A socket API to control Multipath TCP

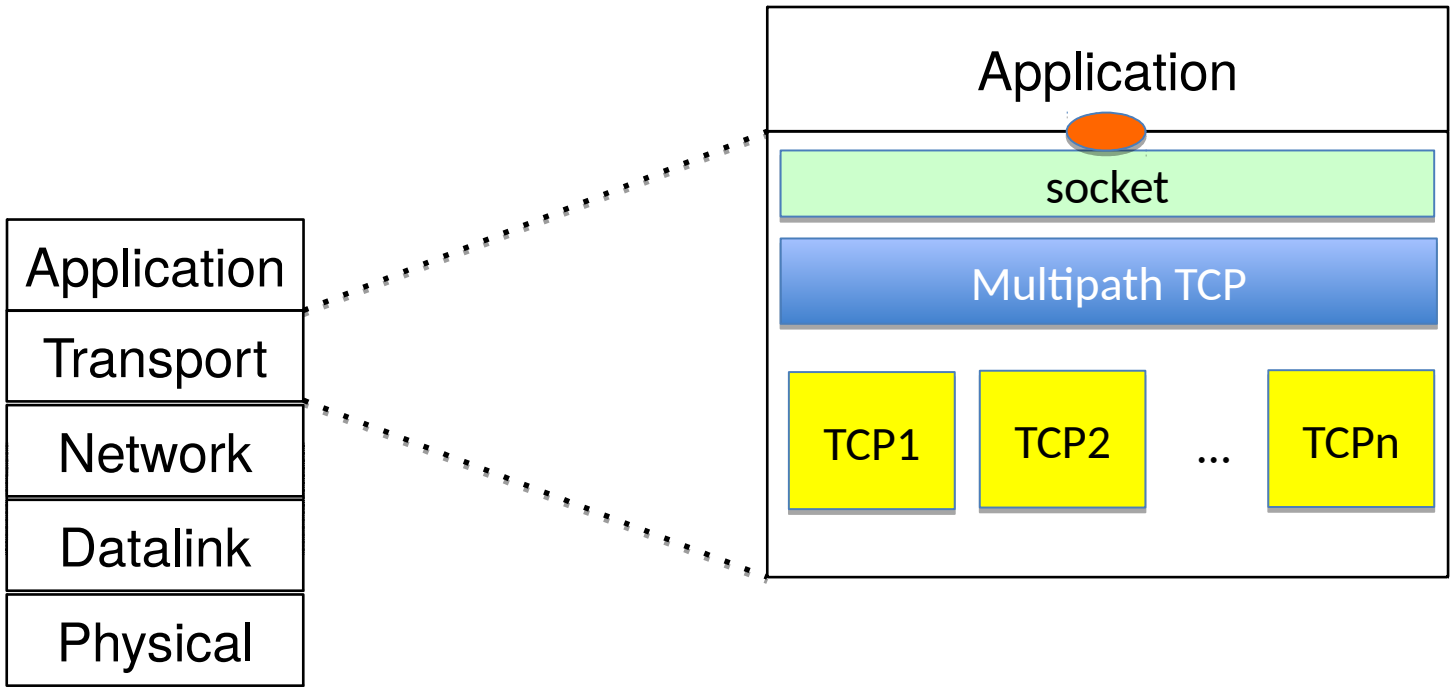
`draft-hesmans-mptcp-socket-00`

Benjamin Hesmans

Olivier Bonaventure

UCL, Belgium

Multipath TCP and the architecture



- Backward compatibility : socket unchanged

A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, “Architectural guidelines for multipath TCP development", RFC6182 2011.

Multipath TCP deployments

- Initial assumption
  - Backward compatible replacement for TCP
    - Use by some researchers and `multipath-tcp.org`
- Existing deployments
  - Siri (Apple)
  - SOCKS (KT, OVH, …)
  - Hybrid Access Networks (Tessares, …)
- Current Multipath TCP users need to control the utilisation of the subflows

How to control the subflows ?

- Current reference implementation on Linux
  - Unmodified standard socket API to support existing applications
- Subflows are managed by the path manager kernel module
  - Full-mesh
    - use all available interfaces as soon as they are available
  - NDiffports
    - Use N flows per interface (assumes single-homed hosts)

What was wrong with this approach ?

In theory, kernel path manager can be tuned to the user's needs but

- User needs vary a lot
  - Prefer A over B if C is down
  - Use B only for a given app
  - Start over C and establish A if flow is long enough
- Writing a new path manager is difficult
- New path manager kernel must be shipped to support specific needs

How to control these subflows ?


```
/* socket creation */
s = socket(AF_MULTIPATH, 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 *)
                                daddr->ai_
endpoints.sae_dstaddrlen = dad_->ai_addrlen;

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

Special AF

Other system calls

Towards a standardised MPTCP API using socket options

- Why 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
- Initially suggested in RFC6897, but not supported by any implementation

Implemented 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

```
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(op

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_n  a(sin->sin_addr),
                                          ntohs  n->sin_port));
```

Local endpoint

Remote endpoint

Creating a subflow

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

optlen = sizeof(struct mptcp_sub_tuple,
             2 * sizeof(struct sockaddr_i
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);
```

Local endpoint

Remote endpoint

Status

- Implemented in Linux
  - Create/delete/query subflows, apply socket options
  - non-blocking I/O and events, e.g. with `select`, `recvmsg` and `sendmsg`
- Seeking cooperation with application developers
  - Better understand their requirements
  - Expose the right abstractions

- Next steps in IETF
  - Add socket API to WG charter
  - WG interest ?