USER GUIDE

Sniffer v3 User Guide

Version 3.0.26

January 25, 2022

aria CYBERSECURITY SOLUTIONS

ARIA Cybersecurity Solutions, which includes ARIA SDS, Myricom network adapters, and nVoy security appliances, are designed and manufactured by the High Performance Products Division of CSP Inc.

ARIA Cybersecurity Solutions Product Development
C/O CSPi
175 Cabot Street, Suite 210
Lowell, MA 01854
Tel: (800) 325-3110
ARIA_support@ariacybersecurity.com
https://www.ariacybersecurity.com/support/

Contents:

This document describes the run-time SNF software for the ARIA Cybersecurity Solutions Myricom 10-Gigabit Ethernet products.

The document is intended for system and networking architects looking for a tailored solution with a focus on low latency combined with increased throughput. In this context, the SNF product provides a deployable solution by using a combination of advanced software stacks and 10GbE adapters. The document assumes that readers are familiar with the C language, using the GNU development tools, and general computer maintenance.

Software developers interested in directly utilizing the advanced features of ARIA Cybersecurity Solutions Myricom products through the SNF interfaces, should refer to the SNF API Reference Manual. Software documentation and downloads are available from the ARIA Customer Portal.

Additional product information can be found on the ARIA website at:
https://www.ariacybersecurity.com/network-adapters

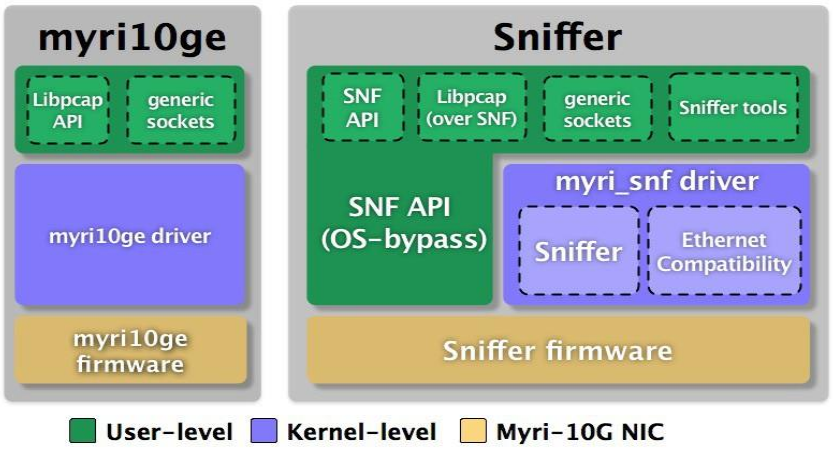
Contact Technical Support via the ARIA Customer Portal:
https://www.ariacybersecurity.com/ support/
or via email:
ARIA_support@ariacybersecurity.com

Chapter 1 Introduction

The Sniffer v3 or Sniffer version 3.0.26 software uses Myri-10G programmable Network Adapters, a firmware extension, and a user-level library to enable sustained capture with merging of 10-Gigabit Ethernet traffic. It also provides high-rate packet injection functionality. Small-packet coalescing and an efficient zero-copy path to host memory allow Sniffer to capture streams at line rate for all Ethernet packet sizes. Optionally, Sniffer can enable multiple independent processes or threads to concurrently analyze the incoming traffic from one or more network adapter ports.

Important Note: The Sniffer v3.0.26 software is a specialized driver/firmware for a wide range of security applications. It is **not** our standard 10-Gigabit Ethernet driver. For standard TCP/IP and UDP/IP 10-Gigabit Ethernet applications, the Myri-10G 10-Gigabit Ethernet driver called **Myri10GE** is available.

Figure 1. Myri10GE and Sniffer software distributions



The Sniffer software distribution is controlled by a dual-mode driver that operates as a regular 10G Ethernet driver until the device is enabled for packet capture. For packet capture, the library and driver instruct the firmware to divert incoming packets to library-managed user-level receive rings instead of the regular Ethernet driver. All the while, the Ethernet driver remains available to send raw Ethernet packets.

Figure 1 shows some of the differences between using the generic myri10ge Ethernet driver and Sniffer:

- **The myri10ge and the myri_snf drivers are *incompatible***. That is, a network adapter port can run either the myri10ge driver and associated firmware or its equivalent components under the Sniffer distribution. Sniffer is not an extension to myri10ge, it wholly replaces the functionality typically provided by myri10ge. From the user's perspective, this is fairly transparent since the firmware is loaded by the network adapter when the myri10ge or myri_snf modules are loaded by the user.
- Sniffer's myri_snf contains two components:
  1. **Sniffer** refers to the driver features and privileged operations that are necessary to allow an application to directly receive packets into userspace. Other aspects such as optional interrupt generation would be covered under this component as well. The Sniffer component is only active when the device is opened for capture through the SNF API (or indirectly from Libpcap/SNF or WinPcap/SNF).
  2. **Ethernet compatibility** refers to the driver-level functionality that is necessary to operate a Myri-10G Network Adapter as a 10Gbit Ethernet adapter. Not all users will need the same level of functionality from this component. At a minimum, it allows capture devices to be brought up as regular Ethernet devices and as such, allows familiar OS-provided tools such as ethtool to interact with the capture device. When the underlying Ethernet device is enabled, it will receive packets until the device is enabled for SNF capture. While the SNF capture device is enabled, it is the exclusive beneficiary of all incoming packets and there is no duplication possible to the Ethernet driver until the SNF capture device is closed. This means that tools and utilities that tap into the OS stack to extract information won't see any traffic. To this end, Sniffer provides its own set of tools and counters that are also usable while packets are being captured. As a special case, the Ethernet driver's send functionality is still available while the underlying SNF capture device is enabled. This permits the use of RAW sockets for sending packets.
- Sniffer's packet capture abilities can be leveraged through the Libpcap or WinPcap library or directly through the SNF API, available as a set of C programming language functions. Using a Sniffer-aware Libpcap/ WinPcap, users reference a Myri-10G Network Adapter through its Ethernet interface name and can run existing Libpcap/WinPcap-dependent applications and continue to rely on Libpcap/WinPcap's portable interface.
-

Direct (SNF API) and indirect (Libpcap/SNF or WinPcap/SNF) users benefit from full userspace access to all incoming packets without requiring any intervention from the OS. This is an important difference when comparing Sniffer to other packet capture solutions. At extremely high packet rates, a single system call on every packet would be sufficient to cause packet drops.

This is why one of Sniffer's main goals is to allow user applications to spend as much time in userspace without relying on the kernel for packet delivery.

1.1 Libpcap/WinPcap over SNF

The easiest way to realize performance improvements using Sniffer is to re-link existing **Libpcap/WinPcap** applications against a Sniffer-capable **Libpcap/WinPcap** (as included in the Sniffer distribution). When this Libpcap/ WinPcap encounters a Sniffer-capable device, it uses the SNF API to obtain user-level zero-copy packets provided by the Myri-10G Network Adapter instead of the usual kernel-based approach.

For details, please read **Chapter 3: Using Sniffer from Libpcap/SNF or WinPcap/SNF.**

1.2 Native SNF API

The SNF API is also available for applications that require tighter integration with Sniffer. When used in its simplest form, the library resembles **Libpcap/ WinPcap** in that the implementation expects a single thread to make successive calls to a receive function (**snf_ring_recv**) to obtain the next available packet. Under a more advanced form, Sniffer implements a variation of the Receive- Side Scaling (RSS) feature that is present in some 10-Gigabit Ethernet drivers.

HTML documentation for the SNF API is available with the Sniffer distribution in **/opt/snf/share/doc/index.html**.

1.3 New Features in the Sniffer v3.0.26 Software

See the SNF v3.0.26 Release Notes (SNFv3.0.26_ReleaseNotes.pdf) for details on the new features and bug fixes included with this release.

Chapter 2 Installation

Use of the Myricom High-Speed, Low-Latency networking solutions requires the installation of one or more Network Adapters and Sniffer software. Once installed, the software must be activated via appropriate license keys.

2.1 Quickstart - Installation Overview

The installation process consists of three major areas:

1. Physical installation of Network Adapter(s) into host computer(s) and connection of appropriate cable.
2. Installation of Myricom Driver Software products and configuration of run-time operation.
3. Configuration of Software License Keys on each host.

Myricom products are designed to be compatible with prevailing industry standards and typically install quickly without significant user effort.

Generalized instructions about each of the three installation areas can be found in the following sections of this document. These instructions will provide for a successful installation for most current computing platforms.

NOTE: Sniffer contains a **myri_snf** driver that must be run instead of the default **myri10ge** driver typically associated to Myri-10G Ethernet network adapters.

Detailed differences in hardware and networking requirements for individual sites is beyond the scope of this manual and customers should contact ARIA Technical Support if they experience difficulty configuring Myricom solutions in their unique environments.

2.2 Hardware Installation Instructions

Installation of a Network Adapter card is a straight-forward process, as follows:

1. Shutdown host system applications and Operating System.
2. Turn off computer power and unplug power cord.

Open the computer case and locate the PCI-Express expansion card slots on the motherboard. The motherboard may have either PCIe 3.0 (“Gen3”), PCIe 2.0 (“Gen2”), or PCIe 1.1 (“Gen1”) PCI-Express slots. Take care not to be confused with legacy PCI card slots which have different physical dimensions and electrical specification (see Figure 2). For best performance, the network adapter should be placed in a slot which supports an 8-lane PCI-Express link (x8). To verify the hardware installation for maximum performance, please read **Section 7.1: Hardware Installation/Performance**.

Some systems will have longer x16 PCIe slots which may also support short x8 cards (verify motherboard specifications for compatibility as mechanical fit alone does not guarantee electrical operation). Note that not all x8 connectors are actually wired as x8 slots; check motherboard specifications if you are unsure. Check that there is a free slot, then remove the mounting screw of the protective bracket plate (if present) covering the selected slot and set aside the plate. If there are no free slots then it will be necessary to remove an unneeded card to make room for the network adapter.

Figure 2. PCI-Express x1 (yellow), PCI-Express x8 (green), and Legacy PCI (white) Card Slots

3. CAUTION: When handling network adapters, it is important to follow anti-static procedures to avoid accidently damaging integrated circuits on the card due to static electric discharge. Avoid working in damp areas or walking on carpet while transporting bare cards. It is recommended to use an anti-static wrist bracelet.

   Alternatively, hold the card carefully by its metal mounting bracket and gently touch the metal case of the computer power supply with your free hand to safely ground any static charge before proceeding.

4. Locate the shipping package for the Myricom Network Adapter and remove the sealed protective sleeve containing the card. Open the seal on the protective sleeve and remove the adapter (see Figure 3). Be careful to avoid touching the gold connectors on the bottom edge of the card.

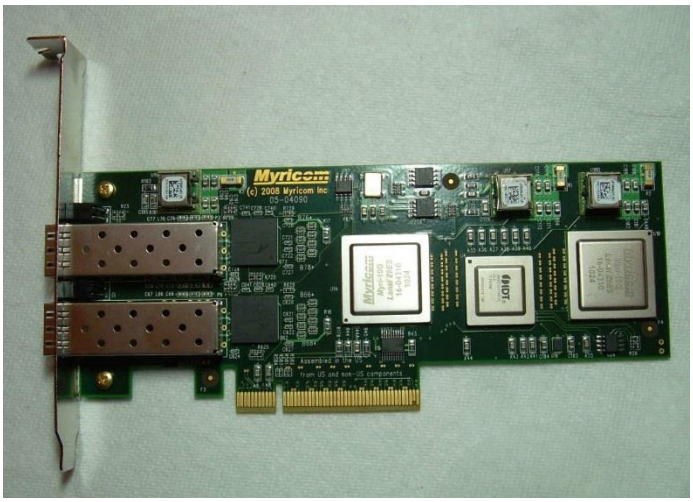


Figure 3. Typical Myri-10G Card

5. Move the card to the empty slot in the computer case. Line up the gold PCI connectors and indexing tab on the edge of the card with the PCI-Express slot, ensuring that the ports and mounting bracket are facing the back of the computer (see Figure 4). Carefully press the card into the slot, making sure to press evenly on both edges until the card is firmly seated. When the card is correctly installed it will “click” into place.

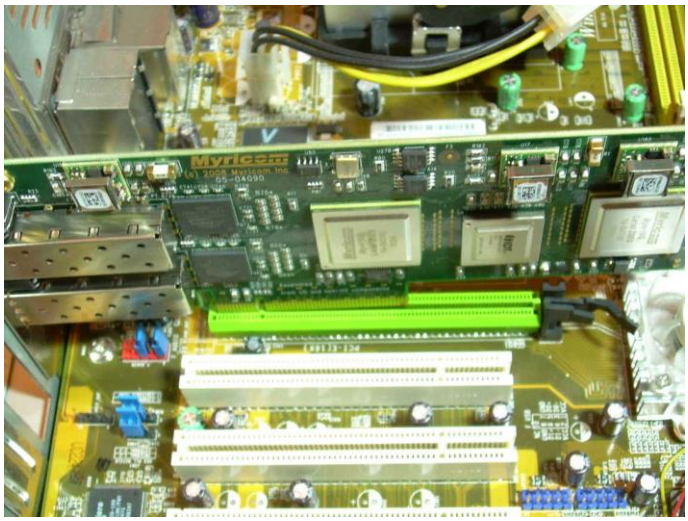


Figure 4. Card Insertion

6. Secure the card to the computer case with the screw removed from the

bracket plate previously (see Figure 5). To verify the hardware installation for maximum performance, please read **Section 7.1: Hardware Installation/ Performance**.

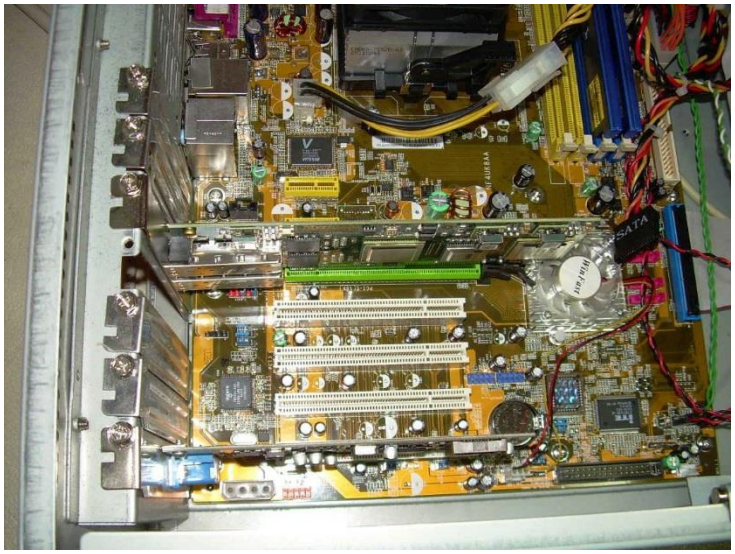


Figure 5. Card Fully Seated

7. Close the computer case and attach the power plug.
8. Attach appropriate cable(s) to the exposed port(s) on the faceplate of the network adapter, taking care not to kink the cable(s).

   Physical network adapter installation is now complete.
9. For the 10G-PCIE2-8C2-2S-SYNC adapter, the adapter will additionally need to be connected to a timecode source using a 50-ohm coaxial cable (such as RG58) with an SMB plug on one end to connect to the 10G-PCIE2-8C2-2S- SYNC adapter, and the other end should be whatever is needed to connect to the timecode generator. Most timecode sources use BNC connectors, so for a source with this connector, the cable would need a BNC male connector.

10G-PCIE2-8C2-2S-SYNC network adapters connect to CDMA or GPS IRIG-B00x timecode sources. These IRIG-B00x timecode outputs may also be called "unmodulated" or "DCLS" or "TTL into 50 ohm" outputs. Units which do not have enough outputs for the number of adapters to be connected will need a fanout buffer to drive multiple outputs from a single input.

By default, the software assumes that the 10G-PCIE2-8C2-2S-SYNC adapter is connected to a timecode generator and timesource (hardware) timestamping is desired. If the SYNC adapter is not connected to a timecode generator, and host timestamping will be utilized, the **myri_snf** driver must be loaded with the **myri_timesource=0** load-time option. Otherwise, Sniffer will return zero for the timestamps. For details, please read **Section 2.3: Sniffer Software Driver Installation Instructions.**

2.3 Sniffer v3.0.26 Software Driver Installation Instructions

The Sniffer v3.0.26 Software is distributed in a single package file containing:

- A special Network Adapter driver to replace the normal Ethernet driver
- A dynamic library of software modules
- Test programs to demonstrate Sniffer functionality

Begin the installation by first obtaining a copy of the appropriate Sniffer software package for your system by following the steps detailed below.

1. Using your normal Internet browser, navigate to the ARIA Download Center: https://www.ariacybersecurity.com/support/downloads/.
2. Scroll down to the SNFv3 section.
3. Click on the link to download the package for your OS. The downloaded file format may be either .rpm or .tgz for Linux or .msi for Windows. A different installation process is required for each format and is explained in the four subsequent sections.

2.4 Linux Installation Process for Binary RPM (.rpm)

This section details the installation process for Linux hosts for RPM-based systems running a standard derivative of a RedHat-based Linux distribution (RHEL, Centos, Fedora).

The RPM is a binary RPM with the exception of the kernel module which is distributed as source. When the RPM is installed, the kernel module is compiled against the currently running kernel. For a successful installation, the **kernel- devel** package that corresponds to the currently running kernel (as reported by **uname -a**) must be installed. The installation will also require the **gcc** package for kernel module compilation and the **jdk** package for java runtime support. Users with alternative installations may prefer the binary tarball method described next.

1. When the download of the file is complete, open a terminal window and type the following command using the actual <version_info> of the downloaded file name:

   ```
   # rpm -i myri_snf-<version_info>.x86_64.rpm
   ```

   As a result of a successful installation, the package will be installed in **/opt/ snf** and the kernel module is automatically started. Upon next reboot, the kernel module will also be automatically started from /etc/init.d.

2. (Optional) The --prefix option to rpm can be used as the package is relocatable.

3. (Optional): Manual stopping and starting of the kernel module can by controlled by using the **start** or **stop** commands with the **myri_start_stop** script:

   ```
   # /opt/snf/sbin/myri_start_stop start
   ```

   The basic installation of Sniffer software is complete.

4. A Local Area Network address must be configured for each port of the Network Adapter card(s) before it can be used on the network. Manually assign the appropriate IP address on all ports on all host systems to establish the Sniffer-enabled network.

   The installation of the Sniffer-enabled Myri-10G Network Adapter is now complete.

Note that with some Linux distributions, the GUI Package Manager can be used to perform the package install rather than the command line used above.

Module Parameters (Optional)

To determine the list of optional module load-time parameters, type:

```
# modinfo /opt/snf/sbin/myri_snf.ko
```

If you are using SFP+-terminated copper “direct attach” cables with the Myri-10G network adapters and you experience link up/down connectivity issues or bad crc errors, please read about the load-time option **myri_serdes_mode** in **Section 7.2 Hardware Cabling Connectivity Issues**.

Timestamping Support

Timestamping support in Sniffer is enabled by default. For details, please read **Section 7.6 Network Adapter Timestamps**.

In the Sniffer release with SYNC adapter support, three extra parameters were added to the SNF module:

- **myri_timesource**

  By default, the Sniffer software assumes that the 10G-PCIE2-8C2-2S-SYNC adapter is connected to a timecode generator (**myri_timesource=1**) and that timesource (hardware) timestamping mode is desired. If the SYNC adapter is not connected to a timecode generator, and host timestamping will be utilized, the **myri_snf** driver must be loaded with the **myri_timesource=0** load-time option. Otherwise, Sniffer will return zero for the timestamps.

  On Linux, the instructions to set **myri_timesource=0** and thus switch to using host timestamps on the SYNC adapter are:

  ```
  # /opt/snf/sbin/myri_start_stop start myri_timesource=0
  ```

  On non-SYNC adapters, the default is **myri_timesource=0** and host timestamping.

- **myri_timesource_loss_mode**

  Set **myri_timesource_loss_mode=1** for free-running (default), and set value to 0 for zeroing timestamps.

- **myri_timesource_pps**

  Set **myri_timesource_pps=0** to disable the exporting of a pps interface that can be used by ntp.

For additional details on timestamping, please read **Section 7.6 Network Adapter Timestamps**.

Arista Timestamping Support (Optional)

Timestamping support with Arista 7150 series 10GbE switches is available in Sniffer v3 and later. Sniffer supports the alternate timestamping provided by the Arista 7150 Switch EOS. For more details, please read:

https://eos.arista.com/timestamping-on-the-7150-series/

and https://www.arista.com/assets/data/docs/Manuals/EOS-4.20.1F-Manual.pdf

The technique supported by Sniffer is the append (before-fcs in the Arista terminology), which will add 4 bytes at the end of each packet, as well as extra packets to describe the time references for computations to be made (keyframes in the Arista terminology).

Arista timestamping support is disabled by default.

(**MYRI_ARISTA_ENABLE_TIMESTAMPING=0**).

- **MYRI_ARISTA_ENABLE_TIMESTAMPING = 0,1**

  MYRI_ARISTA_ENABLE_TIMESTAMPING enables the support for Arista timestamping, and is activated as soon as the process receives its first keyframe. The Arista switch does not guarantee delivery of keyframes when it is under heavy load. It is possible for the arrival of the first keyframe which enables time stamping to begin to be delayed. Also, under these conditions, the arrival interval of keyframes may be greater than the timestamp wrap time of approximately 6.1 seconds. In this case Sniffer will continue to approximate timestamps until a new keyframe arrives. Therefore, a lapse of time may be needed to receive the first keyframe. By default, **MYRI_ARISTA_ENABLE_TIMESTAMPING=0.**

  On Linux, if you are using 10G-PCIE2-8C2-2S-SYNC adapters, hardware timestamping must be disabled. Execute the following command:

  ```
  # /opt/snf/sbin/myri_start_stop start myri_timesource=0
  ```

  To verify that Arista timestamping has been enabled, the following message will appear in the kernel log.

  ```
  myri_snf: INFO: ** Arista timestamping [on] supported modes: append (before-fcs)
      * features: keyframe check min.rate [on] kf check ptp [on] kf kernel dup [off]
  ```

- **MYRI_ARISTA_PARAM_KF_DST_IP = <IP ADDRESS>**

  MYRI_ARISTA_PARAM_KF_DST_IP must be set to the IP address of the keyframe generator (the Arista switch). By default, the address is set to 1.1.1.0. For more details, please consult

  https://www.arista.com/assets/data/docs/Manuals/EOS-4.20.1F-Manual.pdf

The following two environment variables are optional for Arista timestamping support.

1. **MYRI_ARISTA_PARAM_KF_CHECK_PTP = 0,1**

   MYRI_ARISTA_PARAM_KF_CHECK_PTP is an optional environment variable to check if the PTP synchronization is working on the switch. By default, **MYRI_ARISTA_PARAM_KF_CHECK_PTP=1**.

2. **MYRI_ARISTA_PARAM_KF_CHECK_RATE = 0,1**

   MYRI_ARISTA_PARAM_KF_CHECK_RATE is an optional environment variable to check if the keyframes sent are sufficient to provide accuracy. By

default, **MYRI_ARISTA_PARAM_KF_CHECK_RATE=1**.

2.4.1 Uninstalling the Binary RPM

To uninstall the Linux Sniffer RPM, execute the following commands:

```
$ sudo /opt/snf/sbin/myri_start_stop stop
$ sudo rpm -e myri_snf
```

2.5 Linux Installation Process for Binary tarball (.tgz)

This section details the installation process for Linux hosts using the binary tarball.

1. When the download of the file is complete, first remove any existing /opt/snf directory, then type the following commands using the actual <version_info> of the downloaded file name:

```
# cd /opt
# gunzip -c <path>/myri_snf-<version_info>.x86_64.tgz | tar xvf -
# mv myri_snf-<version_info>.x86_64 snf
# cd /opt/snf
# sbin/rebuild.sh
```

2. Now load the drivers by typing:

```
# /opt/snf/sbin/myri_start_stop start
```

   The basic installation of Sniffer software is complete.

3. If you are running Ubuntu 14.04 LTS and want the myri_start_stop script installed in the init scripts area, enter the following:

Running Ubuntu 14.04 LTS

```
# cd /opt/snf
# sbin/myri_local_install
# update-rc.d myri_start_stop start 30 2 3 4 5 . stop 70 0 1 6 .
```

   You can now run system myri_start_stop start or service myri_start_stop stop to start or stop the Sniffer v3 driver.

4. If you are running Ubuntu 16.04 or later, enter the following:

Running Ubuntu 16.04 or Later

```
# cd /opt/snf
# cp sbin/myri_start_stop.service /etc/systemd/system
# systemctl enable myri_start_stop.service
```

   You can now run systemctl start myri_start_stop.service or systemctl stop myri_start_stop.service to start or stop the Sniffer driver.

5. A Local Area Network address must be configured for each port of the Network Adapter card(s) before it can be used on the network. Manually assign the appropriate IP address on all ports on all host systems to establish the Sniffer-enabled network.

   The installation of the Sniffer-enabled Myri-10G Network Adapter is now complete.

Module Parameters (Optional)

To determine the list of optional module load-time parameters, type:

```
# modinfo /opt/snf/sbin/myri_snf.ko
```

If you are using SFP+-terminated copper “direct attach” cables with the Myri-10G network adapters and you experience link up/down connectivity issues or bad crc errors, please read about the load-time option **myri_serdes_mode** in **Section 7.2 Hardware Cabling Connectivity Issues**.

Timestamping Support

Timestamping support in Sniffer is enabled by default. For details, please read **Section 7.6 Network Adapter Timestamps**.

In the Sniffer release with SYNC adapter support, three extra parameters were added to the SNF module:

- **myri_timesource**

  By default, the Sniffer software assumes that the 10G-PCIE2-8C2-2S- SYNC adapter is connected to a timecode generator (**myri_timesource=1**) and that timesource (hardware) timestamping mode is desired. If the SYNC adapter is not connected to a timecode generator, and host timestamping will be utilized, the **myri_snf** driver must be loaded with the **myri_timesource=0** load-time option. Otherwise, Sniffer will return zero for the timestamps.

  On Linux, the instructions to set **myri_timesource=0** and thus switch to using host timestamps on the SYNC adapter are:

  ```
  # /opt/snf/sbin/myri_start_stop start myri_timesource=0
  ```

- **myri_timesource_loss_mode**

  Set **myri_timesource_loss_mode=1** for free-running (default), and set value to 0 for zeroing timestamps.

- **myri_timesource_pps**

  Set **myri_timesource_pps=0** to disable the exporting of a pps interface that can be used by ntp.

For additional details on timestamping, please read **Section 7.6: Network Adapter Timestamps**.

Arista Timestamping Support (Optional)

Timestamping support with Arista 7150 series 10GbE switches is available in Sniffer v3 and later. Sniffer supports the alternate timestamping provided by the Arista 7150 Switch EOS. For more details, please read:

https://eos.arista.com/timestamping-on-the-7150-series/

and

https://www.arista.com/assets/data/docs/Manuals/EOS-4.20.1F-Manual.pdf

The technique supported by Sniffer is the append (before-fcs in the Arista terminology), which will add 4 bytes at the end of each packet, as well as extra packets to describe the time references for computations to be made (keyframes in the Arista terminology).

Arista timestamping support is disabled by default.

(**MYRI_ARISTA_ENABLE_TIMESTAMPING=0**).

- **MYRI_ARISTA_ENABLE_TIMESTAMPING = 0,1**

  MYRI_ARISTA_ENABLE_TIMESTAMPING enables the support for Arista timestamping, and is activated as soon as the process receives its first keyframe. The Arista switch does not guarantee delivery of keyframes when it is under heavy load. It is possible for the arrival of the first keyframe which enables time stamping to begin to be delayed. Also, under these conditions, the arrival interval of keyframes may be greater than the timestamp wrap time

  of approximately 6.1 seconds. In this case Sniffer will continue to approximate timestamps until a new keyframe arrives. Therefore, a lapse of time may be needed to receive the first keyframe. By default, **MYRI_ARISTA_ENABLE_TIMESTAMPING=0.**

  On Linux, if you are using 10G-PCIE2-8C2-2S-SYNC adapters, hardware timestamping must be disabled. Execute the following command:

  ```
  # /opt/snf/sbin/myri_start_stop start myri_timesource=0
  ```

  To verify that Arista timestamping has been enabled, the following message will appear in the kernel log.

  ```
  myri_snf: INFO: ** Arista timestamping [on] supported modes: append (before-
  fcs)
        * features: keyframe check min.rate [on] kf check ptp [on] kf kernel dup
                                                [off]
  ```

- **MYRI_ARISTA_PARAM_KF_DST_IP = <IP ADDRESS>**

  MYRI_ARISTA_PARAM_KF_DST_IP must be set to the IP address of the keyframe generator (the Arista switch). By default, the address is set to 1.1.1.0. For more details, please consult https://www.arista.com/assets/data/docs/Manuals/EOS-4.20.1F-Manual.pdf

The following two environment variables are optional for Arista

timestamping support.

1. **MYRI_ARISTA_PARAM_KF_CHECK_PTP = 0,1**

   MYRI_ARISTA_PARAM_KF_CHECK_PTP is an optional environment variable to check if the PTP synchronization is working on the switch. By default, **MYRI_ARISTA_PARAM_KF_CHECK_PTP=1**.

2. **MYRI_ARISTA_PARAM_KF_CHECK_RATE = 0,1**

   MYRI_ARISTA_PARAM_KF_CHECK_RATE is an optional environment variable to check if the keyframes sent are sufficient to provide accuracy. By default, **MYRI_ARISTA_PARAM_KF_CHECK_RATE=1**.

2.5.1 Driver Fails to Load – Rebuilding Driver

The sniffer kernel driver is built at install time to match the running kernel. If you install software updates which update the kernel, the next time you reboot, the sniffer driver may fail to load and you receive an invalid module format error:

```
# ./myri_start_stop start
Loading myri_snf
insmod: ERROR: could not insert module /opt/snf/sbin/myri_snf.ko: Invalid module format
```

To resolve this, you need to rebuild the driver:

```
# cd /opt/snf
# sbin/rebuild.sh
# sbin/myri_start_stop start
```

2.5.2 Uninstalling the Binary tarball (.tgz)

To uninstall the Linux Sniffer tarball (.tgz) from the standard location (/opt/snf), execute the following commands:

```
$ sudo /etc/init.d/myri_start_stop stop
$ sudo rm -rf /opt/snf/
$ sudo rm -f /etc/init.d/myri_start_stop
```

2.6 FreeBSD Installation Process for Binary TBZ (.tbz)

This section details the installation process for FreeBSD 9, 10 and 11 hosts.

1. When the download of the file is complete, open a terminal window and type the following command using the actual <version_info> of the downloaded file name.

   For FreeBSD 11

   ```
   # pkg add snf-<version_info>_FreeBSD_11_amd64.tbz
   ```

   For FreeBSD 10:

   ```
   # pkg add snf-<version_info>_FreeBSD_10_amd64.tbz
   ```

   Note that the installation for FreeBSD 10 is different from FreeBSD 9. There is no underscore '_' between the 'pkg' and 'add' commands.

   For FreeBSD 9:

   ```
   # pkg_add snf-<version_info>_FreeBSD_9_amd64.tbz
   ```

   The installation instructions are the same for FreeBSD 8, except that the package name is snf-<version_info>_FreeBSD_8_amd64.tbz.

   As a result of a successful installation, the package will be installed in /usr/local/.

2. Load the driver by typing:

   ```
   # /usr/local/opt/snf/sbin/myri_start_stop start
   ```

   Note: If you have configured **mxge** (Myri10GE driver) into your kernel, you will need to remove **mxge** from your kernel config file. Similarly, if you are using **mxge** as a module you will need to remove it (kldunload if_mxge). You should also remove ifconfig_mxgeX lines from /etc/rc.conf if you want to load the Sniffer driver at boot.

   To automatically load the driver at boot time, do the following:

   ```
   # cp /usr/local/opt/snf/sbin/myri_start_stop
   /usr/local/etc/rc.d/myri_start_stop
   ```

   Once the file above is in the location specified, add the following line to /etc/rc.conf

   ```
   myri_enable="YES"
   ```

3. (Optional): Manual stopping and starting of the kernel module can by controlled by using the **start** or **stop** commands with the **myri_start_stop** script:

   ```
   # /usr/local/opt/snf/sbin/myri_start_stop start
   ```

   The basic installation of Sniffer software is complete.

4. (Optional if not enabled at boot): A Local Area Network address must be

configured for each port of the Network Adapter card(s) before it can be used on the network. Manually assign the appropriate IP address on all ports on all host systems to establish the Sniffer-enabled network.

The installation of the Sniffer-enabled Myri-10G Network Adapter is now complete.

2.6.1.1 Uninstalling the Binary TBZ

To uninstall the FreeBSD Sniffer TBZ, execute the following commands.

For FreeBSD 11:

```
$ sudo /usr/local/opt/snf/sbin/myri_start_stop stop
$ sudo pkg delete snf-<version_info>_FreeBSD_11_amd64
```

For FreeBSD 10:

```
$ sudo /usr/local/opt/snf/sbin/myri_start_stop stop
$ sudo pkg delete snf-<version_info>_FreeBSD_10_amd64
```

Note that the uninstall for FreeBSD 10 is different than for FreeBSD 9 or 8. There is no underscore '_' between the 'pkg' and 'delete' commands.

For FreeBSD 9:

```
$ sudo /usr/local/opt/snf/sbin/myri_start_stop stop
$ sudo pkg_delete snf-<version_info>_FreeBSD_9_amd64
```

The uninstall instructions are the same for FreeBSD 8, except that the package name is snf-<version_info>_FreeBSD_8_amd64.

If the driver was automatically loaded at boot time, you will also need to do the following two steps.

Remove the start/stop script.

```
# rm /usr/local/etc/rc.d/myri_start_stop
```

Remove the line of text from rc.conf that enables it by default

```
# sed -ie '/myri_enable/d' /etc/rc.conf
```

2.6.2 Windows Installation

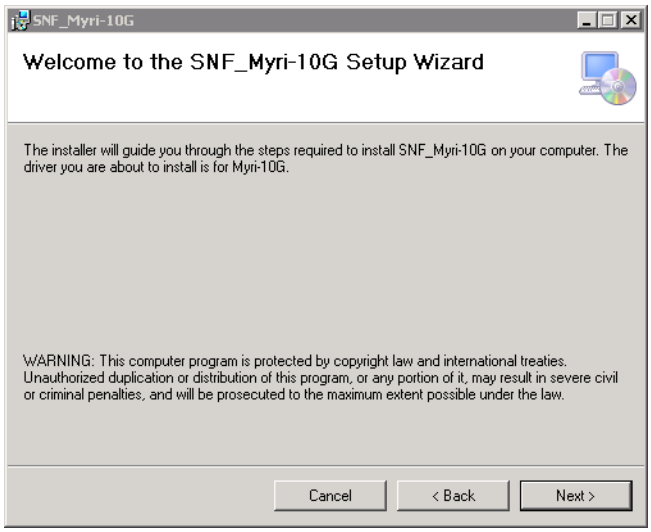
This section details the installation process for Windows hosts.

2.6.2.1 Installation using Windows Installer (.msi)

NOTE:

- Installation must be done from an account with administrator privileges and permissions.
- The .msi file must be on a local or regular network drive; a Remote Desktop- mounted device will not work.
- .NET 2.0 or later is required. If this is not installed, the installer will open a warning and provide a way to download it.

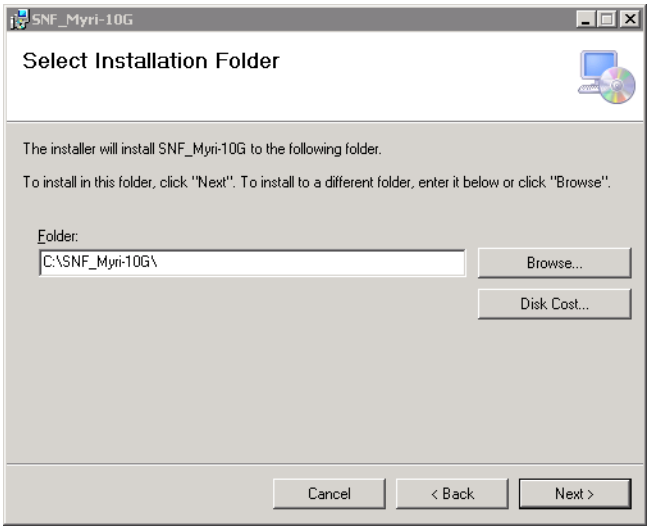
1. When the download of the file is complete, invoke the MSI installer with a GUI and the following splash screens will appear.



2. After clicking **Next**, step through these dialogue pages. License Agreement

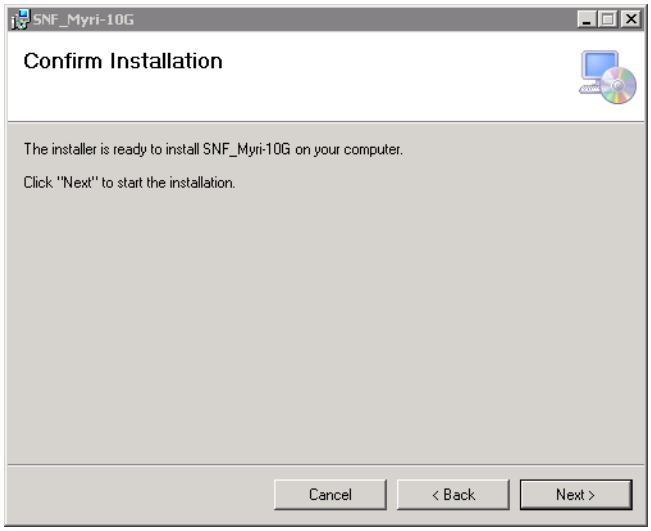
You must agree to the license to continue.

3. Select Installation Folder

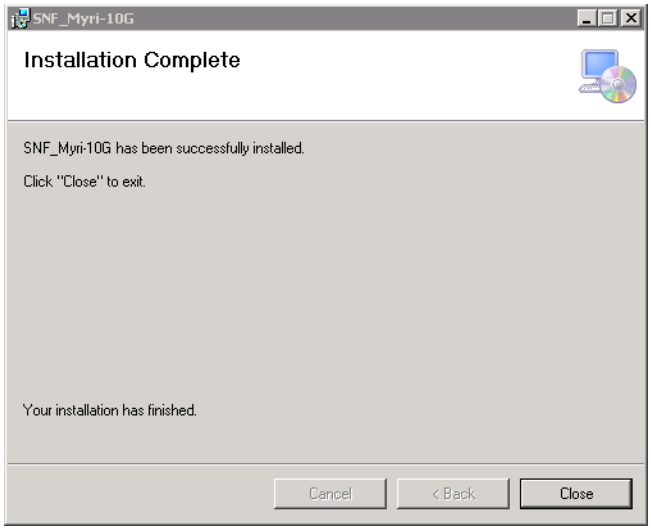


Specify the location to install the software.

4. You are ready to Install

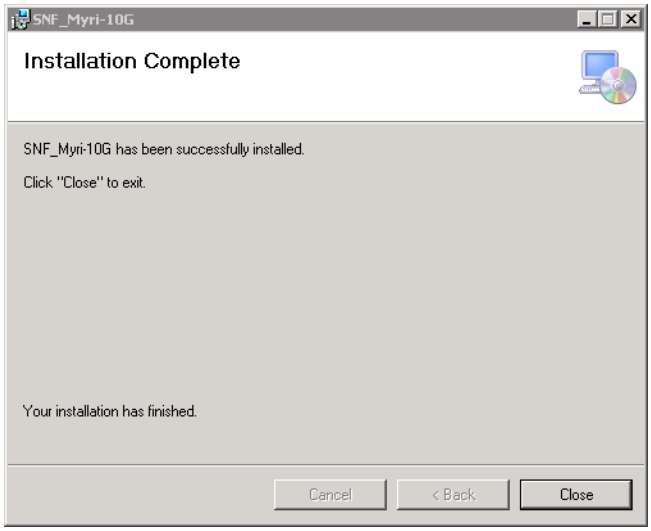


5. After confirming the installation location, click **Next** to install.



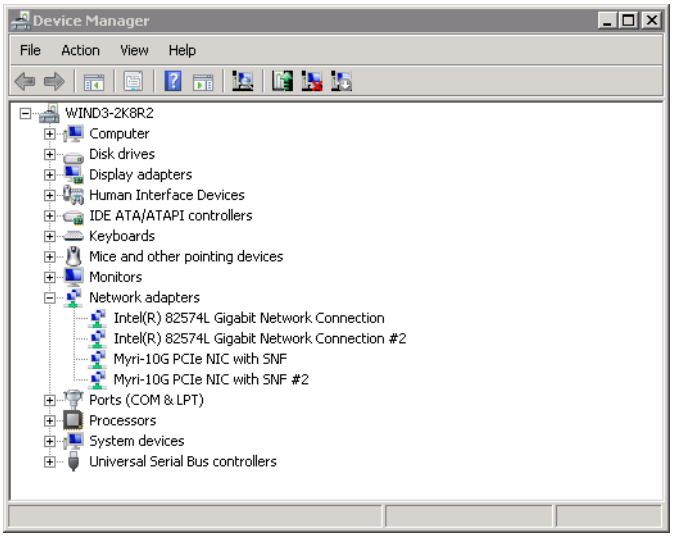
6. If a security alert like the one below appears during the installation, click on Install/Continue.

7. Finish Up
   After the installation wizard is complete, the computer may need to be restarted to complete the installation. If that is needed, there will be a note on the final page of the installer.



8. When finished, set the IP address and other network settings as explained on the Microsoft website.

2.6.2.2 Confirming the MSI Installation



Open the Device Manager (Start->Run->devmgmt.msc) to confirm that the software has been successfully installed. The software installation is successful when the text **Myri-10G PCIe NIC with SNF** appears under **Device Manager -> Network adapters.** (In this example, a two-port Myri-10G network adapter is installed in the machine so there are two entries for SNF.)

If an error has been encountered during the software installation, the output of **Device Manager -> Network adapters** will either show no SNF devices or a yellow exclamation mark.

If the MSI installation fails when running the MSI installer, please send ARIA Technical Support the log obtained by adding /log filename to the MSI install. For example,

```
c:\> snf-<version_info>*x64.msi /log
snf_install_log.txt
```

If you are using SFP+-terminated copper “direct attach” cables with the Myri-10G network adapters and you experience link up/down connectivity issues or bad crc errors, please read about the load-time option **myri_serdes_mode** in **Section 7.2 Hardware Cabling Connectivity Issues**.

2.6.2.3 Installing using Windows msiexec

If the MSI cannot be installed, the files can be extracted from the MSI without running any Custom Actions by using the following command line:

```
$ msiexec /a snf-<version_info>*x64.msi /qb TARGETDIR="C:\SNF_Myri-10G"
INSTALLLEVEL=5
```

Or, for default settings, use the following command line:

```
$ msiexec /i snf-<version_info>*x64.msi
INSTALLLEVEL=5
```

Afterwards, please use the 'Update Driver' Network Adapter option.

2.6.2.4 Deploying for Windows imaging

To deploy Sniffer for Windows imaging, the files can be extracted from the MSI by using the following command line:

```
$ msiexec /a snf-<version_info>.msi /qb TARGETDIR="C:\SNF_Myri-
10G"
```

2.6.2.5 Timestamping Support

Timestamping support in Sniffer is enabled by default. For details, please read **Section 7.6 Network Adapter Timestamps**.

In the Sniffer release with SYNC adapter support, three extra parameters were added to the SNF module:

- **myri_timesource**

  By default, the Sniffer software assumes that the 10G-PCIE2-8C2-2S-SYNC adapter is connected to a timecode generator (**myri_timesource=1**) and that timesource (hardware) timestamping mode is desired. If the SYNC adapter is not connected to a timecode generator, and host timestamping will

  be utilized, the **myri_snf** driver must be loaded with the **myri_timesource=0** load-time option. Otherwise, Sniffer will return zero for the timestamps.

  On Windows, the instructions to set **myri_timesource=0** and thus switch to using host timestamps on the SYNC adapter are:

  Create the following registry key (followed by a reboot):

```
REG ADD HKLM\SYSTEM\CurrentControlSet\services\snf /v myri_timesource /t
REG_DWORD /d 0
```

  The Windows EventViewer will then show:

```
myri_snf INFO: Timesource function disabled by registry entry
```

  On non-SYNC adapters, the default is **myri_timesource=0** and host timestamping.

- **myri_timesource_loss_mode**

  Set **myri_timesource_loss_mode=1** for free-running (default), and set value to 0 for zeroing timestamps.

- **myri_timesource_pps**

  Set **myri_timesource_pps=0** to disable the exporting of a pps interface that can be used by ntp.

For additional details on timestamping, please read **Section 7.6 Network Adapter Timestamps**.

2.6.2.6 Arista Timestamping Support (Optional)

Timestamping support with Arista 7150 series 10GbE switches is available in Sniffer v3 and later. Sniffer supports the alternate timestamping provided by the Arista 7150 Switch EOS. For more details, please read:

https://eos.arista.com/timestamping-on-the-7150-series/

and

https://www.arista.com/assets/data/docs/Manuals/EOS-4.20.1F-Manual.pdf

The technique supported by Sniffer is the append (before-fcs in the Arista terminology), which will add 4 bytes at the end of each packet, as well as extra packets to describe the time references for computations to be made (keyframes in the Arista terminology).

Arista timestamping support is disabled by default (**MYRI_ARISTA_ENABLE_TIMESTAMPING=0**).

- **MYRI_ARISTA_ENABLE_TIMESTAMPING = 0,1**

  MYRI_ARISTA_ENABLE_TIMESTAMPING enables the support for Arista timestamping, and is activated as soon as the process receives its first keyframe. The Arista switch does not guarantee delivery of keyframes when it is under heavy load. It is possible for the arrival of the first keyframe which enables time stamping to begin to be delayed. Also, under these conditions, the arrival interval of keyframes may be greater than the timestamp wrap time of approximately 6.1 seconds. In this case Sniffer will continue to approximate timestamps until a new keyframe arrives. Therefore, a lapse of time may be needed to receive the first keyframe. By default, **MYRI_ARISTA_ENABLE_TIMESTAMPING=0.**

  On Windows, if you are using 10G-PCIE2-8C2-2S-SYNC adapters, hardware timestamping must be disabled. Create the following registry key (followed by a reboot):

  ```
  REG ADD HKLM\SYSTEM\CurrentControlSet\services\snf /v
  myri_timesource /t
  ```

  To verify that Arista timestamping has been enabled, the Windows EventViewer will then show:

  ```
  myri_snf: INFO: ** Arista timestamping [on] supported modes: append (before-
  fcs)
        * features: keyframe check min.rate [on] kf check ptp [on] kf kernel dup
                                          [off]
  ```

- **MYRI_ARISTA_PARAM_KF_DST_IP = <IP ADDRESS>**

  MYRI_ARISTA_PARAM_KF_DST_IP must be set to the IP address of the keyframe generator (the Arista switch). By default, the address is set to 1.1.1.0.
  For more details, please consult
  https://www.arista.com/assets/data/docs/Manuals/EOS-4.20.1F-Manual.pdf

The following two environment variables are optional for Arista timestamping support.

1. **MYRI_ARISTA_PARAM_KF_CHECK_PTP = 0,1**

   MYRI_ARISTA_PARAM_KF_CHECK_PTP is an optional environment variable to check if the PTP synchronization is working on the switch. By default, **MYRI_ARISTA_PARAM_KF_CHECK_PTP=1**.

2. **MYRI_ARISTA_PARAM_KF_CHECK_RATE = 0,1**

   MYRI_ARISTA_PARAM_KF_CHECK_RATE is an optional environment variable to check if the keyframes sent are sufficient to provide accuracy. By default, **MYRI_ARISTA_PARAM_KF_CHECK_RATE=1**.

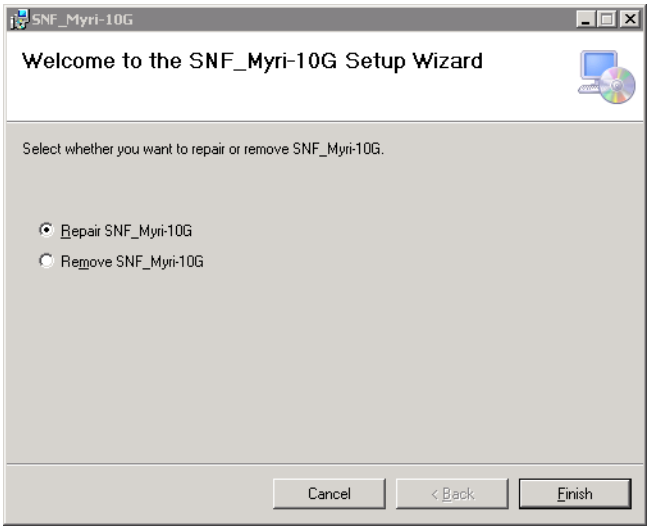
2.6.2.7 Uninstalling on Windows

The recommended method to uninstall the Sniffer software on Windows is using the MSI installer.

Using the Installer

If the software was installed using the installer (.msi), running it again will include an option to uninstall. Select that option and there will be two choices: **Repair** and **Remove**. Select **Remove** and click **Finish**. When the software has been removed, an "Installation Complete" message will appear.

Alternatively, if the Sniffer installation has been corrupted in some manner (for example, a file was accidentally deleted), you can select the **Repair** option to reinstall the software.



Manual Removal

It is possible to uninstall the software manually; however, this method should only be considered if the MSI uninstall method was not successful.The steps must be performed in this order.

1. Open Device Manager (Start->Run->devmgmt.msc), and uninstall all Myri-10G Adapters.
2. **Remove INF and PNF files.** This procedure removes all references to the driver and prevents a stale file from being used in the future.
3. Open a command window (Start->Run->cmd).

```
cd %windir%\inf
findstr Myricom oem*.inf
```

4. **Delete all Myricom oem files**. E.g., if oem6 was found with the findstr,

   ```
   del oem6*
   ```

   You should delete any pnf file as well, so use the above wildcard to remove inf and pnf files.

5. **Delete sys and dat files**

   ```
   cd %windir%\system32\drivers\
   del myri_snf.sys
   ```

6. Restart the machine

   ```
   shutdown /r /t 0
   ```

Once the machine reboots, Windows will try to find the driver for the Myri-10G adapters. It should fail because you deleted the inf and pnf oem files.

2.6.3 Installation in a Virtualized Environment (VMware)

Sniffer 2.0.7 or later is required to install the software in a virtualized environment. The following instructions detail the installation of Sniffer with VMware's ESXi 5.0.0 hypervisor.

Consult your motherboard manual and enable the VT-d (Virtualization Technology for Directed I/O) option in your BIOS.

Login to your ESXi 5 host using vSphere Client and click on the Configuration Tab.

- Under the Hardware Section, select Advanced Settings and the DirectPath I/O Configuration Menu appears.
- Click on Edit and a pop-up window appears to allow marking of devices for passthrough.
- Check the box for the Myricom 10G-PCIE-8B device that will be passed through to the Guest Virtual Machine.
- Click on the OK button to close the pop-up window.
- For adapter models 10G-PCIE2-8B2 ONLY (not single-port 8B and not -8C2-*):

  ACS(Access Control Services) check must be disabled:

  – Select ESXi 5 host in your inventory and click on the Configuration Tab.
  – Under the Software Section, click on Advanced Settings and an Advanced Settings pop-up window appears.
  – Click the VMkernel entry and scroll down to the entry that says VMkernel.Boot.disableACSCheck and click the checkbox to disable the ACS check.
  – Click on the OK button to close the pop-up window.
- Reboot the ESXi 5 host to enable the use of the DirectPath I/O devices. The

DirectPath I/O devices are now available for use by the Guest VMs.

- Power off the Guest VM.
- Select Getting Started Tab
- Click on Edit virtual machine settings.
- Under the Hardware Tab, click on the Add button.
- Select PCI Device and then click on the Next button.
- From the drop-down menu, select the physical PCI/PCIe device (06:00.0 | Myricom 10G-PCIE-8B) and click on the Next button.
- Click on the Finish button.
- Click on the OK button to update the Guest VM settings.
- Power up the Guest VM.

Install the Sniffer software according to the instructions for the Operating System installed inside the Guest VM.

Licensing note: The **myri_license** tool cannot successfully write licenses to the adapter when it is used within a guest. If a new license needs to be written to an adapter, the tool must be used within a 64-bit non-virtualized environment (such as a Linux boot CD).

Important Note:

ESXi 5.x has an MMIO address limitation of 42-bits for passthrough devices. The Myri-10G network adapter can support BIOSes that map our device into a 64-bit BAR0 address. If the BIOS maps the Myri-10G network adapter into an address above 42-bits, ESXi will fail (Error Type: **GenericVmConfigFault**) to start the VM that contains the passthrough device. Also, after the failure to start, if you login to the ESXi hypervisor host and type the command "dmesg", you will see an entry as follows:

```
2014-03-06T00:22:47.347Z cpu20:32874)WARNING: IOMMU: 1978: Unable to
unset device 84:00.0 since it hasn't been set.
```

Where the device number(06:00.0) should match the PCI Passthrough device being used in your Virtual Machine configuration.

To work around the ESXi address limitation, please flash the EEPROM firmware on the Myri-10G network adapter to limit its request for address space into the 32- bit region. You must flash the EEPROM firmware using a LiveCD or on another bare-metal machine.

To flash the EEPROM firmware on the adapter, download the **myri-tools-linux.tar.gz** file from the “Toolkit-LANai” group on the ARIA Customer Portal and untar the file. Execute the command to reflash the EEPROM firmware on all Myri-10G network adapters on this machine.

```
# bin/ze-upgrade -B myri-eeprom-1.4.57.mcp2 [reboot]
```

2.7 License Key Configuration

A separate license key will be provided by ARIA Customer Support for each purchased network adapter and related licensed software products. A one-time activation step must be performed to record the license key information in the flash memory of the network adapter. This step ensures proper functioning of all licensed features on that card and its use with related software products.

If you purchased Sniffer v3 licenses at the same time as your Myri-10G network adapters, the licenses will be pre-installed on the adapters and you should skip the **myri_license -f** procedure in this section of the document. If you are installing Sniffer on an adapter that was not purchased with the Sniffer licenses pre-installed, then you must install the license keys in the system using the **myri_license -f** procedure below. Similarly, if you purchased Sniffer 2.x licenses and are now upgrading to Sniffer v3, you will need to install new license keys.

Note that the purchase of a Sniffer v3 license is required to use the Sniffer v3 software. A Sniffer 2.0 or 1.0 license key is not valid for the Sniffer v3 software. A Sniffer 2.0 or 1.0 license may be upgraded to a Sniffer v3 license. Contact mailto:ariasales@ariacybersecurity.com for details.

2.7.1 License Key File

A license key file consists of one record for each network adapter. The file is a simple text format that can be viewed with any common text editor. Each record consists of an encrypted key which is locked to a specific network adapter serial number, followed by a string describing the licensed features enabled for that card.

The following is a sample license key file.

```
569f-f54d-1fb3-f194:1:123456:SNF:V2 # s/n 123456
6471-6557-e163-954a:1:123456:SNF:V2.3 # s/n 123456
# limited time trial evaluation of new product
c93b-9fc0-3570-d21b:1:123456:SNF:V2:T1299100709 # s/n 123456 (expires mm/
dd/yyyy)
```

Any records with a leading “#” are ignored and can be used to record any local site comments.

2.7.2 License Key Activation Process

Normally an email will be sent from ARIA Customer Support to the customer containing the list of license keys for the products and network adapter serial numbers requested. The license records will be delimited in the email as follows.

```
.
.
.
====  start  license  file
====
    .
    .
    .
===  end  license  file
===
.
.
.
```

Simply copy the text license records between the file delimiters comments in the email and paste into a new file using a text editor. Save the file with any file name, (represented below as <license_file>) in a convenient location.

Open a terminal window as user “root” on Linux. Now run the following command, supplying the assigned file name.

```
#/opt/snf/sbin/myri_license -f <license_file>
```

The application **myri_license** should be present in the sbin directory of the product(s) you have previously installed (e.g., **/opt/snf/sbin**)

This procedure will program the license keys into the network adapters installed in the host on which you run the **myri_license** command. **myri_license** must be run on each machine that contains Myri-10G Network Adapters to be licensed, but a single file may be used to hold all licenses for convenience.

To verify that the software license(s) have been successfully installed, run the following command:

```
$/opt/snf/bin/myri_nic_info
```

If any of the software licenses are listed as invalid, additional diagnostic information may be obtained by examining the kernel log output (dmesg) or running the command:

```
$ /opt/snf/bin/myri_nic_info --license
```

Important note:

To determine the license key string(s) currently programmed on an adapter, you can either run the **myri_license** command without any arguments, or the **myri_info** command without any arguments, and it will return the license key.

If you accidentally overwrite a license key, it is possible to obtain the missing license key string from **ARIA Technical Support.** Please include the six-digit serial number (or the MAC address) for the network adapter in your inquiry.

2.7.3 License Key Upgrade Process

When licenses have been purchased for new features to be used with existing network adapters, it will be necessary to rerun **myri_license** with an updated license file on the host system containing the network adapters.

Just add the new license key records to your local license file using a text editor and rerun the **myri_license** command to activate the new features.

2.7.4 Multiple License Keys

If you have purchased both DBL and Sniffer licenses for the same Myri-10G network adapter(s), both licenses can exist concurrently on the adapter and/or in the myri_license license file. However, only one driver can be loaded at a given time on a specific network port on the adapter. The driver load-time option **myri_bus** or **myri_mac** can be used to selectively load DBL or Sniffer (or Myri10GE) on a specific network port on the adapter. Contact ARIA Technical Support for details.

Before attempting to run Sniffer applications, it should be verified that the Sniffer driver is loaded on the specific adapter's network port. To verify that the Sniffer driver is loaded, run either the **/opt/snf/bin/myri_nic_info** command or the **/opt/snf/sbin/myri_info** command and examine the **Running MCP** section of the output for text that references **snf-<version>**. Otherwise, if the adapter has two ports or there are multiple adapters installed on the same hosts, there may be confusion with the wrong driver loaded on the wrong adapter port at a given time.

2.7.5 License Key Portability

Once a network adapter has been activated using the appropriate license key, it is permanently enabled with those features. The card may be moved to other host systems if desired and will continue to operate.

2.7.6 License key maintenance for defective card (RMA Process)

In the rare event of failure of a network adapter, contact ARIA Technical Support to obtain a replacement card using the Return Merchandise Authorization (RMA) process. A new license key record will be provided with the replacement network adapter.

Just add the new license key record(s) to your local license_file using a text editor and rerun the myri_license command to activate the replacement network adapter.

Note that it is not necessary to delete the original license key record in the file as it will be silently ignored.Running basic tests to verify Sniffer installation.

2.8 Running basic tests to verify Sniffer installation

Sniffer software installation is normally completed without difficulty on the first attempt. However, it is recommended that some simple tests be performed to verify Sniffer operation on your systems and provide basic familiarization with normal operation.

On Linux Sniffer v3, test / example programs are available from:
/opt/snf/bin/tests of the install directory in binary form and in **share/examples** in source form.

On Windows Sniffer v3, these tests / example programs are available in binary and source form from the **[INSTALL_DIR]\bin\tests** directory.

> **Note**: The undocumented example programs in this directory are provided as additional coding examples, and no instructions for their use are provided.

The following test / example programs demonstrate different aspects of the SNF API and how to use its features.

Many of the Sniffer test/example programs require a **-p <port number>** as a command-line argument. The port number refers to the physical PHY connector port on the network adapter. The ports are enumerated starting at 0. If the network adapter has two physical ports, the port closest to the PCI connector is assigned port 0 and the other port is assigned port 1. Port 0 also corresponds to the lower MAC address of a two-port adapter. If there are multiple adapters installed in the server, the port numbers for the successive adapters are assigned according to the order in which the adapters are detected by the BIOS. Refer to the output of **myri_nic_info -B** to determine the port numbering.

Invoke the test program with the option **--help** to obtain the usage summary of the command-line arguments.

2.8.1 snf_simple_recv.c:

Simple example program demonstrating how to receive packets using the Sniffer API.

Usage: ./snf_simple_recv [-v] [-t] [-p <port number>] [-p] [-n <num pkts>] [-d <ring sz>] [-S <snap len>]

Command-line [options]:

```
[ -v ] verbose.
[ -t ] print periodic statistics.
[ -p <port number> ] Myri-10G port number to use.
[ -T <timeout> ] Wait for at most <timeout> milliseconds between calls, 0 never blocks.
[ -n <num pkts> ] number of packet to receive (default: 0 - infinite).
[ -d <ring sz> ] size of the receive ring in bytes.
[ -S <snap len> ] display first <snap len> bytes of each packet.
```

```
[ -R <port number> ] reinject every received packet on port <portnum>
[ -N ] pass every received packet to netdev.
```

2.8.2 snf_multi_recv.c:

Example program demonstrating how to receive packets with multiple rings using the Sniffer API. This program also supports packet borrowing (leaving packets on the ring).

Note: If borrowing packets, make sure to monitor the queue size. If the queue is full, packets will be dropped.

Usage: ./snf_multi_recv –w <num_workers> [:core1,core2,…coren] [options]

Command-line [options]:

```
[ -p <port number> ] Myri-10G port number to use.
[ -n <num pkts> ] number of packet to receive (default: 0 - infinite)
[ -t ] show periodic stats, every second
[ -V ] validate incoming packets
[ -v ] verbose, or -vv for very verbose.
[ -d <ring sz> ] size of the receive ring in bytes, or megabytes.
[ -D <nanosecs> add <nanosecs> of synthetic processing delay in packet
handling.
[ -W <msecs> ] timeout of <msecs> in blocking receive calls.
[ -M ] Consume all packets and advance the tail pointer.
[ -B ] Borrow a percentage of ring memory. See -P option for setting
percentage.
[ -P <percent> ] Set percentage of memory to borrow. Default is 80. Range is
1-99.
[ -S <snap len> ] display first <snap len> bytes of each packet.
[ -R <port number> ] reinject every received packet on port <portnum>.
 [ -A <app_id> ] set application ID, where <app_id> is any integer value except
-1.
```

2.8.3 snf_bridge.c:

Example program demonstrating how to use SNF to create a transparent bridge to analyze traffic on one device and replay it on another.

Usage: ./`snf_bridge [options] -b <...> [ -b <...> ... ]`

Command-line [options]:

```
[ -b <board_source>:<board_dest>:<num_rings>[:<0xcpumask>]
   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 <cpumask>
   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.
```

2.9 Diagnostic Tool Programs

The Sniffer distribution also provides diagnostic tool programs, in **sbin/** and **bin/**, as documented below. Since the **/sbin/** tools must be run as root, you must either add **/opt/snf/sbin/** to your **PATH**, or execute the command using the full path (e.g., **/opt/snf/sbin/myri_license**). The majority of these tools are only needed for obtaining diagnostic information for error reporting. There are undocumented tool programs in these directories which should only be run as instructed by ARIA Technical Support.

- **sbin/myri_bug_report:** (Linux only) A Linux diagnostic/ bug report script for ARIA Technical Support. For example:

  ```
  # /opt/snf/sbin/myri_bug_report > output.txt
  ```

  Please read **Chapter 5: Troubleshooting** for details.

- **sbin\myri_dmesg.ps1**: (Windows only) A diagnostic powershell script for ARIA Technical Support. Please read **Chapter 5: Troubleshooting** for details.

- **sbin/myri_info:** Provides hardware information (e.g., serial number, mac address) for the adapter, as well as the version of firmware running on the adapter.

  Usage:

  ```
  # /opt/snf/sbin/myri_info [options]
  ```

  Command-line [options]:

  ```
  [ -b <board_num> ] Only print info about card instance <board_num>.
  [ -v ] Verbose.
  [ -vv ]Very verbose.
  ```

- **sbin/myri_license:** Programs software license keys into the network adapter(s) installed in the host. Please read **Section 2.7 License Key Configuration** for details. myri_license specified with no arguments reports licenses currently loaded.

  Usage:

  ```
  # /opt/snf/sbin/myri_license [options]
  ```

  Command-line [options]:

  ```
  [ -b <unit> ]
      Is used to specify an adapter for reporting. Default is
      all adapters if -b is not specified.
  [ -c ] Clears/removes license(s) on the adapter.
  [ -f <license_file> ] Load licenses from <license_file> into all
  matching adapters.
  ```

- **sbin/myri_ptpv2d:** Runs the IEEE 1588 PTP daemon for the timestamping protocol in the Sniffer/Linux distribution. For a usage summary, please type:

  ```
  # /opt/snf/sbin/myri_ptpv2d -?
  ```

- **bin/myri_bandwidth**: Shows the instantaneous bandwidth of data going through a given network adapter, which is mostly useful when displayed at an interval (-i option).

  Usage:

  ```
  $ /opt/snf/bin/myri_bandwidth [options]
  ```

  Command-line [options]:

  ```
  [ -b <board_num> ] Board number or MAC address. Default: 0.
  [ -t <secs> ] Time interval (seconds). Default: 1.

  [ -k ] Keep running.
  [ -p ] Display counters for each port on multi-port adapters.
  [ -a ] Also display aggregated counters for each iteration.
  [ -A ] Only display aggregated counters for each iteration.

  [ -h ] Help.
  ```

- **bin/myri_counters**: Generates output for low-level network adapter counters. For details, please read **Appendix: Sniffer v3.0.26 Counters**.

  Usage:

  ```
  $ /opt/snf/bin/myri_counters [options]
  ```

  Command-line [options]:

  ```
  [ -p <board_num> ]
     Board number or MAC address. Default: 0.
  [ -c ]
     Clear/reset the counters.
  [ -q ]
     Quiet: display only non-zero counters.
  [ -i ]
     Display host interrupt counters.
  [ -x ]
     Expert: display all counters.
  [ -h ]
     Help.
  ```

- **bin/myri_dmabench**: .Verifies the PCI DMA read and DMA write bandwidth for the PCIe slot for which the adapter is installed. Please read **Chapter 5: Troubleshooting** for details.

  Usage:

  ```
  $ /opt/snf/bin/myri_dmabench [options]
  ```

Command-line [options]:

```
[ -b <board_num> ]
   Board number or MAC address. Default: 0.
[ -s <dma_size> ]
   DMA size. Default: 4096. Must be a power of two between 1 and 4096.
[ -r ]
   Test only read (send) DMA.
[ -w ]
   Test only write (recv) DMA.
[ -i <iters> ]
   Number of iterations. Default: 64.
[ -a ]
   Test all (power of 2) DMA sizes.
```

- **bin/myri_endpoint_info**: Shows which processes consume some adapter- level resources known as endpoints. It can be useful to know which process IDs are using Sniffer.

  Usage:

  ```
  $ /opt/snf/bin/myri_endpoint_info [options]
  ```

  Command-line [options]:

  ```
  [ -b <board_num> ]
     Board number or MAC address. Default: 0. [ -h ]
     Help.
  ```

- **bin/myri_nic_info:** Provides diagnostic information on the number of adapters installed, which driver is loaded, and the status of the software license(s) for each adapter.

  Usage:

  ```
  $ /opt/snf/bin/myri_nic_info [options]
  ```

  Command-line [options]:

  ```
  [ -B ]Display board numbers.
     [ -m ]Comma separated output (for machine parsing).

  [ -a ] Print for all known adapters.
  [ --license ]Print software licensing status for all known adapters.
  ```

- **bin/myri_port_failover**: A utility that can be used to control the failover policy on the 10G-PCIE-8B-2* adapters and the 10G-PCIE-8B-2I and 10G-PCIE-8B2-4I BladeCenter adapters. Please read **Chapter 5: Troubleshooting** for details.

Usage:

```
$ /opt/snf/bin/myri_port_failover [ -b<unit> ] [ -0 | -1 | -s | -r<0|1>
]
```

Command-line [options]:

```
[ -b <unit> ]
   Uses NIC/Controller rank #unit [Default is 0]. [ -0 ]
   Set P0 as primary link for failover purposes (P1 is backup only).
[ -1 ]
   Set P1 as primary link for failover purposes (P0 is backup only).
[ -s ] Symmetric failover (only switch link if current port is down).
[ -r <0|1> ]
   Disable/enable sending a rarp pkt on failover (to update
   switch tables)
```

2.9.1 Running Sniffer tests on Linux

Tests / example programs are available from **/opt/snf/bin/tests** of the install directory in binary form and in **share/examples** in source form.

Basic Test

As a simple test, we will run **snf_pktgen.c** to generate the packets for injection and run **snf_simple_recv.c** to receive the packets. This test is sufficient to verify correct software installation, the network adapter hardware connectivity, and the expected packet rate for the Sniffer software.

This test assumes that a Myri-10G network adapter has been installed and the Sniffer software loaded on both a "server" host and a "client" host, and that the default Myri-10G adapter port on the "server" is physically connected (via a cable) to the default Myri-10G adapter port on the "client". I.e., the server host and the client host are connected in a point-to-point (switchless) configuration. If both the "server" adapter and the "client" adapter have two physical ports, the default Myri-10G adapter port is port 0 (the port closest to the PCI connector and also the port with the lower MAC address), and it is assumed that port 0 of the “server” adapter is physically connected to port 0 of the “client” adapter.

Important Notes:

- On the Linux test system insure that **/opt/snf/bin/tests/** has been added to the current **$PATH** variable.
- Always start the **snf_simple_recv** process on the “client” before starting the packet generator (**snf_pktgen**) on the “server”.
- Always stop the packet generator (**snf_pktgen**) on the “server” before stopping the **snf_simple_recv** process on the “client”.

- Otherwise, if the packet generator is sending packets, but **snf_simple_recv** is not running, then the overflow counter (**Net overflow drop** in the **myri_counters** output) will be increasing. This situation occurs because there is no Sniffer program consuming the packets through the SNF API, and the packets are instead being delivered to the ethernet portion of the driver, which will attempt to drop them as fast as possible, but will not be able to sustain the packet rate.
- This test assumes that port 0 (if using a two-port adapter) of the "client" and "server" are connected point-to-point.

  On the “server” host system, open a command window and type the following command to generate 1 billion 60-byte packets:

  ```
  $ snf_pktgen -s 60 -n 1000000000
  ```

  On the “client” host system, open a command window and type the following command:

  ```
  $ snf_simple_recv -t
  ```

  and the following output will appear:

```
14843291 pkts (890597460B) in 1.001 secs (14827884 pps), Avg Pkt: 60, BW (Gbps):   7.117
14843663 pkts (890619780B) in 1.001 secs (14828242 pps), Avg Pkt: 60, BW (Gbps):   7.118
14841101 pkts (890466060B) in 1.001 secs (14825697 pps), Avg Pkt: 60, BW (Gbps):   7.116
14845703 pkts (890742180B) in 1.001 secs (14830250 pps), Avg Pkt: 60, BW (Gbps):   7.119
14842322 pkts (890539320B) in 1.001 secs (14826946 pps), Avg Pkt: 60, BW (Gbps):   7.117
14842292 pkts (890537520B) in 1.001 secs (14826902 pps), Avg Pkt: 60, BW (Gbps):   7.117
14843007 pkts (890580420B) in 1.001 secs (14827601 pps), Avg Pkt: 60, BW (Gbps):   7.117
14842392 pkts (890543520B) in 1.001 secs (14826942 pps), Avg Pkt: 60, BW (Gbps):   7.117
```

  The execution of this test will report for each second the packet rate (pps). The packet rate is 14.8 Mpps, or line rate for 60-byte packets.

2.9.2 Running Sniffer tests on Windows

On Windows, the Sniffer tests / example programs are available in binary and source form from the **[INSTALL_DIR]\bin\tests** directory.

If you would like to compile the test programs on Windows, please note that the Sniffer software is currently **x64 only**, and you will need to adjust your build architecture accordingly.

Compiling/linking test programs is as follows:

```
cl /I../../include/ snf_simple_recv.c getopt.obj ../../lib/snf.lib
```

Chapter 3 Packet Generation at Line Rate

3.1 snf_pktgen.c:

Example program demonstrating how to generate packets for injection.

Usage: ./`snf_pktgen [options]`

Command-line [options]:

```
[ -?, --help ] Prints help information and usage.
[ -v ] Print verbose output.
[ -p <config> ] Configure threads (max: 8), where
<config> = <board>[:<nthreads>[:<cpumask>]]
(default: 0:1)
[ -n <count> ] Number of packets to send per thread (default:
infinite)
[ -r <x.y> ] Replay at x.y Gbits/s (per thread, not aggregate)
[ -R <x.y> ] Replay at x.y Mpps (per thread, not aggregate)
[ -W ] Wait instead of busy-poll when injecting packets
[ -E <src mac>-<dst mac> ] Source-destination MAC address
[ -I <src ip>-<dst ip> ] Source-destination IP address
[ -s <size> ] Packet size (default: 60)
[ -S <src port> ] Source port
[ -D <dst port> ] Destination Port

    (-s/S/D only) For rand/seq ranges, specify  <min value> - <max
    value> : {R,S}
```

3.2 PCAP Packet Replay

The snf_replay tool has been designed to be able to replay 10G line rate, any size packets, by using the sniffer API to bypass the kernel. snf_replay can also preload the pcap file into memory to bypass the disk read speed limitation.

Tcpreplay is going to try and use software waits to time the transmit of PCAP packets, and depending on the packet size, may not keep up with the desired rate.

3.2.1 snf_replay.c:

Example program that uses SNF-level injection to replay a .pcap file.

Usage: ./`snf_replay [options] <file.pcap>`

Command-line [options]:

```
[ --help ] Prints help information and usage.
[ -v ] verbose.
[ -p <port> ] Myri10G port number to use
[ -n <count> ] Maximum number of packets to send from pcap file.
[ -t <nthreads> ] Number of threads to concurrently send same file
(max: 16).
[ -i <iters> ] Number of times to replay the file on each thread.
[ -T <vlan_id> ] Insert a vlan tag with vlan_id for packets without
VLAN tags.
[ -r <x.y> ] Replay at x.y Gbits/s (per thread, not aggregate)
[ -R <x.y> ] Replay at x.y Mpps (per thread, not aggregate)
```

```
[ -z ] Replay according to packet timestamp
[ -F ] Disable setting thread affinity
[ -m ] Read the pcap file into memory first. Note: Make sure that
your pcap file fits in available memory.
```

Note: The [-m] command-line option reads the pcap file into memory, and then replays it from memory instead of disk. This option allows it to achieve line rate (if your machine/memory is fast enough) for small packets. You must have enough available memory to accommodate the pcap file you are trying to replay.

Chapter 4 Using Sniffer v3.0.26 from Libpcap/SNF or WinPcap/SNF

Sniffer's packet capture capabilities can be leveraged through the **Libpcap** or **WinPcap** library or directly through the SNF API. Using a Sniffer-aware **Libpcap/WinPcap**, users reference a Myri-10G Network Adapter through its Ethernet interface name and can run existing Libpcap-dependent applications and continue to rely on Libpcap/WinPcap's portable interface. Going through the Sniffer interface instead of the kernel ensures a tighter level of integration with the Myri-10G Network Adapter by leveraging user-level receive mechanisms.

The easiest way to realize performance improvements using Sniffer is to re-link existing Libpcap/WinPcap applications against a Sniffer-capable Libpcap/ WinPcap (as included in the Sniffer distribution). The libpcap library distributed as part of the Sniffer software package is based on libpcap v1.3.0. The WinPcap library distributed as part of the Sniffer software package is based on WinPcap 4.1.3. When this Libpcap/WinPcap encounters a Sniffer- capable device, it uses the SNF API to obtain user-level zero-copy packets provided by the Myri-10G Network Adapter instead of the usual kernel-based approach.

Users can ensure that a Sniffer-aware **Libpcap/WinPcap** library is linked to the application by setting **SNF_DEBUG_MASK=3** in the environment when opening the Sniffer device **ethX**. Setting this variable causes the SNF API to output information when the Sniffer device (**ethX**) is opened by **Libpcap/ WinPcap**.

4.1 Running tcpdump

Here are the instructions for using **tcpdump** with Sniffer. Note that while libpcap remains binary compatible between versions 0.9.x up to 1.1.x, it is probably safest to recompile **tcpdump** against a newer libpcap as demonstrated in step 2 below since binary compatibility is not stated as an explicit goal by the libpcap project.

1. No recompilation with dynamically linked tcpdump. Verify that your **tcpdump** has a dynamically linked **libpcap**:

```
%ldd `which tcpdump` | grep pcap
```

If your **tcpdump** statically links **libpcap**, then there will be no output. Otherwise, the output will look something like:

```
libpcap.so.0.9.4 => /usr/lib64/libpcap.so.0.9.4
(0x00000034c1400000)
```

If there was no output above, continue to step 2 (building a new **tcpdump**)

If there was output, skip to step 3: (running **tcpdump** with **LD_LIBRARY_PATH** set).

2. With recompilation: Building a new **tcpdump**:

   a. Ensure that **libpcap-devel** is installed

   ```
   %sudo yum install libpcap-devel
   ```

   b. Download and build **tcpdump**

   ```
   % wget http://www.tcpdump.org/release/tcpdump-4.1.1.tar.gz

   %tar zxf tcpdump-4.1.1.tar.gz
   %cd tcpdump-4.1.1/
   % ./configure
   % make
   ```

3. Running **tcpdump** with **LD_LIBRARY_PATH** set:

   a. Determine which **libpcap** version that **tcpdump** expects:

   ```
   %ldd tcpdump | grep pcap
   libpcap.so.0.9.4 => /usr/lib64/libpcap.so.0.9.4
   (0x00000034c1400000)
   ```

   b. Create a **symlink** to that version in **/opt/snf/lib**. Assuming the version is 1.5.0:

   ```
   %sudo ln -s /opt/snf/lib/libpcap.so /opt/snf/lib/libpcap.so.1.5.0
   ```

   c. Set **LD_LIBRARY_PATH** to prefer the Sniffer-compatible **libpcap**:

   ```
   csh/tcsh:
   % setenv LD_LIBRARY_PATH /opt/snf/lib
   bash:
   $ export LD_LIBRARY_PATH=/opt/snf/lib
   ```

   d. Set **SNF_DEBUG_MASK=3** to know if incoming packets are going through Sniffer

   ```
   csh/tcsh:
   % setenv SNF_DEBUG_MASK 3
   bash:
   $ export SNF_DEBUG_MASK=3
   ```

   e. Run **tcpdump** to list the valid interfaces:

   ```
   % tcpdump -D
   ```

The list of valid interfaces will be displayed. The sniffer devices will have names “snf0” , “snf1”, “snf2”, or “snf3” depending on the environment variables set. Normally snf0 and snf1 will be the 2 interfaces, but if port merging is enabled,

then the interfaces will be listed as snf1, snf2, and snf3 – where snf3 are all the packets from snf1 and snf2.

f. Run **tcpdump** on the **snf0** interface:

```
% tcpdump -i snf0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode listening on eth0,
link-type EN10MB (Ethernet), capture size 65535 bytes
09:13:10.065406 IP 192.168.1.32 > 192.168.1.250: ICMP echo request, id 21865, seq 7, length 64
09:13:11.065142 IP 192.168.1.32 > 192.168.1.250: ICMP echo request, id 21865, seq 8, length 64
09:13:12.065889 IP 192.168.1.32 > 192.168.1.250: ICMP echo request, id 21865, seq 9, length 64
09:13:13.065632 IP 192.168.1.32 > 192.168.1.250: ICMP echo request, id 21865, seq 10, length 64
09:13:14.065374 IP 192.168.1.32 > 192.168.1.250: ICMP echo request, id 21865, seq 11, length 64

5 packets captured
5 packets received by filter
0 packets dropped by kernel
```

Caveat:

Please be aware that while Sniffer can support up to 14.8Mpps, **tcpdump** will print output to standard out or to a file, which will severely limit achievable packet rates to a couple of gigabits per second.

The easiest way to demonstrate the packet rates that Sniffer can achieve is to run the test program **/opt/snf/bin/tests/snf_simple_recv**.

Note 1:

If there are problems opening the **snf0** device, verify that **ldd** is showing **/opt/snf/**
libpcap and **/opt/snf/libsnf**. Eg:

```
% ldd ./tcpdump
libpcap.so.0.9.4 => /opt/snf/lib/libpcap.so.0.9.4 (0x00002b058049a000)
libc.so.6 => /lib64/libc.so.6 (0x00000034c1000000)
libsnf.so.0 => /opt/snf/lib/libsnf.so.0 (0x00002b05806de000)
/lib64/ld-linux-x86-64.so.2 (0x00000034c0c00000)
libpthread.so.0 => /lib64/libpthread.so.0 (0x00000034c1c00000)
librt.so.1 => /lib64/librt.so.1 (0x00000034c2400000)
```

Note 2:

Using the SNF interface for sniffing causes all traffic to be diverted from the normal ethernet interface to the sniffing application. This will cause hosts to fail to respond on their 10G interfaces when packet sniffing is in progress.

4.2 Compiling tcpreplay

The libpcap library provided with the Sniffer v3 package supports a pcap_inject method that can maintain lower per-packet overhead than PF_PACKET.

To use this libpcap library you must configure your tcpreplay package to use the right libpcap library and the right "injection" method (which is pcap_inject), using a configure line such as:

```
% ./configure --enable-force-inject --with-libpcap=/opt/snf -enable-dynamic-link
```

4.3 Running BRO

The latest versions of BRO, which is now referred to as Zeek, contain support for Myricom. BroControl has built-in support for host-based load-balancing using Myricom and PF_RING. Instead of creating separate entries for each BRO worker, you specify the number of workers and the load-balancing algorithm, and BRO sets up everything, including the appropriate environment variables. In order to achieve the kernel bypass, high performance mode, you must use the “snfn” sniffer interface, and not the kernel interface (such as eth2). Using the kernel interface will result in poor performance and packet loss.

There are 3 new keywords in the node.cfg file: lb_procs, lb_method, and lb_interfaces.

you will need to configure Bro with this option:

./configure --with-pcap=/opt/snf/

Then in your node.cfg file you should make any worker using the Sniffer10G drivers look similar to this:

[worker1]

type=worker

host=1.2.3.4

interface=snf0

lb_method=myricom

lb_procs=10

pin_cpus=3,4,5,6,7,8,9,10,11,12

The flow-based load balancing will be automatically enabled. The pin_cpus will cause each BRO worker to run on a particular core, and stay on that core. That will help cache. As a general rule, you should process as much as possible on the NUMA node that the LANai card is connected to. If the LANAI is on the PCIe bus for socket 0, then use cores on NUMA node 0. Processing packets on another socket will work, but will entail additional overhead to transfer the packet over the inter-socket QPI bus.

Are there any known issues with broctl capstats? The results might not be what you expect when you experience packet loss. If you are running multiple instances of BRO, then the way myricom statistics are reported are that number of received packets is by instance, so if you have 10 instances, you add up the 10 values for received packets to get the total number of packets on the link. However, each instance of BRO also reports the TOTAL number of lost packets. So if worker 1 drops 10 packets and worker 3 drops 25 packets, each worker will report 35 lost packets – that is worker 1 will report 35 lost packets, worker 2 (which didn’t loose any packets) will report 35 lost packets, and worker 3 will also report 35 lost packets.

So to compute the total number of packets BRO processed, you take the number of packets received for each worker, and add the number of lost packets for 1 worker only.

4.3.1 Timestamping Adapter

In order for BRO to process packets, the packets must contain non-zero timestamps. If you are using the LANai timesync adapter, and synchronization is not available, the packets will arrive at BRO with a timestamp of 0. BRO will discard the packets.

If you do not have the sync cable hooked up to the card, then you must load the driver with the option:

myri_timesource=0

You can verify the timestamps by running tcpdump.

4.4 Running YAF

When installing YAF, include reference to the sniffer pcap library.

$ ./configure –with-libpcap=/opt/snf

4.5 Running Suricata

Here are the instructions for using **Suricata,** http://www.openinfosecfoundation.org/index.php/downloads, with the Sniffer libpcap interface.

To use myri-snf with Suricata you need to recompile Suricata with:

```
$ ./configure --with-libpcap=/opt/snf
```

The number of rings is configurable through an environment variable **SNF_NUM_RINGS**.

For example, if **SNF_NUM_RINGS** is set to 2, invoke Suricata with:

```
$ suricata -c suricata.yaml -i snf0 -i snf0
```

4.6 Advanced Libpcap usage (i.e. Parallel Snort)

While **Libpcap** is not thread-safe, it is possible to run multiple processes that use **Libpcap/SNF** in parallel. Under this configuration, if multiple **Libpcap** processes all wish to process incoming data from a single device, they simply need to agree on the total number of rings by exporting the number of desired rings in the environment. This approach effectively translates the number of rings into an equal amount of "virtual" capture devices. That is, if 8 rings are desired, 8 pcap processes should be opened in order to capture all of the incoming data.

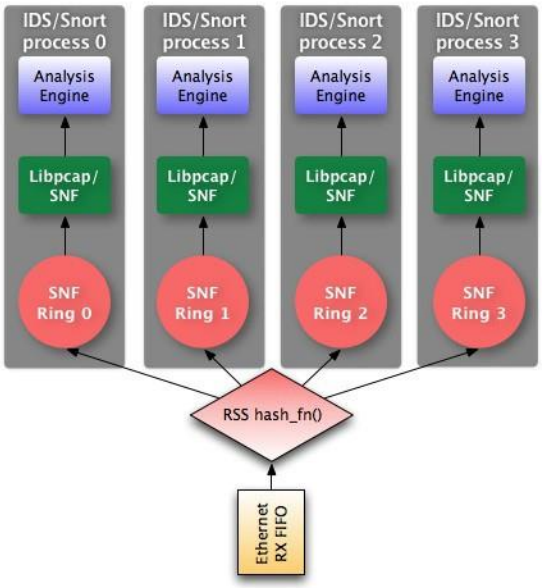


Figure 6. Multi-Process Snort over Libpcap/SNF

Note: Libpcap is not thread-safe to the extent that multiple threads cannot make calls to the same **pcap_t** and expect correct behavior. If each thread opens and accesses its own **pcap_t**, then that will work fine.

Example Code

```
# Simplistic example: start 8 parallel instances of snort each
bound
# to different cores, all using the same configuration file
which
# presumably lists snf0 as an interface.
export SNF_NUM_RINGS=8

# If incoming data is to be duplicated to multiple snort instances, we
# set the SNF_F_RX_DUPLICATE=0x300 and SNF_F_PSHARED=0x1 flags
# export SNF_FLAGS=0x301
```

```
i=0;
while [ $i -lt $SNF_NUM_RINGS ];
do
  taskset -c $i /opt/snort/snort -c /opt/snort/snort.conf &
  sleep 2 # Ensure snort creates different logfile for each
  snort i=$((i+1))
done
```

4.7 Running Wireshark with the Sniffer WinPcap Interface

Here are the instructions for using **wireshark** with the Sniffer WinPcap interface.

Sniffer v3 includes a WinPcap library (**wpcap.dll**) that is installed into the standard Windows OS system location for libraries. This WinPcap library is a Sniffer-Only enabled library that looks for Sniffer devices exclusively. The Sniffer WinPcap library has been tested with wireshark version 1.10.1.

1. Prior to installing Wireshark, prepend the folder SNF_Myri-10G\lib to your PATH.

   ```
   [ set PATH=c:\SNF_Myri-10G\lib;%PATH% ]
   ```

   This step will enable Wireshark to locate the advanced Sniffer-only wcap.dll.

2. Download **Wireshark** from http://www.wireshark.org, and select the installation destination folder, e.g., **C:\opt\wireshark**.

3. During the WireShark installation process, it will ask if you would like to install "WinPcap 4.1.3". Please check this box.

4. After the WireShark installation has completed, run **C:\opt\wireshark\wireshark.exe** and verify in the **Help->About** box that it is running the Sniffer-Only wpcap.dll. You should see text such as "**based on SNFv3 libpcap**". If this text is not present, wireshark may not have located the Sniffer-only **wpcap.dll**. Please copy the Sniffer **wpcap.dll** into the **C:\opt\wireshark** folder and restart wireshark.

The Sniffer devices listed by **wpcap.dll** will be of the form:

1. 192.168.1.5 (Myricom snf1)
2. 10.0.130.77 (Myricom snf0)

where the IPv4 # is the device name and the description field will be Myricom snf<boardnum>.

Note:

- Uninstalling the driver may remove that interface from **Wireshark** (or **WinDump**) permanently, regardless of whether that driver is reinstalled. Customers with interface issues should consult **http://wiki.wireshark.org/CaptureSetup/CapturePrivileges**. We were able to restore the interfaces by restarting the NPF driver. If all else fails, restarting the system can also be used to fix the issue. This issue may be a bug with **Wireshark/WinDump**.
- SNF does **not** replace **wpcap.dll** if there is an existing one provided by the **Wireshark/WinPcap** installation.

Chapter 5 Running Mulitple Applications

5.1 Running Two Applications

There are times when you want to run 2 applications, each of which get all of the packets. That is, every incoming packet goes to both applications.

There are two environment variables you need to set in order to run multiple applications, and have each application get all of the packets.

First, you set the variable SNF_FLAGS = 0x01. This allows >1 application to open a sniffer device at the same time.

Second, you set the variable SNF_APP_ID. If two applications have the same SNF_APP_ID, they will use RSS to get packets based on flow, for example, sorted 5-tuple. The number of applications allowed to get a set of flows is set by the environment variable SNF_NUM_RINGS.

The default SNF_APP_ID is -1.

Set the environment variable SNF_DEBUG_MASK=3 to see what all the variables are set to when your application starts.

Now, to run two instances of TCPDUMP, each one getting all of the packets, you would do:

$ SNF_APP_ID=1 SNF_FLAGS=1 SNF_DEBUG_MASK=3 tcpdump –i snf0

And in another window issue

$ SNF_APP_ID=2 SNF_FLAGS=1 SNF_DEBUG_MASK=3 tcpdump –i snf0

Of course, you could run different applications, instead of both being “tcpdump”.

Note: it is important that all the Sniffer environment variable match between instances of applications. There is only one data ring and one descriptor ring per port. Therefore, you cannot specify a different SNF_DATARING_SIZE or SNF_DESCRING_SIZE between two applications (processes) on the same port. The first application to open the sniffer port will set the variable size, and all subsequent opens to share the port will use the same values, even if the local environment specifies a different value.

If you change the SNF_APP_ID between two processes, you can then change the value of SNF_NUM_RINGS for the SNF_APP_ID. However, all processes using the same SNF_APP_ID will have SNF_NUM_RINGS set by the first sniffer open for the port with that particular SNF_APP_ID.

These environment variables can also be set in your application. See the API reference manual for details on how to set each of the variables prior to opening the sniffer port.

An example of running 2 applications, each with a different number of SNF_NUM_RINGS would be if you ran BRO as SNF_APP_ID=1, and you wanted 16 BRO workers, and SNF_APP_ID=2 was TCPDUMP, and you wanted all the packets to go to the one instance of TCPDUMP.

5.2 Running Three Applications

Follow the steps above, but open a third window and change the SNF_APP_ID to be a third unique number.

$ SNF_APP_ID=1 SNF_FLAGS=1 SNF_DEBUG_MASK=3 tcpdump –i snf0

And in another window issue

$ SNF_APP_ID=2 SNF_FLAGS=1 SNF_DEBUG_MASK=3 tcpdump –i snf0

And in a third window issue

$ SNF_APP_ID=3 SNF_FLAGS=1 SNF_DEBUG_MASK=3 tcpdump –i snf0

The maximum number of applications / processes that can each receive a copy of the packets is 32.

5.3 Running Two Multi-Threaded Applications

Multi-threaded applications will work the same way as running multiple applications. However, each application can set it's own value for SNF_NUM_RINGS. So SNF_APP_ID=1 can set SNF_NUM_RINGS=16 to run 16 workers, and SNF_APP_ID=2 can set SNF_NUM_RINGS=1 to run 1 worker. An example of this would be where APP_ID 2 is a packet capture utility wanting all the packets, and APP_ID 1 is a packet processing application, such as BRO or Suricata.

Chapter 6 Tuning

The performance of the Myricom High-Speed, Low-Latency networking solutions in real customer environments will vary depending upon the particular details of the network configuration, end-user applications, and transaction workloads.

Critical applications often have internal measurements of effective transaction rates. It is recommended that a study be made of performance before installing the Myricom solution and compared with performance after. Ideally a repeatable workload should be used. If that is not possible, sample enough daily traffic to establish typical performance. If a noticeable improvement was not experienced after installation of the Myricom solution then verify that proper installation of network adapter cards and software features was performed.

Contact ARIA Technical Support for further assistance if results are below expectations.

6.1 Sniffer v3.0.26 Performance

Whereas most Internet traffic is usually bimodal in the distribution of packet sizes, Sniffer has been designed to support a worst case scenario where all packets are at the minimum 10-Gigabit Ethernet packet size, 64 bytes. When including the 7-byte preamble, the start byte, and the 12-byte inter-packet gap, a minimum-size packet of 64 bytes requires 84 byte times on the wire. Under a constant stream of minimum packet sizes, a packet arrives at every 67.2 nanoseconds corresponding to maximum packet rate of 14.88 Mpps.

Please read **Section 2.9.1 Running Sniffer tests on Linux** for a detailed description of how to test the packet rate of the Sniffer software.

On our reference platform, a Xeon X5570 at 2.93GHz, running Sniffer in a single ring configuration demonstrates a library overhead of about 32 nanoseconds per packet on average for 64-byte packets. Minimizing library overhead is necessary to achieve high packet rate capture.

Multi-ring performance

The primary goal of using multiple rings is to leverage multiple cores in the packet analysis by effectively reducing the amount of packets each ring has to process. Assuming that the incoming traffic can be fairly well balanced across (say) 8 cores, each core is responsible for processing one eighth of a potential peak 14.88 Mpps, for a worst case of a packet every 537.6 nanoseconds. With the aforementioned library overhead, this leaves roughly 500 nanoseconds of analysis per core under a worst case scenario.

Grossly defective operation of network adapter cards is usually easily determined. When this occurs, contact ARIA Technical Support to initiate the RMA (Return Merchandise Authorization) process to obtain replacements for the defective network adapters.

If the network appears to be functioning correctly but application transaction rates have not significantly improved after installation of the Myri-10G Network Adapter cards and Sniffer software, it is possible that the proprietary software features are not properly enabled.

Remember that the default operation of the Myri-10G Network Adapters is to provide standard 10Gbit Ethernet performance. Only those hosts with valid Sniffer licenses loaded will demonstrate accelerated performance.

It may be necessary to repeat the activation process on all suspect host machines using the latest customer site license file to insure proper status of all network adapters.

The following sections provide guidance in troubleshooting the hardware or software issue.

Chapter 7 Hardware Issues

7.1 Hardware Installation/Performance

Your host/motherboard may have either PCIe 3.0 (“Gen3”), PCIe 2.0 (“Gen2”), or PCIe 1.1 (“Gen1”) PCI-Express slots.

CSP Inc. sells both x8 "Gen2" and x8 "Gen1" 10-Gigabit Network Adapters.

Important Note: Myri-10G "Gen1" PCI Express Network Adapters are compatible with “Gen3” and "Gen2" slots in hosts; the adapter auto-negotiates operation in the widest available mode (x8, x4, x2, or x1) supported by the slot into which it is installed, and at the 2.5 GT/s data rate. Similarly, Myri-10G "Gen2" PCI Express Network Adapters are compatible with “Gen3” slots and auto-negotiate operation in the widest available mode (x8, x4, x2, or x1) supported by the slot into which it is installed, and at the highest data rate (5 or 2.5 GT/s), but these "Gen2" PCI Express Adapters cannot achieve full performance in "Gen1" PCI Express slots in a host.

Which type of Myri-10G network adapter is installed? Gen1 or Gen2?

The Product code for the adapter will indicate if it is a "Gen1" (2.5 GT/s) PCI-Express Network Adapter or if it is a "Gen2" (5.0 GT/s) PCI-Express Network Adapter. If the product code includes PCIE2 in its name, then the adapter is a "Gen2" (5.0 GT/s) PCI-Express Network Adapter. For example, 10G-PCIE2-8B2-2S. Similarly, if the product code has PCIE in its name, then the adapter is a "Gen1" (2.5 GT/s) PCI-Express Network Adapter. For example, 10G-PCIE-8B-S.

For optimal performance, if you have installed an x8 “Gen2” Myri-10G network adapter, verify that the adapter reports x8 "Gen2" (5.0 GT/s) PCIe link speed. There are two ways to determine if the "Gen2" Myri-10G Network Adapter is installed into a "Gen2" PCI Express slot: the output of **/opt/snf/sbin/myri_info**, or (Linux only) the output of **lspci -vvv**.

Example header information from the output of **myri_info** for a “Gen2” Myri-10G Network Adapter:

```
pci-dev at 05:00.0 vendor:product(rev)=14c1:0008(01)
        behind bridge downstream-port: 04:02.0 111d:806a
        (x8.1/x8.2) behind bridge upstream-port: 03:00.0 111d:806a
        (x8.2/x8.2) behind bridge root-port: 00:03.0 8086:340a
        (x8.2/x16.2)
Myri-10G-PCIE-8B -- Link x8
```

The ".2" in the **pci-dev** output indicates that this is a PCIe 2.0 "Gen2" slot.

For PCIE2 adapters, the interesting component in the **pci-dev** output is the second line in the bridge series.

More specifically:

- **behind bridge upstream-port: 03:00.0 111d:806a (x8.2/x8.2)** indicates that our adapter is currently running at x8 Gen2 speed (and that is also its maximum capability).
- **behind bridge root-port: 00:03.0 8086:340a (x8.2/x16.2)** indicates the same link as seen from the motherboard side. The link is observed running at the same width-speed x8.2 from this side than from the adapter side. In addition, the motherboard slot is advertised as x16-able.
- The **x8.1** lines are about the internal links between the PLX/IDT chip and each of our controllers inside our adapter, which are x8.1 (x8 at Gen1 speed), but there is one link for each of the two Z8ES chips.
- The **Myri-10G-PCIE-8B -- Link x8** also indicates that the adapter is optimally running at x8 speed.

Alternatively, if your operating system has the **lspci** command, look at the **lspci - vvv** output to examine the Link speed (**Lnk Sta**) for the PLX or IDT chips.

If you have a two-port **8C** adapter, e.g., **10G-PCIE2-8C2-2S**, the Myri-10G network adapter has a PLX bridge chip. There will be 4 PLX chip entries in the **lspci** output, as well as 2 entries for 10G-PCIE-8B. The extra PLX entry is a downstream port which can be ignored. If the network adapter is installed into an x8 PCIe "Gen2" slot, then the **Lnk Sta** (a.k.a., link status) of one of the PLX chip entries should be listed as **5 GT/s**, and the **Lnk Sta** of the other three PLX chip entries will be listed as 2.5 GT/s. Otherwise, if you only see 2.5 GT/s listed for all 4 PLX chips, the network adapter is installed into a "Gen1" PCIe slot.

If you have a two-port **8B** adapter, e.g., **10G-PCIE2-8B2-2S**, the Myri-10G network adapter has an IDT bridge chip. There will be 3 IDT chip entries in the **lspci** output, as well as 2 entries for 10G-PCIE-8B. If the network adapter is installed into an x8 PCIe "Gen2" slot, then the **Lnk Sta** of one of the IDT chip entries should be listed as **5 GT/s**, and the **Lnk Sta** of the other two IDT chip entries will be listed as 2.5 GT/s. Otherwise, if you only see 2.5 GT/s listed for all 3 IDT chips, the network adapter is installed into a "Gen1" PCIe slot.

For similar performance reasons, if you have installed an x8 “Gen1” Myri-10G network adapter, verify that the adapter reports x8 “Gen1” (2.5 GT/s) PCIe link speed.

7.2 Hardware Cabling Connectivity Issues

If you are using Myri-10G “8C” adapters (10G-PCIE2-8C2-2S or 10G-PCIE2-8C2-2S-SYNC) with SFP+-terminated copper “direct attach” cables and you experience link up/down connectivity issues or bad crc errors, try loading the myri_snf driver with the load-time option **myri_serdes_mode=2**.

On Linux, issue the following command:

```
# /opt/snf/sbin/myri_start_stop start myri_serdes_mode=2
```

On Windows, the equivalent instructions are:

Create the following registry key (followed by a reboot):

```
REG ADD HKLM\SYSTEM\CurrentControlSet\services\snf /v myri_serdes_mode /t
REG_DWORD /d 2
```

We have documented cases where certain direct attach cables work better with 10G-PCIE2-8C2-2S-SYNC adapters when SFP+ settings for the serdes chip on the adapter are used instead of direct attach settings.

All of the possible options to **myri_serdes_mode=X** are as follows:

0: Autodetect (default)

1: Force use of direct attach cable settings.

2: Force use of SFP+ 10GBase-SR/10GBase-LR settings.

3: Force use of SFP+ 10GBase-LRM settings. All other values are reserved

for future use.

\

If you are using Myri-10G “8B” adapters (e.g., 10G-PCIE2-8B2-2S), the load-time option **myri_serdes_mode=X** will be ignored. If you experience link up/down connectivity issues or bad crcs errors with the Myri-10G “8B” adapters and direct attach cables, please try a shorter length direct attach cable or replace the direct attach cable with an SFP+ transceiver module and fiber cable.

7.3 Software Installation/System Configuration

If an issue is encountered with installation, usage, or performance, please send the output of **myri_bug_report** or **myri_dmesg.ps1** to ARIA Technical Support. The output of this script may contain vital information to speed the resolution of the issue.

/opt/snf/sbin/myri_bug_report is a diagnostic script included in the Linux and FreeBSD Sniffer 2.0 software distributions. It is used to collect diagnostic information about a customer's system configuration, such as uname output, processor files such as cpuinfo and interrupts, lspci, kernel messages, ethtool, myri_counters, etc. This script **must** be run as root.

[INSTALLDIR]\sbin\myri_dmesg.ps1 is a powershell script in the Windows Sniffer v3 software distribution which extracts logging information from the Windows Logs and prints them to stdout. It will show whether the Sniffer license is valid, which driver is loaded, and any error statements. This script

must be run as administrator. (If you cannot use powershell, inspect the **Event Viewer -> Windows Logs** for error messages related to the Sniffer software.)

7.3.1 Linux RPM/TGZ Installation failure

If an error occurs during the Linux RPM installation, please send us the full output from the rpm command as well as the output in the kernel log and **/tmp/myri_snf.log**.

If an error occurs during the Linux TGZ installation, please send us the full output from the **sbin/rebuild.sh** command. Please note that to build a Sniffer kernel module, the source kernel tree must be configured to match the running kernel. For example, with RedHat, you must install the kernel-devel package and the kernel-headers package from the RedHat distribution.

7.3.2 Windows MSI Installation failure

If the Windows MSI installation fails when running the MSI installer, please send ARIA Technical Support the log obtained by adding /log filename to the MSI install. For example,

```
c:\> snf-<version_info>*x64.msi /log
snf_install_log.txt
```

If the MSI installation fails with an ambiguous message saying that "There is a problem with this Windows Installer Package. A program run as part of setup did not finish as expected". please do the following.

Please verify that an earlier removal has unlocked all processes from Sniffer related files. A way to determine if Sniffer is still in use is to run from a command prompt:

```
c:\> tasklist /m snf.dll
```

The following output will show process(es) still using Sniffer related software:

```
Image Name                     PID Modules
============ ========== ======================

svchost.exe                    8948
snf.dll
```

A reboot is required.

Alternatively, the Windows Sniffer software can be installed using Windows msiexec as described in **Section 2.6.2.3: Installing with Windows msiexec** and **Section 2.6.2.4: Deploying for Windows imaging**.

7.3.3 Failover and myri_port_failover

myri_port_failover is a utility that can be used to control the failover policy on the 10G-PCIE-8B-2* adapters and the 10G-PCIE-8B2-4I BladeCenter adapter. The usage information for **myri_port_failover** is:

```
myri_port_failover [ -b<unit> ] [ -0 | -1 | -s | -r<0|1> ]
    Options:
     -b <unit>: uses NIC/Controller rank #unit (default 0)
     -0: set P0 as primary link for failover purposes (P1 is backup-
     only)
     -1: set P1 as primary link for failover purposes (P0 is backup-
     only)
     -s: symmetric failover (only switch link if current port is down)
     -r<0|1>: disable/enable sending a rarp pkt on failover
          (to update switches tables)
```

With no options (other than -b<unit>), it will only display settings/status. Without options, it will print the current settings, for example:

```
Currently active-port: P0
Rarp-on-failover:
enabled Port-priority:
symmetric P0-link is UP
P1-link is DOWN
```

As can be seen in the usage information, options -0, -1, and -s choose the failover policy.

This utility is particularly useful with the BladeCenter adapter to determine the I/O Module that is being used by the Myri-10G High Speed Expansion Card (HSEC) in an IBM BladeCenter blade. Instructions for this procedure are described below.

One Myri-10G High Speed Expansion Card (HSEC) types for the IBM BladeCenter H is available for purchase.

The 10G-PCIE-8B2-4I HSEC has 4 network ports for performance and failover (20Gb/s throughput). For the 10G-PCIE-8B2-4I expansion card (which has two interfaces), one interface will be connected to I/O Module slot 7 and 8, and the other will be connected to I/O Module slots 9 and 10.

The mapping of logical ethernet interfaces (ie. ethX) to physical interfaces is set by the PCI bus structure, which is motherboard-dependent. To determine which logical ethernet interface is connected to which I/O Module slot you must look at the interfaces' MAC addresses.

The interface with the lower MAC address will be connected to I/O Module slots 7 and 8, while the interface with the higher MAC address will be connected to I/O Module slots 9 and 10.

Example:

If the 4I adapter has MAC addresses 00:60:dd:46:ef:a8 and 00:60:dd:46:ef:aa, then

```
00:60:dd:46:ef:a8 port 0 connected to I/O Module slot 7
00:60:dd:46:ef:a8 port 1 connected to I/O Module slot 8

00:60:dd:46:ef:aa port 0 connected to I/O Module slot 9
00:60:dd:46:ef:aa port 1 connected to I/O Module slot 10
```

The functions **sbin/myri_info** and **bin/myri_nic_info** will provide the MAC addresses. From this output, you can determine which MAC address is mapped to which interface.

If you are trying to use the failover port, then you need to either use the **myri_ethport** tool from the myri-tools packageor use **myri_port_failover**, included in Sniffer 2.0.5 or later.

Alternatively, you can use the Sniffer API function **snf_getifaddrs()** to invoke the **myri_port_failover** utility to obtain a list of all the known Sniffer devices.

If you sort the list of devices that **snf_getifaddrs()** provides by MAC address, you will retain the property of "lower MAC address" connected to I/O Module slots 7-8 and "higher MAC address" connected to I/O Module slots 9-10. From each device, the **snf_ifa_portnum** (or **snf_ifa_boardnum**) is the number you can use to control the behavior of **myri_port_failover**.

7.3.4 Bad License Key

To test license validity, run **/opt/snf/bin/tests/snf_simple_recv**. If the license key is valid, the following output will appear:

```
snf_recv ready to receive
```

If the license key is invalid, the following output will appear:

```
License check failed on board 0, sn=<serial>: Invalid key signature
Please contact sales@myri.com to obtain an appropriate key
Can't open snf for sniffing: Permission denied
```

Additional diagnostic information may be obtained by examining the kernel log output (dmesg) or running the command:

```
$ /opt/snf/bin/myri_nic_info --license
```

If an error occurs, please contact ARIA Technical Support to obtain an appropriate key.

7.3.5 Expired License Key

If a Sniffer application complains of an expired license key, please examine the

output of **/opt/snf/bin/myri_nic_info** or **/opt/snf/sbin/myri_info** to determine if a valid Sniffer license key is present on the adapter and that the Sniffer driver is loaded properly on the network port. The **Running MCP** line of text in the **myri_info** output will identify which driver and version of software that is running.

7.3.6 Multiple License Keys

If you have purchased both DBL and Sniffer licenses for the same Myri-10G network adapter(s), both licenses can exist concurrently on the adapter and/or in the myri_license license file. However, only one driver can be loaded at a given time on a specific network port on the adapter. The driver load-time option **myri_bus** or **myri_mac** can be used to selectively load DBL or Sniffer (or Myri10GE) on a specific network port on the adapter. Contact ARIA Technical Support for details.

Before attempting to run Sniffer applications, it should be verified that the Sniffer driver is loaded on the specific adapter's network port. To verify that the Sniffer driver is loaded, run either the **/opt/snf/bin/myri_nic_info**

command or the **/opt/snf/sbin/myri_info** command and examine the **Running MCP** section of the output for text that references **snf-<version>**. Otherwise, if the adapter has two ports or there are multiple adapters installed on the same hosts, there may be confusion with the wrong driver loaded on the wrong adapter port at a given time.

If a license key is accidentally removed from an adapter, contact **ARIA Technical Support** with the 6-digit serial number (or MAC address) of the adapter and we will resend the license key to you.

7.3.7 Software Counters

The tool **myri_counters** provides low-level Sniffer hardware and software counters for traffic that goes through the device when it is in Sniffer mode.

On Linux:

```
$ /opt/snf/bin/myri_counters
```

On Windows:

```
$ [INSTALL_DIR]\bin\myri_counters.exe
```

By default, the **myri_counters** output is only displayed for port/board 0. Two-port adapters appear to **myri_counters** as different ports/boards. If you have a two- port network adapter installed in the host, you will need to specify the command- line argument **-p <port_num>** to obtain the counters output for each port. The variable **port_num** is an integer value from 0 to n-1, where n is the number of Myri-10G network adapters installed in the host and running the Sniffer driver.

Example:

```
% /opt/snf/bin/myri_counters -p 0
% /opt/snf/bin/myri_counters -p 1
```

Note that the space between the "p" and the number is optional. Also, if a host contains two two-port adapters, you would use -p0 and -p1 for the ports of the first adapter and -p2 and -p3 for the ports of the second adapter.

To clear / reset the counters requires root privileges. To clear the counters on a specific port of a network adapter use:

```
# /opt/snf/bin/myri_counters -p <port_num> -c
```

For a detailed listing of the command-line arguments to **myri_counters**, please read **Section 2.9: Diagnostic Tool Programs**. The Sniffer software counters reported by **myri_counters** are defined in **Appendix: Sniffer v3.0.26 Counters**.

7.4 Environment Variables

A number of environment variables are available in the Sniffer software for debugging and for customizing the software configuration to the requirements of the application.

7.4.1 Debugging (SNF_DEBUG_MASK=0x3)

Users can verify that a Sniffer-aware Libpcap library is linked to the application by setting **SNF_DEBUG_MASK=0x3** in the environment when opening the Sniffer device snfX. For example,

```
$ SNF_DEBUG_MASK=0x3 SNF_FLAGS=0x2 /path/to/tcpdump -i snf3
```

Setting this **SNF_DEBUG_MASK** variable causes the SNF API to output memory mapping information when the Sniffer device (snfX) is opened by Libpcap. If no information is dumped to the screen, it is likely that your application is linked against a version of libpcap that is not SNF aware. Monitor **myri_counters** to verify that traffic is in fact being received. For more information on the numbering of snfX and how this numbering corresponds to the installed adapters within a host, please read **Section 5.2.9 Numbering of snfX interfaces**.

Similarly, the **SNF_DEBUG_MASK=0x3** can be used to ensure that a Sniffer application is being correctly linked to the Sniffer library. E.g.,

```
$ SNF_DEBUG_MASK=3 SNF_RSS_FLAGS=0x1 ./snf_simple_recv
snf.0.0 P (userset) SNF_PORTNUM = 0
snf.0.0 P (default) SNF_RING_ID = -1 (0xffffffff)
snf.0.0 P (default) SNF_NUM_RINGS = 1 (0x1)
snf.0.0 P (default) SNF_RSS_FLAGS = 1 (0x1)
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
```

7.4.2 Ring Management (SNF_RING_ID, SNF_NUM_RINGS, SNF_DATARING_SIZE, and SNF_DESCRING_SIZE)

The easiest way to view the RING parameter settings is to run the Sniffer API application with the environment variable **SNF_DEBUG_MASK=0x3**. In this way, when snf_open() is called by the program it displays the current parameter settings and how they have been set (i.e. from the environment, hard-coded by the application or the default value).

For example:

```
$ SNF_DEBUG_MASK=3 SNF_RSS_FLAGS=0x1 ./snf_simple_recv
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 = 1 (0x1)
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_RING_ID is the ring to open, with values from **0** to **SNF_NUM_RINGS - 1**. If the value is -1, this function behaves as if snf_ring_open() was called. By default, **SNF_RING_ID = -1**.

The maximum number of rings (**SNF_NUM_RINGS**) supported by Sniffer is 32 per adapter port. By default, **SNF_NUM_RINGS = 1**.

If you try to allocate SNF_NUM_RINGS > 32 per adapter port, you will receive a Sniffer WARN message that looks like:

```
myri_snf WARN: eth2: endpt XX, early enable failed
```

The command 'taskset' (Linux) should be used for each program that uses a Sniffer ring, assuming there are enough cores on the machine. This spreads the load among the CPUs.

Note from the Sniffer v3.0.26 API documentation:

SNF_DATARING_SIZE represents the total amount of memory to be used to store incoming packet data for *all* rings to be opened. If the value is set to 0 or less than 0, the library tries to choose a sensible default unless **SNF_DATARING_SIZE** is set in the environment. The value can be specified in megabytes (if it is less than 1048576) or is otherwise considered to be in bytes. In either case, the library may slightly adjust the user's request to satisfy alignment requirements (typically 2MB boundaries).

By default:

SNF_DATARING_SIZE=256MB

SNF_DESCRING_SIZE=64MB.

As for the maximum amount of memory that can be configured as a ring or for all rings, we allow a maximum of 80% of physical memory to be pinned.

7.4.3 RSS Hashing (SNF_RSS_FLAGS)

The RSS hashing flags are set via the environment variable **SNF_RSS_FLAGS**.

The easiest way to view the RSS parameter setting is to run the Sniffer application with the environment variable **SNF_DEBUG_MASK=0x3**. In this way, when snf_open() is called by the program it displays the current parameter settings and how they have been set (i.e. from the environment, hard-coded by the application or the default value).

For example:

```
$ SNF_DEBUG_MASK=3 SNF_RSS_FLAGS=0x1 ./snf_simple_recv
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 = 1 (0x1)
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_RSS_FLAGS can be specified to let the implementation know which IP/TCP/UDP fields are significant when generating the hash. By default, RSS is computed on IPv4/IPv6 addresses and source/destination ports when the protocol is TCP or UDP.

By default: **SNF_RSS_FLAGS = (SNF_RSS_IP | SNF_RSS_SRC_PORT | SNF_RSS_DST_PORT | SNF_RSS_GTP | SNF_RSS_GRE)**.

In **/opt/snf/include/snf.h** there is this definition:

```
enum snf_rss_mode_flags {
SNF_RSS_IP  = 0x01, /**< Include IP (v4 or v6) SRC/DST addr in hash */
 SNF_RSS_SRC_PORT  =  0x10, /**< Include TCP/UDP/SCTP SRC port in hash */
 SNF_RSS_DST_PORT  =  0x20, /**< Include TCP/UDP/SCTP DST port in hash */
 SNF_RSS_GTP       =  0x40, /**< Include GTP TEID in hash */
   SNF_RSS_GRE       =  0x80, /**< Include GRE contents in hash */
};
```

If the packet is an IP packet and SNF_RSS_IP is desired, the IPs will be considered in the hash. If the IP packet is UDP or TCP or SCTP, then the ports are considered in the hashing. If all the traffic is tunneled through GTP or GRE protocols, the SNF_RSS_GTP or SNF_RSS_GRE flags can be used to consider elements of those headers in the hash. When SNF_RSS_GTP is set, the 'TEID' is used as part of the hash if the first payload byte indicates that the packet is a GTP version 1 or GTP version 2 packet. If SNF_RSS_GRE is used, the headers of the IP or ERSPAN+Ether+IP packet encapsulated in the GRE payload will be used in hashing.

We parse any number of stacked VLAN and QinQ (i.e. VLAN over VLAN) headers, but then any RSS is applied only if what follows is an IPv4/IPv6 header that is followed by a TCP/UDP header. This means that any form of TCP/UDP tunneling considers only the outermost headers.

We also parse up to 2 MPLS labels and then assume an IPv4 header follows before RSS is applied to the inputs.

For Ethernet frames bearing "non-hashable" traffic (e.g., other non-Myricom supported packets), they are distributed to the lowest core id or ring 0.

No packets get dropped based on the type of packet it is. The only drop cases are bad CRC32 or when there is an overflow (either the adapter and/or application cannot sustain the packet rate).

7.4.4 Port Aggregation and Merging (SNF_FLAGS)

Using the Sniffer software, traffic from two or more ports, from the same or different Myri-10G adapters, can be aggregated into a single logical receiver.

SNF_FLAGS is the environment variable that controls process-sharing (0x1), port aggregation (0x2), and packet duplication (0x3).

From **/opt/snf/include/snf.h**, here is more information on the values for **SNF_FLAGS: SNF_F_PSHARED, SNF_F_AGGREGATE_PORTMASK, SNF_F_RX_DUPLICATE**.

```
/
*
*
* Device can be process-sharable. This allows multiple
independent
* processes to share rings on the capturing device. This option can be
```

```
* used to design a custom capture solution but is also used in
libpcap
* when multiple rings are requested. In this scenario, each
libpcap
* device sees a fraction of the traffic if multiple rings are used
unless
* the @ref SNF_F_RX_DUPLICATE option is used in which case each
libpcap
* device sees the same incoming packets.
*
/
#define SNF_F_PSHARED 0x1

/
*
*
* Device can be opened for port aggregation (or merging). When this
flag
* is passed, the @b portnum parameter in @ref snf_open is interpreted as
* a bitmask where each set bit position represents a port number. The
* Sniffer library will then attempt to open every portnum with its bit set
* in order to merge the incoming data to the user from multiple
ports.
* Subsequent calls to @ref snf_ring_open return a ring handle
that
* internally opens a ring on all underlying
ports.
*
/
#define SNF_F_AGGREGATE_PORTMASK 0x2

/
*
*
* Device can duplicate packets to multiple rings as opposed to
applying
* RSS in order to split incoming packets across rings. Users should be
* aware that with N rings opened, N times the link bandwidth is
necessary
* to process incoming packets without drops. The duplication happens in
* the host rather than the NIC, so while only up to 10Gbits of
traffic
* crosses the PCIe, N times that bandwidth is necessary on the
host.
*
* When duplication is enabled, RSS options are ignored since every
packet
* is delivered to every ring.
*
/
#define SNF_F_RX_DUPLICATE 0x300
```

Additional Details

SNF_FLAGS=0x2 (Port aggregation (or merging))

Flag 0x2 says that the port number that is passed to an application is actually a mask of port, not just one port.

For example, when using tcpdump:

```
export SNF_FLAGS=0x2
env SNF_FLAGS=0x2 /path/to/tcpdump -i snf3
```

Without **SNF_FLAGS=0x2**, you would actually try to open snf port 3 (which may not exist if you only have one adapter.) For more information on the numbering of snfX and how this numbering corresponds to the installed adapters within a host, please read **Section 7.4.6 Numbering of snfX interfaces**.

Sniffer aggregation is done in software by the library, not at the NIC level. Thus, when this option is enabled there is more software overhead due to preserve packet ordering (among a few other things).

As stated in the Sniffer API documentation, snf_ring_recv_many is **not** supported with aggregation (it will return EINVAL). The reasons for this are many. Foremost, since the aggregation is done in software, the interface as presented occasions more overhead w.r.t. the metadata that is necessary to release buffers once they are returned to the library. Even aggregation aside, there is no performance advantage to using recv_many instead of the regular receive -- we only added it by user request.

7.4.5 Directing Traffic to Multiple Independent Applications (SNF_APP_ID)

Previous versions of Sniffer could either duplicate or share all traffic to all rings. Sniffer v3 adds more flexibility so that, for example, multi-process Bro and multi-threaded Suricata and tcpdump can run in parallel, each receiving all of the traffic and internally splitting the traffic among their application threads.

The environment variable SNF_APP_ID is used to assign a unique application ID for each independent application to be run.

When the application ID is set, Sniffer duplicates receive packets to multiple applications. Each application must have a unique ID. Then, each application may utilize a different number of rings. The application can be a process with multiple rings and threads. In this case all rings have the same ID. Or, multiple processes may share the same application ID.

The user may not run a mix of processes with valid application IDs (not -1) and processes with no IDs (-1). Either all processes have valid application IDs or none of them do.

7.4.6 Numbering of snfX interfaces

If you have multiple two-port network adapters per server, the tool program **myri_nic_info** can be used to determine the numbering/assignment of snfX interfaces to the installed adapters within a host. This utility also lists the MAC addresses of the snfX interfaces.

Please refer to the following output of **myri_nic_info**, where the board number corresponds to the snf interface number. (I.e., board #X will be snfX under libpcap).

```
$ /opt/snf/bin/myri_nic_info -B
# Serial MAC               ProductCode      Driver
2 277770 00:60:dd:45:ec:e5 10G-PCIE-8B-C    myri_snf 2.0.8.50315
0 363130 00:60:dd:46:b2:8a 10G-PCIE2-8B2-2S myri_snf 2.0.8.50315
1 363130 00:60:dd:46:b2:8b 10G-PCIE2-8B2-2S myri_snf 2.0.8.50315
```

7.5 Packet drops

If the network adapter runs out of receive buffers on the host, all subsequent packets will be dropped until the application can provide more host buffers. Sniffer supports very large receive rings, but these large buffers come in handy only if the application can, on average, keep up with the incoming packet rate. While the network adapter can support a worst-case 14.88Mpps, it can also be the source of dropped packets in some less-common cases. Here are some steps to isolate the problem.

1. Ensure that the Sniffer software is being used by setting **SNF_DEBUG_MASK=3** and opening the **snfX** device when invoking the application. This option will cause the Sniffer library to dump memory mapping information when the device is opened. If no information is dumped to the screen, it is likely that your application is linked against a version of libpcap that is not SNF aware.
2. Monitor **myri_counters** to verify that traffic is in fact being received. For each network adapter, check the runtime counter values.
    a. First examine values of the “SNF_* ” counters to verify that traffic is going directly to a consumer that uses the SNF API. If this count is zero, packets are probably getting delivered to the regular (much slower) kernel stack.

b. Next examine the value of “Receive Queue full”, and “Net Overflow Drop”.

These counters respectively represent host and network adapter drops.

c. Verify that the network adapter is in a PCIe slot that can sustain 10Gbit/s.

The ARIA Cybersecurity Solutions Knowledge base available on the ARIA Support Portal covers issues relative to slot width and the implications of using anything smaller than a x8 PCIe slot. An easy method to check that the PCIe slot can sustain 10Gbit/s is to run the **/opt/snf/bin/myri_dmabench** utility to ensure that the DMA write bandwidth can achieve at least 1250 MB/sec.

By default, the **myri_counters** output is only displayed for port/board 0. Two- port adapters appear to **myri_counters** as different ports/boards. If you have a two-port network adapter installed in the host, you will need to specify the command-line argument **-p <port_num>** to obtain the counters output for each port. The variable **port_num** is an integer value from 0 to n-1, where n is the number of Myri-10G network adapters installed in the host and running the Sniffer driver.

7.6 Network Adapter Timestamps

Timestamping with the Sniffer software is available for all supported Myri-10G network adapters, and is available through the pcap interface. Timestamping support is enabled by default and is available in two modes: host timestamping mode, and timesource timestamping mode. Host timestamping mode is available for all supported Myri-10G network adapters. Timesource timestamping mode is only available for 10G-PCIE2-8C2-2S-SYNC network adapters connected to a timecode generator. Sniffer v2.0.3.50235 or later is required for 10G-PCIE2-8C2-2S-SYNC network adapter support. For software installation details about timestamping and module parameters, please read **Section 2.3 Sniffer Software Driver Installation Instructions**.

The timestamp that is made available through the pcap header is the adapter-based nanoseconds transposed to system host time (as typically returned by **gettimeofday**). When each packet arrives at the network adapter, a raw timestamp (or tick) is taken from a free-running counter and is attached to the packet as it is transferred to the host. In order to make sense of the timestamp with respect to system time, the network adapter tick is transposed to system host time by using a simple arithmetic operation. The parameters for the transposition are determined by periodically synchronizing the host's clock source with the network adapter's clock source to account for clock skew. The native timestamp is the number of nanoseconds elapsed since Jan 1, 1970 00:00:00 UTC. For example code, please refer to the Sniffer test program **snf_simple_recv.c**.

NOTE for non-SYNC network adapters: NTP Daemons should be disabled to ensure monotonically increasing timestamps. Services that implement the network time protocol can cause variations in system host time that are larger than the inter-packet arrival times. While the network adapter's raw timestamps are always monotonically increasing, forward or backward jumps in time may be observed once these timestamps are transposed if relatively large correction factors are applied to the host's clock source.

7.6.1 Synchronization

There are different levels of timestamp synchronization.

1. Network Adapter-to-host synchronization (local sync)
2. Host-to-host synchronization (global sync)
3. Network Adapter-to-global synchronization (global sync, needs extra hardware)

Most systems use something like ntpd to implement (2.) -- there are more precise options by using PTP but that only moves the accuracy from milliseconds to tens of microseconds. Since both protocols are not typically run on dedicated networks and because there is typically a lot of host overhead in processing the protocol, the time can only be so accurate.

Sniffer v3 provides all three levels of timestamp synchronization. The network adapter has its own free-running clock over which timestamps are taken. These raw timestamps are transferred with each packet to the host. Two types of parameters are then necessary:

1. At startup, the network adapter timestamp was read once to compare it against the system's current time.
2. Over time, an asynchronous process synchronizes the network adapter and host clocks every second to account for drift between host and network adapter clocks (since the clock sources are two different crystals).

When packets are returned to users in nanoseconds (SNF API) or timeval (libpcap), the raw timestamps are translated to host-level timestamps based on the two above parameters. The computation is fairly inexpensive, it consists of a multiply and shift. The network adapter timestamps are always monotonically increasing. This property can only be maintained if ntpd/ptpd are disabled since there is no guarantee that these systems will not cause a significant time jump in adjustment when compared to the inter-packet arrival times that can be seen at 10Gbit/s speeds. Sniffer v3 ships with a ptpv2d client that can help in obtaining more accurate synchronization for (2.), but likely not on an order of less than 10-15 microseconds.

When using the 10G-PCIE2-8C2-2S-SYNC adapter, it is possible to implement (3.) by connecting the network adapter with a coax connector that carries an IRIG-B signal. The network adapter internally adjusts its timestamping mechanism to synchronize with the PPS information from the IRIG-B and the supplemental time values that are also encoded in this signal allow the host library to determine the second associated to the pulse. This approach is immune to whatever issues can arise by using a synchronization mechanism (like PTP) that is less accurate than 10Gbit/s frequencies.

Note: Sniffer v2.0.3.50235 or later is required for 10G-PCIE2-8C2-2S-SYNC network adapter support. For details of the hardware requirements with the 10G-PCIE2-8C2-2S-SYNC adapter, please read **Section 2.2 Hardware Installation Instructions** of this document.

7.6.2 Implementation of Time Synchronization

Here are the components involved in time synchronization with Myri-10G “8B” and “8C” network adapters.

Host Timestamping

1. The network adapter timestamps packets using its own free-running clock with an accuracy of ~1.5us on 8B and 8C network adapters. These timestamps are "raw" in that they are subject only to one clock and have not undergone any transformation.
2. The **myri_snf** driver runs a background process every second to synchronize the host and network adapter clock. The host clock is what applications would use when calling gettimeofday() whereas the network adapter clock refers to the adapter's free-running clock. This process

involves sampling both clocks over PCI express and is accurate to ~200ns. The sampling runs asynchronously with respect to all other running applications and produces functional parameters that can be supplied to userspace applications to transform "raw" network adapter timestamps into host timestamps.

Note: This entry only applies to host timestamp mode and has nothing to do with the SYNC adapter when connected to a time source. It describes how the adapter converts adapter time to host time for the purpose of presenting a timestamp on the packet in the host time domain. The host's time is never adjusted by any of our software at any time, even when connected to a time source.

3. Users of the Sniffer API always return absolute timestamps. If the underlying adapter is an 8B network adapter, the raw timestamps will be converted to host timestamps and as such will be accurate to gettimeofday() within ~200ns (because of the limitations explained in #2).

Timesource (Hardware) Timestamping

1. The network adapter timestamps packets using its own free-running clock with an accuracy of ~1.5us on 8B and 8C network adapters. These timestamps are "raw" in that they are subject only to one clock and have not undergone any transformation. If you have a 10G-PCIE2-8C2-2S-SYNC network adapter that is connected to a IRIG-B signal, the free-running clock has the additional property that it is synchronized with an external timesource accurate to ~12ns. In this case, the "raw" timestamps are also absolute timestamps because of the additional hardware on the board.
2. Users of the Sniffer API always return absolute timestamps. With a IRIG- B connected SYNC network adapter, the raw timestamps are absolute and referenced to the IRIG-B signal and will not undergo transformations with respect to the host system time.
3. **Note (for Linux only):**

   Since the host time is never adjusted by the Sniffer software, if you would like to sync the host's clock to the timesource, the software exports a PPS output to the system. This PPS output can be used as an input into ntp. For more details, please contact ARIA Technical Support.

Chapter 8 Appendix: Sniffer v3 Counters

To obtain the values of the Sniffer (SNF) hardware and software counters since the driver was loaded (or since the counters were reset via **myri_counters - c)**, use the tool **myri_counters**.

On Linux:

```
% /opt/snf/bin/myri_counters
```

On Windows:

```
% [INSTALL_DIR]\bin\myri_counters.exe
```

By default, the myri_counters output is only displayed for port 0. Two-port adapters appear to myri_counters as different ports. If you have a two-port network adapter installed in the host, you will need to specify the command-line argument -p <port_num> to obtain the counters output for each port. For example:

```
% /opt/snf/bin/myri_counters -p 0
% /opt/snf/bin/myri_counters -p 1
```

Note that the space between the "p" and the number is optional. Also, for example, if a host contains two two-port adapters, you would use -p0 and -p1 for the ports of the first adapter and -p2 and -p3 for the ports of the second adapter.

Usage Summary:

```
$ ./myri_counters -h
Usage: myri_counters [args]
-p - Port number [0] or Ethernet MAC
-c - clear the counters
-q - quiet: show only nonzero counters
-i - show host interrupt counters
-x - expert: show all counters
-h - help
```

To clear / reset the counters requires root privileges. To clear the counters on a specific port of a network adapter use:

```
# /opt/snf/bin/myri_counters -p <port_num> -c
```

All of the counters are 64-bit counters. The majority of the counters are for developer use only and will not be defined below.

If there are additional questions, please send the output of **/opt/snf/bin/myri_bug_report** or **[INSTALL_DIR]\bin]myri_counters.exe** to ARIA Technical Support.

Chapter 9 Appendix: Multiple Packet Receiving – Memory Borrowing and Returning

SNF supports several API functions for receiving packets. The snf_ring_recv() function is used to receive one packet at a time. Both snf_ring_recv_many() and snf_ring_recv_multiple(), however, are used for receiving more than one packet at a time.

The snf_ring_recv_many() function can be used for applications that handle or process the packets very quickly. When the application has finished processing the packets, it will call snf_ring_return_many() to instruct the SNF driver to free the space used by the packets and adjust the ring pointers.

Some applications, however, need to hold packets for longer time to allow for more extensive processing before the freeing the space the packets used. The function snf_ring_recv_multiple() allows the application to leave the packets in the ring until they are explicitly freed by a call to snf_ring_return_many(). With snf_ring_recv_multiple(), packets are retained on the ring for each thread, and each thread that calls this function is responsible for calling snf_ring_return_many() to release the space. The SNF driver then keeps track of the state of each thread. Once all threads have called snf_ring_return_many(), the SNF driver will free the space used by the packets and adjust the ring pointers.

Using the snf_ring_recv_multiple() function allows an application to borrow a large portion of the rings and defer the return to later. However, it is important to monitor the queue size as the physical ring will start dropping packets once the queue is full.

Chapter 10 Glossary for the Sniffer v3 (SNF) Counters

- **Lanai uptime (seconds**)
  The time (in seconds) since the Sniffer driver was loaded.
- **Counters uptime (seconds)**
  The time (in seconds) since the SNF counters were reset (myri_counters -c).
- **Net send KBytes (Port 0)**
  The amount of data sent (in kilobytes).
- **Net recv KBytes (Port 0)**
  The amount of data received (in kilobytes).
- **Ethernet send**
  The number of Ethernet packets sent through the regular OS driver.
- **Ethernet Small recv**
  The number of small (typically 256 bytes) Ethernet packets passed through the OS driver.
- **Ethernet Big recv**

  The number of big (bigger than small, but less than the Ethernet MTU size) Ethernet packets passed through the OS driver.
- **Ethernet recv down**

  The number of Ethernet packets that could not be delivered because the ethernet interface was down. if this count is high, please check if you have ifconfig up'ed the ethernet device.
- **Ethernet recv overrun**

  The OS Ethernet driver does not consume packets as fast as the network adapter is giving them to it. In normal operation, this should not happen. It may happen if the host is very loaded.
- **Ethernet recv oversized**

  The network adapter received a packet that is larger than the configured MTU.
- **SNF send pkts**

  The number of packets sent through SNF.
- **SNF recv pkts**

  The number of packets received by the Sniffer driver, **myri_snf**. This is a necessary corner case that causes some PCIe transactions to be restarted. Normally, this counter should be close to zero unless the performance of DMA reads is inadequate.
- **SNF drop prefetch race**

This is a necessary corner case that causes some PCIe transactions to be restarted. Normally, this counter should be close to zero unless the performance of DMA reads is inadequate.

- **SNF drop ring full**

  The Sniffer application (or the application using **libpcap**) does not consume the packets fast enough, so packets are dropped because the host ring is full.

- **SNF drop desc credits**

  Same as **SNF drop ring full**. The SNF “drop” credits were split into “desc” and “data” in case the customer experimented with changing the size of the queue.

- **SNF drop data credits**

  Same as **SNF drop ring full**. The SNF “drop” credits were split into “desc” and “data” in case the customer experimented with changing the size of the queue.

- **Net send Raw**

  The amount of raw data sent (in kilobytes).

- **Interrupts**

  The total number of software interrupt events generated by the NIC firmware, MSI hardware interrupts, and legacy hardware interrupts. Interrupts are only generated if they are requested.

  If the incoming packet rate is higher than what the application can ingest and there is no more buffer space, the packets are dropped. An application only requests interrupts when it is done processing its queue and when it sees that there are no new packets to be processed.

  If the interrupt counts appear to be different for port 0 and port 1, it is most probably because the application is able to process more packets from one port rather than the other. This is contrary to typical OS drivers that implement interrupts as a periodic function.

- **Wake interrupt**
  Developer use only.

- **Wake race**
  Developer use only.

- **Wake endpoint closed**
  Developer use only.

- **Net send queued**
  Developer use only.
- **Event Queue full**
  A nonzero value for this counter indicates an application flow control issue. Packets cannot be received because internal buffering is exhausted due to the application not consuming the incoming messages fast enough.
- **RX DataQ race**
  Developer use only.
- **Receive Queue full**

  A nonzero value for this counter indicates an application flow control issue. Packets cannot be received because internal buffering is exhausted due to the application not consuming the incoming messages fast enough.

  - **Fragmented request**
    Developer use only.
  - **Net bad PHY/CRC32 drop (Port 0)**
    Packet dropped due to bad PHY check or bad Ethernet CRC. If the number is high, the link is bad.
  - **Net overflow drop (Port 0)**
    The number of instances where the network adapter hardware had to drop packets due to lack of internal buffering. This count will increase if the incoming packet rate is higher than the network adapter processing rate, and if the link flow control is disabled (always when Sniffer is running).
  - **Net Recv PAUSEs**
    The number of ethernet pause packets received.
  - **Net recv alternate channel**
    Developer use only.
  - **Net Alt drop**
    Developer use only.

- **net filter drop**
  The number of packets dropped by the network adapter multicast/unicast filter. This can occur when a multicast packet is received and the corresponding multicast group has not been joined by the OS, or when the ethernet destination mac address does not match the network adapter mac address and the adapter is not in promiscuous mode.

- **User request type unknown**
  Developer use only.

- **Out of send handle**
  Developer use only.

- **Out of push handles**
  Developer use only.

- **Spurious user request**
  Developer use only.

Chapter 11 Appendix: Ethernet over Sniffer

7.1 Introduction

myri_eosnf is a Linux kernel module that provides a high-speed network interface to the OS via the Sniffer Kernel API.

7.2 Usage

The **myri_eosnf** module is loaded and unloaded via the standard Linux commands of modprobe/insmod and rmmod. One network interface per Myri-10G adapter running Sniffer will be created. All myri_eosnf network interfaces can be configured via the standard ifconfig methods.

myri_eosnf requires that the myri_snf kernel module is loaded prior to attempting to load the module. If the module is not loaded, a "missing symbols" error is printed in the system log and the module will fail to load.

All kernel threads will be run on the NUMA node local to the PCI adapter.

7.3 Module Parameters

myri_eosnf has a number of module parameters to configure the driver for optimal performance.

- **cpu_mask**

  Default Value: 0

  A CPU bitmask specifying to which CPUs the myri_eosnf kernel threads should be bound (where these threads handle packet reception). For example, 0x6 (110b) would bind receive threads to CPU 2 and CPU 1, but not CPU 0.

The value of 0 will specify a thread to be run on each CPU on the NUMA node local to the Myri-10G adapter tied to the Sniffer port, unless the num_rings kernel parameter is specified. If a bit specifies a CPU that is unavailable, the bit will be ignored and an error will be printed in the system log.

- **num_rings**

  Default Value: 0

  Similar to cpu_mask, num_rings specifies the number of raw CPUs to assign receive kernel threads.

  The value of 0 will specify a thread to be run on each CPU on the NUMA node local to the Myri-10G adapter tied to the Sniffer port, unless the cpu_mask kernel parameter is specified. The maximum value available is the number of CPUs on the NUMA node. If a number greater that this value is set, myri_eosnf will print an error in the system log and use the maximum number of CPUs available on the NUMA node.

- **rx_timeout_ms**

  Default Value: 1000

  The number of milliseconds for the receive thread to wait for an incoming packet before timing out, per the Sniffer API. Setting this value too high can result in issues when removing the module. Setting this value too low can result in higher CPU utilization.

11.1 Statistics

myri_eosnf presents a number of statistics for use via the standard Linux network statistics interfaces. The number of packets transmitted, received, errors for receive length, and packets dropped are recorded and can be queried via the standard netstat/snmp methods.

Ethtool has a number of additional statistics which can be

queried. These statistics are updated once every second.

11.2 Overall NIC statistics

nic_pkt_recv - Number of packets received by the adapter

nic_bytes_recv - Number of bytes received by the adapter

nic_pkt_overflow - Number of receive overflow errors on the adapter

nic_pkt_bad - Number of bad packets received by the adapter

snf_pkt_overflow - Number of overflow packets received at the Sniffer layer

nic_pkt_send - Number of packets transmitted by the adapter

nic_bytes_send - Number of bytes transmitted by the adapter

11.3 Receive ring statistics

ring - Receive ring number

ring_pkt_recv - Number of packets received on the ring specified

11.4 Transmit queue statistics

queue - Transmit queue number

inj_pkt_send - Number of packets sent by the transmit queue specified