



MPTCP KTLS Support

Bringing TLS to Multipath TCP

Geliang Tang <geliang@kernel.org>

Linux Plumbers Conference

Prague, Oct 2026



Thanks to Paolo Abeni, Jakub Kicinski, Matthieu Baerts, Mat Martineau, Hannes Reinecke, Zqiang, Gang Yan, and others for their help during the development of MPTCP KTLS support.



Agenda

- Introduction to MPTCP
- Motivation
- Current Status
- Implementation
- Bugfixes
- Limitations
- Selftests and Validation



Introduction to MPTCP

What is MPTCP

- Multipath TCP or MPTCP is an extension to the standard TCP and is described in RFC 8684.
- MPTCP is implemented via TCP Options, kind number of MPTCP is 30.
- It allows a device to make use of multiple interfaces at once to send and receive TCP packets over a single MPTCP connection.
- MPTCP can aggregate the bandwidth of multiple interfaces or prefer the one with the lowest latency. It also allows a fail-over if one path is down, and the traffic is seamlessly reinjected on other paths.
- Introduced in Linux kernel v5.6
- For more details, please visit the project website: <https://www.mptcp.dev>



Introduction to MPTCP

How to Use MPTCP

- Address config: ip mptcp (manual), mptcpd or Network Manager (automatic)

```
# ip mptcp endpoint add 10.1.2.2 subflow
```

- Create MPTCP socket:

```
fd = socket(AF_INET, SOCK_STREAM, IPPROTO_MPTCP);
```

Or force with 'mptcpize' or 'mptcpify-bpfcc' (no code change)

- Use: same as TCP sockets

Kernel creates multiple TCP subflows to send and receive data
MPTCP socket has its own socket options



Introduction to MPTCP

How to Configure TLS

- **set:**

```
setsockopt(fd, IPPROTO_TCP, TCP_ULP, "tls", sizeof("tls"));    (NOT SUPPORTED)
```

```
struct tls12_crypto_info_aes_gcm_128 tls_tx = {
    .info = {
        .version      = TLS_1_3_VERSION,
        .cipher_type = TLS_CIPHER_AES_GCM_128,
    },
};
setsockopt(fd, SOL_TLS, TLS_TX, (void *)&tls_tx, sizeof(tls_tx));
```

```
struct tls12_crypto_info_aes_gcm_128 tls_rx = {
    .info = {
        .version      = TLS_1_3_VERSION,
        .cipher_type = TLS_CIPHER_AES_GCM_128, i
    },
};
setsockopt(fd, SOL_TLS, TLS_RX, (void *)&tls_rx, sizeof(tls_rx));
```

- **get:**

```
getsockopt(fd, IPPROTO_TCP, TCP_ULP, buf, &buflen);    ("mptcp")
```



Motivation

MPTCP-adapted applications need TLS

- **lighttpd:**

https://github.com/multipath-tcp/mptcp_net-next/issues/480

MPTCP and KTLS support #480

@matttbe opened on Mar 19, 2024

Currently, an MPTCP socket cannot support other TCP ULP extensions. This includes KTLS. KTLS is supported in GnuTLS and OpenSSL libraries. It is then tempting to use it without too much of an effort, e.g. lighttpd is using KTLS with OpenSSL and GnuTLS if available since January 2023.

It might be good to see if KTLS and MPTCP subflows hooks can be chained somehow.

@gstrauss on Mar 19, 2024

FYI: lighttpd 1.4.68 (released Jan 2023) supports KTLS in lighttpd mod_openssl and lighttpd mod_gnutls

@matttbe assigned @geliangtang on May 20, 2024

- **NVMe over MPTCP:**

From: Hannes Reinecke <hare@suse.de>
To: Geliang Tang <geliang@kernel.org>
Subject: Re: [RFC 2/3] nvmet-tcp: add mptcp support
Date: Thu, 15 May 2025 12:06:02 +0200 (05/15/2025 06:06:02 PM)

On 5/14/25 12:51, Geliang Tang wrote:

> On the target side, reuse TSAS (Transport Specific Address Subtype)
> field with MPTCP and TLS instead of adding a new one. Since MPTCP
> and TLS are mutually exclusive and cannot be used at the same time,
> they are both implemented through TCP ULP.

>

> Signed-off-by: Geliang Tang <geliang@kernel.org>

> ---

> drivers/nvme/target/configfs.c | 1 +
> drivers/nvme/target/tcp.c | 7 +++++-
> include/linux/nvme.h | 1 +
> 3 files changed, 8 insertions(+), 1 deletion(-)

>

Hmm. Why fold it into 'tsas', and not 'PROTO' itself?

That seems a far more natural choice, and would allow us to run TLS on top of MPTCP.



Current Status

Patchsets:

- mptcp: implement MPTCP-specific helpers for TLS usage
- mptcp: reduce differences between TCP and MPTCP (Paolo Abeni)
- **MPTCP KTLS support**
- selftests: tls: add mptcp test cases
- mptcp: address stall under memory pressure (Paolo Abeni)
- enable KTLS for NVMe over MPTCP
- tlshd: add mptcp support



MPTCP Implementation

MPTCP Socket Option

Set and get ULP on the MPTCP socket itself, not on any subflow sockets

```
static int mptcp_setsockopt_sol_tcp(struct mptcp_sock *msk, int optname,
                                   sockptr_t optval, unsigned int optlen)
{
    switch (optname) {
    case TCP_ULP:
-       return -EOPNOTSUPP;
+       return mptcp_setsockopt_tcp_ulp(sk, optval, optlen);

static int mptcp_getsockopt_sol_tcp(struct mptcp_sock *msk, int optname,
                                   char __user *optval, int __user *optlen)
{
    switch (optname) {
    case TCP_ULP:
+       return mptcp_getsockopt_tcp_ulp(sk, optval, optlen);
    case TCP_CONGESTION:
    case TCP_INFO:
    case TCP_CC_INFO:
        return mptcp_getsockopt_first_sf_only(msk, SOL_TCP, optname,
                                              optval, optlen);
```



MPTCP Implementation

MPTCP-specific Helpers for TLS

9 hardcoded TCP helpers used by TLS need MPTCP-specific versions:

tcp_sendmsg_locked	->	mptcp_sendmsg_locked
tcp_read_sock	->	mptcp_read_sock
tcp_poll	->	mptcp_poll
tcp_inq	->	mptcp_inq

tcp_epollin_ready	->	mptcp_epollin_ready
tcp_recv_skb	->	mptcp_recv_skb
tcp_read_done	->	mptcp_read_done

tcp_rate_check_app_limited	->	mptcp_rate_check_app_limited
TCP_SKB_CB	->	MPTCP_SKB_CB

The direct calls in these places should be replaced with indirect calls to switch between TCP and MPTCP.



TLS Implementation

Generic Socket Ops

Add sk_proto_ops in struct tls_context:

```
struct tls_context {  
    ...  
    struct proto *sk_proto;  
+    const struct proto_ops *sk_proto_ops;  
    struct sock *sk;  
    ...  
}
```

Save sk_proto_ops in tls_ctx_create:

```
static struct tls_context *tls_ctx_create(struct sock *sk)  
{  
    ...  
    mutex_init(&ctx->tx_lock);  
    ctx->sk_proto = READ_ONCE(sk->sk_prot);  
+    ctx->sk_proto_ops = READ_ONCE(sk->sk_socket->ops);  
    ctx->sk = sk;  
    ...  
}
```

Replace the original TCP calls by calling through sk_proto_ops:

```
-    ret = tcp_sendmsg_locked(sk, &msg, size);  
+    ret = ctx->sk_proto_ops->sendmsg_locked(sk, &msg, size);  
  
-    inq = tcp_inq(strp->sk);  
+    inq = ctx->sk_proto_ops->peek_len(strp->sk->sk_socket);
```

```
-    tcp_read_sock(strp->sk, &desc, tls_strp_copyin);  
+    ctx->sk_proto_ops->read_sock(strp->sk, &desc, tls_strp_copyin);  
  
-    mask = tcp_poll(file, sock, wait);  
+    mask = ctx->sk_proto_ops->poll(file, sock, wait);
```



TLS Implementation

Receive-related Ops

Define a `tls_strparser_ops` structure:

```
struct tls_strparser_ops {
    int          protocol;
    bool (*recv_ready)(const struct sock *sk);
    struct sk_buff *(*recv_skb)(struct sock *sk, u32 *off);
    void (*read_done)(struct sock *sk, size_t len);
};
```

Define instances for TCP and MPTCP:

```
static const struct tls_strparser_ops tcp_tls_strp_ops = {
    .protocol      = IPPROTO_TCP,
    .recv_ready    = tls_sw_tcp_recv_ready,
    .recv_skb      = tls_sw_tcp_recv_skb,
    .read_done     = tcp_read_done,
};

const struct tls_strparser_ops mptcp_tls_strp_ops = {
    .protocol      = IPPROTO_MPTCP,
    .recv_ready    = mptcp_recv_ready,
    .recv_skb      = mptcp_recv_skb,
    .read_done     = mptcp_read_done,
};
```

Add a pointer to this structure in `tls_sw_context_rx`:

```
struct tls_sw_context_rx {
    ...
    struct tls_strparser strp;
+   const struct tls_strparser_ops *strp_ops;
    ...
};
```

Set the pointer to a different instance based on the protocol:

```
void tls_sw_strparser_arm(struct sock *sk, struct tls_context *tls_ctx)
{
    ...
+   ops = tls_strp_lookup_ops(tls_proto_id(sk->sk_protocol));
    ...
+   rx_ctx->strp_ops = ops;
    write_unlock_bh(&sk->sk_callback_lock);
}
```

Replace TCP calls with `strp_ops` in rx context:

```
-   if (likely(qshort && !tcp_epollin_ready(strp->sk)))
+   if (likely(qshort && !rx->strp_ops->recv_ready(strp->sk)))

-   first = tcp_recv_skb(strp->sk, &offset);
+   first = rx->strp_ops->recv_skb(strp->sk, &offset);

-   tcp_read_done(strp->sk, strp->stm.full_len);
+   ctx->strp_ops->read_done(strp->sk, strp->stm.full_len);
```



TLS Implementation

The Final Two

- **check_app_limited**
mptcp: track app-limited state in mptcp_sendmsg_locked
tls: drop duplicate check_app_limited in tls_push_sg
- **TCP_SKB_CB**
mptcp: sync mptcp skb cb layout with tcp one (Paolo Abeni)
tls: read skb sequence through a TLS_SKB_CB overlay



Bugfixes

- **A deadlock in mptcp_read_sock**
mptcp: defer read_sock cleanup to mptcp_worker (Paolo Abeni)
- **TLS decryption failed on overlapping skbs**
tls: fix RX desync on overlapping skbs (Maximilian Immanuel Brandtner)
- **TLS operation cache shared by TCP and MPTCP caused a crash**
tls: add protocol dimension to tls operation cache



Limitations

- **TLS is not supported by fallback MPTCP sockets**
Will be supported in the future
- **Hardware TLS offload is not supported**
Per-subflow hardware TLS offload is very difficult to implement on MPTCP
Possible approach: trigger MPTCP fallback

Selftests

Selftest Test: ./mptcp_tls.sh

```
# # Starting 884 tests from 13 test cases.
# # RUN      tls.12_aes_gcm_mptcp.sendfile ...
# #          OK  tls.12_aes_gcm_mptcp.sendfile
# ok 1 tls.12_aes_gcm_mptcp.sendfile
# # RUN      tls.12_aes_gcm_mptcp.send_then_sendfile ...
# #          OK  tls.12_aes_gcm_mptcp.send_then_sendfile
# ok 2 tls.12_aes_gcm_mptcp.send_then_sendfile
# # RUN      tls.12_aes_gcm_mptcp.multi_chunk_sendfile ...
# #          OK  tls.12_aes_gcm_mptcp.multi_chunk_sendfile
# ok 3 tls.12_aes_gcm_mptcp.multi_chunk_sendfile
# # RUN      tls.12_aes_gcm_mptcp.recv_max ...
# #          OK  tls.12_aes_gcm_mptcp.recv_max
# ok 4 tls.12_aes_gcm_mptcp.recv_max
# # RUN      tls.12_aes_gcm_mptcp.recv_small ...
# #          OK  tls.12_aes_gcm_mptcp.recv_small
# ok 5 tls.12_aes_gcm_mptcp.recv_small
# # RUN      tls.12_aes_gcm_mptcp.msg_more ...
# #          OK  tls.12_aes_gcm_mptcp.msg_more
# ok 6 tls.12_aes_gcm_mptcp.msg_more
# # RUN      tls.12_aes_gcm_mptcp.cmsg_msg_more ...
# #          OK  tls.12_aes_gcm_mptcp.cmsg_msg_more
# ok 7 tls.12_aes_gcm_mptcp.cmsg_msg_more
# # RUN      tls.12_aes_gcm_mptcp.msg_more_then_cmsg ...
# #          OK  tls.12_aes_gcm_mptcp.msg_more_then_cmsg
# ok 8 tls.12_aes_gcm_mptcp.msg_more_then_cmsg
... ..
```

```
... ..
# # RUN      tls.12_aria_gcm_256_mptcp.rekey ...
# #          OK  tls.12_aria_gcm_256_mptcp.rekey
# ok 877 tls.12_aria_gcm_256_mptcp.rekey
# # RUN      tls.12_aria_gcm_256_mptcp.rekey_fail ...
# #          OK  tls.12_aria_gcm_256_mptcp.rekey_fail
# ok 878 tls.12_aria_gcm_256_mptcp.rekey_fail
# # RUN      tls.12_aria_gcm_256_mptcp.rekey_peek ...
# #          OK  tls.12_aria_gcm_256_mptcp.rekey_peek
# ok 879 tls.12_aria_gcm_256_mptcp.rekey_peek
# # RUN      tls.12_aria_gcm_256_mptcp.splice_rekey ...
# #          OK  tls.12_aria_gcm_256_mptcp.splice_rekey
# ok 880 tls.12_aria_gcm_256_mptcp.splice_rekey
# # RUN      tls.12_aria_gcm_256_mptcp.rekey_peeksplice ...
# #          OK  tls.12_aria_gcm_256_mptcp.rekey_peeksplice
# ok 881 tls.12_aria_gcm_256_mptcp.rekey_peeksplice
# # RUN      tls.12_aria_gcm_256_mptcp.rekey_getsockopt ...
# #          OK  tls.12_aria_gcm_256_mptcp.rekey_getsockopt
# ok 882 tls.12_aria_gcm_256_mptcp.rekey_getsockopt
# # RUN      tls.12_aria_gcm_256_mptcp.rekey_poll_pending ...
# #          OK  tls.12_aria_gcm_256_mptcp.rekey_poll_pending
# ok 883 tls.12_aria_gcm_256_mptcp.rekey_poll_pending
# # RUN      tls.12_aria_gcm_256_mptcp.rekey_poll_delay ...
# #          OK  tls.12_aria_gcm_256_mptcp.rekey_poll_delay
# ok 884 tls.12_aria_gcm_256_mptcp.rekey_poll_delay
# # PASSED: 884 / 884 tests passed.
# # Totals: pass:884 fail:0 xfail:0 xpass:0 skip:0 error:0
```

All cipher types work on MPTCP for both TLS 1.2 and TLS 1.3.



Validation

Using TLS for NVMe over MPTCP:

```
# nvme discover -a 10.1.1.1 --tos=0x10 --host-iface=ns2eth1 --tls
# Discovery Log Number of Records 2, Generation counter 2
...
# =====Discovery Log Entry 1=====
# trtype: mptcp
# adrfam: ipv4
# subtype: nvme subsystem
# treq: required, sq flow control disable supported
# portid: 20541
# trsvcid: 25256
# subnqn: nqn.2014-08.org.nvmexpress.mptcpdev.497.24223
# traddr: 10.1.1.1
# eflags: none
# sectype: tls13
# Connecting to 10.1.1.1:25256 --tos=0x10 --host-iface=ns2eth1 --tls
nvmet: Created nvm controller 1 for subsystem nqn.2014-08.org.nvmexpress.mptcpdev.497.24223 for
      NQN nqn.2014-08.org.nvmexpress:uuid:f7f6b5e0-ff97-4894-98ac-c85309e0bc77, TLS.
nvme nvme0: creating 24 I/O queues.
nvme nvme0: mapped 24/0/0 default/read/poll queues.
nvme nvme0: new ctrl: NQN "nqn.2014-08.org.nvmexpress.mptcpdev.497.24223", addr 10.1.1.1:25256,
      hostnqn: nqn.2014-08.org.nvmexpress:uuid:f7f6b5e0-ff97-4894-98ac-c85309e0bc77
# PASS: mptcp socket bound to ns2eth1
# nvme list
...
# /dev/nvme0n1 /dev/ng0n1 4ed80418956e8730c1e5 Linux 0x1 536.87 MB / 536.87 MB 512 B + 0 B 7.3.0-rc
# fio randread /dev/nvme0n1
...
# fio-3.41
# Starting 4 threads
#
# Run status group 0 (all jobs):
#   READ: bw=107MiB/s (112MB/s), 107MiB/s-107MiB/s (112MB/s-112MB/s), io=1193MiB (1251MB), run=11202-11202msec
#
# Disk stats (read/write):
#   nvme0n1: ios=9528/0, sectors=2439168/0, merge=0/0, ticks=8649677/0, in_queue=8649677, util=99.20%
...
# fio randwrite /dev/nvme0n1
...
```



References

[1] MPTCP and KTLS support #480

https://github.com/multipath-tcp/mptcp_net-next/issues/480

[2] implement mptcp read_sock

<https://lore.kernel.org/all/20260130-net-next-mptcp-splice-v2-0-31332ba70d7f@kernel.org/>

[3] Reduce the differences between TCP and MPTCP for TLS usage

<https://patchwork.kernel.org/project/mptcp/cover/cover.1788245079.git.tanggeliang@kylinos.cn/>

[4] MPTCP KTLS support

<https://patchwork.kernel.org/project/mptcp/cover/cover.1763438045.git.tanggeliang@kylinos.cn/>

[5] tlshd: add mptcp support

<https://github.com/oracle/ktls-utils/commit/8fd3c7071e195206c4119575f40546f7243cc4c1>



Questions