Understanding Source Code

Topics:

1. Custom Teststeps - How to write custom steps (allowed keywords, format, etc). What testcases should be covered under this?
    a. This is described in detail in VSPERF documentation (see docs/testing/user/userguide/teststeps.rst). There are also a plenty of examples available in both documentation and configuration files (see files conf/integration/01*conf).
    b. In case of Step Driven Tests "usage", there aren't any restrictions. In general, this feature is intended for flexible prototyping of new deployment scenarios, support of new vswitches, etc. In case that some scenario is to be used very often (like p2p and pxp), then it make sense to implement a new deployment scenario under core/vswitch_controller_NewDeploymentName. However in case of rarely used scenarios, one off tests or integration testcases, it make more sense to use Step Driven Testcases. The big advantage of step driven testcases is, that you don't need to modify vsperf framework code to create new testcases. So it is more convenient for vsperf users to write their own internal set of testcases and keep them stored internally, without a need of vsperf framework code change.
    c. documentation was updated: https://gerrit.opnfv.org/gerrit/#/c/56237/
    d. I believe that this topic can be covered by self study by interested vsperf users based on the information provided above.
2. Process involved in deployment and configuration of virtual-switch and Guest-VNFs.
    a. Execution and configuration of vswitches and VNFs is driven by appropriate classes available at **vswtiches** or **vnfs** directories and their controllers available at **core** directory. In case that new functionality is required, then it is possible to use step driven tests to create new or modify existing implementation or to modify or implement new classes and controllers to achieve requested behaviour.
3. Results Management and Upgrade - Adding additional information to results on configured and available resources, test start and end timestamp,
    a. VSPERF collects test results from traffic generator, collector, etc. and these data are used for generation of CSV result file and in case of performance testcase also for generation of final report.
    b. Generic logic of the result reporting is implemented by **TestCase::run_report()** (testcases/testcase.py) and specialized by **IntegrationTestCase::run_report()** (testcases/integration.py) and **PerformanceTestCase::run_report()** (testcases/performance.py).
    c. Implementation of performance report generation and jinja and rst templates used for its generation are available at **tools/report** directory. In case that static content of the report should be modified, please update rst and jinja templates accordingly.
    d. In case of performance tests, a separate MD and RST report files are prepared for each executed test inside result directory. After the execution of all tests, function generate_final_report() is executed from vsperf script. This function will collect all (partial) RST reports into final RST file with results from all executed tests with appropriate header and footer. Note: per test MD report is generated for backward compatibility. It is not used for generation of final report anymore, but it is still used by CI jobs to show TC result summary in "human readable" form. This is a topic for possible improvement/refactoring.
    e. Possible report enhancements:
        i. A new value from the traffic generator should be part of the report - in that case appropriate **send_*** method(s) of affected traffic generators have to be modified to store new value into the **result** dictionary. Please update **ResultsConstants** (core/results/results_constants.py) accordingly and avoid manual specification of new item key. This new value will be automatically added into vsperf output, result CSV file and in case of performance TC also into generated report.
        ii. A new data from collector or new collector should be supported - similar to trafficgen controller also collector controller provides interface to both print and get results. These results are stored inside a dictionary. So new value or data retrieved by new collector will be automatically added into vsperf output and in case of performance TC also into generated report.
        iii. A new OS details should be supported - please update toos/report/report.py with interface to read requested data (see _get_env()) and jinja template to properly type new data into "Test Environment" section (or at desired part of the report).
        iv. A new data collected by VSPERF (e.g. start/end timestamps) should be included - function **generate()** from **tools/report/report.py** has access to whole testcase object through **testcase** argument. It means, that required data should be stored within testcase properties and then it is possible to pass them to jinja template. It is always easier to include new data into existing categories like conf (test configuration), result (data from trafficgen), env (os related) or stats (data from collector). In that case, data are either included in the report automatically or after appropriate update of jinja template. In case that new category of data should be passed and processed by jinja template, then new category has to be added into "tests" dictionary in **generate()** function analogically to existing categories.
4. Testing as part of patch review Process.
    a. Every patch pushed to gerrit must successfully pass a verification CI job. As part of this patch a very basic VSPERF functionality is tested. Check VSPERF CI wiki for additional details. Nevertheless, these tests can detect only severe functionality defects and the fact that verify job pass, doesn't mean that it is well tested.
    b. After the patch is merged, then daily job is executed where a set of performance testcases is executed. However it still doesn't mean that merged changes were correct.
    c. VSPERF project doesn't have any module/unit tests, which would test in detail a functionality of all classes, their methods and various values and boundaries check of their arguments. I've proposed a creation of module/unit test suite as a topic for further improvement. However to prepare and maintain comprehensive set of unit tests would require significant effort.
    d. So it is always required to do a proper testing of the patch by both contributor before patch is pushed for review and also part of the review process should be testing performed by reviewer. Of course, there are changes, which can't be tested by others, e.g. modifications of Traffic Generator support, which might be not be available at OPNFV lab. Nevertheless, proper review should go beyond quick check of the code, but it should consider possible side effects, refactoring or generalization needs and identify testing needs.
5. Configuration parameters - Order of processing, preference, etc.
    a. This is described in detail in VSPERF documentation (see docs/testing/user/userguide/testusage.rst)
    b. documentation was updated: https://gerrit.opnfv.org/gerrit/#/c/56245/
    c. I believe that this topic can be covered by self study by interested vsperf users based on the information provided above.
6. How the code flows doing a simple phy2phy test.
    a. TestCase::run() executes all "elements" involved in the test, i.e. controllers of vswitch, trafficgen and collector
    b. in case of phy2phy, VswitchControllerP2P::setup() will take care about configuration of 2 physical ports and appropriate flows to properly forward traffic.
    c. trafficgen controller will ensure that traffic is generated and results collected by selected traffic generator
    d. after traffic generation ends, all TC components are stopped and retrieved data are reported to the standard output and stored into result directory (csv & rst)
7. Directory structure and what modules go where

Directory Structure

```
 3rd_party                         # placeholder for all scripts, which license is not compatible with
OPNFV licenses
 ci                                         # CI related scripts, i.e. detailed CI job definition and
script for graph generation from results
 conf                                 # all configuration files including a TC definition
    integration                         # integration TC specific configuration files
 core                                 # implementation of controllers (to control vswitch, vnf and
trafficgen configuration and execution);
                                                # implementation of generic classes, which assure
selection of components selected by configuration;
                                                # definition of constants used for results processing
and reporting
 docs                                 # documentation files in RST format; See OPNFVDOCS project for more
details about documentation
                                                # structure, description of local and automatic DOC
generation, etc.
 jobs                                 # not used by VSPERF
 src                                         # external tools required by vsperf are cloned and compiled
here; Directory structure contains a set
                                                # of makefiles, definition of tools' repositories and
versions and in case of DPDK and OVS it contains
                                                # an implementation of basic functions to control DPDK
and OVS. These files should be probably moved
                                                # into vswitch directory.
 systems                                 # installation scripts for various Linux distributions
 testcases                         # implementation of testcases execution logic
 tools                                 # includes a set of functions for interaction with OS; Classes
used for configuration and control of
                                                # external tools are stored in subdirectories (except
VNF & vSwitch)
    collectors                         # wrapper classes for collectors, which monitors and record system
utilization during TC execution
    llc_management                 # wrapper classes for LCC configuration
    load_gen                         # wrapper classes for generators of background load
    opnfvdashboard                 # class used for export of VSPERF CI results into OPNFV result database
    pkt_fwd                                 # class configures testpmd as a packet forwarder for P2P
deployment; It should be generalized
                                                        # to support standard deployments and moved to
vswitches directory
    pkt_gen                                 # wrapper classes for traffic generators
    report                                 # templates and functions used for report generation for
performance testcases
 vnfs                                 # classes for VNF deployment and control (currently only QEMU is
supported)
 vswitches                         # classes for vSwitch configuration and control (OVS with DPDK,
Vanilla OVS and VPP are supported)
 yardstick                         # example testcases for yardstick to demonstrate yardstick and vsperf
possible integration;
                                                # see vsperf documentation for further details
```

I believe that this topic can be covered by self study by interested vsperf users based on the information provided above.

8. Explain the core modules (core folder)

core directory

```
core
 component_factory.py                         # class used for creation of objects, which will control
tools used during TC execution;
                                                        # Component factory is called
from testcase.py to create instances of vSwitch, trafficgen
                                                        # VNFs, etc. It is also called
directly from vsperf script in case of "--mode trafficgen".
```

```
 loader
    loader.py                                       # Support classes which import and provide
information about all implemented vsperf
    loader_servant.py                        # classes for vSwitch, traffcigen, VNF. loadgen, collector
and packet generator.
                                                                        # Loader and LoaderServant
classes are used by testcase.py and vsperf to obtain
                                                                        # data for proper component
factory invocation.
 pktfwd_controller.py                        # Controller which implements P2P scenario with DPDK
testpmd as a packet forwarder.
 results
    results_constants.py                # definition of constants used for results processing and
reporting
    results.py                                       # IResults interface used by traffic controllers
to hold results from traffic generators
 traffic_controller.py                        # Generic implementation of traffic generator controller.
 traffic_controller_rfc2544.py        # RFC2544 specific specialization of traffic controller, which
according to VSPERF/TC
                                                                        # configuration executes RFC2544
Throughput, Back2Back, Continuous stream
                                                                        # or Burst traffic. Required
traffic type must be supported by selected trafficgen.
 traffic_controller_rfc2889.py        # RFC2889 specific specialization of traffic controller, which
according to VSPERF/TC
                                                                        # configuration executes RFC2899
Caching, Forwarding or Learning tests. This is currently
                                                                        # supported only by Spirent
TestCenter traffic generator.
 vnf_controller.py                                # Implementation of VNF controller, which based on TC
configuration (e.g. selected
                                                                        # deployment or TestSteps)
creates, starts and stops all VNFs required for TC execution.
 vswitch_controller_p2p.py                # vSwitch controller which configures vSwitch (currently only
OVS is supported) according
                                                                        # to P2P deployment scenario -
see vsperf doc for additional details about deployments.
 vswitch_controller_pxp.py                # vSwitch controller which configures vSwitch (currently only
OVS is supported) according
                                                                        # to PXP (PVP, PVVP, PVPV, etc.)
deployment scenario - see vsperf doc for additional
                                                                        # details about deployments.
 vswitch_controller_clean.py                # Simple vSwitch controller, which just ensures, that
vswitch and all required processes
                                                                        # (e.g. ovsdb) are properly
executed. This controller is used by step driven testcase
                                                                        # with "clean" deployment, where
vSwitch configuration is driven by TestSteps.
 vswitch_controller_op2p.py                # vSwitch controller used for tunneling protocols, where
vswitch is configured to either
                                                                        # encapsulate or decapsulate the
incoming traffic from the trafficgen to/from selected
                                                                        # tunneling protocol (i.e. GRE,
VxLAN or Geneve). This is used by "op2p" deployment and
                                                                        # it configures unidirectional
traffic for OVS. Trafficgen must support given tunneling
                                                                        # protocol. Currently only IxNet
is supported.
 vswitch_controller_ptunp.py                # vSwitch controller used for testing of tunneling
protocols, where both encapsulation
                                                                        # and decapsulation of selected
tunneling protocol is performed by vSwitch. Currently
                                                                        # only OVS is supported. However
any supported trafficgen can be used.
 vswitch_controller.py                        # Implementation of generic IVswitchController interface
used by specific vswitch
                                                                        # controller implementations.
```

I believe that this topic can be covered by self study by interested vsperf users based on the information provided above.