175 Cabot Street – Suite 210
Lowell, MA 01854 USA
www.cspi.com
NASDAQ: CSPI

How do I run the snf_bridge example program?

Model:

N/A

Software:

Sniffer10G

Operating System:

Supports both Linux and Windows Operating Systems.

Information:

snf_bridge is a simple Sniffer10G example program in the **/bin/tests/** directory that receives packets on an interface and injects them on an interface.

Its main purpose is to demonstrate how to use the Sniffer10G API to receive and inject.

The source code of this example program can be found in the **/share/examples/** directory in the software distribution. Invoke the test program **snf_bridge** with the option --help to obtain the usage summary of the command-line arguments.

Usage Summary:

```
$ ./snf_bridge [args] -b <...> [ -b <...> ... ]

Command-line [args]:

[ -b <board_source>:<board_dest>:<num_rings>[:]
     where
     <board_source> is the location to capture packets,
     <board_dest> is the location to forward packets,
     <num_rings> is the number of rings/workers to dedicate to capture, and
      is an optional binding cpumask in hexadecimal.

[ -n <num_packets> ] Number of packets to forward before exiting.
[ -N <tx_try_again> ] Number of times to try injecting before dropping
packet.
[ -W <tx_wait_msecs> ] Number of milliseconds to wait in
snf_inject_send.
[ -R ] Reflect non UDP and TCP packets to network device.
```

Here is a simple test you can conduct with two machines (machine 1 and machine 2) with dual port network adapters connected point to point (switchless to port0 to port0 and port1 to port1).
On machine2, the bridging machine, run the command:

```
        $ ./snf_simple_recv -v -n 1000 -b 1
```

This command-line says to bridge all traffic between board 0 and board 1 using one receive ring. Because –n was specified, it will stop bridging once 1000 packets have been received.

And then on machine1, open two windows. In the first window run:

```
        $ ./snf_simple_recv -v -n 1000 -b 1
```

And in the second window run:

```
        $ ./snf_pktgen -p 0 -n 1000
```

The result is that **snf_pktgen** will send 1000 packets out of interface 0 on machine1, it will be received by machine2 on interface 0, which will then reinject them on interface 1, and machine 1 will then receive them with **snf_simple_recv** on interface 1. At this point, since we sent 1000 packets, and we told **snf_bridge** and **snf_simple_recv** to end after 1000 packets, everything should terminate and you should see from the snf_bridge counters printed that it did in fact receive and transmit 1000 packets. If you run into packet loss (which would be unexpected in a short point-to-point connection) you may want to increase the number of packets **snf_pktgen** sends.

You could also do the test using a one port network adapter. You would just be injecting back onto the same interface you received from (by passing –b0:0:1 to **snf_bridge** and –b0 to **snf_simple_recv**), making the bridging machine look like a loopback connector.

*****Note:** The **snf_bridge** program is a software tap and forwards all packets by default. If multiple rings are used (say 8), there is quite a lot of time per packet to make forwarding decisions. There is the ability to forward packets to the operating system as is done with non-TCP and UDP packets with the –R option. Timestamping the packets before forwarding to the OS is possible since the sniffer timestamp is available and the packet can be rewritten before forwarding. Forwarding to the OS is not a high performance pathway however, so depending on how much you wish to forward, this may or may not meet your heads. It is much more efficient to just let sniffer handle everything. Libpcap could be linked in and selected packets could be written out in pcap format for example (assuming fast enough disk).
***

Revision	**Date**	**Change**
1	7/5/2016	Initial Draft
2	8/19/2016	Feedback Edit