# **mTCP**

### Get the source

1. git clone ssh://mchowdh1@git-amr-1.devtools.intel.com:29418/mtcp-source && cd "mtcp-source" && git config user.name "Muktadir Chowdhury" && git config user.email "[muktadir.chowdhury@intel.com](mailto:muktadir.chowdhury@intel.com)" && scp -P 29418 [mchowdh1@git-amr-1.devtools.intel.com:hooks/commit-msg .git/hooks/](mailto:mchowdh1@git-amr-1.devtools.intel.com:hooks/commit-msg%20.git/hooks/)

Use you username, name, and windows password.

1. After getting the source need to checkout the libc-wrapper branch: git checkout –b <name of branch of your choice> libc-wrapper

Please note that mTCP does not support setting of ethernet flow rules for children process by the parent process in ixgbe driver. I ran my experiement on ncc7 and interface that is connected to the switch has ixgbe driver. So had to change the code such that each children process sets its own flow rule. The corresponding code is in : /pnpdata/dpdk-mtcp/mtcp-source/mtcp/src/dpdk\_module.c. In i40e nic, the original code will work without any modification.

### Build mTCP

1. Go to mtcp directory: $ cd mtcp
2. Run the following script. If you already have dpdk installed in your system then you can pass the dpdk installation path ($RTE\_SDK) to the script. If you don’t pass the argument the script then will clone dpdk. They have used dpdk-18.02/ as their DPDK driver. FYI, you can pass a different dpdk source directory as command line argument.

$ bash setup\_mtcp\_dpdk\_env.sh [<path to $RTE\_SDK]

The script will go to dpdk directory and run the dpdk set-up script from …./dpdk/usertools/dpdk-setup.sh

After that you have to do the following:

Press [14] to compile x86\_64-native-linuxapp-gcc version (if libnuma is not there: yum install numactl-devel)

Press [17] to install the driver (depmod, )

Press [21] to setup 2048 2MB hugepages

Press [23] to register the Ethernet ports

(**Help**: It is not possible to register \*\*active\*\* interfaces. Need to bring down the interface before registering, e.g. ifdown <interface-name>)

Press [22] to make sure your ports are registerd properly under dpdk driver.

Press [34] to quit the tool

After that the interfaces which are now bound to dpdk-compatible driver will be shown and they are renamed to **dpdkX** mTCP.

1. Next bring the dpdk-registered interfaces up, and then set RTE\_SDK

and RTE\_TARGET environment variables.

$ sudo ifconfig dpdk0 x.x.x.x netmask 255.255.255.0 up

Note: Here you can assign any address to the interface. Better to use private address of the form 192.168.1.x

$ export RTE\_SDK=`echo $PWD`/dpdk

export RTE\_SDK=/pnpdata/dpdk-mtcp/mtcp/dpdk

$ export RTE\_TARGET=x86\_64-native-linuxapp-gcc

1. Setup mtcp library:

$./configure --with-dpdk-lib=$RTE\_SDK/$RTE\_TARGET

$ make

- By default, mTCP assumes that there are 16 CPUs in your system.

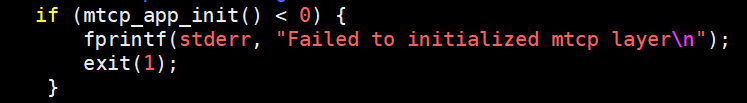
You can set the CPU limit, e.g. on a 32-core system, by using the following command:

# ./configure --with-dpdk-lib=$RTE\_SDK/$RTE\_TARGET CFLAGS="-DMAX\_CPUS=32"

In the system, I installed mtcp has 36 cpus, so I used the following command: ./configure --with-dpdk-lib=$RTE\_SDK/$RTE\_TARGET CFLAGS="-DMAX\_CPUS=36"

### Build redis-intel

1. Change the redis code to init mtcp stack.



1. Get redis-intel from <https://github.intel.com/DSLO/redis-intel>. Make sure to get the redis-4.0.8-dpdk branch.

$ git clone --single-branch -b redis-4.0.8-dpdk <https://github.intel.com/DSLO/redis-intel>

1. Go to redis-intel directory.

$ cd redis-intel

1. Need to change src/Makefile so that compiler flags contain the correct include and library path of dpdk and mtcp.

3.1. $ vi src/Makefile

(**Help**: Search for dpdk or mtcp, wherever they appear you will have to change those lines.)

3.2. Change the include path of dpdk and mtcp. In my case it was the following: -I /pnpdata/dpdk-mtcp/mtcp/dpdk/x86\_64-native-linuxapp-gcc/include/ -I /pnpdata/dpdk-mtcp/mtcp/mtcp/include

3.3. Change library path path of dpdk and mtcp. In my case it was the following: -L /pnpdata/dpdk-mtcp/mtcp/dpdk/x86\_64-native-linuxapp-gcc/lib -L /pnpdata/dpdk-mtcp/mtcp/mtcp/lib

4. $ make

5. **Before running** need to do the following:

5.1. Make sure there is a mtcp.conf file in the system, and MTCP\_CONFIG environment variable is set. There is a sample copy of mtcp.conf in …../mtcp/config/ directory.

$ export MTCP\_CONFIG=<path to mtcp.conf>

e.g. export MTCP\_CONFIG=/pnpdata/dpdk-mtcp/redis-intel/mtcp.conf

You need to make the following changes in your **mtcp.conf**:

5.1.1. port = dpdk0

5.1.2. num\_cores = 1

5.1.3. stat\_print = dpdk0

5.2. Create a “config” directory inside “redis-intel”, then create arp.conf file, which contains the mac address and IP address of the redis-client. There is a sample copy of arp.conf in …../mtcp/config/ directory.

ARP\_ENTRY 1

192.168.4.98/24 3C:FD:FE:A1:35:53

5.3. The above two config files are for mtcp’s use. You also need to edit redis.conf. Look for the keyword “bind” and “port”, and assign ip address (of dpdk0) and port number respectively.

6. Run redis-server. Redis server binary is in (redis-intel/src/redis-server):

$ redis-intel/src/redis-server <path to redis.conf>

Or you can specify the IP and port from the command line:

$ redis-intel/src/redis-server –bind <ip-address of dpdk0> <port-no>

(**Help**: Make sure the port is open , otherwise redis client won’t be able to access it. Check the iptable rule. In fedora a port (say port no. 9001) can be opened for tcp connection like this: iptables -I INPUT 1 -p tcp --dport 9001 -j ACCEPT)

You should be able to see on the terminal that mtcp is initilaizing and redis-server has started. The statistics (tx, rx etc) for the interface (dpdk0) will be printed on terminal.

(**Help**: If you can’t see the statistics then check the IP address of your dpdk0 interface. If it is not there you need to assign it again. You can stop the network manager to prevent it from happening again.)

7. From the client side you can test the connectivity by pinging the IP address. If it works fine, then you can run a redis client by specifying the redis server’s IP address and port number. If your client cannot connect to specified IP and port make sure the port is open on the server side.

### Example mtcp config file and run script

A sample of mtcp configuration and run script is given below. This setting I have for pnp152 and client-29, and client-32. Where pnp152 is the server and the clients are directly connected to server in two ethernet ports. I have bound those two ports to dpdk driver, then run mtcp application.

**Mtcp configuration file:**

############### mtcp configuration file ###############

# The underlying I/O module you want to use.

io = dpdk

#io = netmap #if mtcp is interfaced with netmap driver.

# Number of cores settings (This should equal to the number of server you want to run)

num\_cores = 4

# Number of memory channels per processor socket

num\_mem\_ch = 4

# User port (interfaces bound with dpdk driver)

port = dpdk0 dpdk1

#port = <netmap configure port name>

# Ethernet filter config

eth\_filter\_conf = fdir

# Maximum concurrency per core

max\_concurrency = 8192

# Maximum number of socket buffers per core

max\_num\_buffers = 8192

# Receive buffer size of sockets

rcvbuf = 32768 # 32K tcprmem

# Send buffer size of sockets

sndbuf = 32768 # 32K tcpwmmem

# TCP timeout seconds (-1 can disable the timeout check)

tcp\_timeout = 240

# TCP timewait seconds

tcp\_timewait = 0

# Used in multi-process configuration

# 1 - Master mode, 0 - slave mode, undefine = single instance mode

multiprocess = 1

#Prints the runtime stats from DPDK driver, you can comment it out

stat\_print = dpdk0 dpdk1 #Prints the runtime stats from DPDK driver

**Script for running dodk+mtcp redis-server**

#/bin/bash

noOfServers=4

export RTE\_SDK=/pnpdata/dpdk-mtcp/mtcp-source/dpdk

export RTE\_TARGET=x86\_64-native-linuxapp-gcc

export MTCP\_CONFIG=/pnpdata/dpdk-mtcp/redis-intel/mtcp.conf

cd /pnpdata/dpdk-mtcp/redis-intel/src

# run master process, which will create interfaces and map memory for other processes

export MTCP\_CORE\_ID=0

./redis-server /pnpdata/dpdk-mtcp/redis-dpdk-1.conf &

sleep 20s # wait for sometime to make sure initialization is done

# Now run other processes

j=5

coreNo=1

for i in `seq 1 $((noOfServers-1))`;

do

if [ $noOfServers -gt 2 ]

then

export MTCP\_CORE\_ID=$coreNo

./redis-server /pnpdata/dpdk-mtcp/redis-dpdk-$((i+1)).conf &

coreNo=$((coreNo+1))

export MTCP\_CORE\_ID=$coreNo

./redis-server /pnpdata/dpdk-mtcp/redis-dpdk-$j.conf &

coreNo=$((coreNo+1))

j=$((j+1))

fi

done

# run the last server

export MTCP\_CORE\_ID=$coreNo

./redis-server /pnpdata/dpdk-mtcp/redis-dpdk-$j.conf &

### Redis and mTCP interaction



mtcp\_app\_init() is the function that initializes mTCP stack. During the mTCP stack initialization, an mTCP thread is created which will read new packets from the nic queue in bursts. Whenever there is a new packet, mTCP thread will add that event to listening socket and mTCP event queue. The mTCP epoll has access to the mTCP event queue and to the socket. Simultaneously, Redis thread is monitoring the socket using epoll\_wait() for new event. During the mTCP initialization function pointers of socket functions will point to mTCP socket functions. Consequently, when Redis thread will call epoll\_wait(), mtcp\_epoll\_wait() will be invoked. And the mtcp\_epoll has access to the event queue where mTCP thread is adding events.

### mTCP codeflow



# **f-stack**

### Build f-stack

git clone <https://github.com/F-Stack/f-stack.git>

cd f-stack

cd dpdk/usertools

./dpdk-setup.sh # build and compile dpdk

export FF\_PATH=/pnpdata/f-stack

export FF\_DPDK=/pnpdata/f-stack/dpdk/x86\_64-native-linuxapp-gcc

cd ../../lib

make # build f-stack library

### Build redis app

cd ../app/redis-3.2.8/src

make # build redis app

### Change f-stack code to enable ethernet flow director

To run fstack in multi-process single-nic mode (where multiple processes are using the same ethernet interface), we need to use ethernet flow director. The implementation is in ncc7 (10.242.51.139. path: /pnpdata/f-stack). It is implemented in a separate git branch named ethernet flow director. To see the changes I have made: git diff master..ethernet\_flow\_director lib/ff\_dpdk\_if.c. Checkout the branch and do the following:

f-stack/lib/ff\_dpdk\_if.c:699, change the IP address according to your server’s IP address. [Using this tool](https://www.silisoftware.com/tools/ipconverter.php), convert IP to hexdecimal format, then put it in the line.

Note that, when we set flow director, the port number is strating from 9001. And its hardocded. If you have other port number need to change it in the f-stack/lib/ff\_dpdk\_if.c:953

After that build f-stack and corresponding application.

### Start a F-Stack application

Since F-Stack is multi-process architecture, every F-Stack application process should call ff\_init(argc, argv) to initialize the environments. For example, if lcore\_mask=f in config.ini, you can start your app like this:

${bin} --conf config.ini --proc-type=primary --proc-id=0

${bin} --conf config.ini --proc-type=secondary --proc-id=1

${bin} --conf config.ini --proc-type=secondary --proc-id=2

${bin} --conf config.ini --proc-type=secondary --proc-id=3

For example it can be run as below:

./app/redis-3.2.8/src/redis-server --conf config.ini --proc-type=primary --proc-id=0 redis-1.conf

./app/redis-3.2.8/src/redis-server --conf config.ini --proc-type=secondary --proc-id=1 redis-2.conf

### Port an application to f-stack

1. Change Makefile to include fstack library and header.
2. Initialize f-stack with this line: ff\_init(argc, argv)
3. ff\_mod\_init(): its in anet\_ff.c: it contains the wrapper of standard socket functions (bind, connect, listen etc.)
4. it is different for unix and redis. Why?
5. If the application require command line argument, need to adjust command line arguments.
6. ff\_run(loop, server.el)
7. Define a loop() function in the **int loop(void\* arg)** format.

f-stack codeflow

