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

# Can I use a two-port Myri-10G network adapter as an inline Ethernet analyzer and simultaneously use the Sniffer10G API with the Sniffer10G Libpcap interface?

## Model:

ARC Series C Adapters (10G-PCIE2-8C2-2S)

## Software:

Sniffer10G

## Operating System:

Supports both Linux and Windows Operating Systems.

## Information:

We have a question on the simultaneous use of the Sniffer API with the SNF libpcap. To be specific on what we thought we could do, we looked at using a two-port network adapter as an inline Ethernet analyzer.

To do this, we used the sample **snf-bridge** tool to bridge port 0 to port 1. By bridging in both directions (0->1 and 1<-0) the bridge seems to work in that we can transparently ping through the “appliance”.

To analyze the traffic, we tried to use tcpdump on one of the SNF ports with the expectation that this should act like a tap to the data we bridge. This fails and we get a **snf_open** failed message on the tcpdump if the bridge is up and running:

```
[root@xxxx ~]# tcpdump -ni snf0
snf.0.0 P (userset) SNF_PORTNUM = 0
snf.0.0 P (default) SNF_NUM_RINGS = 1 (0x1)
snf.0.0 P (default) SNF_RSS_FLAGS = 49 (0x31)
snf.0.0 P (default) SNF_DATARING_SIZE = 268435456 (0x10000000) (256.0 MiB)
snf.0.0 P (default) SNF_DESCRING_SIZE = 67108864 (0x4000000) (64.0 MiB)
snf.0.0 P (default) SNF_FLAGS = 0
snf.0.0 P (environ) SNF_DEBUG_MASK = 3 (0x3)
snf.0.0 P (default) SNF_DEBUG_FILENAME = stderr
snf.0.0 P SNF_DEBUG_MASK=0x3 for modes WARN=0x1, PARAM=0x2 QSTATS=0x4
TIMESYNC=0x8 IOCTL=0x10 QEVENTS=0x20
tcpdump: snf_open failed: Permission denied
```

If the bridge is disabled the tcpdump works fine on the snf interface but the bridge can only be opened in one direction so we can’t get a combination bridge A to B, B to A and tcpdump running at the same time.

With “tcpdump -ni snf0” running, then the snf_bridge will load from port 1 to 0 (./snf_bridge -b 1:0:1) but not 0 to 1

(./snf_bridge-b 1:0:1). This reverse if tcpdump is done on snf1. For example, error seen is:

```
[root@xxxx tests]# ./snf_bridge -b 1:0:1
SNF Failure at line 452: (errno=13)
```

Without tcpdump running then the bridge is loaded with "./snf_bridge -b 0:1:1 -b 1:0:1" and we can at least ping through it.

**Answer:**

Perhaps the best way to explain it is that there is only one underlying receive ring and processes or thread can each arrange to have an additional ring.

If there is more than one interested process or thread can each arrange to have an additional ring.

If there is more than one interested process or thread, the **SNF_F_SHARED** bit of the **SNF_FLAGS** and the number of rings in total that will be used must be specified (e.g. with the NUMB_RINGS flag). By default pkts are distributed among the rings via the RSS hash function. You can specify to duplicate the data to all virtual rings but this is not as efficient as the RSS and won't keep up with line rate.

By default, NUM_RINGS is 1 and flags are 0 so by doing:

```
$ ./snf_simple_recv -b0
snf_recv ready to receive
```

In one window then

```
$ ./snf_simple_recv -b0
Can't open snf for sniffing: Permission denied
```

If the bridge is disabled the tcpdump works fine on the snf interface but the bridge can only be opened in one direction so we can't get a combination bridge A to B, B to A and tcpdump running at the same time.

With "tcpdump -ni snf0" running, then the snf_bridge will load from port 1 to 0 (./snf_bridge -b 1:0:1) but not 0 to 1 (./snf_bridge-b 1:0:1). This reverse if tcpdump is done on snf1. For example, error seen is:

You are running into something similar to this when trying to start tcpdump on the snf device when the bridge program is running. The bridge program uses multiple threads (and cores) internally in order to keep up with line rate packet forwarding. It internally sets the number of rings to exactly the number it uses and does not account for another app sharing the ring. You could modify or override to add one to the number of rings, but the bridge needs to run in RSS mode not duplicate mode, so you would only see a small portion of the traffic and the bridge would not forward that traffic.

The two options to sniff the tap both require modifying the bridge example code. You could call the **snf_netdev_reflect()** function for every packet you wanted to sniff, then run tcpdump on the Ethernet device.

This is not efficient since that traffic would be passed to the kernel stack. The other option (and one we've used) is to write packets directly in pcap format file. See 'man 3 libpcap' to see the interface. If you use a ram disk, you can write the packets to output at a fairly good rate (but not line rate).

| **Draft** | **Date** | **Change** |
|---|---|---|
| 1 | 7/11/16 | Initial Draft |
| 2 | 8/2/2016 | Feedback |
| 3 | 8/19/2016 | Further Edits |
| | | |