User Manual for NFV IMS core on kernel bypass setup

NOTE: The instructions given in this manual work only for Linux-used machines, and might not work as expected on other OSes such as Mac and Windows. Please use online references in such cases. For details on understanding the code and various procedures involved in IMS core, please look at the **`developer_manual.pdf`**

Installation of Kernel Bypass:

- Download **`user manual`** from `https://github.com/networkedsystemsIITB/Modified_mTCP/blob/master/mTCP_over_Netmap/docs/netmap_docs/user_manual.pdf`.
- Install netmap on host server using **`step 1 to 4`** given in **`user manual`**.
- Make sure netmap module is loaded on the host.
- You need to set up 5 VMs for running NFV IMS core
- Install netmap on all the guest VMs. Follow **`step 6`** from **`user manual`**.
- You'll need to do the setup in the host for software-based packet distribution. Use steps are given at **`5.1.Setup in the host for software-based packet distribution`** from the user manual. After applying patches you need to create Persistent VALE ports.

  For RAN VM you need only single queue VALE port. For other NFs (like P-CSCF, I-CSCF, S-CSCF, HSS) you need to create the number of queues equal to the number of CPUs you want to give to that VNF.

  Suppose you want to run PCSCF on VM with two cores, then you'll need to create two queue VALE port. Assuming that you are planning to assign MAC address 00:aa:bb:cc:dd:01 to PCSCF VM. Then steps to add interface vi1 are

  ```
  ./vale-ctl -n vi1 -C 2048,2048,2.2
  ./vale-ctl -a vale0:vi1
   ifconfig vi1 hw ether 00:aa:bb:cc:dd:01
  ```

  NOTE: In case if the host is restarted you'll need to again create all VALE ports.

- Once this is done, you need to configure MAC address for VM's which you're going to use. For that use steps given at **`section 5, subsection 5.1, step 3`** from **`user manual`**.
- Once you start the guest VM, load netmap module inside guest VM using the following command.

```
$ cd netmap/LINUX
$ sudo insmod netmap.ko ptnet_vnet_hdr=0
```

You'll see new interface getting created when you're doing it for the first time. At that time go to **`Network Connections`**, **`Edit Connections`** and change IPv4 Settings of the new interface which was just created to **`Link Local Only`**.

- Download MTCP from `https://github.com/eunyoung14/mtcp`. Configure MTCP using following commands.

  ```
  $ ./configure --enable-netmap CFLAGS="-DMAX_CPUS=32"
  $ make
  ```

  You can check `https://github.com/eunyoung14/mtcp/blob/master/README.netmap` for more details about configuring MTCP.

- Steps for changing the number of cores assigned to VNF: (This applies to PCSCF, ICSCF, SCSCF, and HSS.)
  - Shut down VM in question.
  - Modify number of cores assigned to VM as per requirement.
  - Detach and remove the VALE interface using following commands.

    ```
    ./vale-ctl -d vale0:viX
    ./vale-ctl -r viX
    ```

    Where viX is interface you want to delete.
  - Recreate the interface with required number of queues.

SETUP:

- Download repository.
- You will need to have 5 VMs for setup. Assign one VM for each software module (P-CSCF, I-CSCF, HSS, S-CSCF and RAN). Before proceeding ahead, ensure proper communication between all VMs using the ping command.
- Open **`common.h`** file, replace IP address of each component with IP address you're planning to assign to that component.
  If VM on which you're planning to use has **`VALE interface with IP address`** 10.20.30.40 then change

```
#define UEADDR "192.168.122.251" // RAN
```

to

```
#define UEADDR "10.20.30.40" // RAN
```

Similarly for other all NFs.

- Copy downloaded source on all VMs (with updated **`common.h`**) at `mtcp/apps/`, where `mtcp` is folder where you have configured **`mTCP`**.
- Run **`ifconfig`** command. Check the name of interface which was created using netmap. Open **server.config** and replace interface name with name of interface created on your VM. That is if vale interface is eth3. Replace

  ```
  #------- Netmap ports --------#
  port = eth5
  ```

  with

  ```
  #------- Netmap ports --------#
  port = eth3
  ```

- Run following command on all VMs.

  ```
  $ sudo apt-get install libssl-dev
  ```

- Setting up how many cores you want to run NF with (This instruction applies to PCSCF, ICSCF, HSS, SCSCF). VM hosting RAN should have 4 or more cores.
  For example, if you want to run HSS with 1 core, open **`hss.cpp`** and replace

  ```
  #define MAX_THREADS 3
  ```

  with

  ```
  #define MAX_THREADS 1.
  ```

  Similarly open **`server.config`** and change

  ```
  num_cores = 2
  ```

  with

  ```
  num_cores = 1
  ```

  Similarly open **`Makefile`** and change

CFLAGS=–DMAX_CPUS=2

with

CFLAGS=–DMAX_CPUS=1

Make sure VM on which HSS is running has one core. Similar settings if you want to scale HSS to 2 core, 3 cores and so on.

- Run make. Now you will have executables created like **`mtcp_pcscf`** , **`mtcp_icscf`** , **`mtcp_hss`** , **`mtcp_scscf`** .
- For creating executable for RAN simulator, go to the VM on which you're planning to run RAN, rename **`Makefile_RAN`** with **`Makefile`** . Then execute

  ```
  $ make ransim.out
  ```

NOTE: If there are any errors thrown in the compilation, install the corresponding dependency.

Experimentation:

We will use RAN simulator to simulate the number of concurrent UE and make UE perform register, authentication and deregistration procedures.

- Make sure that you've correctly done the setup on all VMs and assigned cores properly. Make sure that **`common.h`** contains correct IP addresses.
- Once the setup is ready, run each executable on its own VM.
  Usage pattern:

MODULE	**USAGE**
HSS	**`sudo ./mtcp_hss.out`**
PCSCF	**`sudo ./mtcp_pcscf.out`**
SCSCF	**`sudo ./mtcp_scscf.out`**
ICSCF	**`sudo ./mtcp_icscf.out`**

- Run following commands on VM on which you're planning to use RAN simulator.

```
sudo su
echo 1 > /proc/sys/net/ipv4/tcp_tw_reuse
```

- RAN: start RAN simulator with appropriate parameters as given below. This will generate the required amount of control traffic for the given time duration.
  `sudo ./ransim.out <#RAN threads><Time duration in #seconds>`
  A sample run is as follows
  `sudo ./ransim.out 100 300`

Performance Results:

Two performance metrics are computed at end of the experiment by RAN and displayed as output.

- **`Throughput`** : Considering a combination of registration, authentication and deregistration request as a single request, throughput is the number of requests completed per second.
- **`Latency`** : Time taken by individual request to complete.