Helpers: a simplification of your scenarios

Overview

Helpers aim to simplify the scripting of your scenarios, by giving wrappers for common functions or tasks.

Convention to follow when developping helpers:

- Helpers should only wrap a single job.
- Helpers can also incorporate logic to stop the jobs after a given amount of time.

In general, there should be one helper per job. There are cases when the server and the client of different jobs are related (e.g. (voip_qoe_src and voip_qoe_dst) or (dash_client and dash_player&server). In these cases, one helper could cover several jobs. This should be avoided whenever possible.

To be exploited, helpers need to be imported in a scenario. An example can be found in the reference scenario transport_tcp_one_flow.

The helpers need to be imported as follows:

```
from scenario_builder.helpers.metrics import fping_measure_rtt, hping_measure_rtt
```

For a description of the available helper and /or pre-built scenarios, got to the exploitation section

In that case, *fping_measure_rtt* and *hping_measure_rtt* allow to launch and stop the fping and hping jobs. Following previous example, we modify the script in order to use the helpers. Moreover, in this example hping and fping are launched sequentially (one after the other)

delay_scenario_with_helpers.py

```
from scenario_builder import Scenario
from auditorium_scripts.scenario_observer import ScenarioObserver
from scenario_builder.helpers.traffic_and_metrics import fping_measure_rtt,
hping_measure_rtt

def build_scenario(client, scenario_name):
    scenario = Scenario(scenario_name, 'Comparison of 2 types of RTT measurements
sequentially')
    scenario.add_argument('ip_dst', 'Target of the pings and server ip adress')

    wait = hping_measure_rtt(scenario, client, '$ip_dst', 60)
    fping_measure_rtt(scenario, client, '$ip_dst', 60, wait)
    return scenario

def main(scenario_name='Delay metrology scenario'):
    observer = ScenarioObserver()
    observer.add_scenario_argument(
            '--client', '--client-entity', default='Client',
            help='name of the entity for the client of the RTT tests')
    observer.add_scenario_argument(
            '--sequential', action='store_true',
            help='whether or not the test should run one after the other')
    observer.add_run_argument(
            'ip_dst', help='server ip address and target of the pings')
    args = observer.parse(default_scenario_name=scenario_name)

    build = build_scenario(args.client, scenario_name)
    observer.launch_and_wait(build)


if __name__ == '__main__':
    main()
```

Key Principles

A few key principle of helpers:

- They must accept the scenario to configure jobs for as first parameter;
- Then come the arguments needed to configure the jobs;
- Lastly they must accept the following defaulted parameters: `wait_finished=None, wait_launched=None, wait_delay=0`.

These last 3 parameters must be used in the first`scenario.add_function` performed. This is to ensure that it is easy to schedule helpers relative to one another.

When all the configuration is done, the helper must return a list containing the awaitable openbach functions that are relevant for scheduling with respect to other helpers.

A simple example: the helper for the `command_shell` job:

command_shell.py

```
def command_shell(
        scenario, entity, command,
        wait_finished=None, wait_launched=None, wait_delay=0):
    command = scenario.add_function(
            'start_job_instance',
            wait_finished=wait_finished,
            wait_launched=wait_launched,
            wait_delay=wait_delay)
    command.configure('command_shell', entity, command_line=command)

    return [command]
```

Extra duties

Helpers also contains a second kind of function, albeit rarer, to help find some specific configuration for a given job. This is *e.g.* the case for `iperf3` where the provided function will help find the server instance of the job. These functions are there to help extract the right openbach function from the scenario afterwards, mainly to add it into a post-processing job; provide them for your jobs if the name itself is not sufficient to retrieve the right ones (in the case of `iperf3`, only the server emits stats, so we do not want to post-process clients).