An enhanced socket API for Multipath TCP

Benjamin Hesmans
Olivier Bonaventure
UCL, Belgium

http://inl.info.ucl.ac.be
http://www.multipath-tcp.org
Outline

• Multipath TCP

• The proposed socket API
What is Multipath TCP?

- A recently standardised TCP extension that allows packets belonging to one connection to be sent over different paths
  - Both WiFi and LTE on smartphones
  - Both IPv6 and IPv4 on dual-stack but single-homed hosts
  - Leveraging Equal Cost Multipath in datacenters
Multipath TCP

• Multipath TCP is an *evolution* of TCP

• Design objectives
  – Support unmodified applications
  – Work over today’s networks (IPv4 and IPv6)
  – Work in all networks where regular TCP works
Multipath TCP and the architecture

Low-latency for Siri

“Hey Siri, what song is this?”

Through the Shazam app, Siri can tell you what song is playing around you.

Sept. 2013
Siri uses MPTCP
WiFi/LTE Bonding

WiFi

Multipath TCP

Regular TCP

SOCKS

4G/LTE

July 2015
KT uses MPTCP

2005  2010  2015  2020
Sending data over different paths?

– A Multipath TCP connection is composed of one or more regular TCP subflows that are combined

• Each host maintains state that glues the TCP subflows that compose a Multipath TCP connection together

• Each TCP subflow is sent over a single path and appears like a regular TCP connection along this path
Multipath TCP
Connection establishment

SYN, MP_CAPABLE(KeyA)

SYN+ACK, MP_CAPABLE(KeyB)

ACK, MP_CAPABLE(KeyA, KeyB)

seq=123, DSeq=1, "abc"

TokenA=H(KeyA)
TokenB=H(KeyB)

First subflow established
Establishment of the second subflow

TokenA = H(KeyA)
TokenB = H(KeyB)

SYN MP_JOIN[TokenB, NonceA = 123]

SYN+ACK MP_JOIN[TokenA, NonceB = 456, HMAC(123 || 456, "keyB || keyA")]

ACK, MP_JOIN [HMAC(456 || 123, "keyA || keyB")]
Seq = 567, Dseq = 4, "def"

2nd subflow established
TCP subflows

• Which subflows can be associated to a Multipath TCP connection?

  – At least one of the elements of the four-tuple needs to differ between two subflows
    • Local IP address
    • Remote IP address
    • Local port
    • Remote port
Subflow agility

- Multipath TCP supports
  - addition of subflows
  - removal of subflows
How to control these subflows?

- Current reference implementation on Linux
  - Standard socket API to support existing applications

- Subflows are managed by the path manager kernel module
  - Full-mesh
  - NDiffports
How to control these subflows?

```c
/* socket creation */
s = socket(AF_MULTI, SOCK_STREAM, IPPROTO_TCP);

/* creation of first subflow */
sa_endpoints_t endpoints;
/* any source interface */
endpoints.sae_srcif = 0;
/* any address of the client */
endpoints.sae_srcaddr = NULL;
endpoints.sae_srcaddrlen = 0;
/* server address */
endpoints.sae_dstaddr = (struct sockaddr *)
endpoints.sae_dstaddrlen = daddr->ai_addrlen;

int rc = connectx(s, &endpoints, SAE_ASSOCID_ANY,
   0, NULL, 0, NULL, NULL);
```
Outline

- Multipath TCP
- The proposed socket API
Why using socket options?

• `getsockopt` and `setsockopt` are well-known and extensible
• Relatively easy to implement a new socket option
• Can pass information from app to stack as memory buffer
• Can retrieve information from stack to app as memory buffer
The MPTCP socket options

- **MPTCP_GET_SUB_IDS**
  - Retrieve the ids of the different subflows
- **MPTCP_GET_SUB_TUPLE**
  - Retrieve the endpoints of a specific subflow
- **MPTCP_OPEN_SUB_TUPLE**
  - Create a new subflow with specific endpoints
- **MPTCP_CLOSE_SUB_ID**
  - Closes one of the established subflows
- **MPTCP_SUB_GETSOCKOPT** and **MPTCP_SUB_SETSOCKOPT**
  - Apply a TCP socket option on a specific subflow
Currently established subflows

```c
int i;
unsigned int optlen;
struct mptcp_sub_ids *ids;

optlen = 42; // must be large enough
ids = (struct mptcp_sub_ids *) malloc(optlen);

err=getsockopt(sockfd, IPPROTO_TCP,
    MPTCP_GET_SUB_IDS, ids, &optlen);

for(i = 0; i < ids->sub_count; i++){
    printf("Subflow id : %i\n",
        ids->sub_status[i].id);
}
```
What are the endpoints of a subflow?

unsigned int optlen;
struct mptcp_sub_tuple *sub_tuple;

optlen = 100; // must be large enough
sub_tuple = (struct mptcp_sub_tuple *)malloc(optlen);

sub_tuple->id = sub_id;
getsockopt(sockfd, IPPROTO_TCP, MPTCP_GET_SUB_TUPLE,
    sub_tuple,&optlen);

sin = (struct sockaddr_in*) &sub_tuple->addrs[0];
printf("\tip src : %s src port : %hu\n", inet_ntoa(sin->sin_addr),
    ntohs(sin->sin_port));

sin = (struct sockaddr_in*) &sub_tuple->addrs[1];
printf("\tip dst : %s dst port : %hu\n", inet_ntoa(sin->sin_addr),
    ntohs(sin->sin_port));
Creating a subflow

```c
unsigned int optlen;
struct mptcp_sub_tuple *sub_tuple;
struct sockaddr_in *addr;

optlen = sizeof(struct mptcp_sub_tuple);
    2 * sizeof(struct sockaddr_in);
sub_tuple = malloc(optlen);
sub_tuple->id = 0; sub_tuple->prio = 0;

addr = (struct sockaddr_in*) &sub_tuple->addrs[0];
addr->sin_family = AF_INET;
addr->sin_port = htons(12345);
inet_pton(AF_INET, "10.0.0.1", &addr->sin_addr);
addr = (struct sockaddr_in*) &sub_tuple->addrs[1];
addr->sin_family = AF_INET;
addr->sin_port = htons(1234);
inet_pton(AF_INET, "10.1.0.1", &addr->sin_addr);
error = getsockopt(sockfd, IPPROTO_TCP, MPTCP_OPEN_SUB_TUPLE, sub_tuple, &optlen);
```
Utilization of the socket API

MPTCP enabled applications will be able to accurately control their usage of the cellular and WiFi interfaces
Conclusion and next steps

• Multipath TCP is getting deployed
  – Special applications (Siri) and on middleboxes

• Socket API will enable application developers to take full control of the underlying MPTCP
  – Create/delete/query subflows, apply options
  – Next steps
    • non-blocking I/O and events with `select`, `recvmsg` and `sendmsg`
    • Address management and advertisement
    • More options to control stack (e.g. scheduler)

• Cooperation with application developers