RIC benchmarking

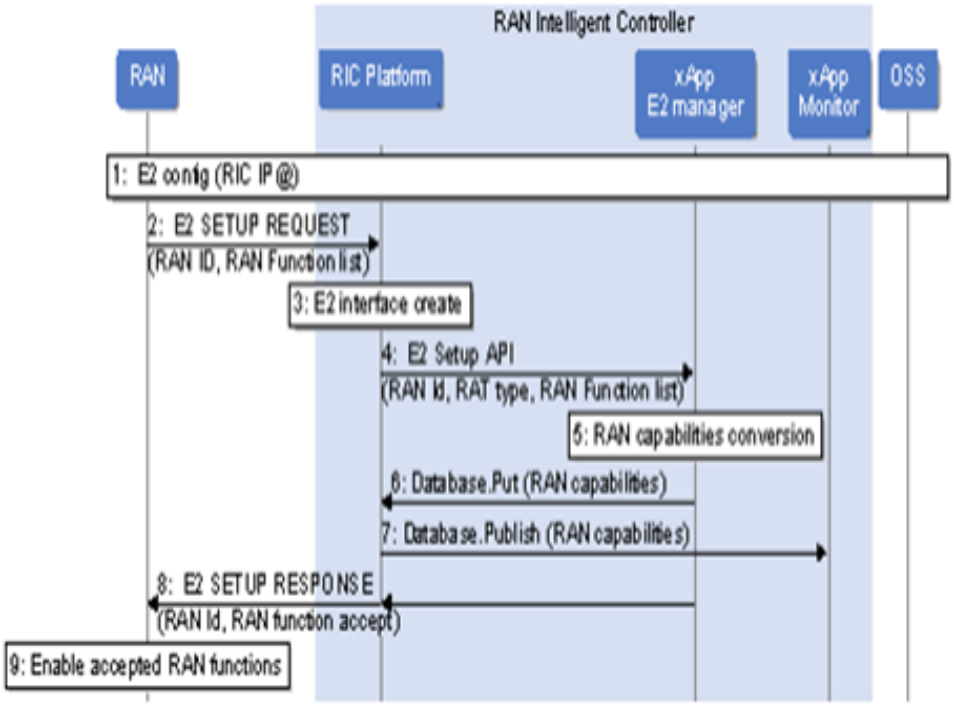
Introduction

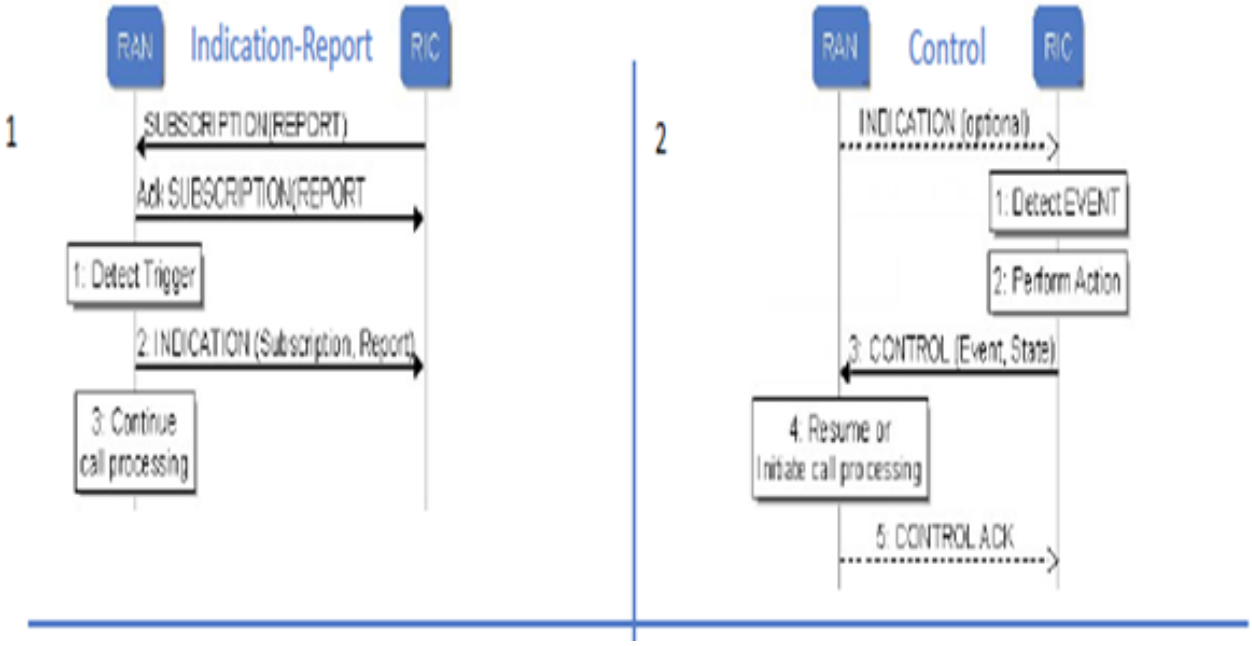
Performance RIC benchmarking is a feature that aims to provide the performance metrics on both the RAN (Radio Access Network) and Near-RT RIC platform. It involves the communication of E2-simulator from RAN side to Near-RT RIC platform infrastructure that uses the xApp onboarded on RIC platform. The xApp that interacts messages with E2simulator through the Near RT RIC platform pods and records the time that provides metrics on latency /throughput on the RIC platform and on the xApp onboarded on RIC platform is termed as Bouncer xApp.

Features:

- The Bouncer xApp demonstrate RIC benchmarking using the E2 simulator communicating with Near-RT RIC platform .
- Bouncer xApp interacts with E2simulator using the Near- RT RIC platform for sending the RIC Subscription Request and receiving Response for the same.
- Bouncer xApp receives the RIC indication and sends the Control Message.
- This communication is stored in a file with timestamp to determine the latency/throughput as part of Performance RIC benchmarking purpose.

Architecture:





Prerequisites:

Near-RT RIC platform setup is required. Link for Near-RT RIC installation.

E2-interface (E2-Simulator) code needs to be build and deployed.

The Bouncer xApp needs to be build, deployed and onboarded on the Near RT RIC platform.

Building E2Sim(E2-interface):

Path: to build Dockerfile : ric_benchmarking/e2-interface/e2sim/e2sm_examples/kpm_e2sm/

build -f Dockerfile -t nexus3.o-ran-sc.org:10002/oran-ric/e2simul:2.0.0 .

docker push nexus3.o-ran-sc.org:10002/oran-ric/e2simul:2.0.0 .

Note: For Running single/multiple instances of E2Sims, Kindly, follow the README file and update the Dockerfile accordingly before building it.

Installing E2Sim(E2-interface):

helm install -n e2sim . --namespace <namespace_name>

check E2Sim pod: *kubectl get pods -n <namespaces_name>*

kubectl logs <e2sim-pod> -n namespace --follow

The logs must show the entries in the end as below:

[E2AP] Unpacked E2AP-PDU: index = 2, procedureCode = 1

[E2AP] Received SETUP-RESPONSE-SUCCESS

This means the E2Sim(RAN) has connected to E2term pod of RIC Platform successfully, and its ready to communicate with xApp. So keep it as it is. and run the Bouncer xApp.

So now you can follow the steps to Deploy the Bouncer xApp.

If, the E2Sim logs doesn't shows above RESPONSE, then it means the E2sim is not connected to E2term pod of RIC Platform and check the status of E2term pods and probe accordingly.

To delete a running E2sim pod:

kubectl delete deployment.apps/e2sim -n namespace

Go to path : ric_benchmarking/e2-interface/e2sim/e2sm_examples/kpm_e2sm/helm

helm del --purge e2sim

To restart the E2Sim pod:

Note: This is done only when either post re-deployment of RIC Platform Pods / RESTART of RIC platform Pods

kubectl -n test rollout restart deployment e2sim

Building xApp:

docker build -f Dockerfile -t nexus3.o-ran-sc.org:10002/oran-ric/bouncer_test:1.0.0 .

docker push nexus3.o-ran-sc.org:10002/oran-ric/bouncer_test:1.0.0

Preparing an xApp for onboarding:

Here we are preparing for API calls into the xApp On-Boarder by providing the locations of the xApp descriptors:

Onboarding of xApp:

STEP -1:

echo '{"config-file.json_url": "https://gerrit.o-ran-sc.org/r/gitweb?p=ric-app/bouncer.git;a=blob;f=Bouncer/init/config-file.json;hb=HEAD"}' > onboard.bouncer.url

STEP -2:

curl --location --request POST "http://$(hostname):32080/onboard/api/v1/onboard/download" --header 'Content-Type: application/json' --data-binary "@./onboard.bouncer.url"

To view onboarded charts:

curl --location --request GET "http://$(hostname):32080/onboard/api/v1/charts"

STEP -3: **Deploying xApp**

Deploy the xApp by invoking the xApp Manager's API.
Note: that the names of the xApp to be deployed must match with what the on-boarder has.
Once receiving the deploy API call, the xApp Manager will make API call into Helm/Kubernetes to deploy the xApp's Helm chart.
The Routing Manager is also involved if the xApp needs to process RMR messages, – it will complete the routes and send out route updates.

curl --location --request POST "http://$(hostname):32080/appmgr/ric/v1/xapps" --header 'Content-Type: application/json' --data-raw '{"xappName": "bouncer-xapp"}'

Now check the bouncer xApp pod:

kubectl get pods -n ricxapp

For eg.

```
NAME                                    READY  STATUS   RESTARTS  AGE
ricxapp-bouncer-xapp-b8f45f4f6-tnvpq   1/1       Running  0              2m2s
```

Once Bouncer xApp pod is spawned, after around 90 secs, the E2sim logs gets updated with RIC Subscription, RIC Response and RIC Indications

To delete an onboarded xApp:

curl --location --request DELETE "http://$(hostname):32080/appmgr/ric/v1/xapps/bouncer-xapp"

Note: where the name: 'bouncer-xapp' is the name of your xApp.

Results:

Round trip time for each interaction are recorded and stored in the timestamp.txt file that gets created in the container.

command to login to container: *kubectl exec -it <bouncer xapp pod> -n ricxapp /bin/bash*

Inside container cd /tmp

cat timestamp.txt

snippet:

[

Received Msg with msgType 28 June: 12011 at time: 1624982927922998
Received Msg with msgType 28 June: 12011 at time: 1624982928923207
Received Msg with msgType 28 June: 12050 at time: 1624982929923397
Send Msg with msgType 28 June: 12040 at time: 1624982929923510
Time diff: 113

]

Similarly, login to the E2sim container and find the timestamp file for E2Sim

kubectl exec -it <E2Sim pod> -n <E2sim namespace> /bin/bash

inside Container cd /tmp

cat Timestamp.txt

snippet:

[

Sent RIC Indication at time: 1606157488066
Received RIC Control Msg at time: 1606157489066
Time diff in Microseconds:1000

]