第三届 eBPF开发者大会

www.ebpftravel.com

BPF on MPTCP

麒麟软件 唐葛亮
geliang@kernel.org

中国·西安

目录

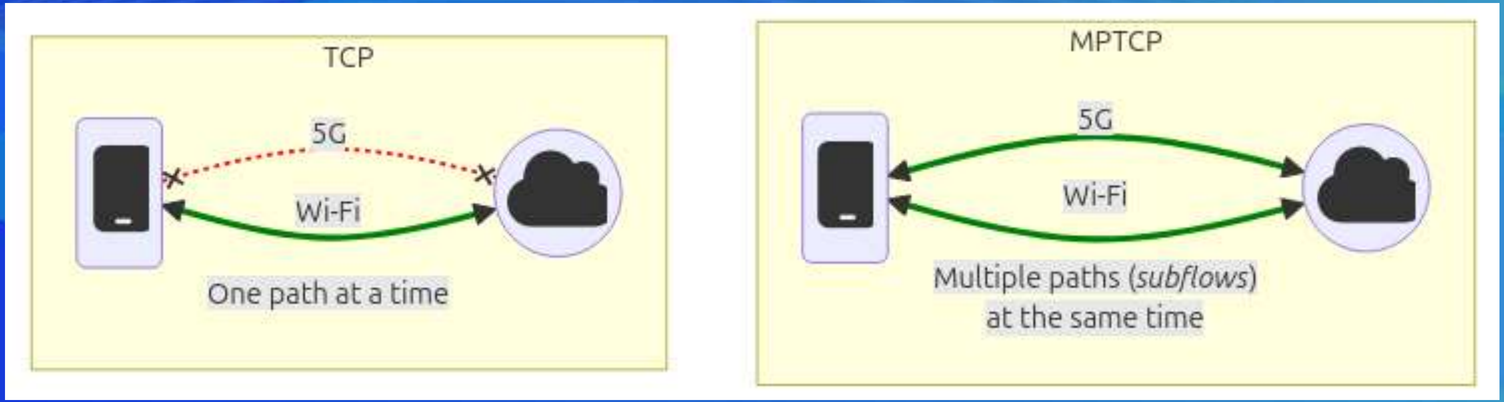
CONTENTS

Geliang Tang

- 麒麟软件研发技术专家
- Linux内核MPTCP（Multipath TCP）Maintainer之一（另外两位是Matthieu Baerts和Mat Martineau）
- openEuler技术委员会委员（2025-2026届）
- 2015年开始贡献Linux内核，累计贡献900+补丁（华人TOP 13）
- 2019年开始专注于MPTCP网络协议栈开发，是该模块的核心开发者，TOP贡献者，Reviewer和Maintainer

MPTCP介绍

- 多路径TCP或MPTCP是标准TCP的扩展，在RFC 8684（TCP Extensions for Multipath Operation with Multiple Addresses）中描述（MPTCP RFC也由我们维护）
- 它允许设备同时使用多个接口通过一条MPTCP连接来发送和接收TCP数据包
- MPTCP可以聚合多个接口的带宽或优先选择延迟最低的接口，如果一条路径发生故障，它还允许故障转移，并且流量会无缝地重新注入其他路径



MPTCP性能演示

为iperf、rsync和Valkey添加MPTCP原生支持

```
fd = socket(AF_INET(6), SOCK_STREAM, IPPROTO_MPTCP)
```

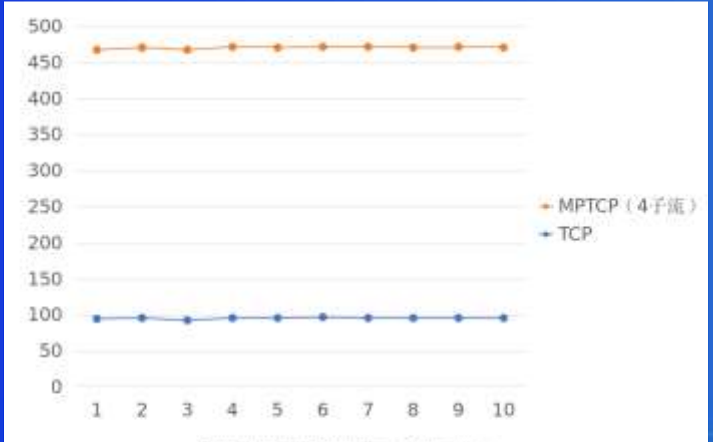
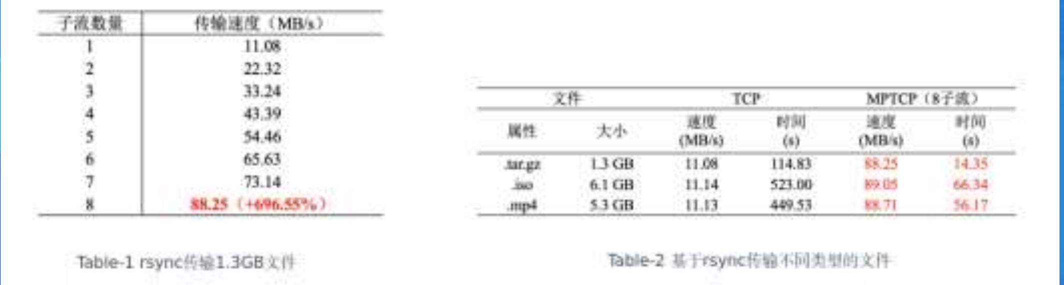


Figure-2. iperf测量MPTCP（4子流）与TCP带宽对比图

iperf带宽测试



子流数量	传输速度（MB/s）
1	11.08
2	22.32
3	33.24
4	43.39
5	54.46
6	65.63
7	73.14
8	88.25（+696.55%）

Table-1 rsync传输1.3GB文件

文件		TCP		MPTCP（8子流）	
属性	大小	速度 (MB/s)	时间 (s)	速度 (MB/s)	时间 (s)
.tar.gz	1.3 GB	11.08	114.83	88.25	14.35
.iso	6.1 GB	11.14	523.00	89.05	66.34
.mp4	5.3 GB	11.13	449.53	88.71	56.17

Table-2 基于rsync传输不同类型的文件

rsync文件传输测试

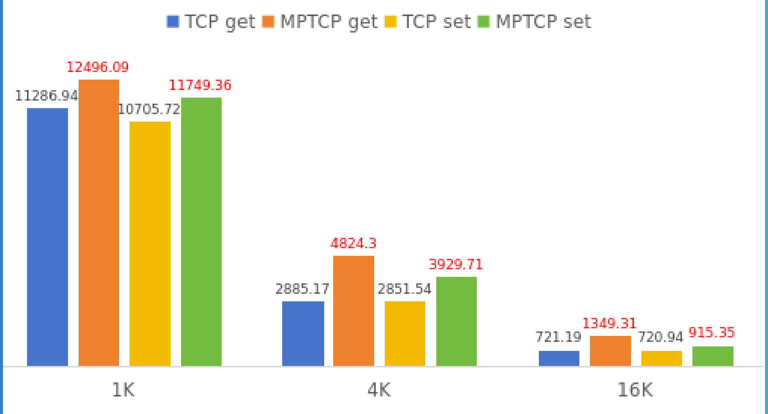


Figure-3. 并发量rpc性能指标

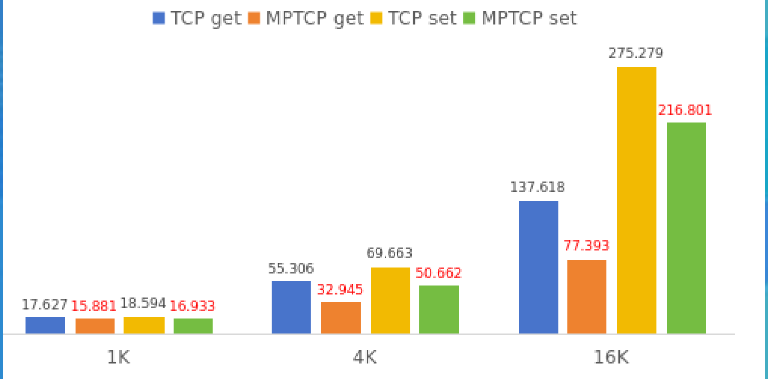
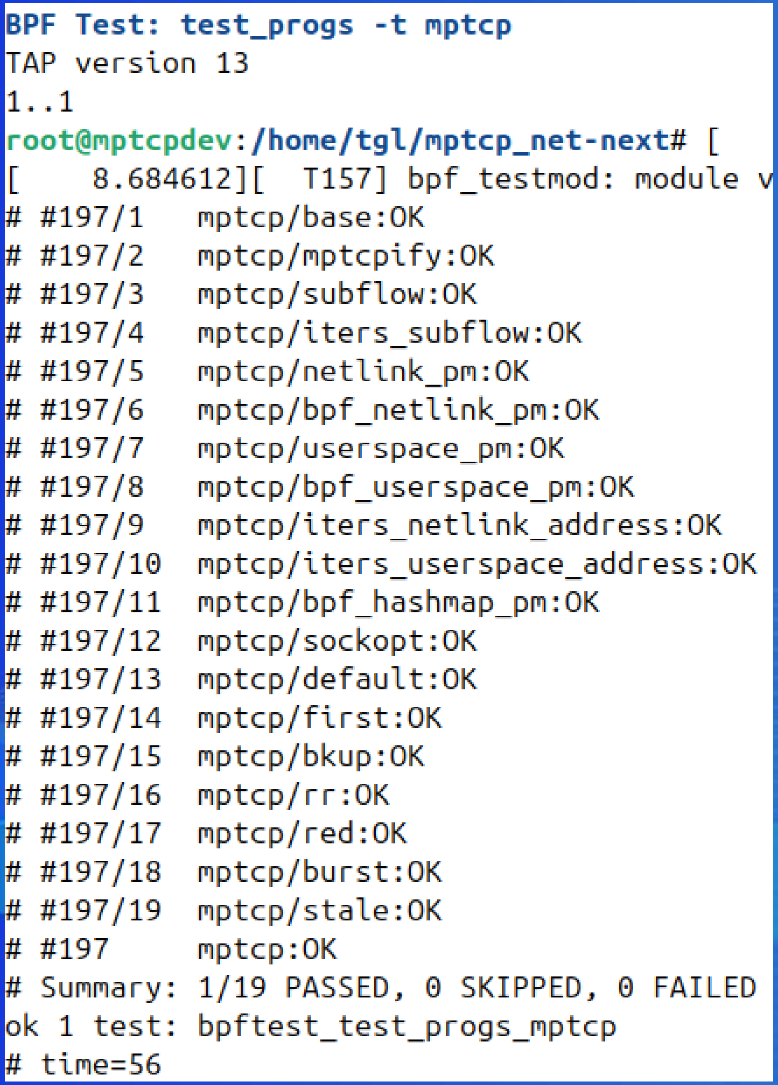


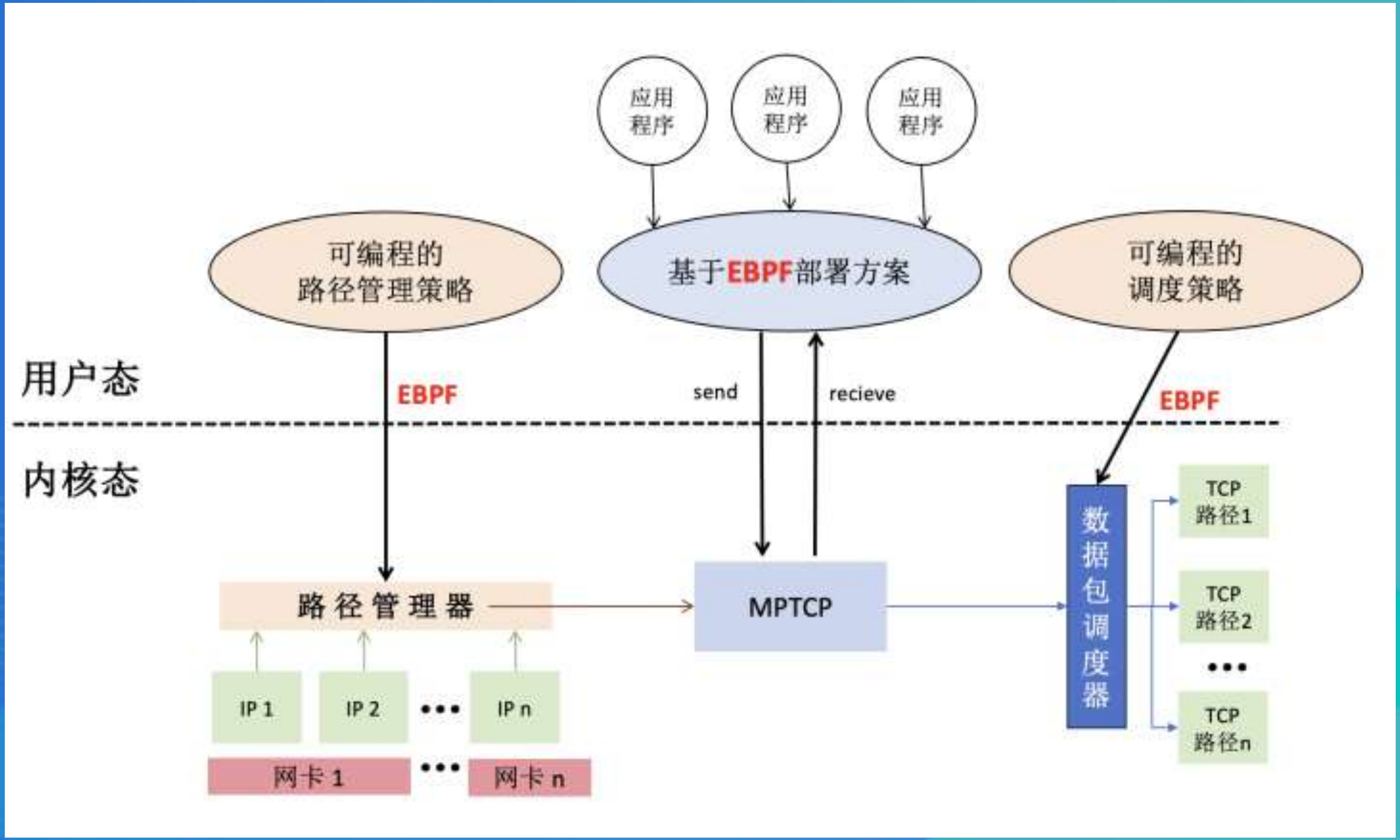
Figure-4. 时间延迟latency指标(ms)

Valkey over MPTCP

MPTCP BPF selftests



```
BPF Test: test_progs -t mptcp
TAP version 13
1..1
root@mptcpdev:/home/tgl/mptcp_net-next# [
[    8.684612][  T157] bpf_testmod: module v
# #197/1    mptcp/base:OK
# #197/2    mptcp/mptcpify:OK
# #197/3    mptcp/subflow:OK
# #197/4    mptcp/iters_subflow:OK
# #197/5    mptcp/netlink_pm:OK
# #197/6    mptcp/bpf_netlink_pm:OK
# #197/7    mptcp/userspace_pm:OK
# #197/8    mptcp/bpf_userspace_pm:OK
# #197/9    mptcp/iters_netlink_address:OK
# #197/10   mptcp/iters_userspace_address:OK
# #197/11   mptcp/bpf_hashmap_pm:OK
# #197/12   mptcp/sockopt:OK
# #197/13   mptcp/default:OK
# #197/14   mptcp/first:OK
# #197/15   mptcp/bkup:OK
# #197/16   mptcp/rr:OK
# #197/17   mptcp/red:OK
# #197/18   mptcp/burst:OK
# #197/19   mptcp/stale:OK
# #197      mptcp:OK
# Summary: 1/19 PASSED, 0 SKIPPED, 0 FAILED
ok 1 test: bpftest_test_progs_mptcp
# time=56
```

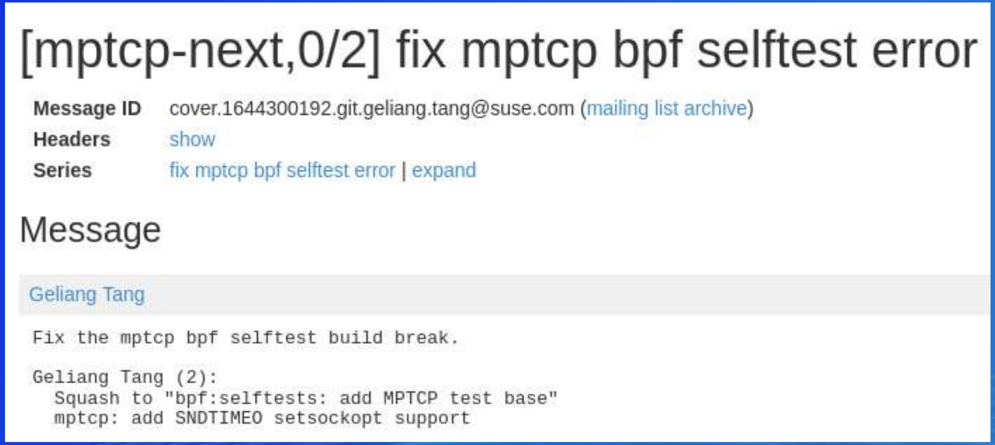


目录

CONTENTS

BPF: Support for mptcp_sock

最初由Nicolas开发，WIP，2022年初本人接手MPTCP BPF开发



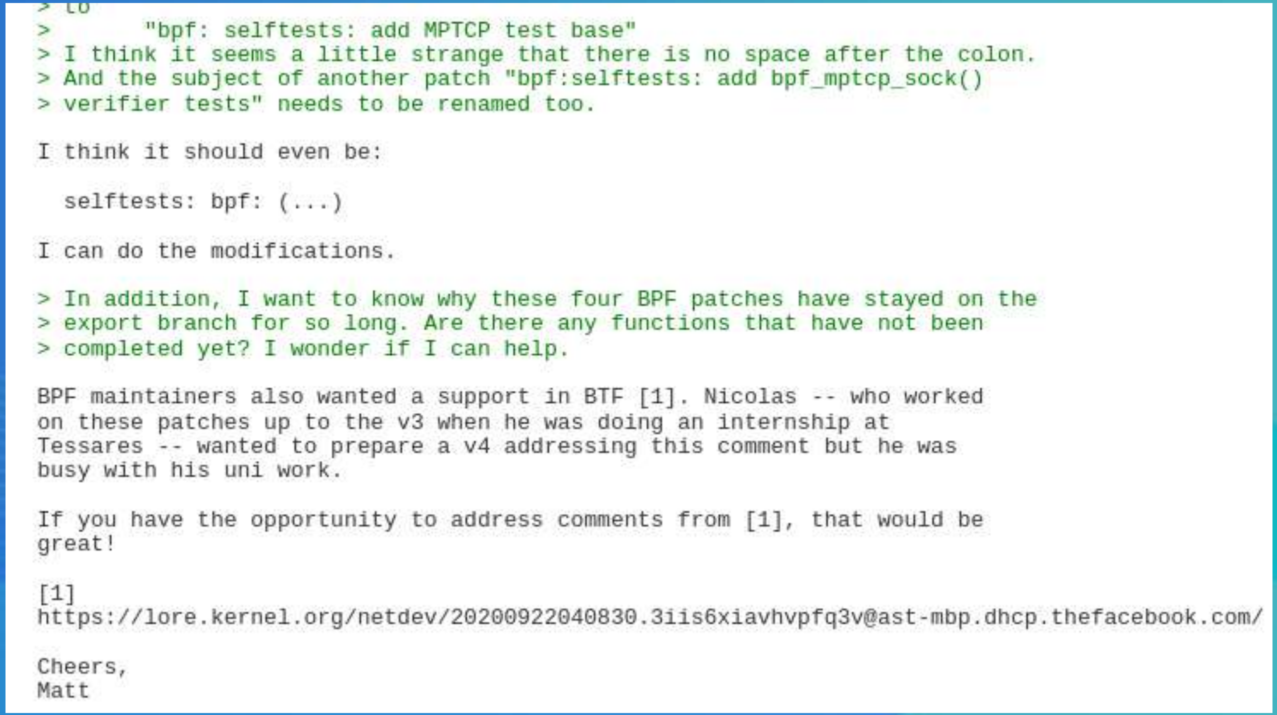
[mptcp-next,0/2] fix mptcp bpf selftest error

Message ID	cover.1644300192.git.geliang.tang@suse.com (mailing list archive)
Headers	show
Series	fix mptcp bpf selftest error | expand

Message

Geliang Tang

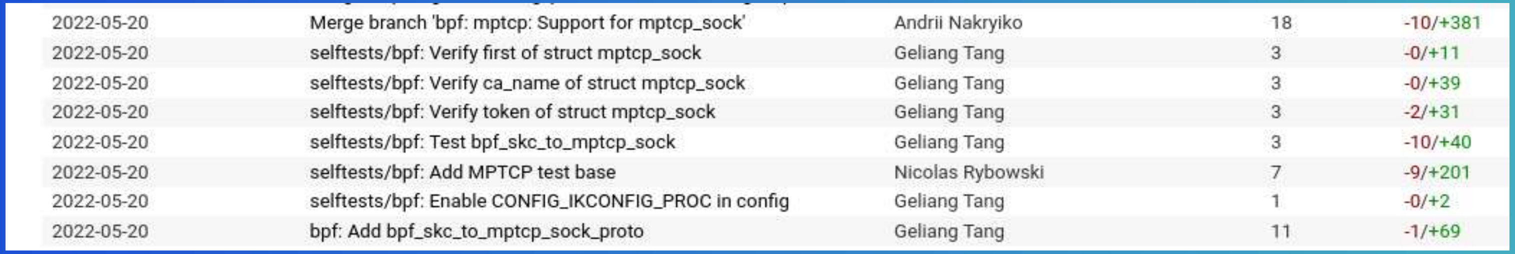
```
Fix the mptcp bpf selftest build break.

Geliang Tang (2):
  Squash to "bpf:selftests: add MPTCP test base"
  mptcp: add SNDTIMEO setsockopt support
```



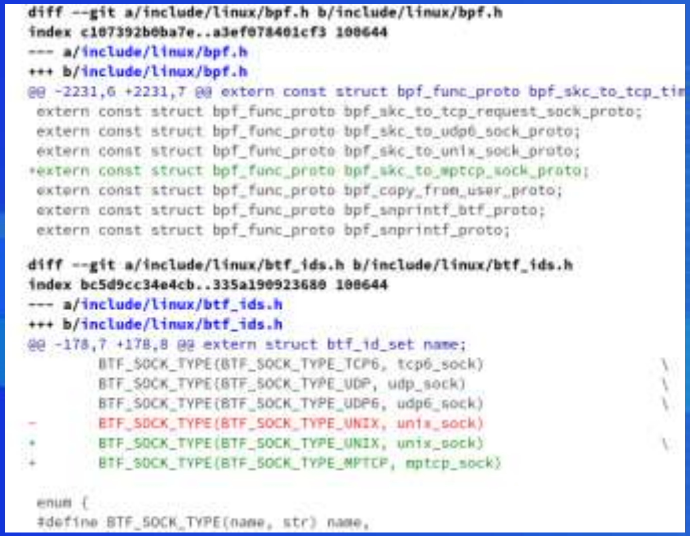
```
>         "bpf: selftests: add MPTCP test base"
> I think it seems a little strange that there is no space after the colon.
> And the subject of another patch "bpf:selftests: add bpf_mptcp_sock()
> verifier tests" needs to be renamed too.

I think it should even be:

  selftests: bpf: (...)

I can do the modifications.

> In addition, I want to know why these four BPF patches have stayed on the
> export branch for so long. Are there any functions that have not been
> completed yet? I wonder if I can help.

BPF maintainers also wanted a support in BTF [1]. Nicolas -- who worked
on these patches up to the v3 when he was doing an internship at
Tessares -- wanted to prepare a v4 addressing this comment but he was
busy with his uni work.

If you have the opportunity to address comments from [1], that would be
great!

[1]
https://lore.kernel.org/netdev/20200922040830.3iis6xiavhvpfq3v@ast-mbp.dhcp.thefacebook.com/

Cheers,
Matt
```

BPF: Support for mptcp_sock

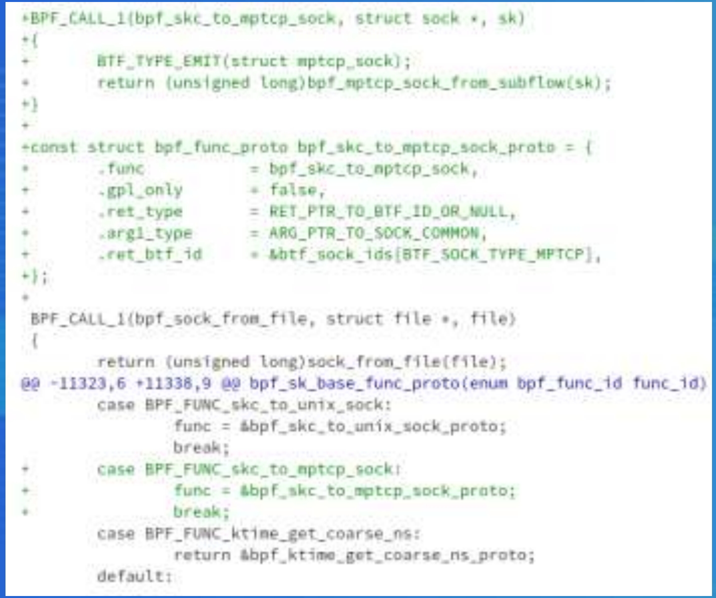
主要实现bpf_skc_to_mptcp_sock函数，从子链路获取mptcp_sock信息



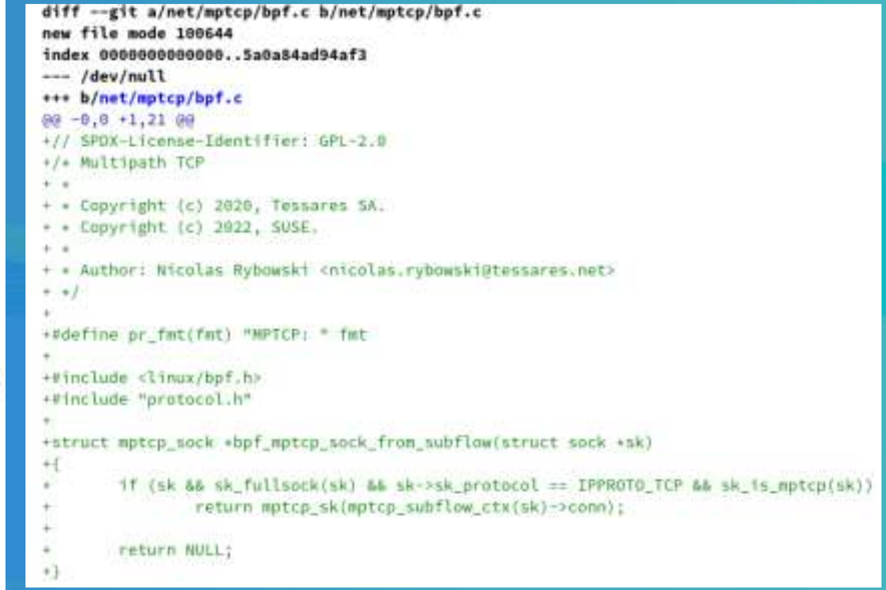
2022-05-20	Merge branch 'bpf: mptcp: Support for mptcp_sock'	Andrii Nakryiko	18	-10/+381
2022-05-20	selftests/bpf: Verify first of struct mptcp_sock	Geliang Tang	3	-0/+11
2022-05-20	selftests/bpf: Verify ca_name of struct mptcp_sock	Geliang Tang	3	-0/+39
2022-05-20	selftests/bpf: Verify token of struct mptcp_sock	Geliang Tang	3	-2/+31
2022-05-20	selftests/bpf: Test bpf_skc_to_mptcp_sock	Geliang Tang	3	-10/+40
2022-05-20	selftests/bpf: Add MPTCP test base	Nicolas Rybowski	7	-9/+201
2022-05-20	selftests/bpf: Enable CONFIG_IKCONFIG_PROC in config	Geliang Tang	1	-0/+2
2022-05-20	bpf: Add bpf_skc_to_mptcp_sock_proto	Geliang Tang	11	-1/+69



```
diff --git a/include/linux/bpf.h b/include/linux/bpf.h
index c107392b0ba7e..a3ef078401cf3 100644
--- a/include/linux/bpf.h
+++ b/include/linux/bpf.h
@@ -2231,6 +2231,7 @@ extern const struct bpf_func_proto bpf_skc_to_tcp_tim
 extern const struct bpf_func_proto bpf_skc_to_tcp_request_sock_proto;
 extern const struct bpf_func_proto bpf_skc_to_udp6_sock_proto;
 extern const struct bpf_func_proto bpf_skc_to_unix_sock_proto;
+extern const struct bpf_func_proto bpf_skc_to_mptcp_sock_proto;
 extern const struct bpf_func_proto bpf_copy_from_user_proto;
 extern const struct bpf_func_proto bpf_snprintf_btf_proto;
 extern const struct bpf_func_proto bpf_snprintf_proto;

diff --git a/include/linux/btf_ids.h b/include/linux/btf_ids.h
index bc5d9cc34e4cb..335a190923680 100644
--- a/include/linux/btf_ids.h
+++ b/include/linux/btf_ids.h
@@ -178,7 +178,8 @@ extern struct btf_id_set name;
         BTF_SOCK_TYPE(BTF_SOCK_TYPE_TCP6, tcp6_sock)                   \
         BTF_SOCK_TYPE(BTF_SOCK_TYPE_UDP, udp_sock)                     \
         BTF_SOCK_TYPE(BTF_SOCK_TYPE_UDP6, udp6_sock)                   \
-        BTF_SOCK_TYPE(BTF_SOCK_TYPE_UNIX, unix_sock)
+        BTF_SOCK_TYPE(BTF_SOCK_TYPE_UNIX, unix_sock)                   \
+        BTF_SOCK_TYPE(BTF_SOCK_TYPE_MPTCP, mptcp_sock)

 enum {
 #define BTF_SOCK_TYPE(name, str) name,
```



```
+BPF_CALL_1(bpf_skc_to_mptcp_sock, struct sock *, sk)
+{
+        BTF_TYPE_EMIT(struct mptcp_sock);
+        return (unsigned long)bpf_mptcp_sock_from_subflow(sk);
+}
+
+const struct bpf_func_proto bpf_skc_to_mptcp_sock_proto = {
+        .func           = bpf_skc_to_mptcp_sock,
+        .gpl_only       = false,
+        .ret_type       = RET_PTR_TO_BTF_ID_OR_NULL,
+        .arg1_type      = ARG_PTR_TO_SOCK_COMMON,
+        .ret_btf_id     = &btf_sock_ids[BTF_SOCK_TYPE_MPTCP],
+};
+
 BPF_CALL_1(bpf_sock_from_file, struct file *, file)
 {
         return (unsigned long)sock_from_file(file);
@@ -11323,6 +11338,9 @@ bpf_sk_base_func_proto(enum bpf_func_id func_id)
         case BPF_FUNC_skc_to_unix_sock:
                 func = &bpf_skc_to_unix_sock_proto;
                 break;
+        case BPF_FUNC_skc_to_mptcp_sock:
+                func = &bpf_skc_to_mptcp_sock_proto;
+                break;
         case BPF_FUNC_ktime_get_coarse_ns:
                 return &bpf_ktime_get_coarse_ns_proto;
         default:
```



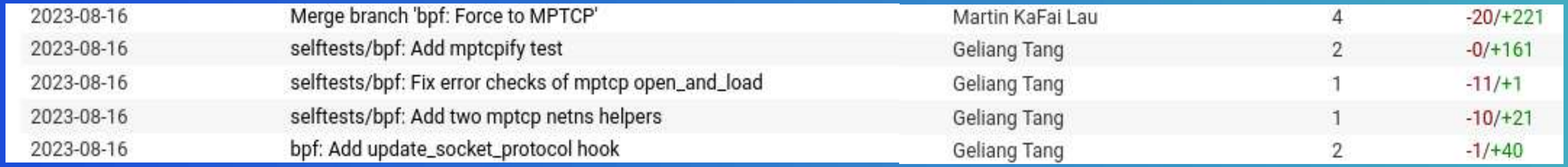
```
diff --git a/net/mptcp/bpf.c b/net/mptcp/bpf.c
new file mode 100644
index 0000000000000..5a0a84ad94af3
--- /dev/null
+++ b/net/mptcp/bpf.c
@@ -0,0 +1,21 @@
+// SPDX-License-Identifier: GPL-2.0
+/* Multipath TCP
+ *
+ * Copyright (c) 2020, Tessares SA.
+ * Copyright (c) 2022, SUSE.
+ *
+ * Author: Nicolas Rybowski <nicolas.rybowski@tessares.net>
+ */
+
+#define pr_fmt(fmt) "MPTCP: " fmt
+
+#include <linux/bpf.h>
+#include "protocol.h"
+
+struct mptcp_sock *bpf_mptcp_sock_from_subflow(struct sock *sk)
+{
+        if (sk && sk_fullsock(sk) && sk->sk_protocol == IPPROTO_TCP && sk_is_mptcp(sk))
+                return mptcp_sk(mptcp_subflow_ctx(sk)->conn);
+
+        return NULL;
+}
```

目录

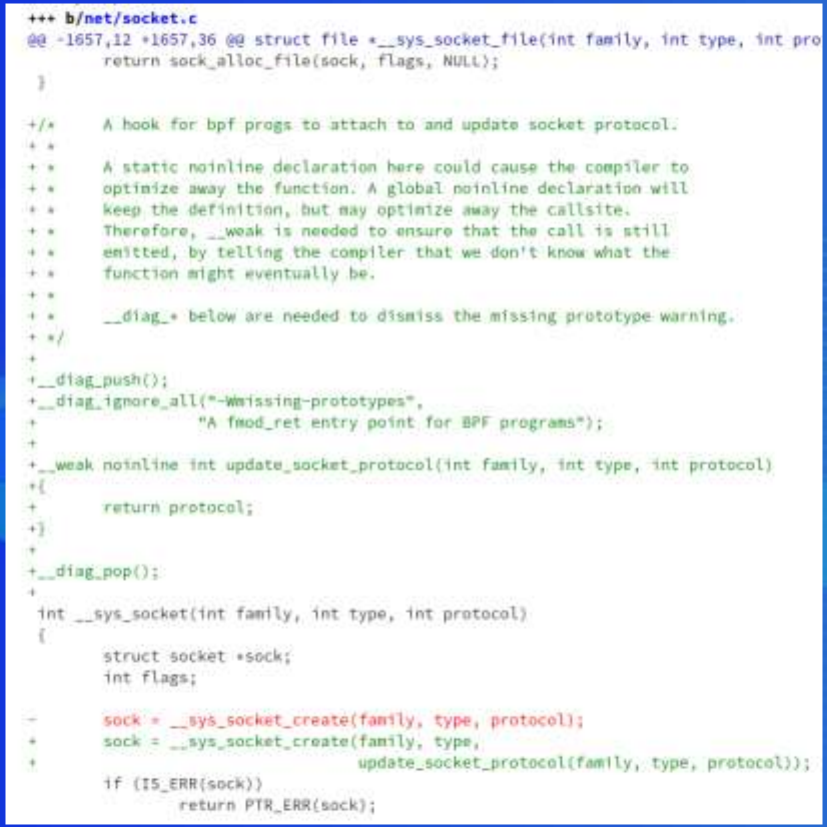
CONTENTS

BPF: Force to MPTCP

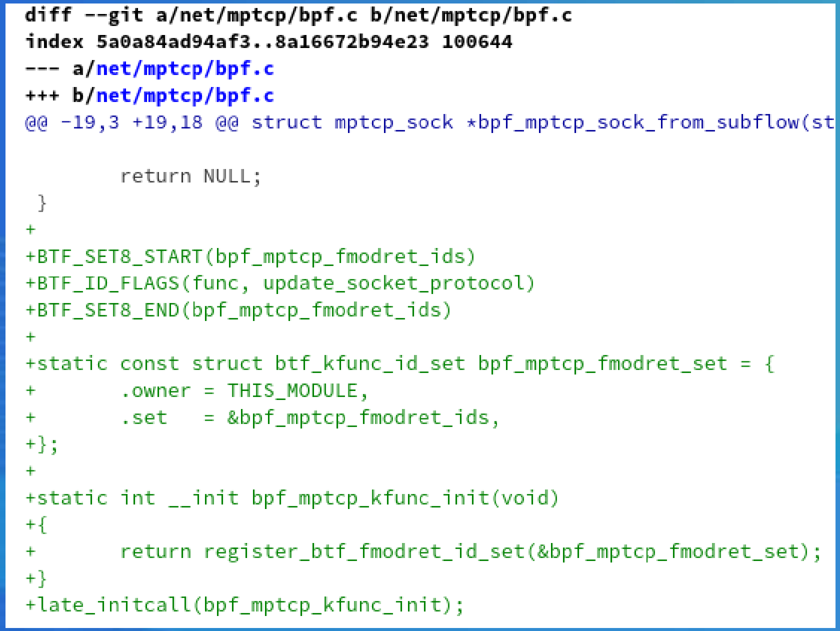
在不修改应用的情况下，使用BPF将原先使用TCP通信的应用转换成使用MPTCP通信



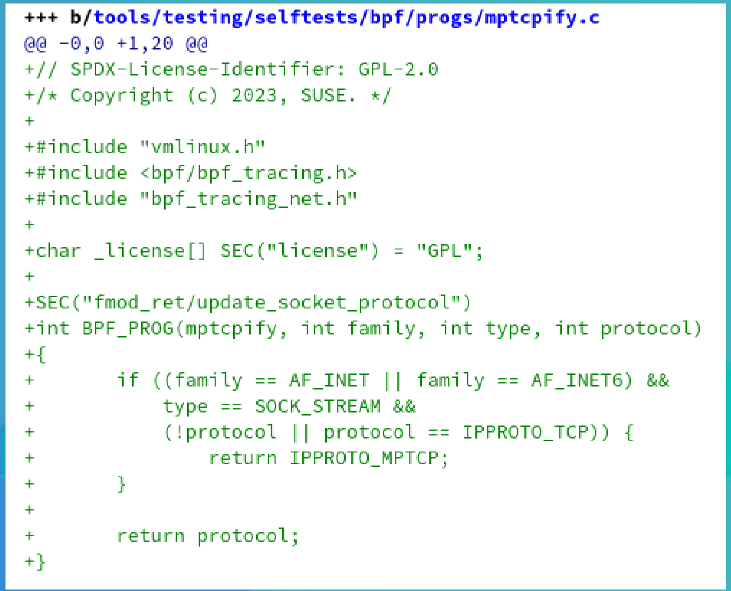
2023-08-16	Merge branch 'bpf: Force to MPTCP'	Martin KaFai Lau	4	-20/+221
2023-08-16	selftests/bpf: Add mptcpify test	Geliang Tang	2	-0/+161
2023-08-16	selftests/bpf: Fix error checks of mptcp open_and_load	Geliang Tang	1	-11/+1
2023-08-16	selftests/bpf: Add two mptcp netns helpers	Geliang Tang	1	-10/+21
2023-08-16	bpf: Add update_socket_protocol hook	Geliang Tang	2	-1/+40



```
+++ b/net/socket.c
@@ -1657,12 +1657,36 @@ struct file *__sys_socket_file(int family, int type, int pro
        return sock_alloc_file(sock, flags, NULL);
 }

+/*      A hook for bpf progs to attach to and update socket protocol.
+ *
+ *      A static noinline declaration here could cause the compiler to
+ *      optimize away the function. A global noinline declaration will
+ *      keep the definition, but may optimize away the callsite.
+ *      Therefore, __weak is needed to ensure that the call is still
+ *      emitted, by telling the compiler that we don't know what the
+ *      function might eventually be.
+ *
+ *      __diag_* below are needed to dismiss the missing prototype warning.
+ */
+
+__diag_push();
+__diag_ignore_all("-Wmissing-prototypes",
+                  "A fmod_ret entry point for BPF programs");
+
+__weak noinline int update_socket_protocol(int family, int type, int protocol)
+{
+       return protocol;
+}
+
+__diag_pop();
+
 int __sys_socket(int family, int type, int protocol)
 {
        struct socket *sock;
        int flags;

-       sock = __sys_socket_create(family, type, protocol);
+       sock = __sys_socket_create(family, type,
+                                  update_socket_protocol(family, type, protocol));
        if (IS_ERR(sock))
                return PTR_ERR(sock);
```



```
diff --git a/net/mptcp/bpf.c b/net/mptcp/bpf.c
index 5a0a84ad94af3..8a16672b94e23 100644
--- a/net/mptcp/bpf.c
+++ b/net/mptcp/bpf.c
@@ -19,3 +19,18 @@ struct mptcp_sock *bpf_mptcp_sock_from_subflow(st

        return NULL;
 }
+
+BTF_SET8_START(bpf_mptcp_fmodret_ids)
+BTF_ID_FLAGS(func, update_socket_protocol)
+BTF_SET8_END(bpf_mptcp_fmodret_ids)
+
+static const struct btf_kfunc_id_set bpf_mptcp_fmodret_set = {
+       .owner = THIS_MODULE,
+       .set   = &bpf_mptcp_fmodret_ids,
+};
+
+static int __init bpf_mptcp_kfunc_init(void)
+{
+       return register_btf_fmodret_id_set(&bpf_mptcp_fmodret_set);
+}
+late_initcall(bpf_mptcp_kfunc_init);
```

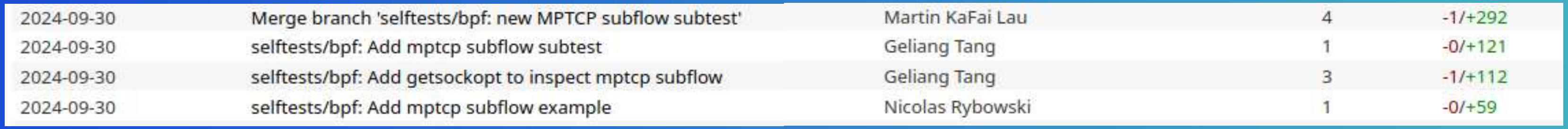


```
+++ b/tools/testing/selftests/bpf/progs/mptcpify.c
@@ -0,0 +1,20 @@
+// SPDX-License-Identifier: GPL-2.0
+/* Copyright (c) 2023, SUSE. */
+
+#include "vmlinux.h"
+#include <bpf/bpf_tracing.h>
+#include "bpf_tracing_net.h"
+
+char _license[] SEC("license") = "GPL";
+
+SEC("fmod_ret/update_socket_protocol")
+int BPF_PROG(mptcpify, int family, int type, int protocol)
+{
+       if ((family == AF_INET || family == AF_INET6) &&
+           type == SOCK_STREAM &&
+           (!protocol || protocol == IPPROTO_TCP)) {
+               return IPPROTO_MPTCP;
+       }
+
+       return protocol;
+}
```

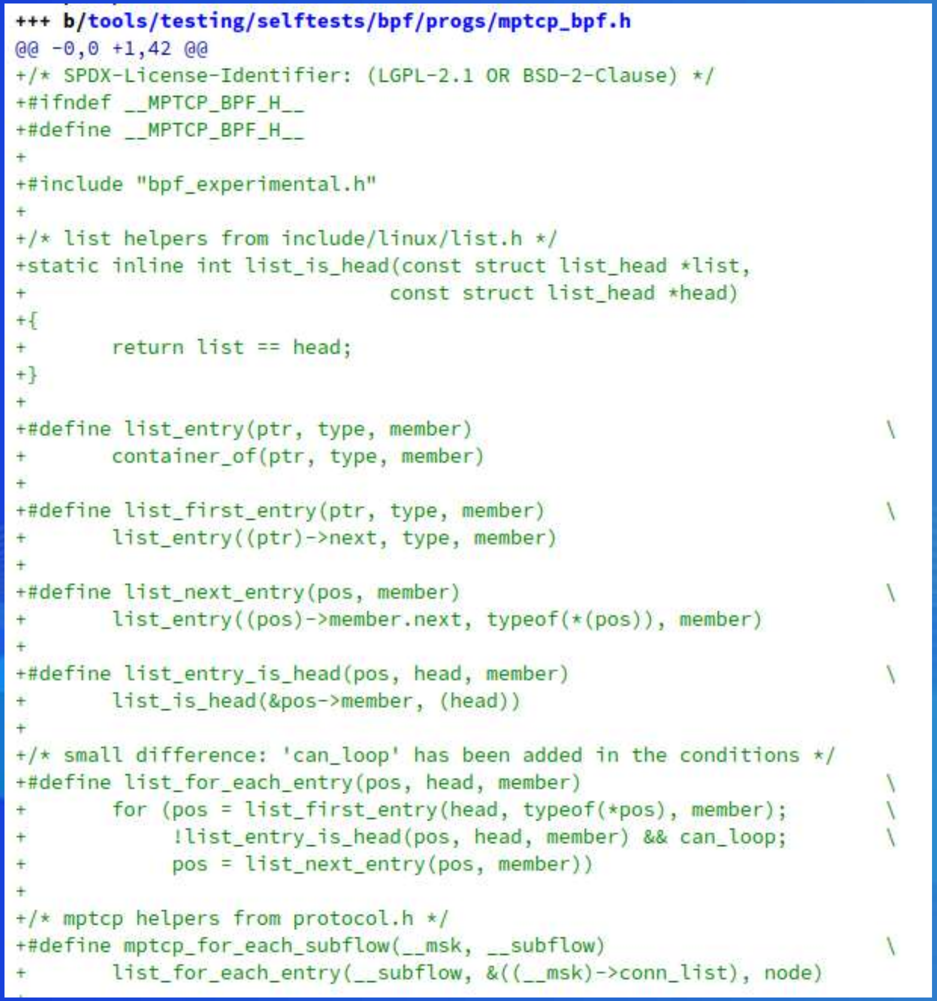
目录

CONTENTS

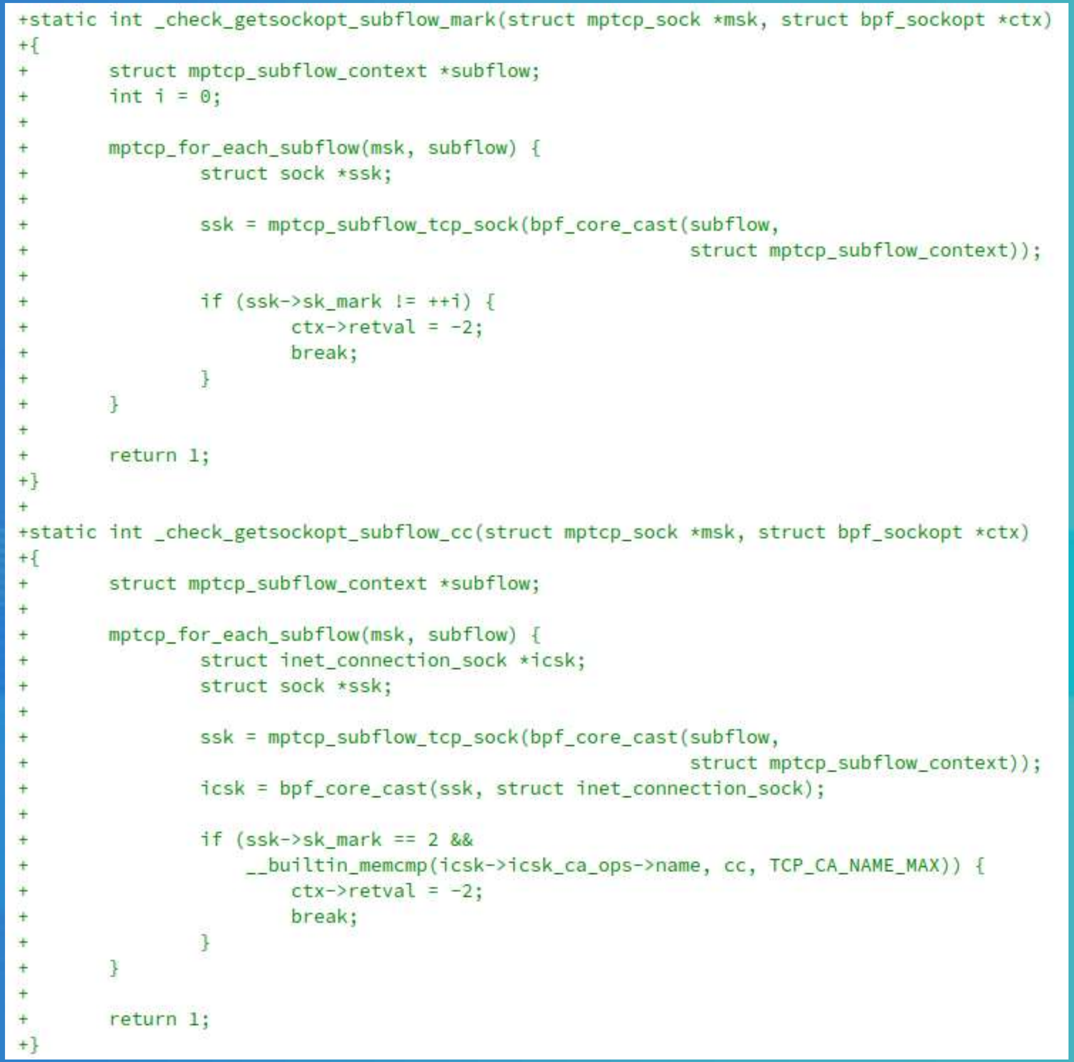
BPF遍历MPTCP subflow



2024-09-30	Merge branch 'selftests/bpf: new MPTCP subflow subtest'	Martin KaFai Lau	4	-1/+292
2024-09-30	selftests/bpf: Add mptcp subflow subtest	Geliang Tang	1	-0/+121
2024-09-30	selftests/bpf: Add getsockopt to inspect mptcp subflow	Geliang Tang	3	-1/+112
2024-09-30	selftests/bpf: Add mptcp subflow example	Nicolas Rybowski	1	-0/+59

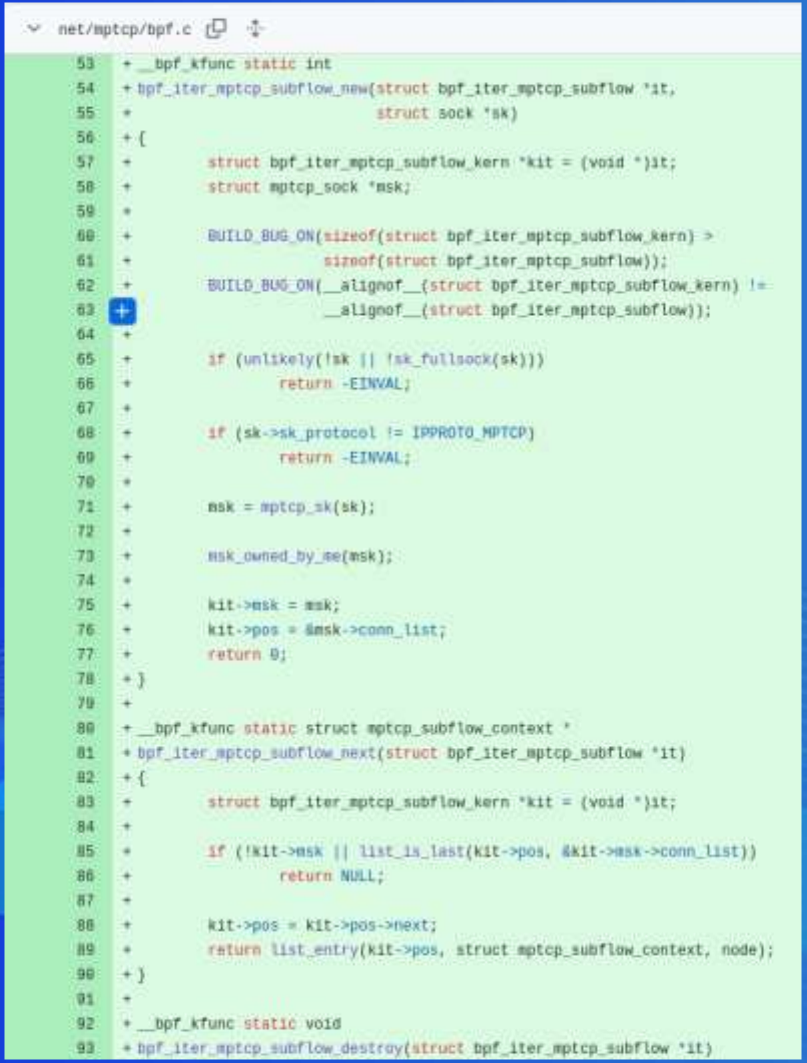


```
+++ b/tools/testing/selftests/bpf/progs/mptcp_bpf.h
@@ -0,0 +1,42 @@
+/* SPDX-License-Identifier: (LGPL-2.1 OR BSD-2-Clause) */
+#ifndef __MPTCP_BPF_H__
+#define __MPTCP_BPF_H__
+
+#include "bpf_experimental.h"
+
+/* list helpers from include/linux/list.h */
+static inline int list_is_head(const struct list_head *list,
+                               const struct list_head *head)
+{
+        return list == head;
+}
+
+#define list_entry(ptr, type, member)                                   \
+        container_of(ptr, type, member)
+
+#define list_first_entry(ptr, type, member)                             \
+        list_entry((ptr)->next, type, member)
+
+#define list_next_entry(pos, member)                                    \
+        list_entry((pos)->member.next, typeof(*(pos)), member)
+
+#define list_entry_is_head(pos, head, member)                           \
+        list_is_head(&pos->member, (head))
+
+/* small difference: 'can_loop' has been added in the conditions */
+#define list_for_each_entry(pos, head, member)                          \
+        for (pos = list_first_entry(head, typeof(*pos), member);        \
+             !list_entry_is_head(pos, head, member) && can_loop;        \
+             pos = list_next_entry(pos, member))
+
+/* mptcp helpers from protocol.h */
+#define mptcp_for_each_subflow(__msk, __subflow)                        \
+        list_for_each_entry(__subflow, &((__msk)->conn_list), node)
```



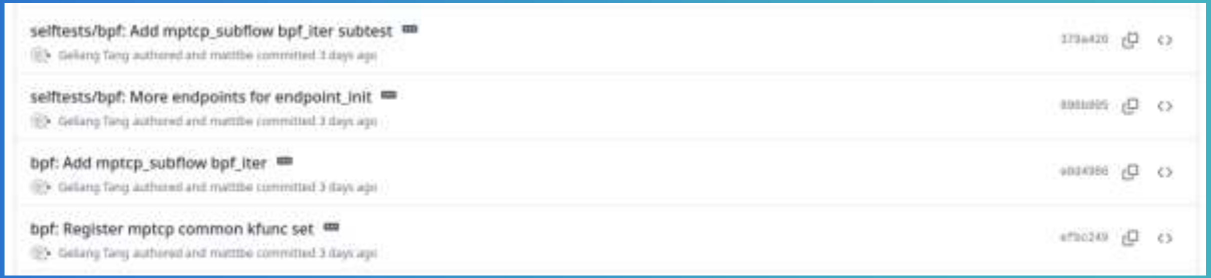
```
+static int _check_getsockopt_subflow_mark(struct mptcp_sock *msk, struct bpf_sockopt *ctx)
+{
+        struct mptcp_subflow_context *subflow;
+        int i = 0;
+
+        mptcp_for_each_subflow(msk, subflow) {
+                struct sock *ssk;
+
+                ssk = mptcp_subflow_tcp_sock(bpf_core_cast(subflow,
+                                                           struct mptcp_subflow_context));
+
+                if (ssk->sk_mark != ++i) {
+                        ctx->retval = -2;
+                        break;
+                }
+        }
+
+        return 1;
+}
+
+static int _check_getsockopt_subflow_cc(struct mptcp_sock *msk, struct bpf_sockopt *ctx)
+{
+        struct mptcp_subflow_context *subflow;
+
+        mptcp_for_each_subflow(msk, subflow) {
+                struct inet_connection_sock *icsk;
+                struct sock *ssk;
+
+                ssk = mptcp_subflow_tcp_sock(bpf_core_cast(subflow,
+                                                           struct mptcp_subflow_context));
+                icsk = bpf_core_cast(ssk, struct inet_connection_sock);
+
+                if (ssk->sk_mark == 2 &&
+                    __builtin_memcmp(icsk->icsk_ca_ops->name, cc, TCP_CA_NAME_MAX)) {
+                        ctx->retval = -2;
+                        break;
+                }
+        }
+
+        return 1;
+}
```

mptcp_subflow bpf_iter



net/mptcp/bpf.c

```
+ __bpf_kfunc static int
+ bpf_iter_mptcp_subflow_new(struct bpf_iter_mptcp_subflow *it,
+                            struct sock *sk)
+ {
+       struct bpf_iter_mptcp_subflow_kern *kit = (void *)it;
+       struct mptcp_sock *msk;
+
+       BUILD_BUG_ON(sizeof(struct bpf_iter_mptcp_subflow_kern) >
+                    sizeof(struct bpf_iter_mptcp_subflow));
+       BUILD_BUG_ON(__alignof__(struct bpf_iter_mptcp_subflow_kern) !=
+                    __alignof__(struct bpf_iter_mptcp_subflow));
+
+       if (unlikely(!sk || !sk_fullsock(sk)))
+               return -EINVAL;
+
+       if (sk->sk_protocol != IPPROTO_MPTCP)
+               return -EINVAL;
+
+       msk = mptcp_sk(sk);
+
+       msk_owned_by_me(msk);
+
+       kit->msk = msk;
+       kit->pos = &msk->conn_list;
+       return 0;
+ }
+
+ __bpf_kfunc static struct mptcp_subflow_context *
+ bpf_iter_mptcp_subflow_next(struct bpf_iter_mptcp_subflow *it)
+ {
+       struct bpf_iter_mptcp_subflow_kern *kit = (void *)it;
+
+       if (!kit->msk || list_is_last(kit->pos, &kit->msk->conn_list))
+               return NULL;
+
+       kit->pos = kit->pos->next;
+       return list_entry(kit->pos, struct mptcp_subflow_context, node);
+ }
+
+ __bpf_kfunc static void
+ bpf_iter_mptcp_subflow_destroy(struct bpf_iter_mptcp_subflow *it)
```

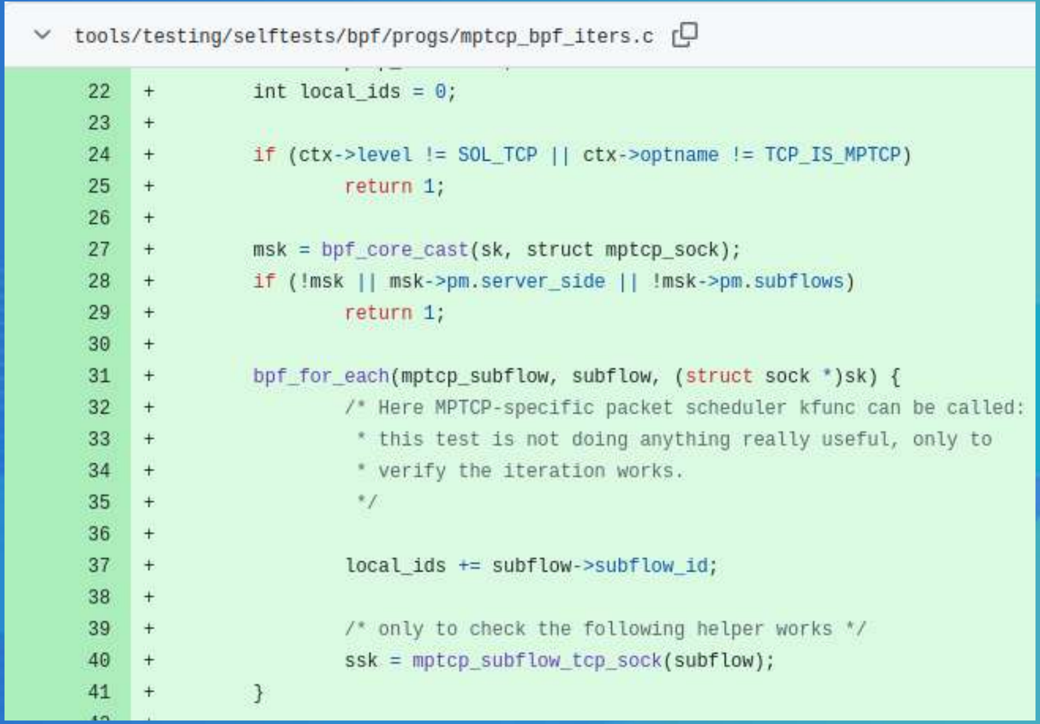


selftests/bpf: Add mptcp_subflow bpf_iter subtest
Geliang Tang authored and matttbe committed 3 days ago — 173a420

selftests/bpf: More endpoints for endpoint_init
Geliang Tang authored and matttbe committed 3 days ago — 8981d05

bpf: Add mptcp_subflow bpf_iter
Geliang Tang authored and matttbe committed 3 days ago — a0d4986

bpf: Register mptcp common kfunc set
Geliang Tang authored and matttbe committed 3 days ago — ef5c249



tools/testing/selftests/bpf/progs/mptcp_bpf_iters.c

```
+       int local_ids = 0;
+
+       if (ctx->level != SOL_TCP || ctx->optname != TCP_IS_MPTCP)
+               return 1;
+
+       msk = bpf_core_cast(sk, struct mptcp_sock);
+       if (!msk || msk->pm.server_side || !msk->pm.subflows)
+               return 1;
+
+       bpf_for_each(mptcp_subflow, subflow, (struct sock *)sk) {
+               /* Here MPTCP-specific packet scheduler kfunc can be called:
+                * this test is not doing anything really useful, only to
+                * verify the iteration works.
+                */
+
+               local_ids += subflow->subflow_id;
+
+               /* only to check the following helper works */
+               ssk = mptcp_subflow_tcp_sock(subflow);
+       }
```

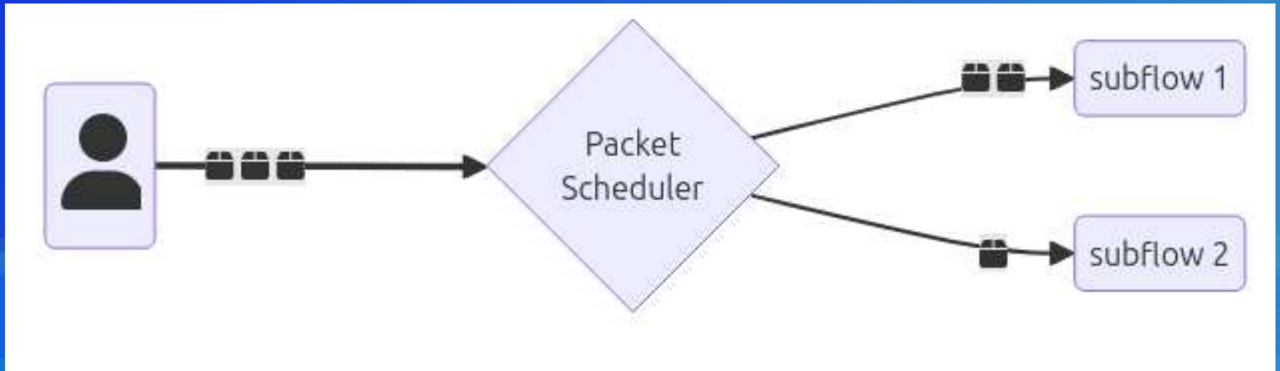
目录

CONTENTS

BPF packet scheduler

Packet scheduler负责选择使用哪一个有效的子链路来发送下一个数据包

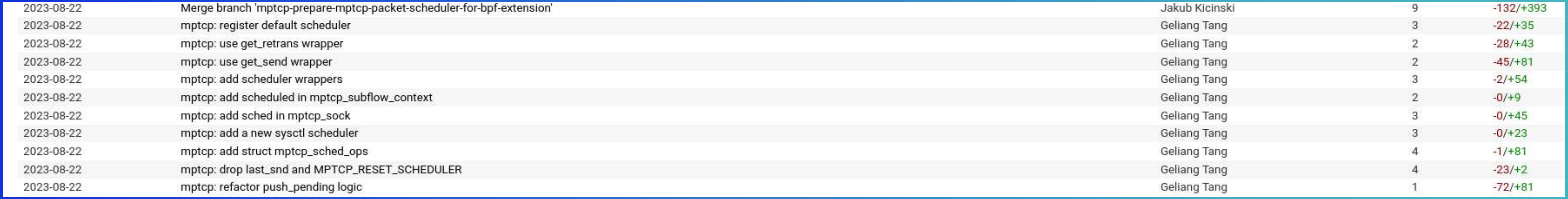
内核中只实现了一个burst packet scheduler，且允许用户通过EBPF自定义packet scheduler策略



Packets distribution between subflows

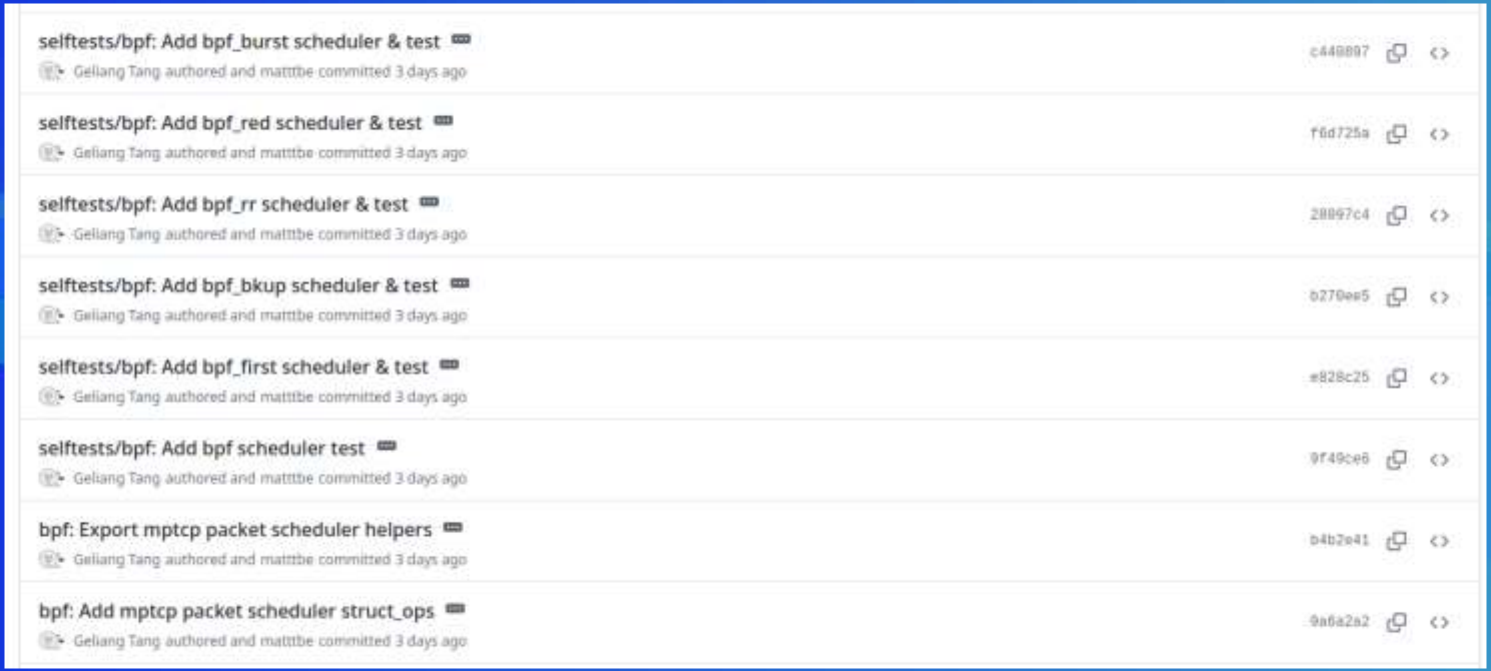
BPF packet scheduler

MPTCP部分已进主线



2023-08-22	Merge branch 'mptcp-prepare-mptcp-packet-scheduler-for-bpf-extension'	Jakub Kicinski	9	-132/+393
2023-08-22	mptcp: register default scheduler	Geliang Tang	3	-22/+35
2023-08-22	mptcp: use get_retrans wrapper	Geliang Tang	2	-28/+43
2023-08-22	mptcp: use get_send wrapper	Geliang Tang	2	-45/+81
2023-08-22	mptcp: add scheduler wrappers	Geliang Tang	3	-2/+54
2023-08-22	mptcp: add scheduled in mptcp_subflow_context	Geliang Tang	2	-0/+9
2023-08-22	mptcp: add sched in mptcp_sock	Geliang Tang	3	-0/+45
2023-08-22	mptcp: add a new sysctl scheduler	Geliang Tang	3	-0/+23
2023-08-22	mptcp: add struct mptcp_sched_ops	Geliang Tang	4	-1/+81
2023-08-22	mptcp: drop last_snd and MPTCP_RESET_SCHEDULER	Geliang Tang	4	-23/+2
2023-08-22	mptcp: refactor push_pending logic	Geliang Tang	1	-72/+81

BPF部分 WIP



selftests/bpf: Add bpf_burst scheduler & test
Geliang Tang authored and matttbe committed 3 days ago
c448897

selftests/bpf: Add bpf_red scheduler & test
Geliang Tang authored and matttbe committed 3 days ago
f6d725a

selftests/bpf: Add bpf_rr scheduler & test
Geliang Tang authored and matttbe committed 3 days ago
28897c4

selftests/bpf: Add bpf_bkup scheduler & test
Geliang Tang authored and matttbe committed 3 days ago
b270ee5

selftests/bpf: Add bpf_first scheduler & test
Geliang Tang authored and matttbe committed 3 days ago
e828c25

selftests/bpf: Add bpf scheduler test
Geliang Tang authored and matttbe committed 3 days ago
9f49ce6

bpf: Export mptcp packet scheduler helpers
Geliang Tang authored and matttbe committed 3 days ago
b4b2e41

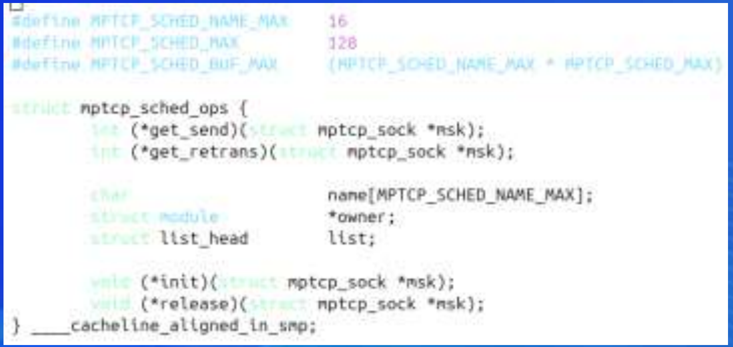
bpf: Add mptcp packet scheduler struct_ops
Geliang Tang authored and matttbe committed 3 days ago
9a6a2a2

BPF packet scheduler

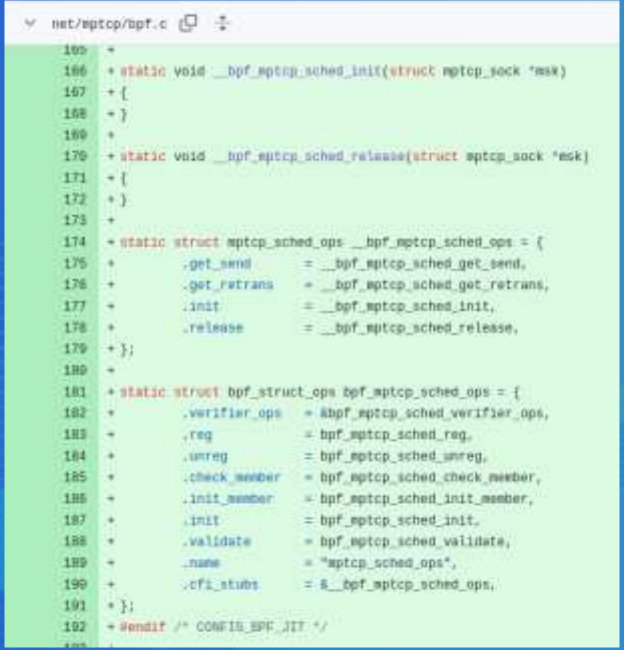
MPTCP中定义mptcp_sched_ops结构

BPF中定义对应的bpf_mptcp_sched_ops

BPF程序自定义mptcp_sched_ops

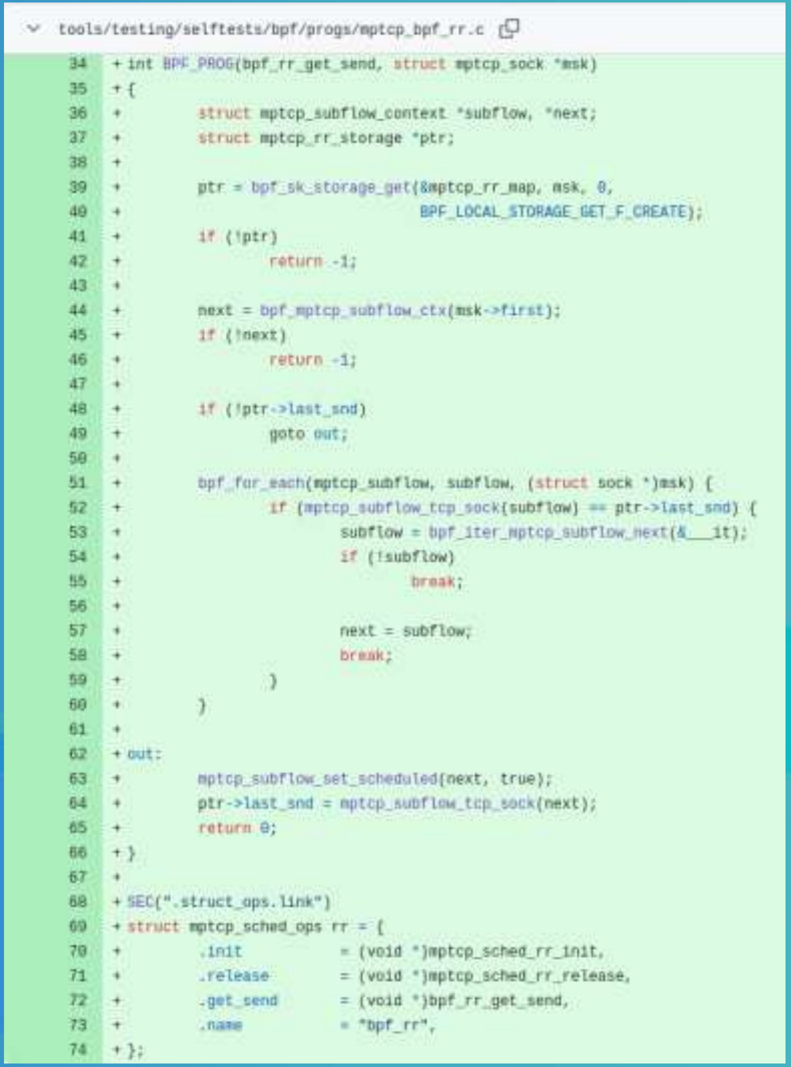


```c
#define MPTCP_SCHED_NAME_MAX    16
#define MPTCP_SCHED_MAX         128
#define MPTCP_SCHED_BUF_MAX     (MPTCP_SCHED_NAME_MAX * MPTCP_SCHED_MAX)

struct mptcp_sched_ops {
        int (*get_send)(struct mptcp_sock *msk);
        int (*get_retrans)(struct mptcp_sock *msk);

        char                    name[MPTCP_SCHED_NAME_MAX];
        struct module           *owner;
        struct list_head        list;

        void (*init)(struct mptcp_sock *msk);
        void (*release)(struct mptcp_sock *msk);
} ____cacheline_aligned_in_smp;
```



net/mptcp/bpf.c

```c
+
+static void __bpf_mptcp_sched_init(struct mptcp_sock *msk)
+{
+}
+
+static void __bpf_mptcp_sched_release(struct mptcp_sock *msk)
+{
+}
+
+static struct mptcp_sched_ops __bpf_mptcp_sched_ops = {
+        .get_send       = __bpf_mptcp_sched_get_send,
+        .get_retrans    = __bpf_mptcp_sched_get_retrans,
+        .init           = __bpf_mptcp_sched_init,
+        .release        = __bpf_mptcp_sched_release,
+};
+
+static struct bpf_struct_ops bpf_mptcp_sched_ops = {
+        .verifier_ops   = &bpf_mptcp_sched_verifier_ops,
+        .reg            = bpf_mptcp_sched_reg,
+        .unreg          = bpf_mptcp_sched_unreg,
+        .check_member   = bpf_mptcp_sched_check_member,
+        .init_member    = bpf_mptcp_sched_init_member,
+        .init           = bpf_mptcp_sched_init,
+        .validate       = bpf_mptcp_sched_validate,
+        .name           = "mptcp_sched_ops",
+        .cfi_stubs      = &__bpf_mptcp_sched_ops,
+};
+#endif /* CONFIG_BPF_JIT */
```



tools/testing/selftests/bpf/progs/mptcp_bpf_rr.c

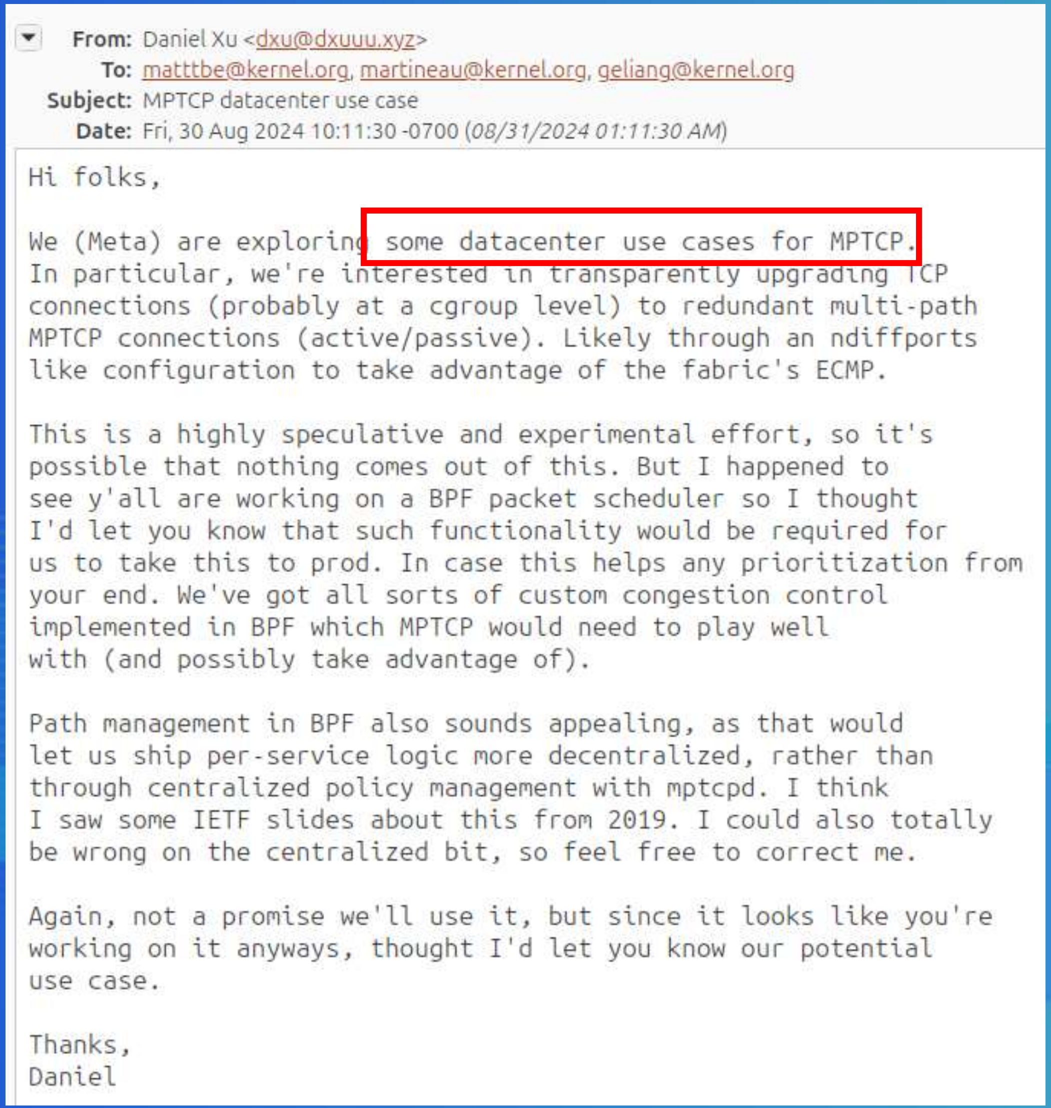
```c
+int BPF_PROG(bpf_rr_get_send, struct mptcp_sock *msk)
+{
+        struct mptcp_subflow_context *subflow, *next;
+        struct mptcp_rr_storage *ptr;
+
+        ptr = bpf_sk_storage_get(&mptcp_rr_map, msk, 0,
+                                 BPF_LOCAL_STORAGE_GET_F_CREATE);
+        if (!ptr)
+                return -1;
+
+        next = bpf_mptcp_subflow_ctx(msk->first);
+        if (!next)
+                return -1;
+
+        if (!ptr->last_snd)
+                goto out;
+
+        bpf_for_each(mptcp_subflow, subflow, (struct sock *)msk) {
+                if (mptcp_subflow_tcp_sock(subflow) == ptr->last_snd) {
+                        subflow = bpf_iter_mptcp_subflow_next(&___it);
+                        if (!subflow)
+                                break;
+
+                        next = subflow;
+                        break;
+                }
+        }
+
+out:
+        mptcp_subflow_set_scheduled(next, true);
+        ptr->last_snd = mptcp_subflow_tcp_sock(next);
+        return 0;
+}
+
+SEC(".struct_ops.link")
+struct mptcp_sched_ops rr = {
+        .init           = (void *)mptcp_sched_rr_init,
+        .release        = (void *)mptcp_sched_rr_release,
+        .get_send       = (void *)bpf_rr_get_send,
+        .name           = "bpf_rr",
+};
```

目录

CONTENTS

BPF path manager

背景——Meta的使用意向

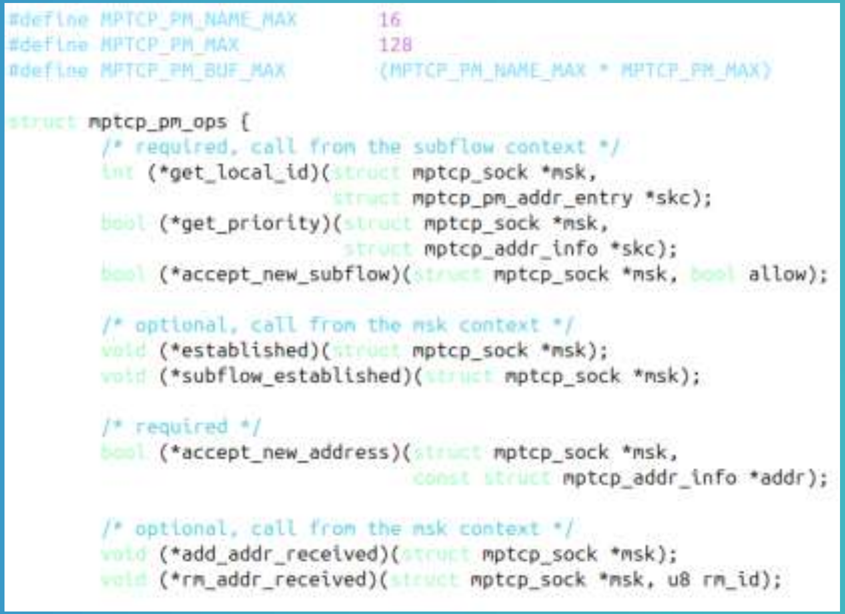


From: Daniel Xu <dxu@dxuuu.xyz>
To: matttbe@kernel.org, martineau@kernel.org, geliang@kernel.org
Subject: MPTCP datacenter use case
Date: Fri, 30 Aug 2024 10:11:30 -0700 *(08/31/2024 01:11:30 AM)*

```
Hi folks,

We (Meta) are exploring some datacenter use cases for MPTCP.
In particular, we're interested in transparently upgrading TCP
connections (probably at a cgroup level) to redundant multi-path
MPTCP connections (active/passive). Likely through an ndiffports
like configuration to take advantage of the fabric's ECMP.

This is a highly speculative and experimental effort, so it's
possible that nothing comes out of this. But I happened to
see y'all are working on a BPF packet scheduler so I thought
I'd let you know that such functionality would be required for
us to take this to prod. In case this helps any prioritization from
your end. We've got all sorts of custom congestion control
implemented in BPF which MPTCP would need to play well
with (and possibly take advantage of).

Path management in BPF also sounds appealing, as that would
let us ship per-service logic more decentralized, rather than
through centralized policy management with mptcpd. I think
I saw some IETF slides about this from 2019. I could also totally
be wrong on the centralized bit, so feel free to correct me.

Again, not a promise we'll use it, but since it looks like you're
working on it anyways, thought I'd let you know our potential
use case.

Thanks,
Daniel
```

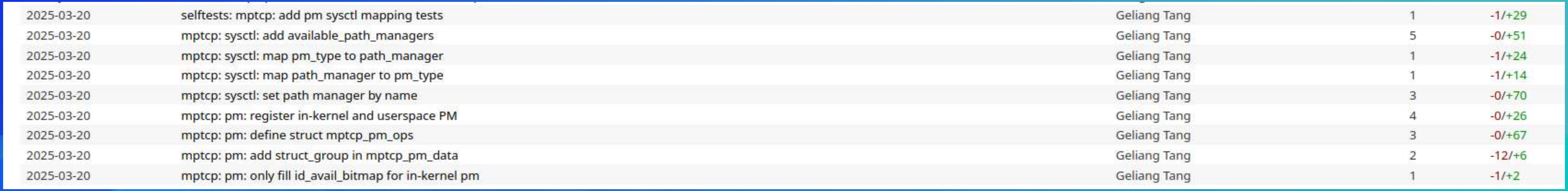
BPF path manager

统一path manager接口，抽象出mptcp_pm_ops，再导出到BPF空间

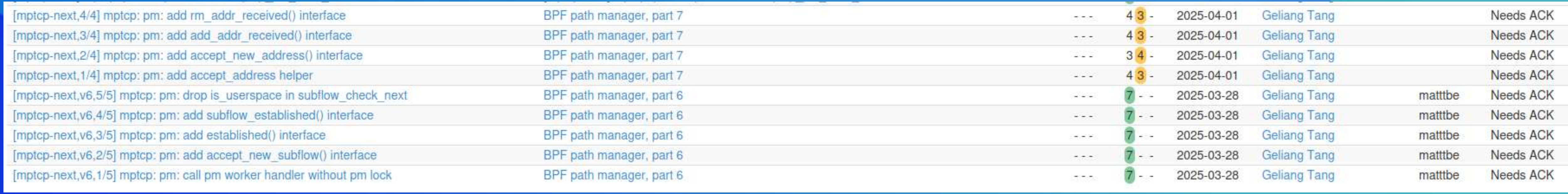


```
#define MPTCP_PM_NAME_MAX       16
#define MPTCP_PM_MAX            128
#define MPTCP_PM_BUF_MAX        (MPTCP_PM_NAME_MAX * MPTCP_PM_MAX)

struct mptcp_pm_ops {
        /* required, call from the subflow context */
        int (*get_local_id)(struct mptcp_sock *msk,
                            struct mptcp_pm_addr_entry *skc);
        bool (*get_priority)(struct mptcp_sock *msk,
                             struct mptcp_addr_info *skc);
        bool (*accept_new_subflow)(struct mptcp_sock *msk, bool allow);

        /* optional, call from the msk context */
        void (*established)(struct mptcp_sock *msk);
        void (*subflow_established)(struct mptcp_sock *msk);

        /* required */
        bool (*accept_new_address)(struct mptcp_sock *msk,
                                   const struct mptcp_addr_info *addr);

        /* optional, call from the msk context */
        void (*add_addr_received)(struct mptcp_sock *msk);
        void (*rm_addr_received)(struct mptcp_sock *msk, u8 rm_id);
```

开发中，部分已进主线



2025-03-20	selftests: mptcp: add pm sysctl mapping tests	Geliang Tang	1	-1/+29
2025-03-20	mptcp: sysctl: add available_path_managers	Geliang Tang	5	-0/+51
2025-03-20	mptcp: sysctl: map pm_type to path_manager	Geliang Tang	1	-1/+24
2025-03-20	mptcp: sysctl: map path_manager to pm_type	Geliang Tang	1	-1/+14
2025-03-20	mptcp: sysctl: set path manager by name	Geliang Tang	3	-0/+70
2025-03-20	mptcp: pm: register in-kernel and userspace PM	Geliang Tang	4	-0/+26
2025-03-20	mptcp: pm: define struct mptcp_pm_ops	Geliang Tang	3	-0/+67
2025-03-20	mptcp: pm: add struct_group in mptcp_pm_data	Geliang Tang	2	-12/+6
2025-03-20	mptcp: pm: only fill id_avail_bitmap for in-kernel pm	Geliang Tang	1	-1/+2



[mptcp-next,4/4] mptcp: pm: add rm_addr_received() interface	BPF path manager, part 7	- - -	4 3 -	2025-04-01	Geliang Tang		Needs ACK
[mptcp-next,3/4] mptcp: pm: add add_addr_received() interface	BPF path manager, part 7	- - -	4 3 -	2025-04-01	Geliang Tang		Needs ACK
[mptcp-next,2/4] mptcp: pm: add accept_new_address() interface	BPF path manager, part 7	- - -	3 4 -	2025-04-01	Geliang Tang		Needs ACK
[mptcp-next,1/4] mptcp: pm: add accept_address helper	BPF path manager, part 7	- - -	4 3 -	2025-04-01	Geliang Tang		Needs ACK
[mptcp-next,v6,5/5] mptcp: pm: drop is_userspace in subflow_check_next	BPF path manager, part 6	- - -	7 - -	2025-03-28	Geliang Tang	matttbe	Needs ACK
[mptcp-next,v6,4/5] mptcp: pm: add subflow_established() interface	BPF path manager, part 6	- - -	7 - -	2025-03-28	Geliang Tang	matttbe	Needs ACK
[mptcp-next,v6,3/5] mptcp: pm: add established() interface	BPF path manager, part 6	- - -	7 - -	2025-03-28	Geliang Tang	matttbe	Needs ACK
[mptcp-next,v6,2/5] mptcp: pm: add accept_new_subflow() interface	BPF path manager, part 6	- - -	7 - -	2025-03-28	Geliang Tang	matttbe	Needs ACK
[mptcp-next,v6,1/5] mptcp: pm: call pm worker handler without pm lock	BPF path manager, part 6	- - -	7 - -	2025-03-28	Geliang Tang	matttbe	Needs ACK

参考

- https://www.rfc-editor.org/rfc/rfc8684.html
- https://github.com/multipath-tcp
- https://www.mptcp.dev