**RAN: (decrease max\_rxgain if LLLLLL occurs, to 95 for 100PRB)(use 2020.w44 branch)**

1. Install ubuntu 18.04 on eNB computer, install low latency kernel
2. Program USIM card with given programmer application, the PLMN should be 22201 for Phone we purchased
3. Disable turbo frequency in BIOS setting, see OAI guides, etc., fix core to 37x, check i7z
4. Install low-latency-hwe-18.04 kernel in RAN computer
5. Download and build OAI code (develop branch); configure “.conf” file accordingly
6. If connect 3 phones with develop branch OAI, crush the eNB, see [here](https://drive.google.com/file/d/1ZmZpSW7HmNi8qidTMAUpXfnZs_f2fY__/view?usp=sharing)
   1. changed targets/COMMON/openairinterface5g\_limits.h from 4 to 128
   2. changed openair2/COMMON/platform\_constant.h from 4 to 128
7. Download flexran and install, see eurcom/flexran-rtc (Note: FlexRAN updated format since July 2020)
   * install dependencies with “/tools/install\_dependencies.sh”
   * If build error, e.g., pistache, its version should be “revert-236-master;

CHECK THIS MODIFIED CODE: [HERE](https://drive.google.com/file/d/1kYqyfUvsN6qlN6pu9eF9OpSyzSCtKYFM/view?usp=sharing)

MODIFICATION NEEDED IN FlexRAN Controller CODE

openair2/COMMON/platform\_constants.h

openair2/ENB\_APP/MESSAGES/V2/config\_common.proto

openair2/ENB\_APP/enb\_config.c

openair2/ENB\_APP/flexran\_agent\_ran\_api.c

openair2/LAYER2/MAC/mac.h

openair2/LAYER2/MAC/main.c

openair2/LAYER2/MAC/pre\_processor.c

openair2/LAYER2/MAC/slicing/slicing.c

openair2/LAYER2/MAC/slicing/slicing.h

targets/COMMON/openairinterface5g\_limits.h

* FlexRAN virtualization with maxmcs for each slice, see compressed [openair2](https://drive.google.com/file/d/1EEGf7f6PvKUyypATs4JHytD4eqskhv4e/view?usp=sharing)/
  + add this line at the line #78 in the file “config\_common.proto” (this file generates .h headers) for both **eNB/gNB** and **FlexRAN controller**  
    “optional uint32 maxmcs = 28;”
  + Add maxmcs in slicing.h in line#69: uint32\_t maxmcs;
  + Add maxmcs in slicing.c in line#184/150: ns->dl\_algo.setup(dl->maxmcs);
  + Add maxmcs in slicing.c in line#217/255: ns->ul\_algo.setup(ul->maxmcs);
  + Add maxmcs in flexran\_agent\_run\_api.c line#3130/3208: ((static\_slice\_param\_t \*)params)->maxmcs = s->maxmcs;
  + Add maxmcs in flexran\_agent\_run\_api.c line#3076/3041: \_algo.setup(28); // 28 for dl or 20 for ul
  + Add maxmcs in flexran\_agent\_run\_api.c line#3290: dl\_pp->dl\_algo.setup(\*(uint32\_t \*)dl\_pp->dl\_algo.data); // *get prev stored data*
  + Add maxmcs in flexran\_agent\_run\_api.c line#3306: ul\_pp->ul\_algo.setup(\*(uint32\_t \*)ul\_pp->ul\_algo.data); // *get prev stored data*
  + Modify enb\_config.c line#272: dl\_pp->dl\_algo.data = dl\_pp->dl\_algo.setup(28);
  + Check dl\_algo.setup() and ul\_algo.setup() to set value
  + Modify main.c (MAC) line#109/117: \_algo.setup(28); // 28 for dl or 20 for ul
  + Modify mac.h line#1184/1197: void \*(\*setup)(uint32\_t); // *not void input anymore*
  + Modify other functions if they use .setup().
  + Modify pre\_process.c for different sched\_algos (.setup(), .run())  
    1) rr\_dl\_setup: \*(int \*) data = maxmcs; // line#138  
    2) rr\_dl\_run: uint32\_t maxmcs = \*(uint32\_t \*) data; // line# 163  
     int s\_ue = 0; // line#164  
     const int mcs = min(maxmcs, cqi\_to\_mcs[cqi]); // line#226  
     int \*start\_ue = &s\_ue; // line#171, *not “data” anymore*

3) rr\_ul\_setup: \*(int \*) data = maxmcs; // line#138  
4) rr\_ul\_run: uint32\_t maxmcs = \*(uint32\_t \*) data; // line# 808  
 min(maxmcs, UE\_info->UE\_sched\_ctrl[UE\_id].phr\_received == 1 ? 20 : 10),   
 int \*start\_ue = &s\_ue; // line#949 *not “data” anymore*

* Usage:
  + Get stats: curl -X GET <http://127.0.0.1:9999/stats/>
  + Config slices: curl -X POST http://127.0.0.1:9999/slice/enb/-1 --data-binary @conf.json (file, [here](https://drive.google.com/file/d/1bQT0n-_mE3oz2ct5J851z8kpupNgzHdQ/view?usp=sharing))
  + Config UE association: curl -XPOST http://127.0.0.1:9999/ue\_slice\_assoc/enb/ --data-binary @ue.json (file, [here](https://drive.google.com/file/d/10FLqQu9iBsJowMSM2apw0B62VoEw3Oo4/view?usp=sharing))
* reliability calculation in RAN based on FlexRAN
  + FlexRAN has interface/predefined variable total\_prb\_retx (total prb used for retransmission for each user)
  + But it is not implemented. We implement it as follow (see here):
  + 1) flrxran\_agent\_ran\_api.c and flrxran\_agent\_ran\_api.h: add flexran\_get\_total\_num\_prb\_retx\_dl\_per\_ue(), just like flexran\_get\_num\_prb\_retx\_dl\_per\_ue()
  + 2) flexran\_agent\_mac.c: call the new function.
  + 3) mac.h #742: add variable for count ul retx,
  + 4) pre\_processor.c #835: record the retx for the users
  + 5)since the interfaces are there, no need to add variables in stats\_common.proto for both OAI RAN and FlexRAN controller
  + The function has to be return uint32\_t since retx number could be large
  + Count the number of retransmission in the given time, to calculate the retx prob

**ANDRIOD:**

1. Add APN (“apn1.carrier.com”) in cellular settings, only ipv4 protocol
2. Install android studio in ubuntu
   * install openjdk11 first
   * go to Tool->SDK Manger, select android 9 &10 to install, then rebuild project;
   * upgrade gradle if you see on screen.
   * Layout is in “res->layout->activity\_main.xml”
3. Install application to all phones in ubuntu in RAN computer
   * Make sure the storage access of the app is ON
   * Copy an image named “1.jpg” to phone internal storage “DCIM/”, see the hardcode in app project
   * change some default server addr&port in Layout

**Transport:**

1. ODL version: 0.5.3-Boron-SR3
2. Main code: tn\_server.py
3. Change “SWITCH\_ID” to communication with the correct openflow switch;
4. Change “DATABASE” for SPGWU addr (192.168.17.3) and RAN&CN VM addr (current is 10.0.0.100 & 102)
5. Change “PORT\_LIST”, currently is just inport 1 and outport 12
6. Take care the static flow id I set hardcode, e.g., [1] for arp
7. REST API PORT: 7777

**CORE:**

* Follow this tutorial to build CUPS based core network, [link](https://github.com/OPENAIRINTERFACE/openair-epc-fed/blob/2021.w06/docs/DEPLOY_HOME.md). Make sure the version is 2021.w06.
* If everything setup correctly (after testing with RAN), then use the following steps to enable core virtualziation.
* We will create a new “test\_oai\_spgwc” and multiple “test-oai-spgwu”, the main changes are all in spgwc container
  + The modified codes are in the folder ” core\_manager\_modified\_codes/”, the CDM ONLY ACCEPT ONE SLICE INFO “PUT” at a time.

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

* + Bring up a new test-oai-spgwc: copy “Dockerfile.SPGWC” into CP VM “openair-epc-fed/”, and refer to “setup\_spgwc.txt” for setup the spgwc container
  + *prerequisites*: install cpprestsdk from source, see [here](https://github.com/Microsoft/cpprestsdk/wiki/How-to-build-for-Linux) ):
  + 1) modify CMakeList.txt in /src/oai-spgwc/.
    - //add these two in the section “add\_library (SPGWC STATIC ...”, line #315

//---------------------------------------------------------------------------------------------------------------

${SRC\_TOP\_DIR}/oai\_spgwc/BasicController.cpp

${SRC\_TOP\_DIR}/oai\_spgwc/BasicService.cpp

//---------------------------------------------------------------------------------------------------------------

* + - //add these three to the line just above the last line

//---------------------------------------------------------------------------------------------------------------

find\_package(Boost REQUIRED COMPONENTS system)

find\_package(cpprestsdk REQUIRED)

message(OPENSSL LIBS " ${OPENSSL\_LIBRARIES}")

//---------------------------------------------------------------------------------------------------------------

* + - Add “cpprestsdk::cpprest ${OPENSSL\_LIBRARIES} ${Boost\_LIBRARIES}” in target\_link\_libraries (last line)
  + 2) modify main.cpp in /src/oai-spgwc/.
    - Add the following at the beginning

//---------------------------------------------------------------------------------------------------------------

#include "BasicService.hpp"

#include "InterruptHandler.hpp"

#include <pplx/threadpool.h>

//---------------------------------------------------------------------------------------------------------------

* + - the following codes are added before “pause()” at the end of main() func

//---------------------------------------------------------------------------------------------------------------

InterruptHandler::hookSIGINT();

BasicService server(0);

server.setEndpoint("http://0.0.0.0:8888");

Logger::pgwc\_app().error("################## REST API started.... ###################");

try {

server.accept().wait();

InterruptHandler::waitForUserInterrupt();

server.shutdown().wait();

}

catch(std::exception & e) {; }

//---------------------------------------------------------------------------------------------------------------

* 3) copy files “BasicService.cpp”, “BasicService.hpp”, “BasicController.cpp”, “BasicController.hpp”, “std\_micro\_service.hpp”, “InterruptHandler.hpp” into the folder /src/oai-spgwc/
* 4) Modify pgw\_pfcp\_association.cpp
* Add this at the beginning: #include "BasicService.hpp"
* at #233, replace pfcp\_associations::select\_up\_node() with following codes. This function selects the spgwu node for the incoming session\_creat\_request from s11 interface. The variable “associations” stores all the associated SPGWUs. The “FOR\_EACH” iterates these associations, we just need to determine which SPGWU we should choose

//----------------------------------------------------------------------------------------------------------------

bool pfcp\_associations::select\_up\_node(pfcp::node\_id\_t& node\_id,

const int node\_selection\_criteria,

const std::string imsi) {

node\_id = {};

if (associations.empty()) {

return false;

}

folly::AtomicHashMap<int32\_t, std::shared\_ptr<pfcp\_association>>::iterator it;

std::cout<<"############ selecting SPGWU node for user imsi "<<imsi<<std::endl;

std::string target\_addr = find\_users\_in\_configs(imsi); // find target addr assigned by slice config

std::cout<<"######### based on slice mapping configs, the target\_addr is "<<target\_addr<<std::endl;

//////////////////////////// select pfcp association ///////////////

node\_id = associations.begin()->second->node\_id; // assign the first SPGWU node by default

std::cout<<"######### by default, pre-allocated addr is "<<inet\_ntoa(node\_id.u1.ipv4\_address)<<std::endl;

FOR\_EACH(it, associations) {

// char \*spgwu\_addr\_str = inet\_ntoa(node\_id.u1.ipv4\_address);

std::string spgwu\_addr\_str(inet\_ntoa(it->second->node\_id.u1.ipv4\_address));

std::cout<<"########## current node\_id addr is "<<spgwu\_addr\_str<<std::endl;

// if the target addr is matched with current node addr

if (target\_addr.compare(spgwu\_addr\_str) == 0) {

node\_id = it->second->node\_id;

std::cout<<"########## found matched spgwu, final selected node\_id addr is "<<inet\_ntoa(node\_id.u1.ipv4\_address)<<std::endl;

return true;

break;

}

}

return true;

}

//---------------------------------------------------------------------------------------------------------------

* 5) change pgw\_pfcp\_association.hpp, add the third argument at the definition of select\_up\_nodes(): “const std::string imsi”
* 6) add “pc->imsi.toString()” in each select\_up\_node() as the third argument in pgwc\_procedure.cpp (three places can be found)
* 7) rebuild the SPGWC as usual, test if it starts (“./build\_spgwc --clean --build-type Release --jobs”)
* 8) others:
* Benchmark for REST C++ framework, see [here](https://github.com/guteksan/REST-CPP-benchmark). Remove “log4cplus” in codes if build error.
* Debug with VS code: <https://code.visualstudio.com/docs/cpp/cmake-linux>
* Client request with curl command: (can ONLY use json.dumps(data) format:  
  curl --header "Content-Type: application/json" -d '{"IMSI": "222010000000003", "ADDR": "127.0.0.2"}' http://127.0.0.1:8888/configs/ -X PUT

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

We then bring up three spgwu containers:

docker run --privileged --name test-oai-spgwu-1 --network vir-oai-public-net --ip 192.168.17.11 -d --entrypoint /bin/bash oai-spgwu-tiny:production -c "sleep infinity"

docker run --privileged --name test-oai-spgwu-2 --network vir-oai-public-net --ip 192.168.17.12 -d --entrypoint /bin/bash oai-spgwu-tiny:production -c "sleep infinity"

docker run --privileged --name test-oai-spgwu-3 --network vir-oai-public-net --ip 192.168.17.13 -d --entrypoint /bin/bash oai-spgwu-tiny:production -c "sleep infinity"

(this following command need spgw0\_ip, get from before hss/mme/etc setting)

python3 component/oai-spgwu-tiny/ci-scripts/generateConfigFiles.py --kind=SPGW-U \

--sxc\_ip\_addr=${SPGW0\_IP} --sxu=eth0 --s1u=eth0 --from\_docker\_file

docker cp ./spgwu-cfg.sh test-oai-spgwu-1:/openair-spgwu-tiny

docker cp ./spgwu-cfg.sh test-oai-spgwu-2:/openair-spgwu-tiny

docker cp ./spgwu-cfg.sh test-oai-spgwu-3:/openair-spgwu-tiny

docker exec -it test-oai-spgwu-1 /bin/bash -c "cd /openair-spgwu-tiny && chmod 777 spgwu-cfg.sh && ./spgwu-cfg.sh"

docker exec -it test-oai-spgwu-2 /bin/bash -c "cd /openair-spgwu-tiny && chmod 777 spgwu-cfg.sh && ./spgwu-cfg.sh"

docker exec -it test-oai-spgwu-3 /bin/bash -c "cd /openair-spgwu-tiny && chmod 777 spgwu-cfg.sh && ./spgwu-cfg.sh"

**DEBUG:**

* If user can ping SGi but not internet connected. Solution: “apt install iptables” in the SPGWU container.