VSPERF Design

//This is work-in-progress draft of a document that is intended to be included in the vsperf source code under /docs _

Intended Audience

This document is intended to aid those who want to modify the vsperf code. Or to extend it - for example to add support for new traffic generators, deployment scenarios and so on.

Usage

Example Command Lines

`$ ./vsperf -h` of course, lists all the cli options

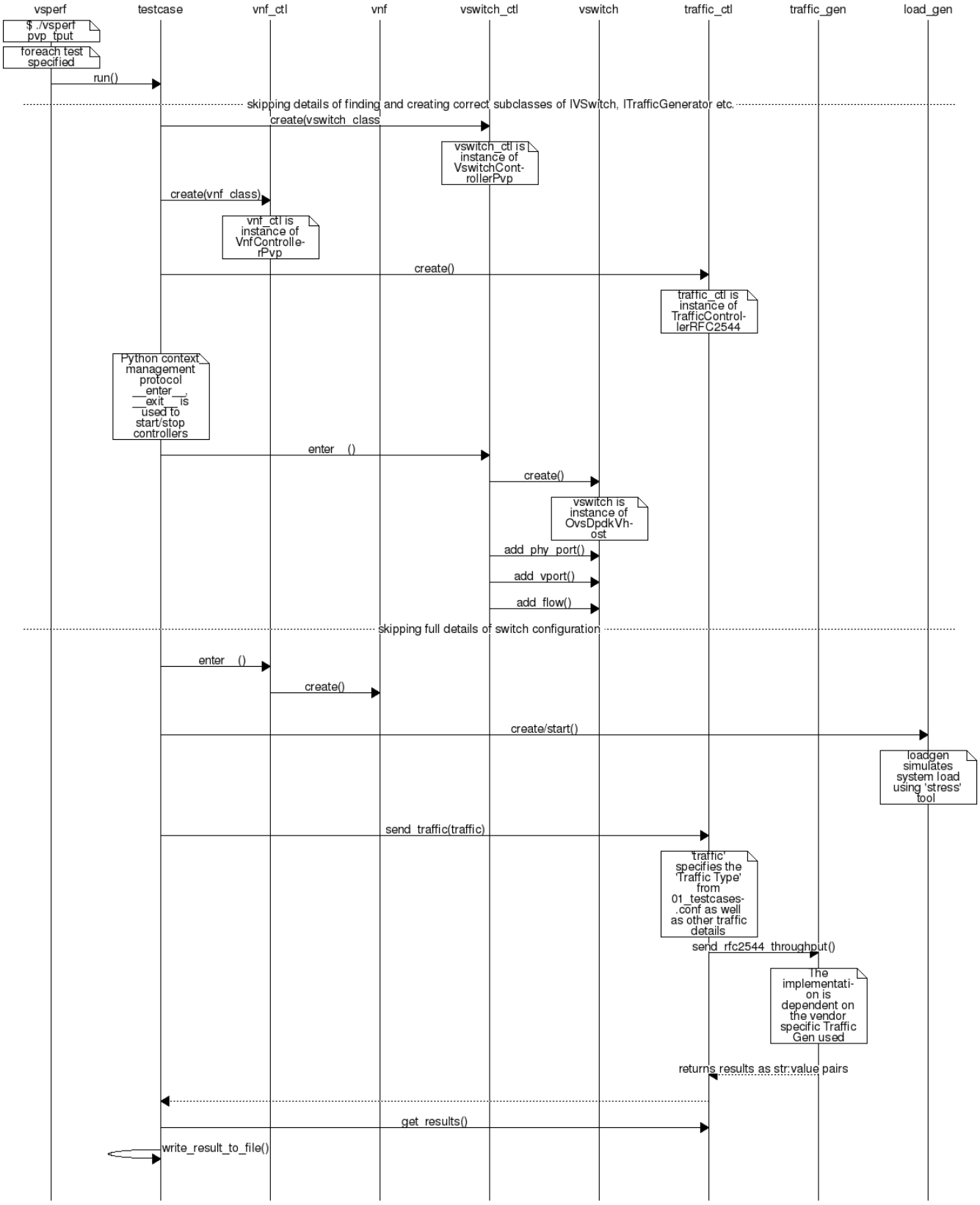
"$ ./vsperf --tests 'tput' " run all tests that have 'tput' in their name - p2p_tput, pvp_tput etc.

"$ ./vsperf --conf-file my_settings.py --tests 'tput' " as above but override default configuration with settings in 'my_settings.py'. This is useful as modifying configuration directly in the configuration files in conf/NN_*.py shows up as changes under git source control

"$ ./vsperf --test-params 'rfc2544_duration=10;rfc2544_trials=1;packet_sizes=64' --tests 'pvp_tput' " override specific test parameters. Useful for shortening the duration of tests for development purposes.

Typical Test Sequence

This is a typical flow of control for a test.



This diagram was generated using mscgen.

Configuration =

The conf package contains the configuration files (*.conf) for all system components, it also provides a ``settings`` object that exposes all of these settings.

Settings are not passed from component to component. Rather they are available globally to all components once they import the conf package.

from conf import settings
...
log_file = settings.getValue('LOG_FILE_DEFAULT')

Settings files (*.conf) are valid python code so can be set to complex types such as lists and dictionaries as well as scalar types:

```
  first_packet_size = settings.getValue('PACKET_SIZE_LIST')[0]
```

{group3}

Precedence

Configuration files follow a strict naming convention that allows them to be processed in a specific order. All the .conf files are named ``NN_name.conf``, where NN is a decimal digit. The files are processed in order from 00_name.conf to 99_name.conf so that if the name setting is given in both a lower and higher numbered conf file then the higher numbered file is the effective setting as it is processed after the setting in the lower numbered file.

The values in the file specified by ``--conf-file`` takes precedence over all the other configuration files and does not have to follow the naming convention.

Other Configuration

``conf.settings`` also loads configuration from the command line and from the environment.

VM, vSwitch, Traffic Generator Independence

VSPERF supports different vSwithes, Traffic Generators and VNFs by using standard object-oriented polymorphism:

- Support for vSwitches is implemented by a class inheriting from IVSwitch.
- Support for Traffic Generators is implemented by a class inheriting from ITrafficGenerator.
- Support for VNF is implemented by a class inheriting from IVNF.

By dealing only with the abstract interfaces the core framework can support many implementations of different vSwitches, Traffic Generators and VNFs.

IVSwitch

```
class IVSwitch:
    start(self)
    stop(self)
    add_switch(switch_name)
    del_switch(switch_name)
    add_phy_port(switch_name)
    add_vport(switch_name)
    get_ports(switch_name)
    del_port(switch_name, port_name)
    add_flow(switch_name, flow)
    del_flow(switch_name, flow=None)
```

IVnf

```
    start(memory, cpus,
          monitor_path, shared_path_host,
          shared_path_guest, guest_prompt)
    stop()
    execute(command)
    wait(guest_prompt)
    execute_and_wait (command)
```

ITrafficGenerator

```
    connect()
    disconnect()

    send_burst_traffic(traffic, numpkts, time, framerate)

    send_cont_traffic(traffic, time, framerate)
    start_cont_traffic(traffic, time, framerate)
    stop_cont_traffic(self):

    send_rfc2544_throughput(traffic, trials, duration, lossrate)
    start_rfc2544_throughput(traffic, trials, duration, lossrate)
    wait_rfc2544_throughput(self)

    send_rfc2544_back2back(traffic, trials, duration, lossrate)
    start_rfc2544_back2back(traffic, , trials, duration, lossrate)
    wait_rfc2544_back2back()
```

Note send_xxx() blocks whereas start_xxx does not and must be followed by a subsequent call to wait_xxx().

Controllers

Controllers are used in conjunction with abstract interfaces as way of decoupling the control of vSwtiches, VNFs and TrafficGenerators from other components.

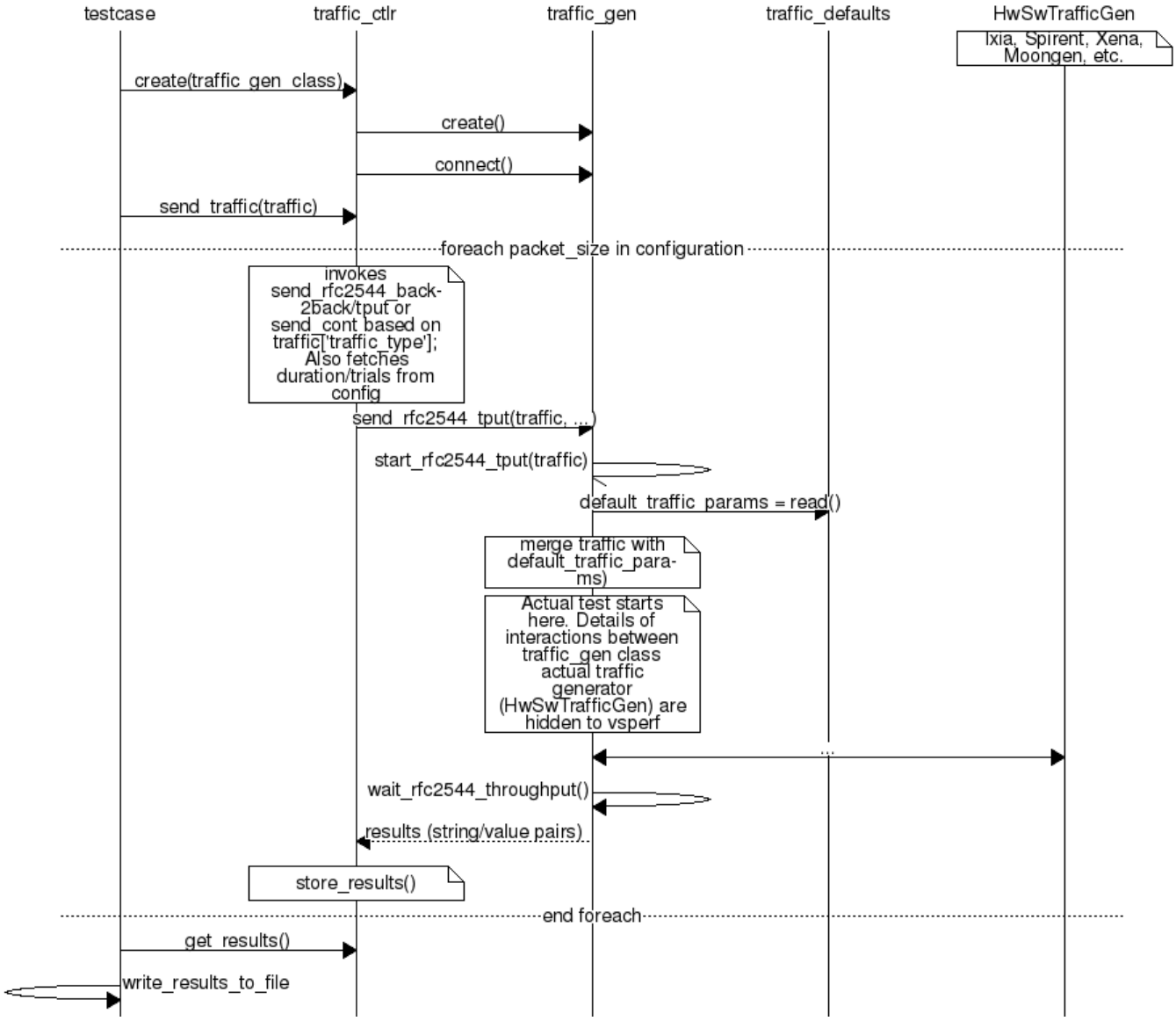
The controlled classes provide basic primitive operations. The Controllers sequence and co-ordinate these primitive operation in to useful actions. For instance the vswitch_controller_PVP can be used to bring any vSwitch (that implements the primitives defined in IVSwitch) into the configuration required by the Phy-to-Phy Deployment Scenario.

In order to support a new vSwitch only a new implementation of IVSwitch needs be created for the new vSwitch to be capable of fulfilling all the Deployment Scenarios provided for by existing or future vSwitch Controllers.

Similarly if a new Deployment Scenario is required it only needs to be written once as a new vSwitch Controller and it will immediately be capable of controlling all existing and future vSwitches in to that Deployment Scenario.

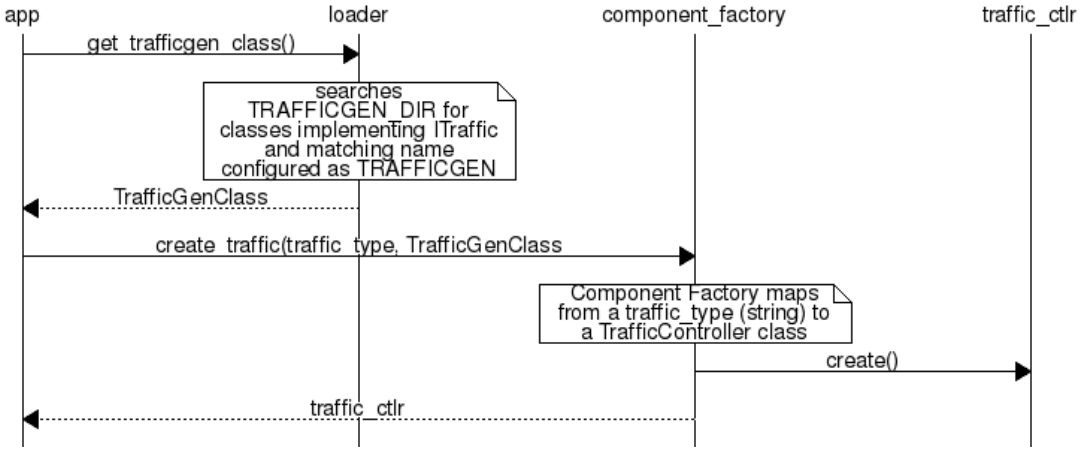
Similarly the Traffic Controllers can be used to co-ordinate basic operations provided by implementers of ITrafficGenerator to provide useful tests. Though traffic generators generally already implement full test cases i.e. they both generate suitable traffic and analyse returned traffic in order to implement a test which has typically been predefined in an RFC document. However the Traffic Controller class allows for the possibility of further enhancement - such as iterating over tests for various packet sizes or creating new tests.

Traffic Controller's Role



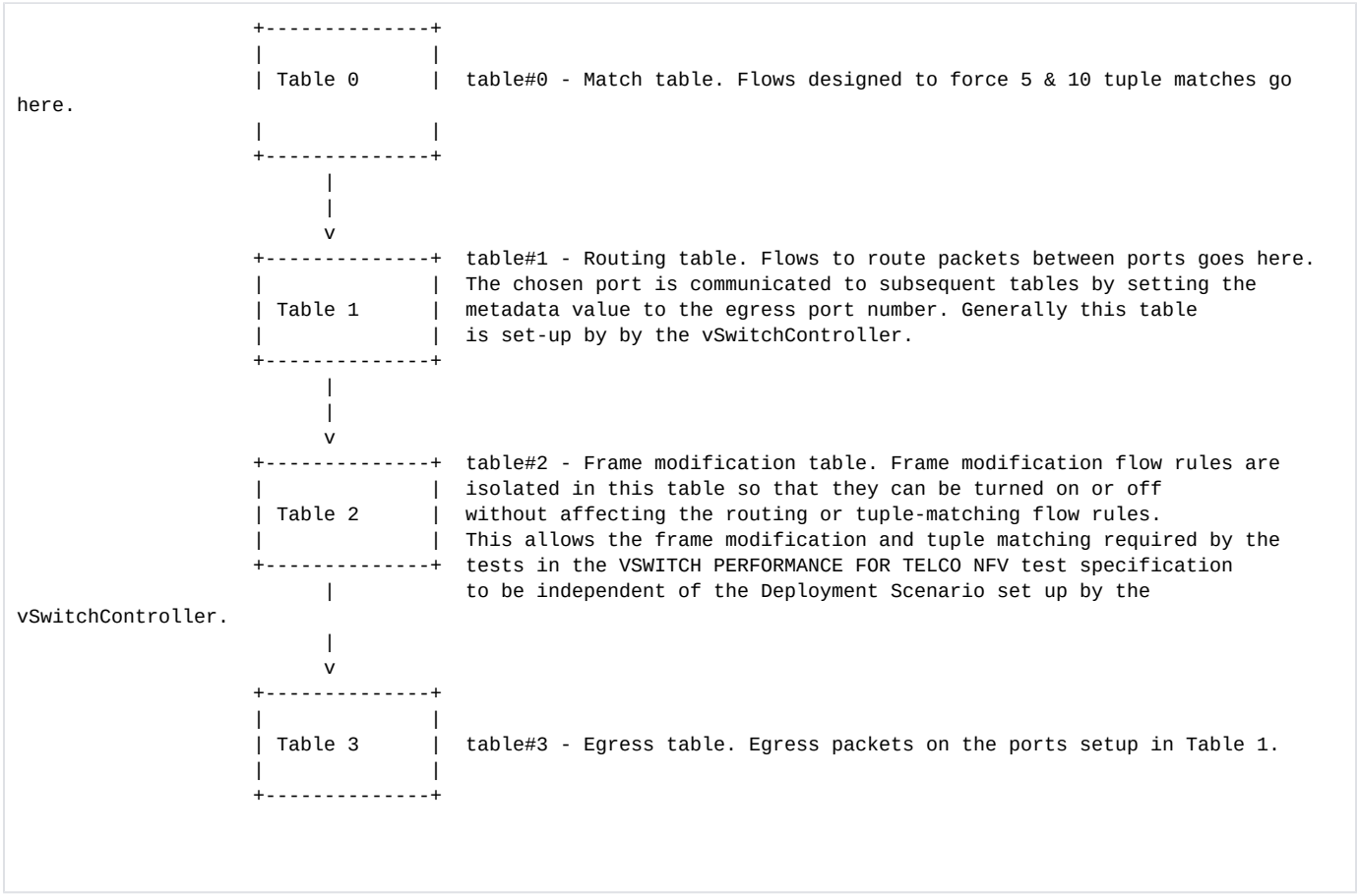
Loader & Component Factory

The working of the Loader package (which is responsible for *finding* arbitrary classes based on configuration data) and the Component Factory which is responsible for *choosing* the correct class for a particular situation - e.g. Deployment Scenario can be seen in this diagram.



Routing Tables

Vsperf uses a standard set of routing tables in order to allow tests to easily mix and match Deployment Scenarios (PVP, P2P topology), Tuple Matching and Frame Modification requirements.



Appendix

Sequence Diagram Text for mscgen

mscgen is available here (http:_www.mcternan.me.uk/mscgen/)

```
msc {

  #Options

  hscale = "2.0";

  #Entities
  vsperf, testcase, vnf_ctl, vnf, vswitch_ctl, vswitch, traffic_ctl, traffic_gen, load_gen;

  #Arcs
  vsperf note vsperf [ label = "$ ./vsperf pvp_tput" ];
  vsperf note vsperf [ label = " foreach test specified" ];
  vsperf => testcase [ label="run()" ];
  --- [ label = " skipping details of finding and creating correct subclasses of IVSwitch, ITrafficGenerator
etc." ];
  testcase => vswitch_ctl [ label="create(vswitch_class" ];
  vswitch_ctl note vswitch_ctl [ label="vswitch_ctl is instance of VswitchControllerPvp"];
  testcase => vnf_ctl [ label="create(vnf_class)" ];
  vnf_ctl note vnf_ctl [ label="vnf_ctl is instance of VnfControllerPvp"];
  testcase => traffic_ctl [ label="create()" ];
  traffic_ctl note traffic_ctl [ label="traffic_ctl is instance of TrafficControllerRFC2544"];
  |||;
  testcase note testcase [ label="Python context management protocol __enter__, __exit__ is used to start/stop
controllers"];
  testcase => vswitch_ctl [ label="__enter__()"];
  vswitch_ctl => vswitch [ label ="create()"];
  vswitch note vswitch [label="vswitch is instance of OvsDpdkVhost"];
  vswitch_ctl => vswitch [ label="add_phy_port()" ];
  vswitch_ctl => vswitch [ label="add_vport()" ];
  vswitch_ctl => vswitch [ label="add_flow()" ];
  --- [ label = " skipping full details of switch configuration " ];
  |||;
  testcase => vnf_ctl [ label="__enter__()"];
  vnf_ctl => vnf [ label="create()"];
  |||;
  testcase => load_gen [ label="create/start()" ];
  load_gen note load_gen [ label="loadgen simulates system load using 'stress' tool "];
  |||;
  testcase => traffic_ctl [ label="send_traffic(traffic)" ];
  traffic_ctl note traffic_ctl [ label="'traffic' specifies the 'Traffic Type' from 01_testcases.conf as well
as other traffic details" ];
  traffic_ctl => traffic_gen [label="send_rfc2544_throughput()"];
  traffic_gen note traffic_gen [label="The implementation is dependent on the vendor specific Traffic Gen
used"];
  |||;
  traffic_ctl << traffic_gen [label="returns results as str:value pairs"];
  testcase << traffic_ctl;
  testcase => traffic_ctl [label="get_results()"];
  testcase => testcase [label="write_result_to_file()"];
  |||;
  |||;
}
```