containing the newly accepted tcp subflow).

MPTCP -> subflow association

Initiator side

- socket(,.. MPTCP), connect()
    - Socket creates the mptcp meta socket
    - The tcp socket gets created on connect() call
    - ULP gets added to it so we can link back to mptcp socket from tcp socket
- association as a subflow occurs via mptcp inet_connection_sock_af_ops.sk_rx_dst_set.
- new subflow gets added to MPTCP socket conn_list after it is established.

Ongoing work (1)

ULP extensions

- Improve MPTCP diagnostics, e.g. show which tcp flows are part of same logical MPTCP “connection” via tcp diag (ss tool)
    - See Davide Caratti’s work: “extend INET_DIAG_INFO with information specific to TCP ULP”
- independent/decoupled from MPTCP: kTLS as first user, so MPTCP can make use of this ULP diag too

Ongoing work (2)

MP_JOIN support

- Allow incoming new connection to be associated with existing (logical) connection in transparent way in case it provides mptcp token
- Additional subflows will not be used for data transfer initially (“backup path”) to simplify implementation
  - Will still need to be able to receive data on such tcp subflows
  - But avoids need for full traffic scheduler
  - No need for logical congestion control (if that is needed at all)

Ongoing work (3)

DATA_FIN support

- Similar to fin in standard tcp, except will close down the logical mptcp connection
  - Typically triggered in response to close()

- Sets a flag in DSS option, i.e. signalled via tcp option on the subflow

Ongoing work (4)

Socket state update

- MPTCP sk_socket state doesn’t reflect logical state at the moment, is just syn/established/close
- mptcp socket state is unrelated from subflow states
    - Could even have connected mptcp socket with no established subflow
    - Plan is to re-use tcp states for mptcp

Future work (1):

- Ipv6 support
  - Current implementation directly calls some tcp ipv4 functions
  - Be32 used for addr storage/in function prototypes
  - No known blockers wrt. Ipv6 at this time
- Coupled receive windows
  - Receive window is no longer per subflow, it indicates buffer space for whole logical connection
  - Its therefore relative to MPTCP DATA_ACK, not subflow acks
  - Given middlebox interference, receivers need to be liberal (use largest value seen on a subflow)

Future work (2):

path/subflow management

- peers announce additional addresses/ports in tcp option
  - From RFC/protocol pov, responder could even connect to initiator
  - In “initiator connections to alternate address announced by server” case: kernel might lack proper info to make such decision (e.g. because alternate path is slow, or not desirable for other reasons)
- Out-of-tree implementation offers different “path managers”
  - similar to tcp congestion control plugins
  - E.g. “full-mesh”: “try to establish everything”
  - most interesting one at this time: netlink based
    - Uses Genl multicast to inform userspace about coming/going peers
    - Userspace can add/remove addresses, perform joins, etc.

Future work (3):

Testing

- Current mptcp kselftest is not enough
  - Limited by the feature set available
  - Can’t test MP_JOIN itself, since we will only do it passively initially
  - Needs “manual” testing vs. out-of-tree MPTCP implementation

- Mptcp packetdrill: https://github.com/multipath-tcp/packetdrill_mptcp
- Syscall level testing is non-existent

Future work (4):

MPTCP UAPI

- RFC 6897 - Multipath TCP (MPTCP) Application Interface Considerations
- No changes at this time (except need to pass IPPROTO_MPTCP in socket(2))
    - Reusing sctp api? (connectx and friends)
    - Need to plumb highlevel set/getsockopts to subflows (e.g. SO_RCVBUF, PRIORITY, NONBLOCK, etc).
        - “Replay” on later JOIN

- RFC requires getpeername/getsockname to not change during lifetime of MPTCP connection
    - Even if the initial subflow was already closed
    - Even allows to close entire MPTCP connection in that case (Fate-Sharing)
- Lack of MPTCP availability in most servers/clients - little to no real operational experience

redhat.

THANK YOU

RFCs

- RFC 6182 - Architectural Guidelines for Multipath TCP Development
- RFC 6824 - TCP Extensions for Multipath Operation with Multiple Addresses
  - This describes the protocol/on-wire details
  - Some hints and considerations (heuristics) for subflow establishment decisions in 3.8 (heuristics).
- RFC 6897 - Multipath TCP (MPTCP) Application Interface Considerations
  - get/setsockopt interface