|  |
| --- |
| Storage Processor EP Demo |
|  |
|  |
| Feb. 2014 |

Legal Information

Copyright

Copyright 2014 PMC-Sierra, Inc. All rights reserved.

The information in this document is proprietary and confidential to PMC-Sierra, Inc., and for its customers’ internal use. In any event, no part of this document may be reproduced or redistributed in any form without the express written consent of PMC-Sierra, Inc.

[PMC-xxxxxxx]

Disclaimer

None of the information contained in this document constitutes an express or implied warranty by PMC-Sierra, Inc. as to the sufficiency, fitness or suitability for a particular purpose of any such information or the fitness, or suitability for a particular purpose, merchantability, performance, compatibility with other parts or systems, of any of the products of PMC-Sierra, Inc., or any portion thereof, referred to in this document. PMC-Sierra, Inc. expressly disclaims all representations and warranties of any kind regarding the contents or use of the information, including, but not limited to, express and implied warranties of accuracy, completeness, merchantability, fitness for a particular use, or non-infringement.

In no event will PMC-Sierra, Inc. be liable for any direct, indirect, special, incidental or consequential damages, including, but not limited to, lost profits, lost business or lost data resulting from any use of or reliance upon the information, whether or not PMC-Sierra, Inc. has been advised of the possibility of such damage.

Trademarks

For a complete list of PMC-Sierra’s trademarks and registered trademarks, visit: <http://www.pmc-sierra.com/legal/>

Patents

The technology discussed in this document is protected by one or more of the following patent grants:

Revision History

|  |  |  |
| --- | --- | --- |
| Issue No. | Issue Date | Details of Change |
| 1 | Feb, 2014 | Initial release |

Table of Contents

[Legal Information 2](#_Toc380684128)

[Copyright 2](#_Toc380684129)

[Disclaimer 2](#_Toc380684130)

[Trademarks 2](#_Toc380684131)

[Patents 2](#_Toc380684132)

[Revision History 3](#_Toc380684133)

[1 Storage Processor EP Demo Scope 6](#_Toc380684134)

[2 Storage Processor EP Demo hardware Configuration 7](#_Toc380684135)

[2.1 SP EP demo configuration 7](#_Toc380684136)

[2.2 Storage Processor Evaluation board 7](#_Toc380684137)

[2.3 x86 host configuration 7](#_Toc380684138)

[2.4 Test client configuration 7](#_Toc380684139)

[2.5 Demo resources 8](#_Toc380684140)

[3 Storage Processor EP Demo Setup 10](#_Toc380684141)

[3.1 Storage Processor Evaluation Board Setup 10](#_Toc380684142)

[3.1.1 Program the flash image 10](#_Toc380684143)

[3.1.2 Prepare the SP EP root file system drive 11](#_Toc380684144)

[3.2 x86 host setup 12](#_Toc380684145)

[3.2.1 Install Debian Linux 12](#_Toc380684146)

[3.2.2 Build the SP EP host driver 12](#_Toc380684147)

[3.2.3 Install SP evaluation board 12](#_Toc380684148)

[3.3 Test client 12](#_Toc380684149)

[4 Run demo application 13](#_Toc380684150)

[4.1 Demo scripts 13](#_Toc380684151)

[4.2 Demo preparation 13](#_Toc380684152)

[4.2.1 Hardware setup 13](#_Toc380684153)

[4.2.2 Demo Script 13](#_Toc380684154)

[4.2.3 Network setup 13](#_Toc380684155)

[4.2.4 TTY Preparatioin for demo 14](#_Toc380684156)

[4.2.5 Prepare Storage 15](#_Toc380684157)

[4.3 Demo Scenario #1: None SP EP offloading 15](#_Toc380684158)

[4.3.1 Data flow 15](#_Toc380684159)

[4.3.2 Demo sequence 16](#_Toc380684160)

[4.4 Demo Scenario #2: EP offloading 17](#_Toc380684161)

[4.4.1 Data flow 17](#_Toc380684162)

[4.4.2 Demo sequence 17](#_Toc380684163)

[4.5 Demo Scenario #3: EP offloading with host Linux network forwarding 18](#_Toc380684164)

[4.5.1 Data flow 18](#_Toc380684165)

[4.5.2 Demo sequence 19](#_Toc380684166)

[5 SP EP Demo result 21](#_Toc380684167)

[6 References 22](#_Toc380684168)

# Storage Processor EP Demo Scope

The SP EP demo is to show the functionality of the storage processor as an EP device, and demonstrate how EP offloading helps the host CPU loading, demo scenarios include:

* Demonstrate the test client stream data to host’s local storage.
* Demonstrate the test client stream data to host with EP storage offloading.
* Demonstrate the test client stream data to host with EP storage offloading using Linux kernel network forwarding.

The demo can conclude:

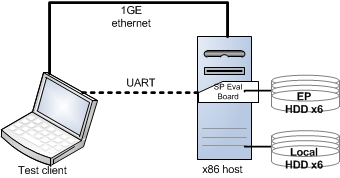
* The application performance is not impacted by introducing EP.
* EP offloading benefits the host CPU utilization shown in the above three scenarios.

# Storage Processor EP Demo hardware Configuration

The storage processor EP demo hardware configuration includes one x86 host, one storage processor evaluation board (i.e. adaptec 7 series card), one x86 test client, SATA drives and necessary cabling for interconnect. A GHS probe is only required for programming the flash of the SP evaluation board; it is not required for running the demo.

* Storage processor evaluation board is an Adaptec 7 series card loaded with SP EP firmware. It shall be connected to 6x SATA HDDs(one system drive, five data drives). SP evaluation board is also connected with a UART cable for console. A GHS probe is required to connect to the SP evaluation board for flash programming.
* The x86 host and the SP evaluation board is connected through the PCIe slot. It shall also have an 1GE NIC interface and be connected to 6x SATA HDDs(one system drive, five data drives) through native SATA port or an HBA.
* The x86 test client is to generate data stream to the x86 host for storage in local connected SATA drive or through EP. It shall be connected to the x86 host through 1GE network for data path. The test client can also act as the UART console of the SP EP.

## SP EP demo configuration



## Storage Processor Evaluation board

The SP evaluation board is an Adaptec 7 series card, it is programmed with the SP EP firmware. It is connected to 6x SATA HDDs through SFF-8643 to 4x SATA fan out cables, one SATA drive for system, the others for data storage.

## x86 host configuration

The x86 host connects to the SP EP evaluation board through the PCIe slot, it receives the network data stream from the test client and store the data in local SATA drive or through the SP EP. It includes one 1GE NIC for interconnect to the test client. It may or may not include a display.

The x86 host is installed with a Debian Linux OS.

## Test client configuration

The test client serves as the test data generator, and a serial console to the SP Linux.

## Demo resources

The demo resources can be retrieved from link below:

* //diqing/shareddocs/engineering/ESD/Projects/Janus/firmware/build/sp-sdk-0.06.20.25-ep/demo

|  |  |  |
| --- | --- | --- |
| **info** | **name** | **package** |
| SP flash image | flash\_block.data | \_internal-sp-sdk-binary-0.06.20.25-ep.tar.bz2 |
| SP rootfs | (rootfs) | rootfs-0.06.20.25-ep.cpio.xz |
| Host VNET driver | applications/pmc\_vnet/rc/ | sp-sdk-0.06.20.25-ep.tar.xz |
| Demo scripts | scripts/ | sp-sdk-demo-script-0.06.20.25-ep.tar.xz |

# Storage Processor EP Demo Setup

## Storage Processor Evaluation Board Setup

## Program the flash image

* Power up in the SP evaluation board by plug to a PCIe slot of a mother board or a PCIe backplane.
* Burn the SP flash image "flash\_block.data" using the probe.
* Reboot and the SP shall boot and prints to the UART console.

To burn the flash image using the probe:

* Launch GHS Multi and connect to SP board
* Reset target
* From GHS MULTI menu, select Target -> Flash...
* In MULTI Fast Flash Programmer window
  + Base address of bank = 0xBA000000, then press Add New
  + Select the flash image file, File type = Raw Bin, Program offset = 0
  + Use 128KB of Target RAM at 0xBC000000
  + Check “Erase”, “Program” and “Verify”
  + Press Program Flash

## Prepare the SP EP root file system drive

The SP EP rootfs(root file system) drive shall be prepared on an x86 host.

* Connect the SP EP rootfs HDD to the x86 host.
* Create a primary partition on the system HDD using fdisk (e.g. */dev/sdb1*)

*fdisk /dev/sdb*

* Create ext3 file system on the partition

*mkfs.ext3 /dev/sdb1*

* Label the partition with “debian”

*tune2fs –L “debian” /dev/sdb1*

* Extract the rootfs content to the just created ext3 file system

*mount /dev/sdb1 /mnt/sdb1* *cp host/path/to/rfs/* *rootfs-N.NN.NN.NN-ep.cpio.xz /mnt/sdb1  
 cd /mnt/sdb1  
 xzcat rootfs-N.NN.NN.NN-ep.cpio.xz | cpio –id*

* Connect the SP EP rootfs HDD to the SP EP evaluation board.

## x86 host setup

## Install Debian Linux

The x86 host in the SP EP demo is tested with Debian Linux, but it should also work on other Linux distributions.

Following packages need to be installed:

|  |  |
| --- | --- |
| **package** | **info** |
| gcc | compile the host VNET driver |
| linux-headers (matches the running kernel) | compile the host VNET driver upon running kernel |
| mdadm | RAID device management |
| sysstat | CPU/DISK usage monitor |
| nuttcp | TCP stream generator and receiver |
| redir | redirect TCP stream between client and EP |
| openssh | ssh client/server |

## Build the SP EP host driver

* Copy and extract SDK package to the x86 host path – /*host/path/to/sdk*
* Make sure the directory */host/path/to/sdk/applications/pmc\_vnet/rc/* exists

## Install SP evaluation board

* Plug the SP evaluation board in the PCIe slot.

## Test client

The test client is an x86 Linux PC, Debian Linux is tested but other Linux distributions shall also work.

Following packages need to be installed:

|  |  |
| --- | --- |
| **package** | **info** |
| nuttcp | TCP stream generator and receiver |
| redir | redirect TCP stream between client and EP |
| minicom | Connect to SP EP serial console thru UART |
| openssh | ssh client/server |

# Run demo application

## Demo scripts

The "client\_\*" scripts aim to run in test client TTYs.

The "host\_\*" scripts aim to run in host TTYs.

The "ep\_\*" scripts aim to run in SP EP TTYs.

|  |  |
| --- | --- |
| **script** | **info** |
| do-raid-create.sh | Raid device create script, use with caution |
| do-raid-delete.sh | Raid device delete script, use with caution |
| client\_init\_net.sh | Client network init script NIC1G |
| client\_do\_service.sh | Client data generator for all stream demos |
| host\_init\_vnet.sh | Host network init script for NIC1G and VNET |
| host\_do\_app\_relay.sh | Host demo script for host-application-relay |
| host\_do\_kernel\_forward.sh | Host demo script for host-kernel-forward |
| host\_do\_local\_store.sh | Host demo script for host-local-store |
| ep\_init\_vnet.sh | SP EP network init script for VNET |
| ep\_do\_store\_app\_relayed.sh | SP EP demo script for host-application-relay |
| ep\_do\_store\_kernel\_forwarded.sh | SP EP demo script for host-kernel-forward |

## Demo preparation

## Hardware setup

Set up the demo as described in section 2 and section 3

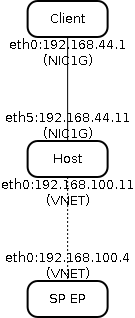
## Demo Script

Copy and extract the test script package to:

* Test client Linux (*client/path/to/script*)
* Host Linux(*host/path/to/script*)
* SP Linux (*sp/path/to/script*)

## Network setup

Use demo scripts to configure the network as below:



Following steps need to be strict followed to ease of operation:

1. **Attention**: Please change the interface name in all demo scripts, if your NIC name is different than the network figure mentioned above.
2. **Important**: Edit the variable "VNET\_SRC\_" in "host\_init\_vnet.sh" script to "/host/path/to/sdk/applications/pmc\_vnet"
3. Run "client\_init\_net.sh" in test client local TTY
4. Run "host\_init\_vnet.sh" in host local TTY
5. Run "ep\_init\_vnet.sh" in SP EP Serial-Console (attached to test client, 115200,8n1)
6. All scripts shall be executed without error

## TTY Preparatioin for demo

The TTYs are required to issue the demo scripts and system commands.

To ease of execution, recommend to use PTS (except those TTYs for running setup/init script) by ssh from host machine to test client and SP EP.

To be simplifying, this document use "TTY" for all terminal sessions from test client, host and SP EP.

Test client TTYs requirement:

* TTY1 to start the stream server services

Host TTYs requirement:

* TTY1 to run "iostat -d -m 10" for monitoring RAID IO
* TTY2 to run "mpstat 10" for monitoring CPU usage
* TTY3 to run demo session#1
* TTY4 to run outstanding demo session#2 (if requires)
* TTY5 to run outstanding demo session#3 (if requires)

SP EP TTYs requirement:

* TTY1 to run "iostat -d -m 10" for monitoring RAID IO
* TTY2 to run "mpstat 10" for monitoring CPU usage
* TTY3 to run demo session#1
* TTY4 to run outstanding demo session#2 (if requires)
* TTY5 to run outstanding demo session#3 (if requires)

## Prepare Storage

To create a RAID5 volume over the SATA drives in host Linux and SP Linux.

* Host Linux local storage preparation

(**Attention**: Please modify "DEVS" and "DEVS\_ADD" variable in "do-raid-create.sh" to match the real spare drive names. Default is for devices */dev/sd[bcdef]*)

* + For example, use "DEVS=/dev/sd[mnop]" and "DEVS\_ADD=/dev/sdq" if the spare drives are "/dev/sdm /dev/sdn /dev/sdo /dev/sdp /dev/sdq"

*./do-raid-create.sh*

* SP Linux storage preparation

(**Attention**: Please modify "DEVS" and "DEVS\_ADD" variable in "do-raid-create.sh" to match the real spare drive names. Default is for devices */dev/sd[bcdef]*)

* + For example, use "DEVS=/dev/sd[mnop]" and "DEVS\_ADD=/dev/sdq" if the spare drives are "/dev/sdm /dev/sdn /dev/sdo /dev/sdp /dev/sdq"

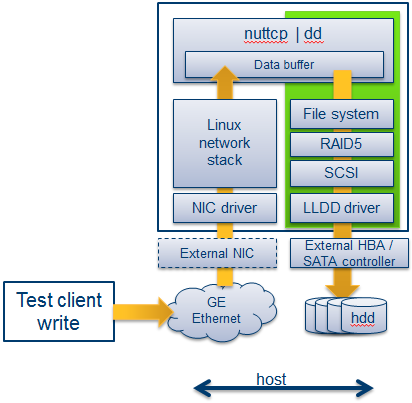
*./do-raid-create.sh*

## Demo Scenario #1: None SP EP offloading

The None SP EP offloading demo is to demonstrate the none-SP-EP offloading mode, i.e. using a legacy HBA or local SATA ports in backend. All storage kernel stack and driver is running on x86 host. The host CPU utilization is recorded for comparison to the SP EP offloading.

## Data flow

* The test client generates the data stream to host x86;
* The host x86 application receives the data stream from the network stack to application buffer
* The host x86 application writes the data to the backend storage(an RAID5 volume over 5 SATA drives).



## Demo sequence

**One data session**

1. Run commands in Client:

*TTY1# client\_do\_service.sh 1*

1. Run commands in Host:

*TTY1# iostat -d -m 10 /dev/md0*

*TTY2# mpstat 10*

*TTY3# host\_do\_local\_store.sh 1*

1. Check disk IO status and CPU usage on TTY1, TTY2 from Host

**Two data sessions**

1. Run commands in Client:

*TTY1# client\_do\_service.sh 1*

*TTY1# client\_do\_service.sh 2*

1. Run commands in Host:

**Attention**: The second session shall be started as quick as possible after the first one

*TTY1# iostat -d -m 10 /dev/md0*

*TTY2# mpstat 10*

*TTY3# host\_do\_local\_store.sh 1*

*TTY4# host\_do\_local\_store.sh 2*

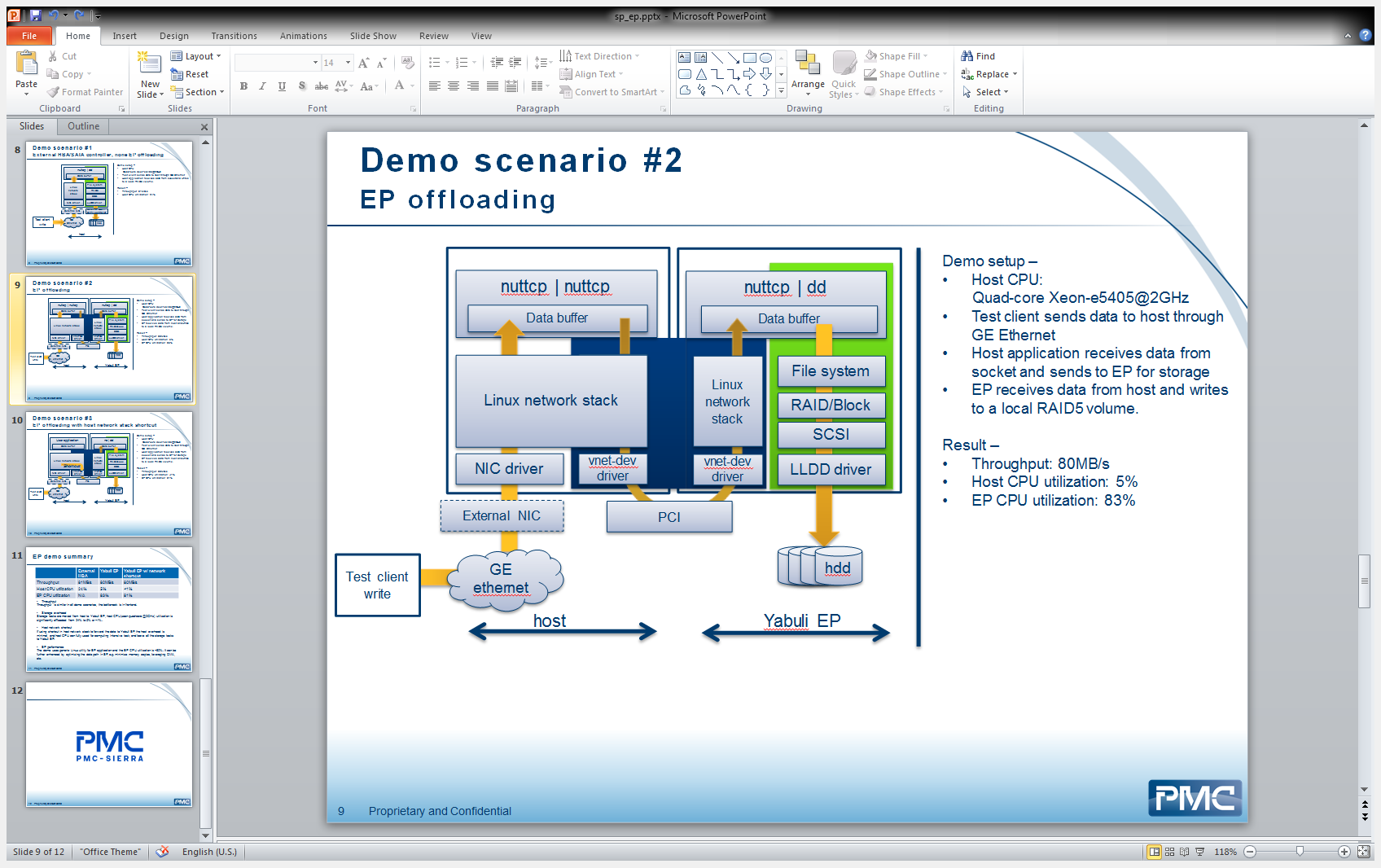
1. Check disk IO status and CPU usage for Host

## Demo Scenario #2: EP offloading

The EP offloading demo is to demonstrate the EP offloading mode. Host APPLICATION receives and forwards the data stream to the EP for storage. All storage kernel stack and driver is running on SP EP Linux. The host CPU utilization is recorded for comparison.

## Data flow

* The test client generates the data stream to host x86;
* The host x86 application receives the data stream from the network stack to application buffer
* The host x86 application sends the data to the SP EP card for storage through the virtual network interface
* The SP application receives the data stream from the virtual network interface to application buffer
* The SP application writes the data to backend storage (a RAID5 volume over 5 SATA drives).



## Demo sequence

**One data session**

1. Run commands in Client TTY:

*TTY1# client\_do\_service.sh 1*

1. Run commands in Host TTY:

*TTY2# mpstat 10*

*TTY3# killall relay && host\_do\_app\_relay.sh 1*

1. Run commands in SP EP TTY:

*TTY1# iostat -d -m 10 /dev/md0*

*TTY2# mpstat 10*

*TTY3# killall nuttcp && ep\_do\_store\_app\_relayed.sh 1*

1. Check disk IO status and CPU usage for Host and SP EP

**Two data sessions**

1. Run commands in Client:

*TTY1# client\_do\_service.sh 1*

*TTY1# client\_do\_service.sh 2*

1. Run commands in Host:

*TTY2# mpstat 10*

*TTY3# killall relay && host\_do\_app\_relay.sh 1*

*TTY4# host\_do\_app\_relay.sh 2*

1. Run commands in SP EP:

**Attention**: The second session shall be started as quick as possible after the first one

*TTY1# iostat -d -m 10 /dev/md0*

*TTY2# mpstat 10*

*TTY3# killall nuttcp && ep\_do\_store\_app\_relayed.sh 1*

*TTY4# ep\_do\_store\_app\_relayed.sh 2*

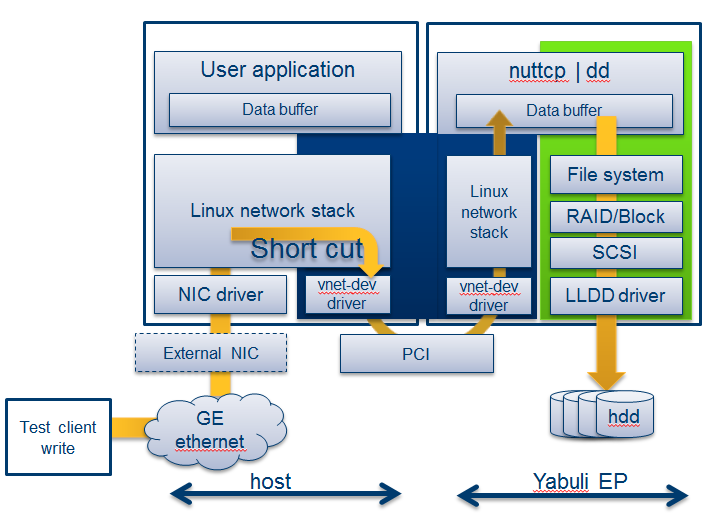
1. Check disk IO status and CPU usage for Host and SP EP

## Demo Scenario #3: EP offloading with host Linux network forwarding

The EP offloading with host Linux network forwarding demo is to demonstrate the EP offloading mode with minimal host Linux overhead. Host KERNEL network stack receives and forwards the data stream to the SP EP for storage, the data stream does not surface out to user application. All storage stack and driver is running on SP EP Linux. The host CPU utilization is recorded for comparison.

## Data flow

* The test client generates the data stream to host x86;
* The host x86 kernel receives and forwards the data stream to the SP EP for storage through the virtual network interface.
* The SP application receives the data stream from the virtual network interface to application buffer
* The SP application writes the data to backend storage (a RAID5 volume over 5 SATA drives).



## Demo sequence

**One data session**

1. Run commands in Client TTY:

*TTY1# client\_do\_service.sh 1*

1. Run commands in Host TTY:

*TTY2# mpstat 10*

*TTY3# host\_do\_kernel\_forward.sh*

1. Run commands in SP EP TTY:

*TTY1# iostat -d -m 10 /dev/md0*

*TTY2# mpstat 10*

*TTY3# killall nuttcp && ep\_do\_store\_app\_relayed.sh 1*

1. Check disk IO status and CPU usage for Host and SP EP

**Two data sessions**

1. Run commands in Client:

*TTY1# client\_do\_service.sh 1*

*TTY1# client\_do\_service.sh 2*

1. Run commands in Host:

*TTY2# mpstat 10*

*TTY3# host\_do\_kernel\_forward.sh*

*TTY4# (not necessary)*

1. Run commands in SP EP:

**Attention**: The second session shall be started as quick as possible after the first one

*TTY1# iostat -d -m 10 /dev/md0*

*TTY2# mpstat 10*

*TTY3# killall nuttcp && ep\_do\_store\_kernel\_forwarded.sh 1*

*TTY4# ep\_do\_store\_kernel\_forwarded.sh 2*

1. Check disk IO status and CPU usage for Host and SP EP

# SP EP Demo result

|  |  |  |  |
| --- | --- | --- | --- |
|  | **None SP EP offloading** | **SP EP offloading** | **SP EP offloading w/ host Linux network forwarding** |
| Throughput | 81MB/s | 80MB/s | 80MB/s |
| Host CPU util | 24% | 5% | <1% |
| SP CPU util | N/A | 83% | 81% |

* Throughput

Throughput is similar in all demo scenarios, the bottleneck is in frontend.

* Storage overhead

Storage tasks are moved from host to SP EP; host CPU(xeon quad-core @2GHz) utilization is significantly offloaded: from 24% to 5% or <1%.

* Host network shortcut

If using shortcut in host network stack to forward the data to SP EP, the host overhead is minimal, and host CPU can fully use for computing intensive task, and leave all the storage tasks to SP EP.

* EP performance

The demo uses generic Linux utility for EP application and the EP CPU utilization is ~80%. It can be further enhanced by optimizing the data path in EP, e.g. minimize memory copies, leveraging DMA, etc.

# References

All demo packages are put at SP release folder

//diqing/shareddocs/engineering/ESD/Projects/Janus/firmware/build/sp-sdk-0.06.20.25-ep/demo

* SDK package:
  + sp-sdk-0.06.20.25-ep.tar.xz
* Root file system package:
  + rootfs-0.06.20.25-ep.cpio.xz
* Demo script package:
  + sp-sdk-demo-script-0.06.20.25-ep.tar.xz
* binary package:
  + \_internal-sp-sdk-binary-0.06.20.25-ep.tar.bz2