Télécom ParisTech
Promotion 2017
Sylvain DASSIER

RAPPORT DE STAGE

Étude de l'apport du protocol MPTCP dans l'optimisation du trafic

Département :	***Département d'Informatique***
Option :	***INFRES***
Encadrants :	***M. Luigi IANNONE, M. Antoine FRESSANCOURT***
Dates :	***18/07/2016 - 17/01/2017***
Adresse :	***Télécom ParisTech, 23 Avenue d'Italie, 75013 Paris***

Declaration d'intégrité relative au plagiat

Je soussigné DASSIER Sylvain *certifie sur l'honneur :*

1. Que les résultats décrits dans ce rapport sont l'aboutissement de mon travail.
2. Que je suis l'auteur de ce rapport.
3. Que je n'ai pas utilisé des sources ou résultats tiers sans clairement les citer et les référencer selon les règles bibliographiques préconisées.

Je déclare que ce travail ne peut être suspecté de plagiat.

17 janvier 2017 Signature : Sylvain Dassier

Abstract

English

Résumé

Français

Table des matières

1 Introduction

1.1 Context

Today, connected vehicles make use of 2G, 3G or 4G networks in order to connect to the internet while in motion. Whether it be for GPS, simple browsing or music, every consumer has his/her own needs.
Apart from the usual connection glitches, such connectivity is rather expensive with limited bandwidth. Even though workarounds have been implemented, most of them are either inefficient or are not completely transparent. These limitations stand in the way of developement of connected vehicles.
MultiPath TCP (MPTCP) is an effort towards enabling the simultaneous use of several IP-addresses/interfaces by a modification of TCP. It presents a regular TCP interface to applications, while in fact spreading data across several sub-flows. Benefits of this include better resource utilisation, better throughput and smoother reaction to failures. The project CarFi, aims to exploit these advantages of MPTCP. A potential add-on would be the usage of the WiFi network when available. Most urban areas are covered via Mobile Network Operator or ISP WiFi hotspots. One may envisage a scenario where the default connection is established over Wifi and when it is no longer available, the communication carries on over 3G.

1.2 Document Outline

This document is divided into two main parts comprising different sections. The first part involves section 2 where we describe how to set up a ***debugging environment for MPTCP***. This will help us to follow the different system calls during the establishment of a flow or a sub-flow. The next sections form the other part, dealing with the new socket API that enables us to control the MPTCP stack from user space. Section 3 gives a description of the socket API. Section 4 elaborates a use case of this API, in our case a **Netcat** with **MPTCP**. Section 5 summarises our results 5.1, elucidates certain statistics 5.2 and emphasises on the utility 5.3 of our work.

2 Setting up a debugging environment for MPTCP :

In order to understand the different stages of running of the MPTCP linux kernel, we have put in place a debugging environment. This has been done with [1, LibOS] (an MPTCP version of the library operating system of the linux kernel) and [2, DCE] (Direct Code Execution). Everything was done on a XUbuntu 14.04 64bit virtual machine with DCE 1.8. The following illustrates how :

1. **Install the dependencies :**

   sudo apt-get install vim git mercurial gcc g++ python python-dev qt4-dev-tools libqt4-dev bzr cmake libc6-dev libc6-dev-i386 g++-multilib gdb valgrind gsl-bin libgsl0-dev libgsl0ldbl flex bison libfl-dev tcpdump sqlite sqlite3 libsqlite3-dev libxml2 libxml2-dev libgtk2.0-0 libgtk2.0-dev vtun lxc uncrustify doxygen graphviz imagemagick texlive texlive-extra-utils texlive-latex-extra texlive-font-utils dvipng python-sphinx dia python-pygraphviz python-kiwi python-pygoocanvas libgoocanvas-dev ipython libboost-signals-dev libboost-filesystem-dev openmpi-bin openmpi-common openmpi-doc libopenmpi-dev libncurses5-dev libncursesw5-dev unrar unrar-free p7zip-full autoconf libpcap-dev cvs libssl-dev wireshark

2. **Build DCE using bake :**

   (a) hg clone http://code.nsnam.org/bake bake

   (b) export BAKE_HOME=‘pwd‘/bake

   (c) export PATH=$PATH :$BAKE_HOME

   (d) export PYTHONPATH=$PYTHONPATH :$BAKE_HOME

   (e) mkdir dce

   (f) cd dce

   (g) bake.py configure -e dce-ns3-1.8

   (h) bake.py download

   (i) bake.py build

3. **Build the *mptcp_trunk_libos* branch of *net-next-nuse***

   (a) git clone -b mptcp_trunk_libos https://github.com/libos-nuse/net-next-nuse.git

   (b) cd net-next-nuse

   (c) make menuconfig ARCH=lib

   (d) make library ARCH=lib

(e) Since DCE by default, calls the library *liblinux.so* (not exactly the correct one), and that the correct library is *libsim-linux.so* found at *$HOME/net-next-nuse/arch/lib/tools* we rename the existing *liblinux.so* to *liblinux0.so* and create a symbolic link for the correct library as follows :
ln -s $HOME/net-next-nuse/arch/lib/tools/libsim-linux.so $HOME/net-next-nuse/liblinux.so. This will "mislead" DCE into loading the correct library.

4. **Build *iproute2* version *2.6.38***

(a) Download the compressed source code from
https://kernel.googlesource.com/pub/scm/linux/kernel/git/shemminger/iproute2/+archive/fcae78992cab7bd267785b392b438306c621e583.tar.gz , extract it and rename the folder to *iproute2-2.6.38*.

(b) cd iproute2-2.6.38

(c) patch -p1 -i ../ns-3-dce/utils/iproute-2.6.38-fix-01.patch

(d) $(KERNEL_INCLUDE) should point to the liblinux.so directory (for me it is $HOME/net-next-nuse)
Hence I modified the following part in the Makefile :
Config :
sh configure /home/lawrence/net-next-nuse
 # sh configure $(KERNEL_MODULE)

(e) LDFLAGS=-pie make CCOPTS='-fpic -D_GNU_SOURCE -O0 -U_FORTIFY_SOURCE'

5. **Set the *DCE_PATH***
export DCE_PATH=$HOME/net-next-nuse :$HOME/iproute2-2.6.38/ip

6. **Build *ns-3-dce***

(a) hg clone http://code.nsnam.org/ns-3-dce -r dce-1.8

(b) cd ns-3-dce

(c) ./waf configure –with-ns3=$HOME/dce/build –enable-kernel-stack=$HOME/net-next-nuse/arch –prefix=$HOME/dce/build

(d) ./waf build

7. **Run *dce-iperf-mptcp* with or without *GDB***

(a) cd ns-3-dce

(b) Without *GDB* : *./waf –run dce-iperf-mptcp*

(c) With *GDB* : *./waf –run dce-iperf-mptcp –command-template="gdb –args %s"*
Once we enter the *GDB* prompt we must put a breakpoint at one of the functions in the *mptcp* folder to stop there. Kindly refer to the files found at *$HOME/net-next-nuse/net/mptcp* to choose the function to define as a breakpoint.
An example :
Suppose we would like to stop the execution at the function *mptcp_set_default_path_manager()* found at *$HOME/net-next-nuse/net/mptcp/mptcp_pm.c*, then we give the following command at the *GDB* prompt :
b mptcp_set_default_path_manager
GDB will ask the following :
Function "mptcp_set_default_path_manager" not defined.
Make breakpoint available on future shared library load ? (y or [n])
Type in ***y*** and press enter. We may run the program by typing ***r*** and then pressing enter. *GDB* will pause at the necessary breakpoint.

3 An Enhanced socket API for Multipath TCP :

In our project CarFi, we would like to control the MPTCP kernel stack from the application layer i.e. manage(open/close) sub flows according to the kind of application that uses it. For example, for a streaming application it is preferable to communicate over the Wifi channel. In the current Linux Kernel implementation of MPTCP, the path managers may not be fit for all kinds of applications. For optimum usage, advanced applications may want to know the number of sub flows available or the state of the active sub flows. When the application possesses such information it may want to create a new sub flow, terminate an existing one, change a sub flow's priority etc.

3.1 Implementation

The enhanced socket API has been implemented over the existing ***getsockopt()*** and ***setsockopt()*** system calls. The following figure illustrates the MPTCP socket structure [3] :

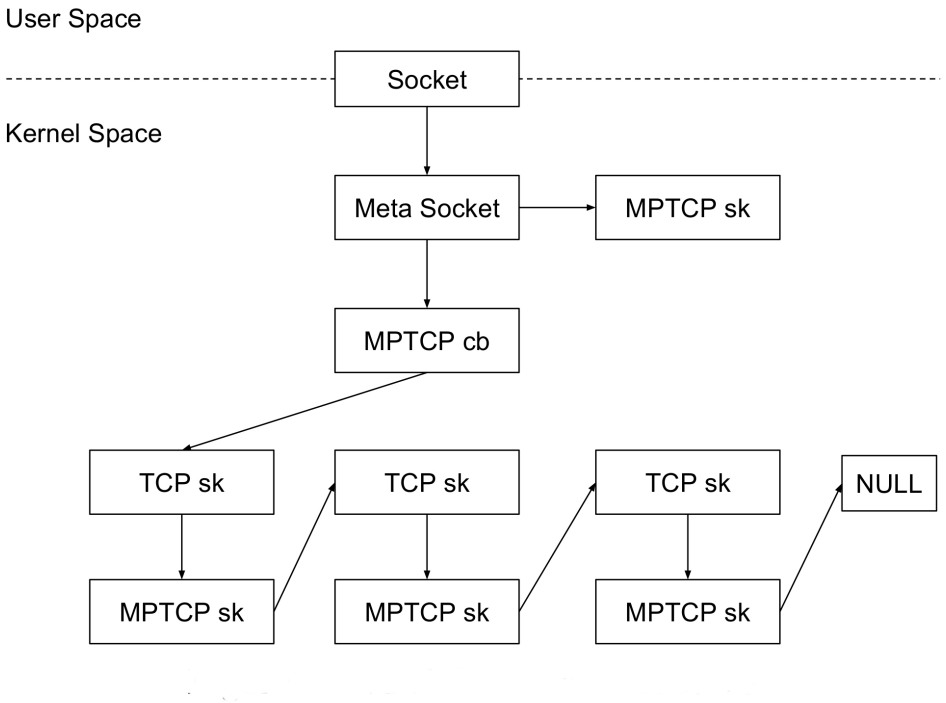

FIGURE 1 – MPTCP socket structure

From the application's point of view, no other socket other than the **Meta Socket** is visible. Underneath the **Meta Socket** lie several subsockets, each representing a sub flow. The structure **mptcp_cb** points towards the head of the subflow list. The structure

mptcp_sk hence points indirectly towards the next subflow. Till now there is no way for the application to know what hides beyond the **Meta Socket**. This is where the socket options come into play. The enhanced socket API lists the following socket options for the user [3] :

Name	Input	Output	Description
MPTCP_GET_SUB_IDS	-	subflow list	Get the current list of subflows viewed by the kernel
MPTCP_GET_SUB_TUPLE	id	sub tuple	Get the ip and ports used by the subflow identified by id
MPTCP_OPEN_SUB_TUPLE	tuple	-	Request a new subflow with pair of ip and ports
MPTCP_CLOSE_SUB_ID	id	-	Close the subflow identified by id
MPTCP_SUB_GETSOCKOPT	id, sock opt	sock ret	Redirects the getsockopt given in input to the subflow identified by id and return the value returned by the operation
MPTCP_SUB_SETSOCKOPT	id, sock opt	-	Redirects the setsockopt given in input to the subflow identified by id

TABLE 1 – Implemented MPTCP socket options

The following example shows how we may use the socket option **MPTCP_OPEN_SUB_TUPLE** and **getsockopt()**to open a sub flow :
First we introduce the **mptcp_sub_tuple** structure which represents the subflow :

```
struct mptcp_sub_tuple {
    _u8 id;       // this is an output signifying the ‘‘id’’ of the
        subflow
    _u8 prio;     // this field determines if the sub flow is backup or
        not
    _u8 addrs[0]; // pair array of size two depicting (source,
        destination)
}
```

Now we use this structure to open a sub flow as follows :

```
int i;
unsigned int optlen;
struct mptcp_sub_ids *ids;
optlen = 42;

int error;

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++;

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);
```

4 Netcat with MPTCP (netcat-mptcp) :

5 Results, Statistics and Utility

5.1 Results

5.2 Statistics

5.3 Utility

6 Conclusion

Conclusion

7 Further developments

In the above experiments

8 Acknowledgements

Acknowledgement

9 Bibliography

Références

[1] Hajime Tazaki. libos-nuse : net-next-nuse.
https://github.com/libos-nuse/net-next-nuse.

[2] NS-3 : Direct Code Execution.
https://www.nsnam.org/overview/projects/direct-code-execution.

[3] Olivier Bonaventure Benjamin Hesmans. An enhanced socket api for multipath tcp. 2016.

10 Appendix

Here we have the different

11 Glossary

RA : *Département d'Informatique*
IETF : *Internet Engineering Task Force*
L2 : *Layer 2/Link Layer of the OSI model*
L3 : *Layer 2/IP Layer of the OSI model*
DHCP : *Dynamic Host Configuration Protocol*
DNS : *Domain name system*
MLD : *Multicast Listener Discovery*
IP : *Internet Protocol*
RFC : *Request for Comment*
ARP : *Address Resolution Protocol*
VLAN : *Virtual local area network*
AP : *Access Point*
RS : *Router Solicitation*
NS : *Neighbour Solicitation*
NA : *Neighbour Advertisement*
mDNS : *multicast Domain Name System*
LLMNR : *Link-Local Multicast Name Resolution*
SLAAC : *Stateless Address Autoconfiguration*