NVMe over MPTCP

Multi-Fold Acceleration for NVMe over TCP

in Multi-NIC Environments

Geliang Tang - MPTCP Community
Kylin Software

LSF/MM/BPF 2026
Zagreb, May 2026

Thanks to Hannes Reinecke, Matthieu Baerts, Mat Martineau, Paolo Abeni, Coly Li, Barry Song, Ming Lei, Zhenwei Pi, Jason Xing, Hui Zhu, Gang Yan, and others for their help during the development of NVMe over MPTCP.

Agenda

- Introduction to MPTCP
- Motivation
- Current Status
- Performance Evaluation
- Implementation
- Discussion

Introduction to 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
- How to use it?
  - Address config: **ip mptcp** (manual); **mptcpd** daemon or NetworkManager (automatic).
  - Create socket: socket(AF_INET, SOCK_STREAM, **IPPROTO_MPTCP**) or force with **BPF**.
  - Use: same as TCP sockets; multiple TCP **subflows** are created in kernel space to send and receive data.
- For more details, please visit the project website: https://www.mptcp.dev

Motivation

- **Provide a kernel-space use case for MPTCP**
  -> Drives missing MPTCP features forward: .read_sock, KTLS, etc.

- **Achieve multi−fold bandwidth aggregation in multi−NIC environments**
  -> Break the single-device bottleneck of NVMe TCP

- **Improve reliability with transparent fail-over across network paths**
  -> Without adding complexity to NVMe layer

Current Status

The modifications in the NVMe subsystem are minimal, most of the changes are on the MPTCP side.

Here are the patchsets:

- mptcp: implement .read_sock **(Mainlined)**
- **NVMe over MPTCP** **(Under Review)**
- MPTCP KTLS support **(Under Review)**
- Enable KTLS for NVMe MPTCP **(Under development)**
- User space changes **(Under development)**
- Fixes for MPTCP corner cases **(Under Review)**

Performance Evaluation (NVMe TCP)

This demo is the selftest script for NVMe over MPTCP

```
# ns1 (target)  ns2 (host)
# 10.1.1.1      10.1.1.2
# 10.1.2.1      10.1.2.2
# 10.1.3.1      10.1.3.2
# 10.1.4.1      10.1.4.2
```

Each network device is limited to a speed of **125 MB/s** (1000 mbit/s)

```
# ./mptcp_nvme.sh tcp numa tls
  READ: bw=112MiB/s (118MB/s), 112MiB/s-112MiB/s (118MB/s-118MB/s),
                io=1123MiB (1177MB), run=10018-10018msec
  WRITE: bw=112MiB/s (117MB/s), 112MiB/s-112MiB/s (117MB/s-117MB/s),
                io=1118MiB (1173MB), run=10018-10018msec
```

Performance Evaluation (NVMe MPTCP)

target:
ip mptcp endpoint add 10.1.2.1 signal
ip mptcp endpoint add 10.1.3.1 signal
ip mptcp endpoint add 10.1.4.1 signal

host:
ip mptcp endpoint add 10.1.2.2 subflow
ip mptcp endpoint add 10.1.3.2 subflow
ip mptcp endpoint add 10.1.4.2 subflow

./mptcp_nvme.sh **mptcp** numa tls
READ: bw=427MiB/s (**448MB/s**), 427MiB/s-427MiB/s (448MB/s-448MB/s),
io=4286MiB (4494MB), run=10039-10039msec
WRITE: bw=387MiB/s (**406MB/s**), 387MiB/s-387MiB/s (406MB/s-406MB/s),
io=3885MiB (4073MB), run=10043-10043msec

In a four-network-device environment, MPTCP achieves a **four times** performance improvement.

Implementation

TCP is currently hardcoded in a few places:

Target Side

- ret = sock_create(port->addr.ss_family, SOCK_STREAM, **IPPROTO_TCP**, &port->sock);
- tcp_sock_set_nodelay(port->sock->sk); (**list_add corruption**)
- sock_set_reuseaddr(port->sock->sk); (**Address already in use**)
- sock_set_priority(port->sock->sk, so_priority);
- sock_no_linger(sock->sk);
- ip_sock_set_tos(sock->sk, inet->rcv_tos);

Host Side

- ret = sock_create_kern(current->nsproxy->net_ns, ctrl->addr.ss_family, SOCK_STREAM, **IPPROTO_TCP**, &queue->sock);
- tcp_sock_set_syncnt(queue->sock->sk, 1);
- tcp_sock_set_nodelay(queue->sock->sk);
- sock_no_linger(queue->sock->sk); (**nvme disconnect**)
- sock_set_priority(queue->sock->sk, so_priority);
- ip_sock_set_tos(queue->sock->sk, nctrl->opts->tos);

Target Implementation

- Define a **target tcp_proto** structure:

```
+struct nvmet_tcp_proto {
+       int                     protocol;
+       void (*set_reuseaddr)(struct sock *sk);
+       void (*set_nodelay)(struct sock *sk);
+       void (*set_priority)(struct sock *sk, u32 priority);
+       void (*no_linger)(struct sock *sk);
+       void (*set_tos)(struct sock *sk, int val);
+       const struct nvmet_fabrics_ops *ops;
+};
```

- Add a pointer to this structure in **tcp_port:**

```
 struct nvmet_tcp_port {
+       struct rcu_head         rcu;
        ... ...
        void (*data_ready)(struct sock *);
+       const struct nvmet_tcp_proto *proto;
 };
```

- Initialize the pointer to a different instance based on transport type (**TCP or MPTCP**) in add_port callback:

```
 if (nport->disc_addr.trtype == NVMF_TRTYPE_TCP) {
        port->proto = &nvmet_tcp_proto;
 } else if (nport->disc_addr.trtype == NVMF_TRTYPE_MPTCP) {
        port->proto = &nvmet_mptcp_proto;
 }
```

- Define **nvmet_tcp_proto / nvmet_mptcp_proto** for TCP / MPTCP:

```
+static const struct nvmet_tcp_proto nvmet_mptcp_proto = {
+       .protocol               = IPPROTO_MPTCP,
+       .set_reuseaddr          = mptcp_sock_set_reuseaddr,
+       .set_nodelay            = mptcp_sock_set_nodelay,
+       .set_priority           = mptcp_sock_set_priority,
+       .no_linger              = mptcp_sock_no_linger,
+       .set_tos                = mptcp_sock_set_tos,
+       .ops                    = &nvmet_mptcp_ops,
+};
```

- New **transport type and fabrics ops** for MPTCP:

```
+static const struct nvmet_fabrics_ops nvmet_mptcp_ops = {
+       .owner                  = THIS_MODULE,
+       .type                   = NVMF_TRTYPE_MPTCP,
+       .msdbd                  = 1,
+       .add_port               = nvmet_tcp_add_port,
        ... ...
+};
```

- Access the instance **through the pointer in tcp_port**, and use this instance afterwards:

```
        ret = sock_create(port->addr.ss_family,
                                SOCK_STREAM,
+                               port->proto->protocol,
                                &port->sock);

+       port->proto->set_reuseaddr(port->sock->sk);
+       port->proto->set_nodelay(port->sock->sk);
```

Host Implementation

- Define a **host tcp_proto** structure:

```
+struct nvme_tcp_proto {
+       int                     protocol;
+       int (*set_syncnt)(struct sock *sk, int val);
+       void (*set_nodelay)(struct sock *sk);
+       void (*no_linger)(struct sock *sk);
+       void (*set_priority)(struct sock *sk, u32 priority);
+       void (*set_tos)(struct sock *sk, int val);
+       const struct nvme_ctrl_ops *ops;
+};
```

- Add a pointer to this structure in **tcp_ctrl:**

```
 struct nvme_tcp_ctrl {
+       struct rcu_head         rcu;
        ... ...
        u32                     io_queues[HCTX_MAX_TYPES];
+       const struct nvme_tcp_proto *proto;
 };
```

- Initialize the pointer to a different instance based on transport name (**tcp or mptcp**) in nvme_tcp_alloc_ctrl:

```
 if (!strcmp(ctrl->ctrl.opts->transport, "tcp")) {
        rcu_assign_pointer(ctrl->proto, &nvme_tcp_proto);
 } else if (!strcmp(ctrl->ctrl.opts->transport, "mptcp")) {
        rcu_assign_pointer(ctrl->proto, &nvme_mptcp_proto);
 }
```

- Define **nvme_tcp_proto / nvme_mptcp_proto** for TCP / MPTCP:

```
+static const struct nvme_tcp_proto nvme_mptcp_proto = {
+       .protocol               = IPPROTO_MPTCP,
+       .set_syncnt             = mptcp_sock_set_syncnt,
+       .set_nodelay            = mptcp_sock_set_nodelay,
+       .no_linger              = mptcp_sock_no_linger,
+       .set_priority           = mptcp_sock_set_priority,
+       .set_tos                = mptcp_sock_set_tos,
+       .ops                    = &nvme_mptcp_ctrl_ops,
+};
```

- New **transport ops** for MPTCP:

```
+static struct nvmf_transport_ops nvme_mptcp_transport = {
+       .name                   = "mptcp",
+       .module                 = THIS_MODULE,
+       ...
+       .create_ctrl            = nvme_tcp_create_ctrl,
+};
```

- Access the instance **through the pointer in tcp_ctrl**, and use this instance afterwards:

```
        ret = sock_create_kern(current->nsproxy->net_ns,
                        ctrl->addr.ss_family, SOCK_STREAM,
+                       proto->protocol, &queue->sock);

+       proto->set_syncnt(queue->sock->sk, 1);
+       proto->set_nodelay(queue->sock->sk);
```

Discussion

- **mptcpize / mptcpify tools do not work on NVMe TCP**

Keep the NVMe code unchanged, can we just use MPTCP tools to convert NVMe TCP to use MPTCP by force? **The answer is no**.

- **NVMe MPTCP vs. NVMe multipath**

Can NVMe MPTCP work with NVMe multipath? Both names contain 'multipath'. **The answer is yes**. (I/O policy)

- **NVMe MPTCP vs. Block multiqueue**

Can NVMe MPTCP work with Block multiqueue? **The answer is yes**. All queues - both admin queue and I/O queues - are MPTCP connections.

- **How to select MPTCP?**

-> A new transport type?
-> A TCP variant?

A new transport type:
+ straightforward
- Update of the NVMe protocol -> How?

A new TCP 'variant':
+ smaller update of the protocol
-> How: using Transport Specific Address Subtype (TSAS) to represent a new Protocol Type

Contact information

- **GitHub:**
  @geliangtang

- **Email:**
  geliang@kernel.org

- **MPTCP Community:**
  Website: https://www.mptcp.dev/
  Repositories: https://github.com/multipath-tcp/
  Mailing List: mptcp@lists.linux.dev

References

- **NVMe over MPTCP**

https://lore.kernel.org/all/cover.1775047736.git.tanggeliang@kylinos.cn/

- **mptcp: implement .read_sock and .splice_read**

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

- **MPTCP KTLS support**

https://lore.kernel.org/all/cover.1776469068.git.tanggeliang@kylinos.cn/

- **libnvme: add mptcp trtype**

https://lore.kernel.org/all/99f6e63b5c9677f29a9bc8cdd87b2064b258435f.1764206766.git.tanggeliang@kylinos.cn/

- **fabrics: add mptcp support**

https://github.com/linux-nvme/nvme-cli/commit/f468531d0592ad22b71760d883409363b1f8a9d6

- **tlshd: add mptcp support**

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

- **[LSF/MM/BPF TOPIC] NVMe over MPTCP**

https://lore.kernel.org/linux-nvme/a9f115aa5719e1088702a3fdeee766a3166611b1.camel@kernel.org/

Questions