DPDK Test Suite

Release 17.08.0

Nov 17, 2017

Contents

CHAPTER 1

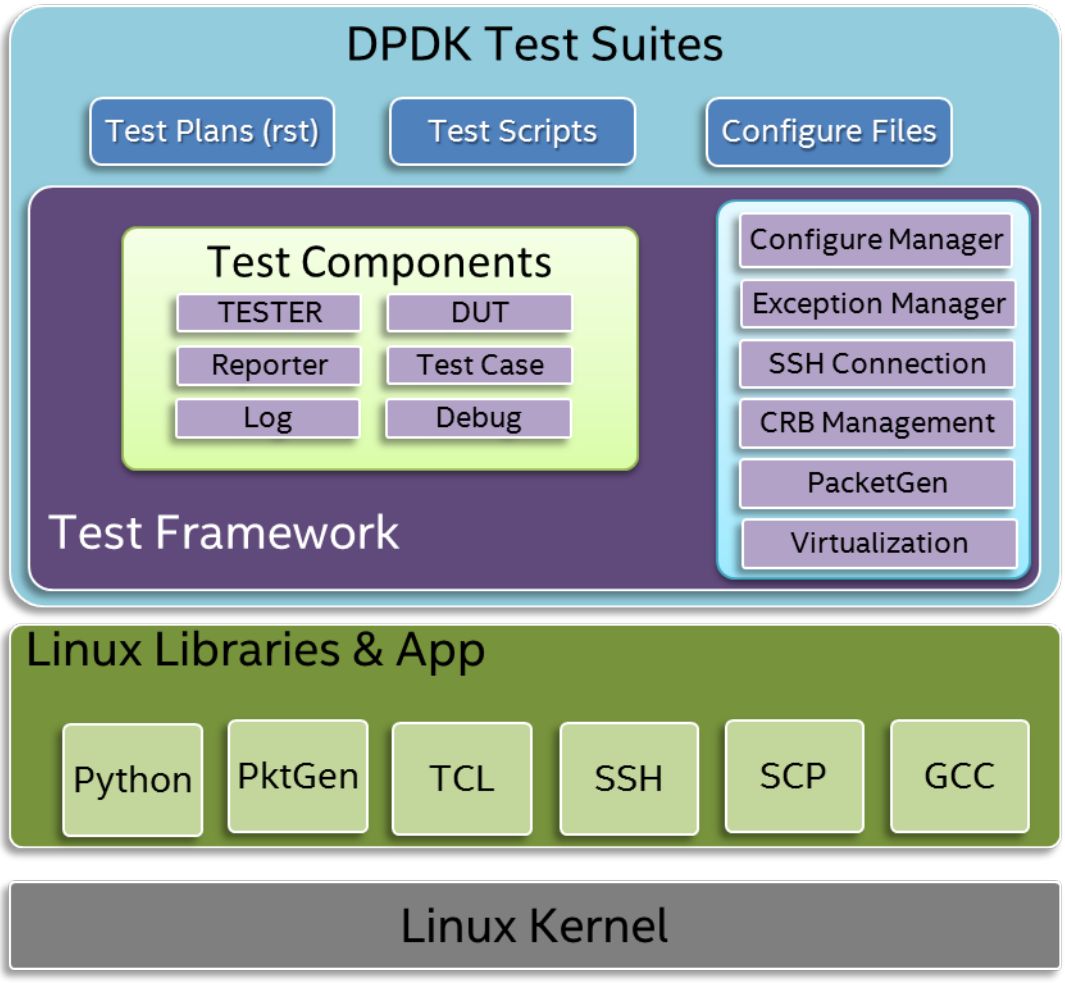
Introduction

This document describes how to install and configure the Data Plane Development Kit Test Suite (DPDK Test Suite) in a Linux environment. Users can refer this document to enable this test infrastructure in their environment and don't need go deeply with too much details about this framework. DPDK Test Suite is an automation test tool for DPDK software, a python-base library. It can run on the tester machine, and communicate/manage DUT by SSH connection. DTF supports different kind of traffic generators, including DPDK-based PacketGen, third-party professional tester equipment (IXIA®). Data Plane Development Kit Test Suite (DPDK Test Suite) includes one set of test cases and DPDK generic test framework. DPDK Test Suite provides test example, references and framework for open source community. Based on DPDK Test Suite, everyone can develop their test plan, automation script and configuration for own features and platform. In addition, DPDK Test Suite provides a solution to allow that DPDK developers contribute their function test to certify their patch integration. It only requires limitation effort to maintain test cases once merged into DPDK Test Suite. Everyone can utilize DPDK Test Suite to measure performance and functionality for features.

Please see DPDK Test Suite architecture in the following figures:

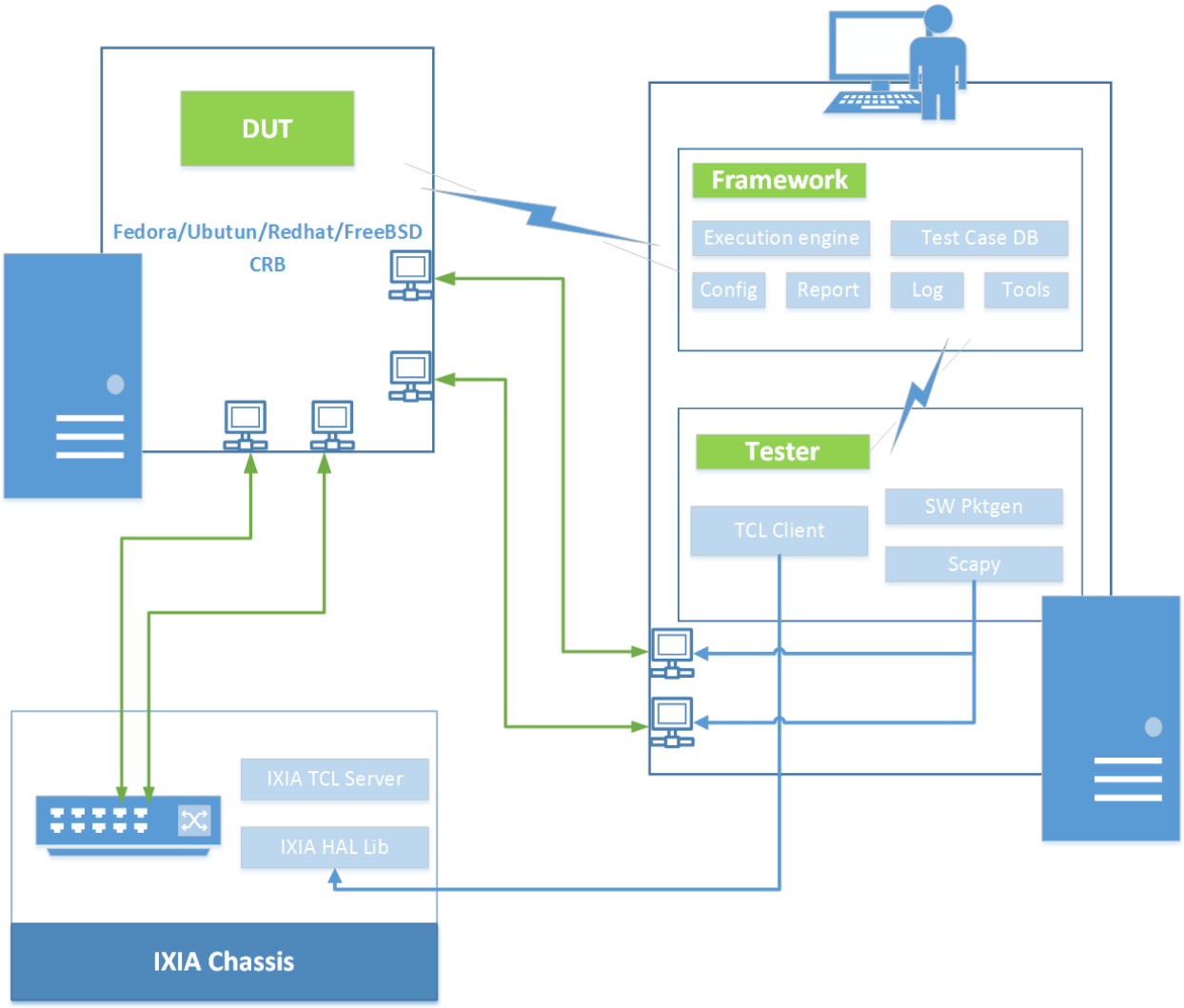
As generic test framework, DPDK Test Suite provides the following functions:

- Able to work with DUT (Device Under Test), which installed Fedora, Ubuntu, WindRiver, FreeBSD, RedHat and SUSE.
- Support virtualization hypervisors like Xen and Qemu.
- Support both software and hardware traffic generators, including Scapy, DPDK-based PacketGen and IXIA traffic generator, even third party packet generator via TCL or Python library.
- Provide configure files to customize test suite and test cases to run under DUT.
- Provide debug and log functionalities for tracking test cases execution process.
- Support to output test result by excel, log text file, etc.
- DPDK Test Suite provides one set of basic library to manage tester, DUT, test case, exception, generate report, and configure test plan by user configure files. It's easy to develop user-defined test suite and test plan by self, and these test cases are able to integrate into DPDK Test Suite.
- With this test framework, user can automatically identify network topology, and easy to configure/deploy environment for DUT and tester, and provides flexibility to scale test capability by configuration.


DPDK Test Suites
Test Plans (rst)
Test Scripts
Configure Files
Test Components
TESTER
DUT
Reporter
Test Case
Log
Debug
Test Framework
Configure Manager
Exception Manager
SSH Connection
CRB Management
PacketGen
Virtualization
Linux Libraries & App
Python
PktGen
TCL
SSH
SCP
GCC
Linux Kernel

DPDK Test Suite environment includes DUT (Device under Test), Tester and packet generator. DPDK software will deployed and run on DUT. DPDK Test Suite should run on the Tester.

Please see architecture in the following figures:



This architecture provides automatically mechanism to manage tester, DUT and packet generators, and remove dependency between test script and test environment/hardware. It defines one abstraction layer for DPDK Test Suite, and provides extensibility to add more test script. In the DPDK Test Suite Test Framework, it provides the following modules to help to manage device, platform, configure and test results.

File Name	Description
dts.py	Main Application for DPDK Test Suite
main.py	Test script to parse input parameter
dut.py	Setup device under test including tool chain, IP address
tester.py	Provide API to setup tester environment including IP, port, etc.
project_dpdk.py	Provide running environment for DPDK.
exception.py	Manage User-defined exceptions used across the framework
test_cases.py	Provide a base class for creating DPDK Test Suite test cases
logger.py	Deal with different log files to record event or message

Table 1.1 – continued from previous page

serializer.py	Provide wrapper class to manage temporary variables during execution
settings.py	Setting for default network card and its identifiers supported by the framework
utils.py	Provide shared simple functions like IP address covertion and mask creation
ssh_connection.py	Create session to host, implement send_expect and copy function
ssh_pexpect.py	Handle ssh sessions between tester and DUT, Implement send_expect function to send command and get outp
pmd_output.py	Module for get all statics value by port in testpmd
rst.py	Generate Rst Test Result Report
stats_reporter.py	Simple text file statistics generator
test_result.py	Generic result container. Useful to store/retrieve results during a DTF execution
excel_reporter.py	Excel spreadsheet generator
plotgraph.py	Generate graphs for each test suite
plotting.py	Generate Plots for performance test results
etgen.py	Software packet generator
ixia_buffer_parser.py	Helper class that parses a list of files containing IXIA captured frames extracting a sequential number on then
ixiaDCB.tcl	Third party Library which provided by IXIA, used to configure IXIA tester
ixiaPing6.tcl	Third party Library which provided by IXIA, used to ping IXIA tester
IxiaWish.tcl	Third party Library which provided by IXIA, set up TCL environment to use correct multiversion-compatible
texttable.py	Third party Library , create simple ASCII tables
qemu_kvm.py	Provide functionality for management and monitoring QEMU hypervisor
qemu_libvirt.py	Provide functionality for usage of libvirt library
virt_base.py	Base class for virtual machine, supply basic management functions
virt_dut.py	Generate instance for virtual machine, usage model is like DUT
virt_resource.py	Provide resource management for virtual machine
virt_scene.py	Generate virtualization scenario based on configuration file

Beside Framework tool, DPDK Test Suite also defines one set of test cases. It includes basic test suite to verify basic functionality of DPDK library. These test script provides example and reference. Everyone can develop their test cases, verify their features functionality, and commit generic test report to maintainer. However, user-defined test cases, plan and script must follow DPDK Test Suite standard including code standard, naming conventions, configure format, rst test plan, API.

Please see test cases, which included in the DPDK compliance test suites:

Test Suite	Descriptions
Command line	Define a demo example of command line interface in RTE
hello_world	Print a `helloworld` message on every enabled logic core.
Multi process	Demonstrates the basics of sharing information between DPDK processes.
Timer	Shows how timer can be used in a RTE application.
Black-list/WhiteList	Tests Whitelist/Blacklist Features by Poll Mode Drivers.
check-sum_offload	Tests RX/TX L3/L4 Checksum offload features by Poll Mode Drivers
jumbo_frame	Tests jumbo frames features by Poll Mode Drivers
testpmd	Provides benchmark tests for the Intel Ethernet Controller (Niantic) Poll Mode Driver.
l2fwd	Provides a basic packet processing application using DPDK. It is a layer-2 (L2) forwarding application which takes traffic from a single RX port and transmits it with few modification on a single TX port.
L3fwd	Verifies Layer-3 Forwarding results using `l3fwd` application.
IP fragment	Verifies IPv4 fragmentation using `ipv4_frag` application.
Flow direction	Verifies the Flow Director feature of the Intel 82599 10GbE Ethernet Controller
link_flowctrl	Verifies Ethernet Link Flow Control Features by Poll Mode Drivers
ieee1588	Tests the IEEE1588 Precise Time Protocol offload supported in Poll Mode Drivers.

CHAPTER 2

System Requirements

The board assigned to be tester should be installed the latest Fedora distribution for easily installed DPDK Test Suite required python modules. Tester board needs plug at least 2 x Intel® 82599 (Niantic) NICs (2x 10GbE full duplex optical ports per NIC) in the PCI express slots, then connect these four Niantic ports to the DUT board and make sure the link has been started up and speed is 10000Mb/s.

Beside the four Niantic ports, tester and DUT should also have one interface connected to the same intranet. So that they can be accessed by each other from local IP address.

Note: Firewall should be disabled that all packets can be accepted by Niantic Interface.

```
systemctl disable firewalld.service
```

2.1 Setup Tester Environment

Note: Please install the latest Fedora distribution on the tester before install DPDK Test Suite on tester. Currently we recommend Fedora 20 for tester. The setup instruction and required packages may be different on different operation systems.

To enable tester environment, you need to install script language, tool chain and third party packet generator, etc.

Please follow the guidance to finish install as the below section.

2.1.1 SSH Service

Since DPDK Test Suite Tester communicates with DUT via SSH, please install and start sshd service in your tester.

```
yum install sshd                # download / install ssh software
systemctl enable sshd.service   # start ssh service
```

For create authorized login session, user needs to generate RSA authentication keys to ssh connection.

Please use the following commands:

```
ssh-keygen -t rsa
```

2.1.2 TCL Language Support modules

Since some third party tools required TCL (Tool Command Language) supports, please install TCL package to control and connect third party package generator. (For example, third-party professional tester IXIA required TCL support)

```
yum install tcl                 # download / install ssh software
```

2.1.3 Install Third Party python modules

With third party module, DPDK Test Suite is able to export test result as MS Excel file or graphs. To support this feature, please install the following modules in the tester. Python Module "xlwt": this module is used to generate spreadsheet files which compatible with MS Excel 97/2000/XP/2003 XLS files. Python Module "numpy": this module provides method to deal with array-processing test results. Python Module "pexpect": this module provides API to automate interactive SSH sessions. Python Module "docutils": Docutils is a modular system for processing documentation into useful formats, such as HTML, XML, and LaTeX. Python Module "pcapy": Pcapy is a Python extension module that interfaces with the libpcap packet capture library. Pcapy enables python scripts to capture packets on the network. Python Module "xlrd": Xlrd is a Python module that extracts data from Excel spreadsheets.

Please see installation instruction as the following:

```
yum install python-xlwt
yum install python-pexpect
yum install numpy
yum install python-docutils
yum install pcapy
yum install python-xlrd
```

2.1.4 Setup and configure Scapy

Scapy is a powerful interactive packet manipulation program. It is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more. It can easily handle most classical tasks like scanning, tracerouting, probing, unit tests, attacks or network discovery.

DTCS uses python module scapy to forge or decode packets of a wide number of protocols, send them over the wire, capture them, and analyse the packets.

```
yum install scapy
```

Fedora20 default kernel will strip vlan header automatically and thus it will cause that scapy can't detect vlan packet normally. To solve this issue, we need to configure scapy use libpcap which is a low-level network traffic monitoring tool.

```
vim /usr/lib/python2.7/site-packages/scapy/config.py  # open configure python files
use_pcap = True                                       # find use_pcap and set it to␣
↪True
```

2.1.5 Install DPDK Test Suite on tester

After configure environment, we need to install DPDK Test Suite into tester. First of all, download the latest DPDK Test Suite code from remote repo.

```
[root@tester ~]#  git clone http://dpdk.org/git/tools/dts
[root@tester ~]#  cd dts
[root@tester dts]#  ls
[root@tester dts]# conf  dep  doc  dts  executions  framework  nics  output  test_
↪plans  tests  tools
```

High Precision Timer (HPET) must be enabled in the platform BIOS if the HPET is to be used. Otherwise, the Time Stamp Counter (TSC) is used by default. The user can then navigate to the HPET option. On the Crystal Forest platform BIOS, the path is: **Advanced -> PCH-IO Configuration -> High Precision Timer**

The DPDK Test Suite is composed of several file and directories:

- dts: Main module of DPDK Test Suite suite
- exectution.cfg: configuration file of DPDK Test Suite suite
- framework: folder with dts framework modules
- nics: folder with different network device modules
- output: folder which contain running log files and result files
- test_plans: folder with rst files which contain the description of test case
- tests: folder with test case scripts

2.2 Setup Target Environment

This section describes how to deploy DPDK Test Suite packages into DUT target.So far, DPDK Test Suite supports the following OS on DUT:

- Fedora18/19/20/23/24/25
- Ubuntu12.04/14.04/16.04
- WindRiver 6.0/7.0
- FreeBSD 10
- RedHat 6.5/7.0/7.3
- SUSE 11

Before run DPDK Test Suite on target, we need to configure target environment, it includes BIOS setting, Network configure, compiler environment, etc.

2.2.1 BIOS setting Prerequisite

In general, enter BIOS Menu by pressing F2 while the platform is starting up.

Note: It is strongly recommended to use DPDK with the latest generation of Intel platforms and processors.

The High Precision Timer (HPET) must be enabled in the platform BIOS if the HPET is to be used. Otherwise, the Time Stamp Counter (TSC) is used by default. The user can then navigate to the HPET option. On the Crystal Forest platform BIOS, the path is:

Advanced -> PCH-IO Configuration -> High Precision Timer

Enhanced Intel SpeedStep® Technology must be disabled in the platform BIOS, to ensure the processor voltage and core frequency do not change. This is necessary for consistency of data. On the Crystal Forest platform BIOS the path is:

Advanced -> Processor Configuration -> Enhanced Intel SpeedStep

Processor state C3 and C6 must be disabled for performance measure too. On the Crystal Forest platform BIOS, the path is:

Advanced -> Processor Configuration -> Processor C3 Advanced -> Processor Configuration -> Processor C6

Hyper-Threading Technology must be enabled. On the Crystal Forest platform BIOS, the path is:

Advanced -> Processor Configuration -> Intel® Hyper-Threading Tech

If the platform BIOS has any particular performance option, select the settings for best performance.

2.2.2 DPDK running Prerequisite

Compilation of DPDK need GNU maker, gcc, libc-header, kernel header installed. For 32-bit compilation on 64-bit systems, there're some additional packages required. For Intel® C++ Compiler (icc) additional libraries may be required. For more detail information of required packets, please refer to Data Plane Development Kit Getting Started Guide.

The DPDK igb_uio kernel module depends on traditional Linux kernel uio support to operate. Linux traditional uio support may be compiled as a module, so this module should be loaded using the modprobe program. Kernel must support the allocation of hugepages. Hugepage support is required for the large memory pool allocation used for packet buffers. By using hugepage allocations, performance will be improved since only fewer pages are needed, and therefore less Translation Lookaside Buffers (TLBs, high speed translation caches), which reduce the time it takes to translate a virtual page address to a physical page address. Without hugepages, high TLB miss rates would occur, slowing performance.

The DPDK igb_uio kernel module depends on traditional Linux kernel `uio` support to operate. Linux traditional `uio` support may be compiled as a module, so this module should be loaded using the `modprobe` program. Kernel must support the allocation of hugepages. Hugepage support is required for the large memory pool allocation used for packet buffers. By using hugepage allocations, performance will be improved since only fewer pages are needed, and therefore less Translation Lookaside Buffers (TLBs, high speed translation caches), which reduce the time it takes to translate a virtual page address to a physical page address. Without hugepages, high TLB miss rates would occur, slowing performance.

For more detail information of system requirements, also refer to Data Plane Development Kit Getting Started Guide.

2.3 Authorized login session

In DPDK Test Suite, support communication be established based on authorized ssh session. All ssh connection to each other will skip password interactive phase if remote server has been authorized.

In tester, you can use tool ssh-copy-id to save local available keys on DUT, thus create authorise login session between tester and DUT. By the same way, you can create authorise login session between tester and itself.

```
ssh-copy-id -i "IP of DUT"
ssh-copy-id -i "IP of tester"
```

In DUT, You also can use tool ssh-copy-id to save local available keys in tester, thus create authorise login session between DUT and tester.

```
ssh-copy-id -i "IP of Tester"
```

CHAPTER 3

Configuring DPDK Test Suite

3.1 DPDK Test Suite command line

DPDK Test Suite supports multiple parameters and these parameters, which will select different of working mode of test framework. In the meantime, DPDK Test Suite can work with none parameter, then every parameter will set to its default value. For Example, please see specific usage, you can get these information via DPDK Test Suite help messages.

```
usage: main.py [-h] [--config-file CONFIG_FILE] [--git GIT] [--patch PATCH]
               [--snapshot SNAPSHOT] [--output OUTPUT] [-s] [-r] [-p PROJECT]
               [--suite-dir SUITE_DIR] [-t TEST_CASES] [-d DIR] [-v]
               [--virttype VIRTTYPE] [--debug] [--debugcase] [--re_run RE_RUN]
               [--commands COMMANDS]
```

DPDK Test Suite supports the following parameters:

parameter	description	Default Value
-h,–help	show this help message and exit	
–config-file CONFIG_FILE	configuration file that describes the test cases, DUTs and targets	execution.cfg
–git GIT	Indicate git label to use as input	None
–patch PATCH	apply a patch to the package under test	None
–snapshot SNAPSHOT	snapshot .tgz file to use as input	dep/dpdk.tar.gz
–output OUTPUT	Output directory where DPDK Test Suite log and result saved	output
-s –skip-setup	Skips all possible setup steps done on both DUT and tester boards.	
-r	Reads the DUT configuration from a cache. If not specified, the DUT configuration will be calculated as usual and cached.	
-p PROJECT –project PROJECT	Specify that which project will be tested dpdk	
-t TEST_CASES [TEST_CASES ...] –test-cases TEST_CASES [TEST_CASES ...]	Executes only the followings test cases	None
-d DIR –dir DIR	Output directory where dpdk package is extracted	dpdk
–suite-dir	Test suite directory where test suites will be imported	tests
-v, –verbose	Enable verbose output, all log shown on screen	
–virttype	Set virtualization hypervisor type. Support kvm and libvirtd by now.	
–debug	Enable debug mode, running process can be interrupted and enter debug mode.	
–debugcase	Enter into debug mode before running every test case.	
–re_run TIMES	Rerun failed test cases for stable result	0
–commands COMMANDS	Run self assigned commands at different stages of exection. Format is [commands]:dut|tester:pre-init|post-init:check|ignore E.g. [/root/setup.sh]:dut:pre-init:check	

Please see more information about some critical parameters as the following:

–config-file

DPDK Test Suite configure file defines some critical parameters. It must contain the DUT CRB IP address, wish list of test suites, DPDK target information and test mode parameter.

–git

When we use –-git parameter, DPDK Test Suite will clone the source code from dpdk.org git repository, then checkout branch specified by the parameter.

–patch

DPDK Test Suite also support apply specified patch list by –patch parameter before build DPDK packet.

–skip-setup

If DPDK source code doesn't changed, you can use –skip-setup to skip unzip and compilation of DPDK source code, just reuse original source code.

–project

Parameter –-project can load customized project model and do its own project initialization.

–output

If we perform multiple validation at the same time, result files in output folder maybe overwritten. Then we can use –-output parameter to specify the output folder and save execution log and result files. This option will make sure that all test result will be stored in the different excel files and rst files, doesn't conflict each other.

Note: The current working folder of DPDK Test Suite is "DTS root directory" and default output folder is "output"

–t

You can only run some specified cases in test suites.

We can use parameter –-t to determine those cases.

–suite-dir

DPDK Test Suite support load suites from different folders, this will be helpful when there's several projects existing in the same time.

–verbose

DPDK Test Suite support verbose mode. When enable this mode, all log messages will be output on screen and helpful for debug.

–virttype

Choose virtualization hypervisor type. By now this configuration is useless.

–debug

DPDK Test Suite support debug mode. After keyboard ctrl+c message to DTS process, will run into this mode. User can do further debug by attached to sessions or call pdb module by interact interface.

Debug interact support commands as below:

```
help(): show help message
list(): list all connected sessions
connect(name): connect to session directly
exit(): exit dts
quit(): quit debug mode and into noraml mode
debug(): call python debug module
```

–debugcase

Another approach to run into debug mode. With this option on, DTS will hang and wait for user command before execution of each test case.

–re_run

Some cases may failed due to miscellaneous packets, rerun those test cases can generate the stable result.

–commands

Allow user specify some commands which can be executed on DUT or Tester in the process of DPDK Test Suite preparation.

3.2 DPDK Release Preparation

Firstly, you need to download the latest code from dpdk.org, then archive and compress it into zipped file. After that, please move this zipped file to DPDK Test Suite "dep" folder. Once launch test framework, DPDK Test Suite will copy this zipped file to root folder on DUT. Finally this source code zip file will be unzipped and built.

```
[root@tester dts]#  ls
[root@tester dts]#  conf dep doc dts executions framework nics output test_plans␣
↪tests tools
```

If enables patch option, DPDK Test Suite will also make patch the unzipped folder and compile it.

```
[root@tester dts]# ./dts --patch 1.patch --patch 2.patch
```

3.3 Create your own execution configuration

First of all, you must create a file named execution.cfg as below.

```
[Execution1]
crbs=192.168.1.1
test_suites=
hello_world,
l2fwd
targets=
x86_64-default-linuxapp-gcc,
parameters=nic_type=niantic:func=true
scenario=pf_passthrough
```

- crbs: IP address of the DUT CRB. The detail information of this CRB is defined in file crbs.py.
- test_suites: defines list of test suites, which will plan to be executed.
- targets: list of DPDK targets to be tested.
- parameters: you can define multiple keywords
- scenario: Senario of DPDK virtualization environment for this execution.
  - **nic_type** [is the type of the NIC to use. The types are defined in the file settings.py.] There's one special type named as **cfg**, which mean network information will be loaded from file.
  - func=true run only functional test
  - perf=true run only performance test

Then please add the detail information about your CRB in **conf/crbs.conf** as follows:

```
[192.168.1.1]
dut_ip=192.168.1.1
dut_user=root
dut_passwd=
os=linux
tester_ip=192.168.1.2
tester_passwd=
ixia_group=group1
channels=4
bypass_core0=True
```

Item	description
dut_ip	IP address of DUT
dut_user	UserName of DPDK Test Suite used to login into DUT
dut_passwd	Password of DPDK Test Suite used to login into DUT
os	Distribution of operation system
tester_ip	IP address of tester
tester_passwd	Password to login into Tester
ixia_group	IXIA group name for DUT
channels	number of memory channels for DPDK EAL
bypass_core0	skip the first core when initialize DPDK

If you need to configure network topology, please add it in **conf/ports.cfg**, e.g.:

```
[192.168.1.1]
ports =
    pci=0000:06:00.0,peer=0000:81:00.0;
    pci=0000:06:00.1,peer=0000:81:00.1;
    pci=0000:08:00.0,peer=IXIA:1.1;
    pci=0000:08:00.1,peer=IXIA:1.2;
```

Item	description
pci	Device pci address of DUT
peer	Device pci address of Tester port which connected to the DUT device

3.4 Launch DPDK Test Suite

After we have prepared the zipped dpdk file and configuration file, just type the followed command "./dts", it will start the validation process.

DPDK Test Suite will create communication sessions first.

```
DUT 192.168.1.1
INFO: ssh root@192.168.1.1
INFO: ssh root@192.168.1.1
INFO: ssh root@192.168.1.2
INFO: ssh root@192.168.1.2
```

Then copy snapshot zipped dpdk source code to DUT.

```
DTS_DUT_CMD: scp dep/dpdk.tar.gz root@192.168.1.1:
```

Collect CPU core and network device information of DUT and tester.

Automatically detect the network topology of DUT and tester.

```
DTS_TESTER_RESULT: DUT PORT MAP: [4, 5, 6, 7]
```

Build dpdk source code and then setup the running environment.

```
DTS_DUT_CMD: make -j install T=x86_64-native-linuxapp-gcc
DTS_DUT_CMD: awk '/Hugepagesize/ {print $2}' /proc/meminfo
DTS_DUT_CMD: awk '/HugePages_Total/ { print $2 }' /proc/meminfo
DTS_DUT_CMD: umount `awk '/hugetlbfs/ { print $2 }' /proc/mounts`
DTS_DUT_CMD: mkdir -p /mnt/huge
DTS_DUT_CMD: mount -t hugetlbfs nodev /mnt/huge
DTS_DUT_CMD: modprobe uio
```

```
DTS_DUT_CMD: rmmod -f igb_uio
DTS_DUT_CMD: insmod ./x86_64-native-linuxapp-gcc/kmod/igb_uio.ko
DTS_DUT_CMD: lsmod | grep igb_uio
DTS_DUT_CMD: usertools/dpdk_nic_bind.py --bind=igb_uio 08:00.0 08:00.1 0a:00.0 0a:00.1
```

Begin the validation process of test suite.

```
TEST SUITE : TestCmdline
                  INFO: NIC :        niantic
    SUITE_DUT_CMD: make -j -C examples/cmdline
    SUITE_DUT_CMD: ./examples/cmdline/build/app/cmdline -n 1 -c 0x2
                           INFO: Test Case test_cmdline_sample_commands Begin
```

Clean-up DUT and tester after all validation finished.

```
   DTS_DUT_CMD: rmmod igb_uio
   DTS_DUT_CMD: modprobe igb
   DTS_DUT_CMD: modprobe ixgbe
   DTS_DUT_CMD: modprobe e1000e
   DTS_DUT_CMD: modprobe e1000
   DTS_DUT_CMD: modprobe virtio_net
DTS_TESTER_CMD: rmmod igb_uio
DTS_TESTER_CMD: modprobe igb
DTS_TESTER_CMD: modprobe ixgbe
DTS_TESTER_CMD: modprobe e1000e
DTS_TESTER_CMD: modprobe e1000
DTS_TESTER_CMD: modprobe virtio_net
```

CHAPTER 4

Review Test Result

4.1 Browse the result files

After DPDK Test Suite finished the validation, we can find the result files as below in output folder. The files in output folder maybe different when change the CRB or choose different suites.

For Example, You can find the following in output folder after execution.

```
[root@tester output]# ls
CrownPassCRB1  dts.log  statistics.txt  TestHelloWorld.log  test_results.xls
```

Please see details about these files:

- CrownPassCRB1: contains the result RST file and graph of performance data
- dts.log: Full execution log of DPDK Test Suite framework
- statstics.txt: summary statistics of DPDK Test Suite executed suites
- TestHelloWorld.log: log message of TestHelloWorld case
- test_result.xls: excel format result file

4.2 Check test result of DPDK Test Suite

You can go through the summary of execution result via statistic.txt. This file includes the number of passed test cases, the number of failed case, the number of blocked and pass ratio.

Please see example as the following. You can cat the sample file, then show this information of execution, totally executed two test cases, all cases passed the criterion and no failed or blocked cases.

```
[root@tester output]# cat statistics.txt
Passed     = 2
Failed     = 0
```

```
Blocked    = 0
Pass rate  = 100.0
```

If you need more detail information of test result, please open excel formatted file test_result.xls. This file contains of both detailed case information and case results. Also you can find description of the failure reason if DPDK Test Suite can track it.

DUT	Target	NIC	Test suite	Test case	Results
10.239.128.117	x86_64-native-linuxapp-gcc	niantic			
			hello_world	hello_world_all_cores	PASSED
				hello_world_single_core	PASSED

If you want to track more details about the process of each suite, please go to log file which named by this suite, all related information will stored in this file.

DPDK Test Suite log module provides several levels to track event or output in log file. Every message level will have its own scopes. Separated log messages will help us get to known what happening in DPDK Test Suite and what happening in DUT and tester.

Level	description
INFO	DPDK Test Suite system level log, show start and stop process in this suite
SUITE_DUT_CMD	Commands send to DUT CRB
SUITE_DUT_OUTPUT	Output after the send the commands to DUT
SUITE_TESTER_CMD	Commands send to tester, most of they are Scapy commands which will send packet to DUT port
SUITE_TESTER_OUTPUT	Output after the send the commands to tester

Please see example for TestHelloWorld suite log as the following. This log file showed that application helloworld sent hello message from core1, and finally matched the pass criterion.

```
22/08/2014 11:04:45                 INFO:
TEST SUITE : TestHelloWorld
22/08/2014 11:04:45                 INFO: NIC :        niantic
22/08/2014 11:04:45        SUITE_DUT_CMD: make -j -C examples/helloworld
22/08/2014 11:04:45     SUITE_DUT_OUTPUT: make: Entering directory `/root/dpdk/
↪examples/helloworld'^M
  CC main.o^M
  LD helloworld^M
  INSTALL-MAP helloworld.map^M
  INSTALL-APP helloworld^M
make: Leaving directory `/root/dpdk/examples/helloworld'
22/08/2014 11:04:45                 INFO: Test Case test_hello_world_single_core Begin
22/08/2014 11:04:45        SUITE_DUT_CMD: ./examples/helloworld/build/app/helloworld -
↪n 1 -c 0x1fffffffff
22/08/2014 11:04:48     SUITE_DUT_OUTPUT: EAL: Detected lcore 0 as core 0 on socket 0^
↪M
...
hello from core 1
22/08/2014 11:05:08                 INFO: Test Case test_hello_world_single_core␣
↪Result PASSED:
22/08/2014 11:05:09        SUITE_DUT_CMD: uname
22/08/2014 11:05:09     SUITE_DUT_OUTPUT:
22/08/2014 11:05:09     SUITE_TESTER_CMD: killall scapy 2>/dev/null; echo tester
22/08/2014 11:05:09  SUITE_TESTER_OUTPUT: tester
22/08/2014 11:05:10     SUITE_TESTER_CMD: uname
22/08/2014 11:05:10  SUITE_TESTER_OUTPUT:
```

```
22/08/2014 11:05:10                    INFO:
TEST SUITE ENDED: TestHelloWorld
```

4.3 Generate PDF doc from RST

Since DPDK Test Suite stores test result as RST by default, you may be want to transfer it to PDF formatted which make it more readable. Firstly, please enter the folder which contained the RST results, then use python tool `rst2pdf` to convert RST. If there's no error return, you can find the pdf file generated with same name.

```
[root@tester dts]# cd output/CrownPassCRB1/x86_64-native-linuxapp-gcc/Niantic
[root@tester niantic]# rst2pdf TestResult_hello_world.rst
[root@tester niantic]# ls
 TestResult_hello_world.pdf  TestResult_hello_world.rst
```

CHAPTER 5

Virtualization Framework

5.1 Design Concept

DTS virtualization framework is based on extendible design which can support different types of hypervisors and their parameters.

Suite for virtual feature should handle virtual machine creation and destruction. It only need to execute some most basic operations on guest like start, stop. There're two ways to configure virtual machine. One is configuration file, the other one is to call hypervisor internal API. Suite can choose anyway of them based on its implementation.

Note: If VM configurations are different between test cases, those configurations shared with all cases should be defined in suite local configuration file. Those dynamical configurations should be set by hypervisor module's API.

Virtual machine DUT class is inherited from host DUT class. This mean that for vm DUT, can call the same API as DUT object.

5.1.1 Flow Chart

Below picture show the virtualization related modules working flow.

5.2 System Requirements

5.2.1 Host Preparation

Kernel should enable KVM. In bios feature Intel(R) Virtualization Technology should be enabled. Emulator qemu must be installed in host.

Note: Some features like virtio cuse request higher version than default qemu release with linux distribution. For those features, qemu should be updated to version later than 2.1.

5.2.2 Guest Preparation

SSH connection

DTS create ssh connection to guest based on redirect guest ssh port to host port. Ssh connection will require one interface in guest which can connect to host. User should setup one default interface (e1000) in guest and make sure it can be up after guest boot up.

Note: Qemu process contained with one virtual DHCP server on 10.0.2.2 and will be allocated an address starting from 10.0.2.15. For more information about qemu network, please reference to https://en.wikibooks.org/wiki/QEMU/Networking.

In qemu module, the default e1000 device's pci address is assigned to 00:1f.0 which is the very end of guest pci address. In guest, we configure udev rule and assign the default interface named as "host_connect".

Add udev rule for default e1000 device:

```
vim /etc/udev/rules.d/70-persistent-net.rules
KERNELS=="0000:00:1f.0",SUBSYSTEMS=="pci", ACTION=="add", DRIVERS=="?*" ,KERNEL=="eth*
↪", NAME="host_connect"
```

Enable dhcp on default host_connect interface.

```
vim /etc/sysconfig/network-scripts/ifcfg-host_connect
    TYPE="Ethernet"
    BOOTPROTO="dhcp"
    DEFROUTE="yes"
    DEVICE="host_connect"
    NAME="host_connect"
    ONBOOT="yes"
    PEERDNS="yes"
    PEERROUTES="yes"
    IPV6_PEERDNS="yes"
    IPV6_PEERROUTES="yes"
```

```
chkconfig --level 2345 network on
```

Install qemu guest agent for DTS monitor guest os.

```
yum install qemu-guest-agent.x86_64
```

For network access, should disable guest firewall service.

```
systemctl disable firewalld.service
```

5.3 Suite Programing

5.3.1 Add Configuration File

Configuration file should be placed in conf/{suite_name}.cfg and in test suite this file will be loaded for VM configurations. Below is one sample for virtualization suite configuration file.

The section name between [] is the VM name. Here we changed default cpu, mem, disk configurations. And add two local configurations login and vnc into configuration file. For cpu parameter, we changed core number to 2 and pin these two cores to socket 1 cores for performance concern. For mem parameter, we changed guest using with hugepage backend memory. It also concerned about performance. For disk parameter, we should change it local disk image absolute path.

Login parameter should be added when guest login username and password not same as host. VNC parameter should be added when need debug guest with vnc display.

```
# vm configuration for vhost sample case
[vm0]
cpu =
   model=host,number=2,cpupin=24 25;
mem =
    size=4096,hugepage=yes;
disk =
    file=/home/img/vm0.img;
login =
    user=root,password=tester;
vnc =
    displayNum=1;
```

5.3.2 Add Parameters

Below is the brief view of the qemu parameters of vxlan sample virtual machine. These parameters are gathered into one list of python dictionary.

```
[{'name': [{'name': 'vm0'}]},
{'enable_kvm': [{'enable': 'yes'}]},
{'qga': [{'enable': 'yes'}]},
{'daemon': [{'enable': 'yes'}]},
{'monitor': [{'path': '/tmp/vm0_monitor.sock'}]},
{'net': [{'opt_addr': '1f', 'type': 'nic', 'opt_vlan': '0'}, {'type': 'user', 'opt_
↪vlan': '0'}]},
{'device': [{'opt_mac': '00:00:20:00:00:20', 'opt_path': './vhost-net', 'driver':
↪'vhost-user'}, {'opt_mac': '00:00:20:00:00:21', 'opt_path': './vhost-net', 'driver
↪': 'vhost-user'}]},
{'cpu': [{'model': 'host', 'number': '4', 'cpupin': '24 25 26 27'}]},
{'mem': [{'hugepage': 'yes', 'size': '4096'}]},
{'disk': [{'file': '/storage/vm-image/vm0.img'}]},
{'login': [{'password': 'tester', 'user': 'root'}]},
{'vnc': [{'displayNum': '1'}]}]
```

In vxlan sample suite, we need to support socket based vhost-user network devices. Qemu command for vhost-user device will like as below.

```
-chardev socket,path=/path/to/socket,id=chr0 \
-netdev type=vhost-user,id=net0,chardev=chr0 \
-device virtio-net-pci,netdev=net0,mac=00:00:20:00:00:20
```

For user configuration, we should only care about socket path and mac address. Configuration of vhost-user device should like below.

```
device =
        driver=vhost-user,opt_path=./vhost-net,opt_mac=00:00:20:00:00:20
        driver=vhost-user,opt_path=./vhost-net,opt_mac=00:00:20:00:00:21
```

Python code should like below, vxlan_sample suite chosen this way.

```
vm_params = {}
vm_params['driver'] = 'vhost-user'
vm_params['opt_path'] = './vhost-net'
vm_params['opt_mac'] = "00:00:20:00:00:20"
self.vm.set_vm_device(**vm_params)
vm_params['opt_mac'] = "00:00:20:00:00:21"
self.vm.set_vm_device(**vm_params)
```

If parameter is device, function add_vm_{device} will be called and device options will passed as arguments. The options will be formatted into python dictionary.

```
{'opt_mac': '00:00:20:00:00:20', 'opt_path': './vhost-net', 'driver': 'vhost-user'}

def add_vm_device(self, **options):
    if options['driver'] == 'vhost-user':
        self.__add_vm_virtio_user_pci(**options)
```

In internal add virtio user device function, qemu module will generate chardev, netdev and virtio-net-pci command line in sequence.

```
def __add_vm_virtio_user_pci(self, **options):
if 'opt_path' in options.keys() and options['opt_path']:
    # add socket char device command line
        dev_boot_line = '-chardev socket'
    char_id = 'char%d' % self.char_idx
    dev_boot_line += separator + 'id=%s' % char_id + separator + 'path=%s' %
↪options['opt_path']
    self.char_idx += 1
    self.__add_boot_line(dev_boot_line)

    # add netdev command line
    netdev_id = 'netdev%d' % self.netdev_idx
    self.netdev_idx += 1
    dev_boot_line = '-netdev type=vhost-user,id=%s,chardev=%s,vhostforce' %
↪(netdev_id, char_id)
    self.__add_boot_line(dev_boot_line)

    # add virtio-net-pci command line
    opts = {'opt_netdev': '%s' % netdev_id}
        if 'opt_mac' in options.keys() and \
            options['opt_mac']:
        opts['opt_mac'] = options['opt_mac']
            self.__add_vm_virtio_net_pci(**opts)
```

5.3.3 VM Management

Suite should have known that which hypervisor type based on. With virtual machine instance, suite can start, stop and check status of the guest. All infrastructures required have been made up by virtualization module, suite should not handle that.

Note: Test case should focus in feature validation. VM management related codes recommended to be placed in set_up_all or set_up functions.

In vxlan sample suite, all test cases request to restart virtual machine. The backend application will be started with different parameters. So VM start up code is placed in set_up function.

```
def set_up(self):
    """
    Run before each test case.
    """
    tep_cmd = tep_cmd_temp % {
        'COREMASK': self.coremask,
        'CHANNELS': self.dut.get_memory_channels(),
        'VXLAN_PORT': self.vxlan_port, 'NB_DEVS': vm_num * 2,
        'FILTERS': self.OUTER_INNER_VNI, 'TX_CHKS': chksum,
        'ENCAP': encap, 'DECAP': decap}
    self.prepare_vxlan_sample_env(tep_cmd)
```

Before VM operations, vxlan sample feature request to start tep_termination application first. To initialized hypervisor kvm instance, there're three parameters required. The first is DUT object, the second is VM name and the third is suite name.

```
def prepare_vxlan_sample_env(self, tep_cmd):
    # remove unexpected socke
    self.dut.send_expect("rm -rf vhost-net", "# ")

    # start tep_termination first
    self.dut.send_expect(tep_cmd, "VHOST_CONFIG: bind to vhost-net")

    # start one vm
    self.vm = QEMUKvm(self.dut, 'vm0', 'vxlan_sample')
```

Before VM start up, suite still can change VM parameters and in this case suite will add two vhost-user devices.

```
# add two virtio user netdevices
vm_params = {}
vm_params['driver'] = 'vhost-user'
vm_params['opt_path'] = './vhost-net'
vm_params['opt_mac'] = "00:00:20:00:00:20"
self.vm.set_vm_device(**vm_params)
vm_params['opt_mac'] = "00:00:20:00:00:21"
self.vm.set_vm_device(**vm_params)
```

Add exception handler in VM start, it is critical function and better to handle the exception.

```
try:
    self.vm_dut = self.vm.start()
    if self.vm_dut is None:
        raise Exception("Set up VM ENV failed!")
except Exception as e:
```

```
    print dts.RED("Failure for %s" % str(e))

    return True
```

VM start function will just return VM DUT object and support all DUT APIs.

```
def vm_testpmd_start(self, vm_id=0):
    """
    Start testpmd in virtual machine
    """
    if vm_id == 0 and self.vm_dut is not None:
        # start testpmd
        self.vm_dut.send_expect(self.vm_testpmd, "testpmd>", 20)
        # set fwd mac
        self.vm_dut.send_expect("set fwd io", "testpmd>")
        # start tx_first
        self.vm_dut.send_expect("start tx_first", "testpmd>")
```

When case has been run, need kill guest dpdk application and stop VM.

```
def clear_vxlan_sample_env(self):
    if self.vm_dut:
        self.vm_dut.kill_all()
        time.sleep(1)

    if self.vm:
        self.vm.stop()
        self.vm = None
```

5.4 KVM Module

5.4.1 Default Parameters

Enable KVM

DTS enable KVM full virtualization support as default. This option will significant improve the speed of virtual machine.

Qemu Guest Agent

Qemu monitor supply one method to interact with qemu process. DTS can monitor guest status by command supplied by qemu guest agent. Qemu guest agent is based on virtio-serial devices.

```
-device virtio-serial -device virtserialport,chardev=vm_qga0,name=org.qemu.guest_
↪agent.0
-daemonize -monitor unix:/tmp/vm_monitor.sock,server,nowait
```

Check whether guest os has been started up.

```
qemu-ga-client address=/tmp/{vm_name}_qga0.sock ping 120
```

Note: We only wait two minutes for guest os start up. For guest os only has few hardware and we has disabled most services, so 2 minutes is enough. This command will be return when guest os is ready, so DTS will not wait 2 minutes for each time.

Check whether guest os default interface has been up.

```
qemu-ga-client address=/tmp/{vm_name}_qga0.sock ifconfig
```

DTS will wait for guest os default interface upped and get auto dhcp address. After that DTS can connect to guest by ssh connections.

```
lo:
          inet 127.0.0.1  netmask 255.0.0.0
                inet6 ::1  prefixlen 128
host_connect:
                inet 10.0.2.15  netmask 255.255.255.0
                inet6 fe80::200:ff:feb9:fed7  prefixlen 64
                ether 00:00:00:b9:fe:d7
```

Power down guest os.

```
qemu-ga-client address=/tmp/{vm_name}_qga0.sock powerdown
```

Note: For more information about qemu guest agent, please reference to http://wiki.qemu.org/Features/QAPI/GuestAgent.

Qemu Monitor

After guest started, there's no way to known that host pci devices assigned-into guest. When assign host pci device into guest, we also add "id" string for this device.

```
-device pci-assign,host=07:10.0,id=pt_0
```

With this command, we assign host VF device 07:10.0 into guest and it named as "pt_0". "pt_0" mean it's the first device pass through into guest os. After guest os started, we use dump pci command and generate guest and host pci mapping by "id".

```
Bus  0, device   4, function 0:
        Ethernet controller: PCI device 8086:10ed
          BAR0: 64 bit memory at 0xfebb0000 [0xfebb3fff].
          BAR3: 64 bit memory at 0xfebb4000 [0xfebb7fff].
          id "pt_0"
Bus  0, device   5, function 0:
        Ethernet controller: PCI device 8086:10ed
          BAR0: 64 bit memory at 0xfebb8000 [0xfebbbfff].
          BAR3: 64 bit memory at 0xfebbc000 [0xfebbffff].
          id "pt_1"
```

Connection to monitor socket on DUT.

```
nc -U /tmp/{vm_name}_monitor.sock
```

Note: For More detail information about qemu monitor. https://en.wikibooks.org/wiki/QEMU/Monitor#info

5.4.2 Configured Parameters

Net

Net parameter is used to create one new Network Interface. There're few types of network interface supported by net parameter.

Type supported: nic: Network Interface Card user: connect the user mode network stack tap: connect the host TAP network interface bridge: connects a host TAP network interface to a host bridge device

Each type has different options, detail information shown in below tables.

Options for nic:

Option of nic	Description	Default value	Must have
opt_vlan	vlan of virtual nic	0	No
opt_macaddr	If not assgin, nic will generate random mac	N/A	No
opt_model	model of virutal nic	e1000	No
opt_name	name be assigned for use in monitor command	N/A	No
opt_addr	pci address in virtual machine	N/A	No
opt_vectors	number v of MSI-X vectors	N/A	No

Options for user:

Option of user	Description	Default value	Must have
opt_vlan	vlan of virtual nic	0	No
opt_hostfwd	Redirect incoming TCP or UDP connections to the host port	N/A	No

Options for tap:

Option of tap	Description	Default value	Must have
opt_vlan	vlan of virtual nic	0	No
opt_br	bridge which tap device bound to	br0	Yes
opt_script	script for tap device network configure	/etc/qemu-ifup	No
opt_downscript	script for tap device network deconfigure	/etc/ qemu-ifdown	No

Device

Device parameter is used to add one emulated device into guest. Now DTS support few types of driver based devices.

Driver supported: pci-assign: pci passthrough host devices into vm virtio-net-pci: virtio devices vhost-user: vhost-user network device based on socket vhost-cuse: vhost-user network device based on tap

Options for pci-assign:

Options of pci-assign	Description	Default value	Must have
opt_host	host pci device address	N/A	Yes
opt_addr	pci address in virtual machine	N/A	No

Options for virtio-net-pci:

Option of virtio-net-pci	Description	Default value	Must have
opt_netdev	name for virtio netdev	N/A	Yes
opt_id	netdevice id for virtio	N/A	Yes
opt_mac	mac address on virtio-net device	N/A	No
opt_bus	pci bus of virtio-net device in vm	N/A	No
opt_addr	pci address of virtio-net-pci device in vm	N/A	Yes

Options for vhost-user:

Option of vhost-user	Description	Default value	Must have
opt_path	unix socket path of character device	N/A	Yes
opt_mac	mac address on virtio-net-pci device	N/A	Yes

Options for vhost-cuse:

Option of vhost-cuse	Description	Default value	Must have
opt_mac	mac address on virtio-net-pci device	N/A	No
opt_settings	virtio device settings	N/A	No

VNC

Vnc parameter is used for add vnc display in qemu process. There's only one option "displayNum" supported by this parameter. This parameter is added for user debug.

Note: Display number should be different between virtual machines.

User Command

User command parameter is used for add one special command for qemu. Some qemu command lines are so unique, there's no value to add certain parameter support for them. Those command lines can be implemented by this parameter.

Login

Login parameter is used for specify guest login username and password.

CHAPTER 6

Virtualization Scenario

When enable virtualization scenario setting in execution cfg, DTS will load scenario configurations and prepare resource and devices for VMs. After VMs started, scenario module will prepare test suite running environment. After all suites finished, scenario module will stop VMs and then clean up the scene.

6.1 Configuration File

With below configuration, DTS will create one scenario which created one VM with two VF devices attached. In scene section and according to configurations defined in suite. DUT object in suite will be VM DUT object, tester and DUT port network topology will be discovered automatically. Now DTS only support kvm typed hypervisor to create virtualization scenario.

```
# vm configuration for vf passthrough cases
# numa 0,1,yes yes mean cpu numa match the first port
# skipcores list mean those core will not be used by vm
# dut=vm_dut; mean vm_dut act as dut
# dut=dut; mean host dut act as dut
# portmap=cfg; mean vm_dut port map will be load from cfg
# portmap=auto; mean vm_dut will create portmap automatically
# devices = dev_gen/host/dev_gen+host not useful now
[scene]
suite =
        dut=vm_dut,portmap=auto;
        tester=tester;
type=kvm;
```

Virtual machine "vm0" section configured cpu, memory, disk and device settings in VM. As below configurations, VM will not use the first four lcores on DUT. DTS will generate two VF devices from first two host PF devices. These two VF devices will be pass-through into guest and their pci address will be auto assigned by qemu.

```
[vm0]
cpu =
        model=host,number=4,numa=auto,skipcores=0 1 2 3;
```

```
mem =
        size=2048,hugepage=no;
disk =
        file=/storage/vm-image/vm0.img;
dev_gen =
        pf_idx=0,vf_num=1,driver=default;
        pf_idx=1,vf_num=1,driver=default;
device =
        vf_idx=0,pf_dev=0,guestpci=auto;
        vf_idx=0,pf_dev=1,guestpci=auto;
vnc =
displayNum=1;
```

All suites will be run in scenario like below picture.

6.2 Scenario Parameters

Options for suite:

option	Description	Options	Default value	Must have
dut	type of dut for dts suite	vm_dut,dut	dut	No
dut->portmap	method to generate dut port maps	auto, cfg	auto	No
tester	type of tester for dts suite[Not used by now]	N/A	N/A	No
type	type of hypervisor	kvm,libvirtd	kvm	No

Options for cpu:

option	Description	Options	Default value	Must have
model	type of dut for dts suite		host	Yes
number	number of cores in virtual machine		4	Yes
numa_aware	numa id of cores allocated from resource module	0,1,auto	0	Yes
skipcores	cores should not be used, most time for those cores will be used by dpdk on host			No

Options for mem:

option	Description	Options	Default value	Must have
size	virtual machine memory size in MBs		2048	Yes
hugepage	whether allocate memory from hugepages		No	No

Options for dev_gen:

option	Description	Options	Default value	Must have
pf_idx	PF device index of host port		0	Yes
pf_inx->vf_num	number of VFs created by this PF device		0	Yes
pf_inx->driver	Allocate VF devices from which PF host driver	igb_uio,default vfio-pci	default	Yes

Options for device:

option	Description	Options	Default value	Must have
pf_idx	PF device index of host port		0	Yes
pf_idx->guestpci	pci address in virtual machine			No
vf_idx	VF devices index of all VFs belong to same PF devices			No
vf_idx->pf_dev	PF device index of this VF device			Yes
vf_idx->guestpci	pci address in virtual machine			No

Options for ports:

option	Description	Options	Default value	Must have
dev_idx	device index of virtual machine ports			No
dev_idx->peer	tester peer port's pci address			No