Developing and using reference scenarios

Overview

Reference scenarios are basics scenario bricks to help build more complex scenarios. They are maintained in OpenBACH with naming conventions and formal organisation as described above.

Each scenario aims to standardize a metrology test; for instance, the `network_delay` reference scenario gives a “standard” way to evaluate a link delay in OpenBACH.

These scenario can be launched on their own using executors or can be integrated in upper code as an API. Each scenario module define two kind of functions:

- reference scenario functions: they create and return a scenario. The scenario must be configured using helpers for consistency. Only the metrology part of the scenario should be defined in those functions.
- the `build` functions: it selects the right scenario function based on “meta” parameters and can add post-processing jobs if required based on the presence of the `post_processing_entity` parameter.

Each reference scenario must define at least one reference scenario function and the `build` function.

How to configure and launch subscenarios

You can also import reference scenarios (or some parts) and add them as subscenarios. In the same example network_delay which:

- Launches the subscenarios `network_delay_simultaneous_core` or `network_delay_sequential_core` (allowing to compare the RTT measurement of fping, hping and owamp jobs).
- Launches two postprocessing helpers (launching postprocessing jobs) to compare the time-series and the CDF of the delay measurements.

As you can see in the import modules of the scenario, we are importing the helpers and the openbach functions StartJobInstance/StartScenarioInstance to launch our reference subscenario and the postprocessing jobs.

```
from scenario_builder import Scenario
from scenario_builder.helpers.network.owamp import owamp_measure_owd
from scenario_builder.helpers.network.fping import fping_measure_rtt
from scenario_builder.helpers.network.hping import hping_measure_rtt
from scenario_builder.helpers.postprocessing.time_series import time_series_on_same_graph
from scenario_builder.helpers.postprocessing.histogram import cdf_on_same_graph,
pdf_on_same_graph
from scenario_builder.openbach_functions import StartJobInstance, StartScenarioInstance
```

Below, you can see how to use subscenarios thanks to the :

- *start_scenario_instance function*

(`scenario.add_function('start_scenario_instance')`) and
- the configuration of these subscenario functions available in the same script (in our case: `network_delay_simultaneous_core` and `network_delay_sequential_core`) with their arguments (`start_scenario_core.configure(scenario_core, srv_ip=srv_ip, duration=duration)`).

Subscenarios from other scripts could be also imported and launched.

```
if simultaneous:
   scenario_core = network_delay_simultaneous_core(clt_entity)
else:
   scenario_core = network_delay_sequential_core(clt_entity)

start_scenario_core = scenario.add_function(
        'start_scenario_instance')

start_scenario_core.configure(
        scenario_core,
        srv_ip=srv_ip,
        duration=duration)
```