SolarCapture™ User Guide

Copyright © 2017-2019 Xilinx Inc. All rights reserved.

The information disclosed to you hereunder (the “Materials”) is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx’s limited warranty, please refer to Xilinx’s Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx’s Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS “XA” IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE (“SAFETY APPLICATION”) UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD (“SAFETY DESIGN”). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

A list of patents associated with this product is at http://www.solarflare.com/patent

SF-108469-CD Last Revised: October 2019

Trademark

© Copyright 2019 Xilinx, Inc. Xilinx, the Xilinx logo, OpenOnload®, EnterpriseOnload®, SolarCapture™ and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

Table of Contents

© Copyright 2017-2019 Xilinx Inc.

© Copyright 2017-2019 Xilinx Inc

© Copyright 2017-2019 Xilinx Inc

© Copyright 2017-2019 Xilinx Inc

© Copyright 2017-2019 Xilinx Inc

1 What's New

This document is the user guide for the SolarCapture™ Linux based network packet capture applications. This issue of the user guide is applicable to SolarCapture version 1.6.7.

SolarCapture is supported on Solarflare Flareon™ SFN7000, XtremeScale™ SFN8000 series and XtremeScale™ X2 series adapters.

To identify feature availability, refer to SolarCapture Features by Distribution on page 9.

For details of AppFlex™ Technology activation key requirements, refer to AppFlex™ Technology on page 13.

1.1 SolarCapture SDK

Description

The SolarCapture Software Development Kit is a free library for developers who want to implement efficient fast packet processing of network traffic, at high packet rates, in a Linux user mode application. SolarCapture SDK can be used to receive and send packets with the minimum number of CPU cycles for packet capture, network security, NFV or other packet processing (C, C++, python) applications.

The SDK can be used on any of the supported adapters.

Features

- C and python bindings, examples and documentation
- Ability to receive and transmit network packets direct from user-mode
- Software timestamps
- Polled and interrupt modes
- Line-rate packet capture
- Capture packets with intact FCS
- `solar_capture_monitor`
- `solar_capture_doc`
- `solar_debug`

© Copyright 2017-2019 Xilinx Inc.

1.2 SolarCapture Pro

Description

SolarCapture Pro provides line-rate high performance packet capture and includes all SolarCapture features including hardware timestamping of packets. SolarCapture Pro also provides the `solar_capture` command line interface and `solar_replay` interface.

SolarCapture Pro can be run on Solarflare Flareon™ SFN7000, XtremeScale™ SFN8000 adapters and XtremeScale™ X2 series adapters.

Features

- SolarCapture Pro includes all features from the SDK
- Command line interface (`solar_capture`)
- Packet replay interface (`solar_replay`)
- Hardware timestamps
- Capture packets with intact FCS
- Line-rate packet capture
- Third party timestamps (Arista)
- Data Acquisition Module

1.3 New Features in SolarCapture v1.6.6

X2 Series Support

SolarCapture v1.6.6 adds support for XtremeScale™ X2 series adapters.

Arista 7280 48bit Timestamps

With existing support for Arista 7150 64bit and Arista 7280 64bit timestamps, SolarCapture 1.6.6 adds support for the Arista 7280 switch 48bit timestamp formats.

For configuration and decode details, refer to Chapter 14 on page 76.

CPacket Trailer Timestamps

When inter-working with the Arista-Metamako switch, SolarCapture 1.6.6 adds support for the 48bit cpacket timestamp. The `sc_cpacket_ts` node is used to replace the NIC arrival timestamp with a cpacket trailer timestamp.

For configuration and decode details, refer to Chapter 14 on page 76.

© Copyright 2017-2019 Xilinx Inc

RX Batching

A new attribute, supported by the X2 series adapter, allows the user to control the batching of packets in the adapter before delivery to the host.

The `rx_batch_nanos` attribute will enable/disable batching and is used in conjunction with the `batch_timeout_nanos` attribute. Use the following command for more information.

```
# solar_capture_doc attr rx_batch_nanos
```

1.4 New Features in SolarCapture v1.6.4

For availability of these new features, refer to SolarCapture Features by Distribution on page 9.

Support Arista 7280 Series

SolarCapture 1.6.4 adds support for Arista 7280 switch timestamping. With support for the Arista 7150 and 7280 timestamps formats, the `arista_ts` field `switch_model` is used to identify the Arista timestamping format to be decoded.

For details, refer to Arista Timestamps on page 76.

SolarCapture C Bindings User Guide

The *SolarCapture C Bindings User Guide* (SF-115721-CD) has been expanded to cover all publicly available nodes, and all exposed API. The guide is supplied with the SolarCapture SDK product. Source code is embedded in header files for the C bindings. A pre-built PDF version of the guide is included.

Shared Memory Channels

SolarCapture shared memory channels allow the transfer of packets between applications. An application can publish a packet stream which is then consumed by one or more subscribers.

See Using a shared memory channel on page 28.

For details of shared memory nodes that provide this feature, see the *SolarCapture C Bindings User Guide* (SF-115721-CD).

Tunnel Nodes

Tunnel nodes allow the transfer of packets between two or more SolarCapture instances through a TCP socket interface. A tunnel can support multiple input and output links, creating multiple separate channels.

For details see the *SolarCapture C Bindings User Guide* (SF-115721-CD).

© Copyright 2017-2019 Xilinx Inc

SFN8000 Series Support

SolarCapture v1.6.3.2 added support for SFN8000 series adapters.

SFN5000 and SFN6000 Series Support

SolarCapture v1.6.x does not support SFN5000 or SFN6000 series adapters.

SFA6902 Support

SolarCapture v1.6.x does not support the SFA6902 adapter. There is no release of AOE SolarCapture Pro for this version.

© Copyright 2017-2019 Xilinx Inc

2 Quick Start

This chapter provides a summary of SolarCapture installation and commands.

2.1 Installation Summary

This summary covers the following scenarios:

- Installation on a machine with a newly installed OS, that has all required OS dependencies already in place
- Upgrade to an existing machine that has a previous version of SolarCapture already installed.

Comprehensive install instructions are available in Installation on page 18.

To install SolarCapture SDK, or Pro:

1 Ensure you have the following:

- access to the Solarflare download site at https://support.solarflare.com

If necessary, contact your Solarflare sales channel.

2 Download these Solarflare products from https://support.solarflare.com:

SF-112972-LS	Version 1.6	SolarCapture SDK
SF-112974-LS	Version 1.6	SolarCapture Live and Pro
SF-107601-LS	Issue 37 or later	Linux Utilities (and firmware)

Download from http://www.openonload.org/:

openonload-<version>.tgz or EnterpriseOnload	**Onload must be installed to run SolarCapture.** **Recommend to use the latest available released version.**

Put the downloads in an empty temporary directory.

3 Uninstall any previous versions of the above software.

For detailed instructions see Remove Existing SolarCapture Installs on page 19, the *Solarflare Server Adapter User Guide*, and the *User Guide*.

© Copyright 2017-2019 Xilinx Inc

4 Remove any obsolete driver distributed with the OS:

```
# find /lib/modules/`uname -r` -name 'sfc*.ko' | rm -rf
# rmmod sfc
```

Then make the change permanent over a reboot. The command to do so is OS-specific, but *one* of the following is typically correct:

```
# dracut -f
```

```
# mkinitrd --force /boot/initramfs-`uname -r`.img `uname -r`
```

5 Build and install . This also installs a new driver:

```
# tar -zxvf openonload-*.tgz
# cd openonload-*/scripts
# ./-install
# -tool reload
```

6 Install the Linux Utilities:

```
# unzip SF-107601-LS-*.zip
# rpm -ivh sfutils-*.rpm
```

7 Update the firmware on the adapter:

```
# sfupdate --write
```

8 Install any new activation keys:

```
# sfkey --install <activation key file>
```

9 **Install SolarCapture SDK**:

- **rpm** package (e.g. RHEL, SLES):

```
# unzip SF-112972-LS.zip
# rpm -ivh solar_capture-core-*.x86_64.rpm
# rpmbuild --rebuild solar_capture-python-<version>.src.rpm
```

This will produce some output including a “Wrote…” line identifying the newly built `.rpm` binary package. Install this package as follows:

```
# rpm -ivh <copy the location from the “Wrote…” line>.rpm
```

- **deb** package (e.g. Ubuntu, Debian):

```
# unzip SF-112972-LS.zip
# dpkg -i solar-capture-core_*_amd64.deb
# dpkg-buildpackage solar-capture-python_<version>.dsc
```

This will produce some output including a “building package…” line identifying the newly built `.deb` binary package. Install it as follows:

```
# dpkg -i <copy the location from the “building package…” line>.deb
```

10 **Install SolarCapture Live and Pro** packages (if d):

- **rpm** package (e.g. RHEL, SLES):

```
# unzip SF-112974-LS.zip
# rpm -ivh solar_capture-live-<version>.x86_64.rpm
# rpm -ivh solar_capture-pro-<version>.x86_64.rpm
```

- **deb** package (e.g. Ubuntu, Debian):

```
# unzip SF-112974-LS.zip
# dpkg -i solar-capture-live_<version>_amd64.deb
# dpkg -i solar-capture-pro_<version>_amd64.deb
```


© Copyright 2017-2019 Xilinx Inc

2.2 Capturing Packets

Use the `solar_capture` command to capture traffic:

```
solar_capture interface=<interface> output=<pcap_filename>
```

For example:

```
solar_capture interface=enp1s0f0 output=enp1s0f0.pcap
```

- For online help, type `solar_capture --help`
- For full details, including the available options, see Command Line Interface on page 30.

2.3 Using libpcap

Use the `solar_libpcap` command to accelerate existing libpcap-aware applications, and add SolarCapture functionality to them:

```
solar_libpcap <command>
```

For example:

```
solar_libpcap tcpdump -i eth2
```

- For full details, including the available options, see Libpcap Support on page 52.

2.4 Replaying Captures

Use the `solar_replay` command to replay captured traffic:

```
solar_replay interface=<interface> input=<pcap_filename>
```

For example:

```
solar_replay interface=enp1s0f0 input=enp1s0f0.pcap
```

- For online help, type `solar_replay --help`
- For full details, including the available options, see SolarReplay on page 61.

© Copyright 2017-2019 Xilinx Inc

2.5 Monitoring Captures

Use the `solar_monitor` command to monitor captures:

```
solar_capture_monitor [options] [sessions] [commands]
```

For example:

```
solar_capture_monitor
solar_capture_monitor dump
solar_capture_monitor line_total
solar_capture_monitor line_rate
```

- For online help, type `solar_capture_monitor --help`
- For full details, including the available options, see SolarCapture Monitor on page 68.

2.6 Getting Details of Attributes

Use the `solar_capture_doc` command to read detailed information for all attributes which can be set on the command line or exported to the solar_capture environment.

```
solar_capture_doc [options] <command>
```

For example:

```
solar_capture_doc list_attr
solar_capture_doc attr
```

- For online help, type `solar_capture_doc --help`
- For full details, including the available options, see SolarCapture Attributes on page 106.

© Copyright 2017-2019 Xilinx Inc

3 Feature Availability Matrix

3.1 SolarCapture Features by Distribution

The following table identifies the SolarCapture features available in the SolarCapture distribution packages.

Feature	SDK	Pro
C and python bindings	Y	Y
example applications	Y	Y
solar_capture_monitor	Y	Y
solar_capture_doc	Y	Y
solar_debug	Y	Y
solar_replay		Y
solar_capture *(cmd line)*		Y
libpcap support		Y
DAQ		Y
Mode: steal	Y	Y
Mode: sniff		Y
polling & interrupt modes	Y	Y
Line rate packet capture	Y	Y
SW timestamp RX	Y	Y
HW timestamp RX/TX		Y
Software Filters (BPF)	Y	Y
Application Clustering	Y	Y
Shared Memory nodes		Y
Tunneling		Y
Arista timestamp 7150		Y
Arista timestamp 7280		Y

© Copyright 2017-2019 Xilinx Inc

4 Introduction

4.1 Purpose

This document describes the SolarCapture family of packet capture applications for Solarflare network adapters. SolarCapture captures received packets from the wire, and either writes these to a capture file, or forwards the packets to user-supplied logic for processing. SolarCapture assigns accurate timestamps to received packets, and is able to capture at line rate.

SolarCapture can be run as a command line tool which captures packets received from network interfaces and writes them to a file. A monitoring tool is included that provides visibility of configuration and statistical data.

SolarCapture includes a library with C and Python bindings which can be embedded in the user's own applications. Users can also extend SolarCapture by providing processing nodes which can be integrated into SolarCapture's packet processing pipeline.

Alternatively, the libpcap bindings can be used to interface with any existing application that is written to the pcap API.

4.2 Definitions, Acronyms and Abbreviations

The table below lists definitions, acronyms, and abbreviations:

Data striping	A technique used in RAID systems where logically sequential data is segmented such that consecutive segments are stored on different physical storage devices. By spreading segments across multiple devices which can be accessed concurrently, total data throughput is increased. Data striping is also a useful method for balancing I/O load across an array of disks.
Deadline	Deadline scheduler. An I/O scheduler which guarantees a service start time for a request.
DMA	Direct Memory Access. Allows the adapter to write data directly to a portion of the system memory without having to use the system CPU.
ext3	Third extended file system. A type of journaling file system commonly used by the Linux kernel.

© Copyright 2017-2019 Xilinx Inc

ext4	Fourth extended file system. A type of journaling file system commonly used by the Linux kernel which extends the capabilities of ext3.
Journaling file system	A file system which keeps track of changes which have been made in a journal. A journal could, for example, be a circular log.
NUMA	Non-Uniform Memory Access. A computer memory design found in recent server designs. Memory access time is dependent on where, in relation to the processor, the memory is located on the server (some memory is “local” to a processor, some is “remote”).
Onload	Solarflare open source OpenOnload and EnterpriseOnload accelerated network middleware.
PCAP	The file format for storing captured packets.
PTP	IEEE 1588-2008 Precision Time Protocol.
RAID	Redundant Array of Independent Disks. A storage technology allowing multiple physical disks to be presented to the system as a single logical unit.
read_expire	One of the tuning options available for the deadline scheduler. When a read request first enters the I/O scheduler, it is assigned a deadline that is the current time + the read_expire value (in units of milliseconds).
RSS	Receive Side Scaling distributes data processing across the available CPU cores.
SSD	Solid state drive or solid state disk.
Strip or stripe	The unit into which data is segmented in data striping.
Stripe size	This is the amount of data a segment used in data striping can store.
VI	The virtual interface between the adapter and the software application. This provides the transmit queue, receive queue and event queue for network packets.
VM	Virtual Memory. A memory management technique that is implemented using both hardware and software.
write_expire	One of the tuning options available for the deadline scheduler. When a write request first enters the I/O scheduler, it is assigned a deadline that is the current time + the write_expire value (in units of milliseconds).
xfs	Type of journaling file system optimized for parallel I/O.

© Copyright 2017-2019 Xilinx Inc

4.3 Software Support

SolarCapture is currently supported on the following distributions:

- Red Hat Enterprise Linux 6 (6.5 or later)
- Red Hat Messaging Realtime and Grid 2 update 5
- Red Hat Enterprise Linux 7.x
- Red Hat Enterprise Linux for Realtime 7.x
- SUSE Linux Enterprise Server 11 (SP3 or later), and 12 (base release)
- SUSE Linux Enterprise Real Time 11 (SP3 or later).
- Ubuntu 12.04 LTS, 14.04 LTS, 14.10 and 15.04, 15.10, 16.04 and 18.04 LTS.
- Debian 7.x, 8.x and 9.x
- kernel.org Linux kernels 3.10 to 4.18.

Solarflare are not aware of any issues preventing SolarCapture installation on other Linux variants such as Centos and Fedora.

- SolarCapture capture files conform to libpcap 1.3.0 format. Refer to http://www.tcpdump.org/ for details.

CAUTION: Onload must be installed to use SolarCapture. Refer to the *Onload User Guide* (SF-104474-CD) download from https://support.solarflare.com/ for installation instructions.

4.4 Firmware Variants

The Solarflare adapter firmware can be configured for specific performance optimization using firmware variants:

- **full-featured** - recommended when capturing a subset of traffic, but also using features
- **ultra-low-latency** - low latency throughput without support for hardware-multicast-loopback, sniffing of transmit traffic and filtering on VLAN-Id.
- **capture-packed-stream** - delivers line-rate packet capture. This firmware variant does not support 'sniff' mode.

A firmware variant is configured using the sfboot utility from the Solarflare Linux Utilities package (SF-107601-LS).

© Copyright 2017-2019 Xilinx Inc

4.5 Hardware Support

SolarCapture is supported on the following enabled Solarflare adapters:

Product	SFN7000	SFN8000	X2 Series
SolarCapture SDK	Y	Y	Y
SolarCapture Pro	Y	Y	Y

SolarCapture is supported on all Intel x86 and AMD 64bit processors.

4.6 AppFlex™ Technology

- Solarflare 'PLUS' adapters do not require any additional activation keys for running SolarCapture, PTP or to enable adapter hardware timestamping.
- Non PLUS adapters require a SolarCapture-Pro AppFlex activation key file to enable SolarCapture 'sniff mode' features.
- Non PLUS adapters require a PTP/HW AppFlex activation key file to enable adapter hardware timestamping.

An activation key, if required, is installed on the adapter using the sfkey utility from the Solarflare Linux Utilities package (SF-107601-LS) .

For detailed instructions for installing the key, refer to the *Solarflare Sever Adapter User Guide* (SF-103837-CD).

Customers should contact their Solarflare sales channel for SolarCapture download site access and to obtain the appropriate AppFlex activation key.

© Copyright 2017-2019 Xilinx Inc

5 Overview

This section describes the SolarCapture operation, identifies the various constituent parts of the SolarCapture product and presents a number of use-cases.

5.1 How it Works

SolarCapture performs a similar task to the tcpdump utility, but achieves a much higher level of performance by employing the kernel-bypass features on Solarflare adapters.

Whereas packets captured by tcpdump are processed by the network stack in the OS kernel, SolarCapture receives packets directly from the network adapter via a dedicated channel. Packets are delivered directly into the address space of the user-level application, bypassing the network stack.

The advantage of this architecture is that SolarCapture is able to capture packets at much higher rates, and can assign very accurate timestamps. Hardware timestamps are generated by Solarflare adapters with a PTP/hardware timestamping activation key.

SolarCapture's default capture mode is ***steal***. In this mode, packets are consumed by the capture process and are no longer delivered to host applications. It is common to use SolarCapture in conjunction with a mirror port or span port on a switch in order to capture unicast traffic flowing between other hosts in the network. SolarCapture can also capture multicast traffic.

SolarCapture Pro also supports a ***sniff*** capture mode. In this mode, packets continue to be delivered to host applications. The adapter delivers each packet a second time directly to the SolarCapture Pro application.

On a fast server, SolarCapture can process millions of packets per second on just two CPU cores, and can also be configured to spread the load over a larger number of cores using receive-side scaling (RSS).

5.2 Threading Model

SolarCapture applications usually include at least two threads:

- One or more *capture threads*, which manage the network interface and assign accurate timestamps.
- One or more *write-out threads*, which write captured packets to disk or perform application processing.

© Copyright 2017-2019 Xilinx Inc

In custom configurations there can also be other threads as needed to provide processing functions.

CAUTION: Although it is possible to perform capture and write-out in the same thread, this is not recommended as delays in write-out can cause packet loss and inaccuracy in software timestamps.

5.3 SolarCapture Components

- *Command Line Interface*

  The command line interface is a complete application for capturing received packets and writing these to files. The command line interface includes options for installing filters to select packets, joining multicast groups and managing buffering etc. Refer to Command Line Interface on page 30 for details.

- *SolarReplay*

  The solar_replay feature provides a playback facility allowing packets captured in libpcap format to be transmitted through a Solarflare adapter interface.

  Command line options provide flexible control over replay speed and bandwidth whilst preserving inter-packet pacing. For more details refer to SolarReplay on page 61.

- *SolarCapture Monitor*

  The solar_capture_monitor utility provides visibility of configuration and runtime state. Refer to SolarCapture Monitor on page 68 for details.

- *SolarCapture Extensions Interface*

  SolarCapture includes a plug-in interface that allows developers to define custom processing for packets handled by SolarCapture. Custom processors are known as *nodes*. Refer to Extending SolarCapture on page 86 for details.

- *SolarCapture C Library*

  SolarCapture can be embedded in applications by linking to the SolarCapture C library. Refer to Embedding SolarCapture on page 85 for details.

- *SolarCapture Python Module*

  SolarCapture includes Python bindings for the C library. This provides a convenient interface for constructing custom configurations of SolarCapture and to make use of features not available via the command line interface.

- *libpcap Library*

  A SolarCapture enabled libpcap library (binary) allowing existing applications, written to the pcap API, to access SolarCapture functionality. Refer Libpcap Support on page 52 to for further information.

- *Snort Data Acquisition Module*

  The Solarflare DAQ is a library module developed to work with the Snort Data Acquisition Module framework. Supported features and configuration options are detailed in Data Acquisition Module on page 57.

© Copyright 2017-2019 Xilinx Inc

5.4 Use Cases

SolarCapture can be used in a number of different ways:

- *Command Line Interface*

  The command line interface is sufficient for most packet capture needs. Received packets are captured, timestamped and written to file. The command line interface is written in Python, and so interacts with SolarCapture via the Python module.

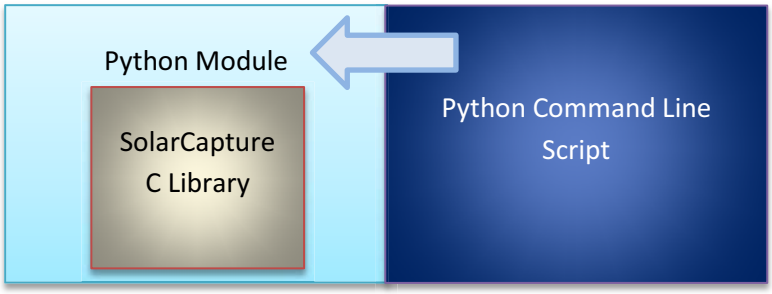


Figure 1: Python Command Line application

- *Embedding SolarCapture*

  SolarCapture can be embedded in user applications via the C bindings:

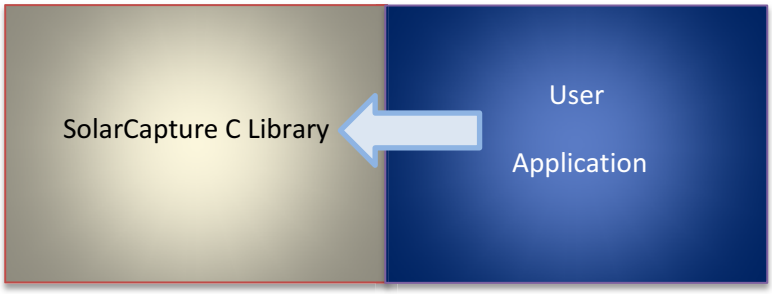


Figure 2: SolarCapture embedded in a C application

…or the Python bindings:

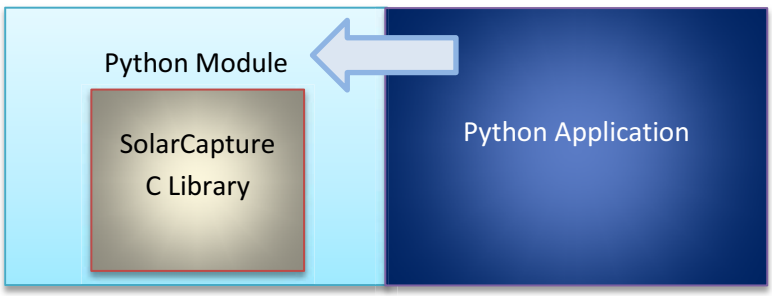


Figure 3: SolarCapture embedded in a Python application

© Copyright 2017-2019 Xilinx Inc

- *Extending SolarCapture*

  SolarCapture can be extended by providing a shared library which conforms to the SolarCapture node interface. Nodes can be inserted into the packet processing pipeline via the C or Python bindings.

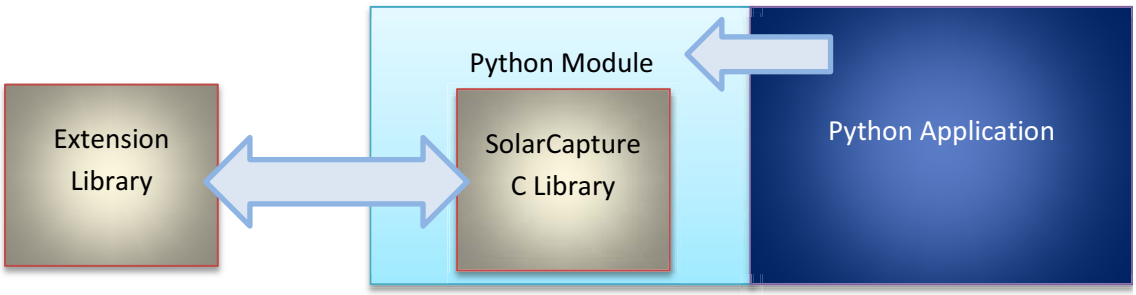


Figure 4: Extending SolarCapture

- *libpcap Library*

  The SolarCapture enabled libpcap library exposes pcap API function calls and allows existing applications written to the pcap API to use SolarCapture functionality. See Libpcap Support on page 52 for the methods of using the libpcap library.

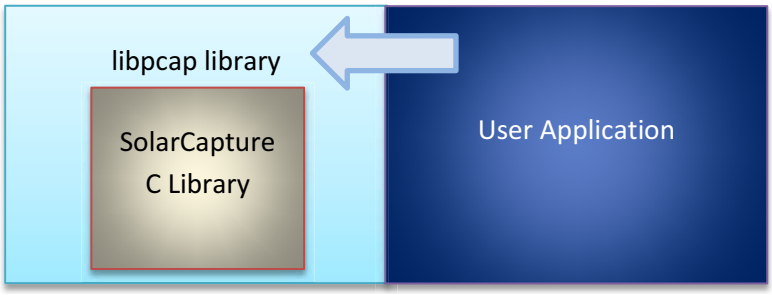


Figure 5: Using the Modified libpcap library

- *Application Clustering*

  Application clustering allows the captured traffic load, from one or more physical interfaces to be spread among multiple receiving applications.

  For more details including configuration procedures refer to Application Clusters on page 45.

© Copyright 2017-2019 Xilinx Inc

6 Installation

6.1 Get Access to Downloads

Solarflare drivers, utilities packages, application software packages and user documentation can be downloaded from: https://support.solarflare.com.

OpenOnload distributions can be downloaded from: http://www.openonload.org/.

SolarCapture Pro customers should contact their Solarflare sales channel to obtain download site access.

6.2 The SolarCapture v1.6 Distributions

The tables below show the SolarCapture v1.6 distributions.

Table 1: Distributions

SF-112972-LS	SolarCapture SDK
SF-112974-LS	SolarCapture Live and Pro

Table 2: Packages for SolarCapture SDK and Pro

solar_capture-core-<version>.x86_64.rpm	C library and core functionality
solar_capture-core-<version>_amd64.deb	
solar_capture-live-<version>.x86_64.rpm	SolarCapture Live
solar_capture-live-<version>_amd64.deb	
solar_capture-pro-<version>.x86_64.rpm	SolarCapture Pro
solar_capture-pro-<version>_amd64.deb	
solar_capture-python-<version>.src.rpm	python bindings and utilities
solar_capture-python-<version>.dsc solar-capture-python_<version>.debian.tar.gz solar-capture-python_<version>.orig.tar.gz	

© Copyright 2017-2019 Xilinx Inc

6.3 Install Dependencies

The install process will build and install Onload, the Solarflare adapter net driver and SolarCapture software on the host.

Before software and firmware can be built and installed, the host server:

- must support a general build environment i.e. have gcc, make, libc and libc-devel, python-devel
- must be capable of compiling kernel modules if building OpenOnload from source, i.e. have the correct kernel-devel package for the installed kernel version
- must have libpcap and libpcap-devel installed
- must have libaio and libaio-devel installed.

6.4 Remove Existing SolarCapture Installs

CAUTION: It is important that any previous SolarCapture versions are uninstalled before installing SolarCapture.

1 To identify a previous installation e.g.

```
rpm -qa | grep solar_capture
solar_capture-python-1.6.4.8-0.x86_64
solar_capture-live-1.6.4.8-0.x86_64
solar_capture-core-1.6.4.8-0.x86_64
solar_capture-pro-1.6.4.8-0.x86_64
```

2 Remove installed packages (note the order of removal):

```
rpm -e solar_capture-pro-1.6.4.8-0.x86_64
rpm -e solar_capture-live-1.6.4.8-0.x86_64
rpm -e solar_capture-python-1.6.4.8-0.x86_64
rpm -e solar_capture-core-1.6.4.8-0.x86_64
```

6.5 Install

CAUTION: Onload should be updated or installed before installing SolarCapture:

- It is recommended to use the latest OpenOnload or EnterpriseOnload distribution available from www.openonload.org.
- refer to the *Onload User Guide* (SF-104474-CD) for install instructions.
- the Onload distribution will also install the solar_clusterd daemon.

© Copyright 2017-2019 Xilinx Inc

6.6 Update firmware

Solarflare recommend an update of adapter firmware to the latest version. Firmware is available from the Solarflare Linux Utilities package (SF-107601-LS). This package also includes the `sfupdate`, `sfkey`, and `sfboot` utilities.

- Once installed, adapter firmware can be configured into different firmware variants for specific optimizations. See Firmware Variants on page 12 for a summary of the different variants available.
- Refer to the *Solarflare Server Adapter User Guide* (SF-103837-CD) for further information, and for update instructions.

6.7 Install SolarCapture SDK, Pro

This section describes how to install SolarCapture SDK, Pro.

- Install SolarCapture SDK on page 21
- Install SolarCapture Pro on page 22

Which SolarCapture Packages to Install

Copy the appropriate distribution zipfiles to the target server, unzip to reveal the RPMs, license file and release notes.

- SolarCapture SDK: install the SDK package.
- SolarCapture Pro: install the SDK, Live and Pro packages.

© Copyright 2017-2019 Xilinx Inc

Install SolarCapture SDK

To install SolarCapture SDK:

1 Unzip package SF-112972-LS.zip

```
# unzip SF-112972-LS.zip
Archive:  SF-112972-LS.zip
  inflating: solar_capture-<version>-ChangeLog.txt
  inflating: solar_capture-<version>-LICENSE.txt
  inflating: solar_capture-<version>-README.txt
  inflating: solar_capture-<version>-ReleaseNotes.txt
  inflating: solar_capture-core-<version>.x86_64.rpm
  inflating: solar_capture-python-<version>.src.rpm
  inflating: solar-capture-core_<version>_amd64.deb
  inflating: solar-capture-python_<version>.orig.tar.gz
  inflating: solar-capture-python_<version>.debian.tar.gz
  inflating: solar-capture-python_<version>.dsc
```

2 Install the `solar_capture-core-<version>` package:

- **`.rpm`** package (e.g. RHEL, SLES):

  ```
  # rpm -ivh solar_capture-core-*.x86_64.rpm
  ```

- **`.deb`** package (e.g. Ubuntu, Debian):

  ```
  # dpkg -i solar-capture-core_*_amd64.deb
  ```

3 Build a binary package from the `solar_capture-python-<version>` source package, and install it:

- **`.rpm`** package (e.g. RHEL, SLES):

  ```
  # rpmbuild --rebuild solar_capture-python-<version>.src.rpm
  ```

  This will produce some output including a *"Wrote…"* line identifying the newly built `.rpm` binary package. Install this package as follows:

  ```
  # rpm -ivh <copy the location from the "Wrote…" line>.rpm
  ```

- **`.deb`** package (e.g. Ubuntu, Debian):

  ```
  # dpkg-source -x solar-capture-python_<version>-1.dsc
  # cd solar-capture-python-<scver>
  # debuild -us -uc
  # cd ..
  # dpkg -i solar-capture-python_<version>-1_amd64.deb
  ```

Example Applications

When the core and python packages have been installed, example applications can be found in the following directory:

```
/usr/share/doc/solar_capture-<version>
```

for example:

```
# ls /usr/share/doc/solar_capture-1.6.6.12
c_api  ChangeLog  examples  LICENSE.txt  README  ReleaseNotes
```


© Copyright 2017-2019 Xilinx Inc

Install SolarCapture Pro

To install SolarCapture Pro:

1 Unzip package SF-112974-LS.zip

```
# unzip SF-112974-LS.zip
Archive:  SF-112974-LS.zip
  inflating: solar_capture-live-<version>.x86_64.rpm
  inflating: solar_capture-pro-<version>.x86_64.rpm
  inflating: solar-capture-live_<version>_amd64.deb
  inflating: solar-capture-pro_<version>_amd64.deb
```

2 Install both Live and Pro packages:

- **.rpm** package (e.g. RHEL, SLES):

```
# rpm -ivh solar_capture-live-<version>.x86_64.rpm
# rpm -ivh solar_capture-pro-<version>.x86_64.rpm
```

- **.deb** package (e.g. Ubuntu, Debian):

```
# dpkg -i solar-capture-live_<version>_amd64.deb
# dpkg -i solar-capture-pro_<version>_amd64.deb
```

6.8 Install PTP

SolarCapture, running on any Solarflare adapter, uses software to generate the timestamp of packets in the capture file.

SolarCapture Pro, running on adapters having an AppFlex PTP/hardware timestamping activation key use hardware timestamps generated as packets are received by the adapter. Solarflare 'PLUS' adapters include the PTP/hardware timestamping function.

Using the Solarflare Enhanced PTP daemon, the adapter time and server system clock can be synchronized to an external PTP clock source.

Install Solarflare PTP

1 Download the Solarflare Enhanced PTP distribution package SF-108910-LS to the target server.

2 Unzip the package to create the sfptpd-<version> subdirectory containing the sfptpd binary and all supporting files.

```
# tar -zxvf SF-108910-LS.tgz
```

3 For configuration and operation of sfptpd refer to the *Solarflare Enhanced PTP User Guide* (SF-109110-CD).

© Copyright 2017-2019 Xilinx Inc

7 SolarCapture Functionality

7.1 Line Rate Packet Capture

Solarflare network adapters, together with SolarCapture software, can capture packets from the network at very high bandwidths and packet rates. Users should appreciate the difference between capturing line-rate to memory and being able to sustain that high rate of capture to disk. Refer to Capture performance on page 87.

Packet capture to disk involves the following separate tasks:

- Capture to memory
- Storage to disk.

Capture to memory

Performance of capture from the network to memory depends on the performance characteristics of the network adapter, the host CPU and the memory subsystems. Best capture performance is achieved with the `capture-packed-stream` firmware variant. Performance capabilities of Solarflare network adapters are given in the table below:

Network adapter	Bandwidth	Packet rate
X2541	1 x 100Gbps	1518 bytes: 7,500,000 pps (92Gbps) 56 bytes: 7,400,000 pps (16 Gbps) 64 bytes: 7,600,000 pps (5 Gbps)
SFN8542	2 × 40Gbps	2 × 31Mpps
	4 × 10Gbps	4 × 14.9Mpps[1]
SFN8*x*22	2 × 10Gbps	2 × 14.9Mpps[1]
SFN7142Q	2 × 40Gbps	2 × 16Mpps
	4 × 10Gbps	32Mpps aggregate[1]
SFN7*x*22	2 × 10Gbps	16Mpps aggregate[1]

1. This corresponds to line rate capture at minimum packet size on all ports.
2. This corresponds to line rate capture at minimum packet size on a subset of the ports

© Copyright 2017-2019 Xilinx Inc

Note that some packet rates in the table above are an aggregate across all ports on the adapter. For example the SFN7*x*22 adapter has a packet rate of 16Mpps aggregate (or 8Mpps per port if split evenly), meaning that 2 × 10Gbps cannot be achieved with smaller packet sizes, but 1 × 10Gbps can be achieved with minimum packet sizes.

Storage to disk

The rate at which packets are stored to disk depends on a number of factors including the storage technology, block layer and filesystem used. Best capture performance is achieved on filesystems that support asynchronous I/O.

Solarflare cannot predict or guarantee the performance of storage to disk on customer systems. For further details, see Capture performance on page 87.

Using the capture-packed-stream firmware variant

The number of pages required to sustain line-rate packet capture will depend on the ability of the application to keep up with the packet arrival rate and write packets to the storage device. Enabling a greater number of huge pages and packet buffers can help sustain line-rate capture during traffic burst periods.

To enable huge pages on Red Hat Enterprise Linux, the kernel must be compiled with the `CONFIG_HUGETLB_PAGE` flag set.

- The size of huge pages allocated by the OS can be identified:

  ```
  # cat /proc/meminfo | grep Hugepagesize
  ```

- The number of huge pages available can be identified:

  ```
  # cat /sys/kernel/mm/hugepages/hugepages-*/nr_hugepages
  ```

  or, using a less recent interface:

  ```
  # cat /proc/sys/vm/nr_hugepages
  ```

- To allocate 100 huge pages when the huge page size is 2048kB:

  ```
  # echo 100 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
  ```

  or, using a less recent interface:

  ```
  # echo 100 > /proc/sys/vm/nr_hugepages
  ```

- To ensure this setting is persistent across restarts add the following line to the sysctl.conf file:

  ```
  vm.nr_hugepages = 100
  ```

Users of other Linux OS variants should refer to the appropriate operating system documentation for huge pages configuration instructions.

The adapter firmware variant can be selected using the `sfboot` utility included in the Solarflare Linux Utilities package (SF-107601-LS).

```
# sfboot --adapter=eth<N> firmware-variant=capture-packed-stream
```

To display the current boot configuration for an adapter:

```
# sfboot
```


© Copyright 2017-2019 Xilinx Inc

Following a firmware variant change it is necessary to reboot the adapter and to reload the adapter drivers before the change becomes effective:

```
# onload_tool reload
```

7.2 Operating Modes

Polling vs Interrupt Mode

The command line option `capture_busy_wait` is used to configure SolarCapture to either 'spin' (busy_wait) on a capture thread or to block on that thread when no traffic is being received.

- A blocked capture thread will be 'unblocked' by an interrupt to signal that there is further incoming traffic to process.
- A polling (busy_wait) thread will never block.

Polling is recommended for applications receiving high traffic rates, that are latency sensitive or can be subject to prolonged bursts of traffic. This is the default SolarCapture behavior.

```
# solar_capture interface=eth2 output=eth2.pcap capture_busy_wait=1....
```

Interrupt driven mode can be used by applications receiving lower traffic rates, that are not latency sensitive and not expected to receive large or sustained bursts of traffic.

```
# solar_capture interface=eth2 output=eth2.pcap capture_busy_wait=0....
```

When using interrupt mode, the number of interrupts can be controlled by configuring interrupt coalescing and interrupt moderation options `rx_usecs` and `adaptive-rx` via ethtool. For more information refer to the *Solarflare Server Adapter User Guide* (SF-103873-CD).

A companion option `writeout_busy_wait` controls the same behavior for writeout threads.

7.3 Capture Frame Check Sequence

Solarflare adapters can be configured to deliver received packets with the FCS intact. This is a per network port setting. When set, all packets will be delivered to all receiving applications with the FCS intact.

To enable the driver to deliver packets and packet FCS, write a '1' to the following file:

```
# echo 1 > /sys/class/net/<interface>/device/forward_fcs
```

To configure SolarCapture to retain the packet FCS use the `strip_fcs` option:

```
# solar_capture strip_fcs=0 interface=eth4 output=eth4.pcap
```


© Copyright 2017-2019 Xilinx Inc

7.4 Capture file timestamp format

By default packets are written to files in PCAP format, which supports microsecond resolution timestamps. There is a well-known variant of PCAP that uses nanosecond resolution for the timestamps. Use the 'format' solar_capture command line option to select one of the following values:

```
format=pcap
format=pcap-ns
```

For details of the nanosecond pcap timestamp format refer to http://anonsvn.wireshark.org/viewvc/trunk/wiretap/libpcap.h?revision=3754.

7.5 Hardware Timestamps

SolarCapture will use hardware generated timestamps if the adapter has a PTP/HW timestamping AppFlex activation key.

These keys are preinstalled on certain adapters, such as the SFN7322F, or the Plus versions of the SFN8000 and X2 series.

If hardware timestamps are not available, SolarCapture will silently fallback to use software timestamps, taking time from the host system clock. Specific attributes are available to control timestamp options:

- `force_sw_timestamps` - When non-zero this will always use software timestamps even when hardware timestamps are available.
- `require_hw_timestamps` - When non-zero, this will cause VI allocation to fail if hardware timestamps are not available.

Use `solar_capture_monitor` to check that hardware timestamping is enabled on the adapter (0=not enabled, 1=enabled):

```
$ solar_capture_monitor dump | grep hw_timestamps
hw_timestamps                   1
```

Refer to Limited Availability of Hardware Timestamping VIs on page 88.

For more information about setting SolarCapture attributes see SolarCapture Attributes on page 106.

Solarflare recommend running PTP (sfptpd) to synchronize the adapter clock with an external PTP/NTP time source, however, even if synchronized time is not a requirement, it may be necessary to run sfptpd briefly in 'freerun' mode to set the initial adapter clock time to the system clock time. Refer to the *Solarflare Enhanced PTP User Guide* (SF-109110-CD) for instructions.

© Copyright 2017-2019 Xilinx Inc

7.6 Ingress Packet Capture

SolarCapture, by default, will capture incoming packets from the specified interface. To explicitly capture received traffic identify the capture_point using the following syntax:

```
solar_capture interface=eth4 output=eth4.pcap mode=sniff capture_point=ingress
```

All captured packets are hardware timestamped - if hardware timestamps are available. See Hardware Timestamps on page 26.

7.7 Egress Packet Capture

SolarCapture, by default, will capture incoming packets from the specified interface. To capture outgoing packets identify the egress capture point using the following syntax:

```
solar_capture interface=eth4 output=eth4.pcap mode=sniff capture_point=egress
```

All captured packets are hardware timestamped - if hardware timestamps are available. See Hardware Timestamps on page 26.

The adapter must be configured to use the full-feature firmware variant.

7.8 Sniff Mode

In sniff mode, solar_capture will install a filter on the adapter to capture all traffic arriving at the capture interface.

With hardware filters

The sniff mode cannot be combined with additional hardware filters to limit capture to only a subset of traffic i.e. **none** of the following will prevent all traffic arriving at the interface from being delivered to the capture file, disk or shared memory:

```
solar_capture mode=sniff join_mcasts="224.100.100.1" interface=enp5s0f0
output=mycap.pcap
```

```
solar_capture mode=sniff join_streams="224.100.100.1" interface=enp5s0f0
output=mycap.pcap
```

```
solar_capture mode=sniff streams=udp:224.1.2.3:8102 join_mcasts=224.1.2.3
interface=enp5s0f0 output=mycap.pcap
```

With software filters

The sniff mode can be combined with software (bpf) filters to ensure that only a subset of the traffic delivered to solar_capture is forwarded to capture file, disk or to shared memory.

The following are example software filters using bpf syntax.

© Copyright 2017-2019 Xilinx Inc

```
solar_capture mode=sniff filter="port 9123 or port 8001"
interface=enp5s0f0
solar_capture mode=sniff filter="ip[9]=0x11" interface=enp5s0f0
solar_capture mode=sniff filter="ip[9]=0x6" interface=enp5s0f0
solar_capture mode=sniff filter="vlan 101" interface=enp5s0f0
solar_capture mode=sniff filter="ip proto \udp" interface=enp5s0f0
solar_capture mode=sniff filter="ip proto \tcp" interface=enp5s0f0
solar_capture mode=sniff filter="src host 170.18.1.76" interface=enp5s0f0
solar_capture mode=sniff filter="src net 172.18" interface=enp5s0f0
```

Users should also refer to the following related topics:

- Sniff Mode in Packed Stream Firmware on page 90
- Sniff Mode Transmitted PTP Packets on page 90

Software filters do increase processing overheads and may reduce the maximum capture packet rate.

For further information about Berkeley Packet Filters syntax, refer to available online documentation.

7.9 Using a shared memory channel

SolarCapture shared memory channels allow the transfer of packets between applications. An application can publish a packet stream which is then consumed by one or more subscribers.

- The publisher can be the `solar_capture` command line tool, the `solar_balancer` tool, or a custom application using the `sc_shm_export` or `sc_shm_broadcast` nodes.
- The consumers can be the libpcap bindings, or a custom application using the `sc_shm_import` node.

Ensure the hugepages filesystem is mounted

The shared memory channel uses a memory-mapped file in the filesystem. When using packed-stream mode this must be a `hugetlbfs` filesystem (and using hugepages can improve performance even when not using packed-stream).

To confirm that a `hugetlbfs` filesystem is mounted:

```
# mount | grep hugetlbfs
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime)
```

If there is no output from the above command, the filesystem is not available - so must be mounted:

```
# mount -t hugetlbfs none /dev/hugepages
```

Ensure that some huge pages are allocated. See Allocating Huge Pages on page 97.

 © Copyright 2017-2019 Xilinx Inc

Publishing to a shared memory channel

To capture packets from a network interface and write to a shared memory channel:

```
solar_capture interface=<interface> output_shm=/dev/hugepages/<interface>
```

For example:

```
solar_capture interface=eth4 output_shm=/dev/hugepages/eth4
```

Consuming a shared memory channel

To consume a shared memory channel using the libpcap bindings:

```
solar_libpcap tcpdump -i scshm:/dev/hugepages/<interface>
```

For example:

```
solar_libpcap tcpdump -i scshm:/dev/hugepages/eth4
```

7.10 Receive Only Configuration

It is possible to use SolarCapture with the adapter configured in a rx-only configuration. This may be required when capturing from a network tap or splitter device.

The user should use ethtool to disable auto negotiation and the link will be up when it receives a valid RX signal. It may also be necessary to explicitly set the interface speed when using a 1G interface.

```
# ethtool -s <interface> autoneg off
```

```
# ethtool -s <interface> speed 1000
```

Solarflare adapters support a RX only configuration on 1G and 10G interfaces.

© Copyright 2017-2019 Xilinx Inc

8 Command Line Interface

8.1 Introduction

The `solar_capture` command line application captures and timestamps packets received at one or more network interfaces, and writes these to files. The interface includes a number of configuration options:

- Set the capture file format
- Join multicast groups
- Control resources used by SolarCapture
- Install filters to select streams to be captured
- Affinitize threads to specific CPU cores
- Control the types of packets captured
- Capture file rotation

NOTE: The command line interface enables the most commonly used features of SolarCapture. For more complex deployments the user can use either the SolarCapture C library or Python module.

The Command Configuration File is an alternative to the standard command line configuration method. Refer to Command Configuration File on page 38 for details.

8.2 Getting Help

To get help:

```
$ solar_capture --help
$ solar_capture help <option>
$ solar_capture help all
```


© Copyright 2017-2019 Xilinx Inc

8.3 Capture Command Line

Syntax

To run the command line interface:

```
$ solar_capture [global-options] <capture>...
```

Each capture instance starts with `interface=` and may provide further options specific to the capture instance. Capture instance options override global options.

Each capture instance must specify an output, which can be a file:

```
$ solar_capture interface=eth4 output=/captures/eth4.pcap
```

Alternative the output can be a shared memory channel:

```
solar_capture interface=eth4 output_shm=/dev/hugepages/eth4 shm_fanout=2
```

Options

Table 3 lists all SolarCapture command line options..

Table 3: Command Line Options

Option	Description
streams	; separated list of streams to capture. `# solar_capture help streams`
join_mcasts	; separated list of multicast groups to join. Note that if all traffic is captured, then IGMP traffic will be captured, and so group membership will lapse. To prevent this, set the ptp option to 1. `# solar_capture help join_mcasts`
join_streams	; separated list of streams to join and capture. Setting this option is identical to setting the streams and join_mcasts options. `# solar_capture help join_streams`
capture_cores	List of cores to capture on. • The number of capture cores cannot exceed the number of RSS queues configured by the net driver module option rss_cpus • The number of cores must be a power of 2 value. `# solar_capture help capture_cores`
writeout_core	The core to use for capture file write-out. `# solar_capture help writeout_core`

© Copyright 2017-2019 Xilinx Inc

Table 3: Command Line Options (continued)

Option	Description
format	Packet capture file format: pcap = microsecond resolution timestamps pcap-ns = nanosecond timestamps `# solar_capture help format`
rotate_seconds	Capture file rotation (like tcpdump -G). • Files are not created before packets are received. The file name specified on the command line is the name of all files created[1]. `# solar_capture help rotate_seconds`
rotate_file_size	Capture file rotation (like tcpdump -C). • Files are not created before packets are received. An incrementing index starting at 0 is appended to each file created[2]. `# solar_capture help rotate_file_size`
packet_count	Stop capture after receiving N packets. `# solar_capture help packet_count`
filter	Specify a software filter in the Berkeley Packet Filter (BPF) specification. Packets not matching the filter are discarded. See examples: Sniff Mode on page 27. `# solar_capture help filter`
arista_ts	Forces SolarCapture to use Arista 7150 or 7280 hardware timestamps. For detailed information refer to Arista Timestamps on page 76 or run the following command: `# solar_capture help arista_ts`
cpacket_ts	Forces SolarCapture to use cpacket hardware timestamps. For detailed information run the following command: `# solar_capture help cpacket_ts`
snap	Max bytes from each packet to store to capture file. `# solar_capture help snap`

© Copyright 2017-2019 Xilinx Inc

Table 3: Command Line Options (continued)

Option	Description
append	Set to 1 to append to capture file (otherwise truncate). `# solar_capture help append`
write_mode	Choose between fast (default) and safe (for concurrent access). `# solar_capture help write_mode`
on_write_error	Set to exit (default), abort, message or silent. `# solar_capture help on_write_error`
mode	steal (default)|sniff • In steal mode, packets are captured by SolarCapture, but do not continue to their destination. • In sniff mode, packets are captured by SolarCapture and continue to their destination: - 'sniff' mode captures all traffic at the specified interface. • sniff mode captures all traffic at an interface and cannot be combined with more specific hardware filters to forward only a subset of traffic. • Refer to Sniff Mode on page 27 for limitations and further details. `# solar_capture help mode`
capture_point	Specifies which datapath to capture. Set to ingress (default) or egress. `# solar_capture help capture_point`
delivery_interface	Interface on which captured packets are delivered to the host.
cluster	Name of application cluster to join. `# solar_capture help cluster`
discard	List of packet errors that should be discarded. `# solar_capture help discard`
rx_ring_max	Maximum fill level for RX descriptor rings. `# solar_capture help rx_ring_max`
capture_buffer	Amount of buffering per capture interface, in bytes `# solar_capture help capture_buffer`

© Copyright 2017-2019 Xilinx Inc

Table 3: Command Line Options (continued)

Option	Description
writeout_buffer	Amount of buffering per disk writer, in bytes `# solar_capture help writeout_buffer`
promiscuous	Enable/disable promiscuous mode when using 'sniff' mode. 1 - all packets received at the capture port are captured. 0 - only packets that would arrive at the host are captured. `# solar_capture help promisuous`
user	Drop privileges to lower user name (like tcpdump -Z). `# solar_capture help user`
ptp	Set to 1 to not capture IGMP because PTP is in use. `# solar_capture help ptp`
capture_busy_wait	Set to 1 to busy-wait in capture thread(s), 0 to block. `# solar_capture help capture_busy_wait`
writeout_busy_wait	Set to 1 to busy-wait in writeout thread, 0 to block. `# solar_capture help writeout_busy_wait`
strip_fcs	Set to 1 to remove FCS from captured packets, set to 0 to capture packets with FCS intact. Also refer to Arista Timestamps on page 76 when using Arista timestamps. `# solar_capture help strip_fcs`
config_file	Identify a command configuration file to configure solar_capture. See Command Configuration File on page 38 `# solar_capture help config_file`
output_shm	Deliver captured packets to shared memory channel. `# solar_capture help output_shm`
shm_fanout	Maximum number of shared memory consumers that connect to a capture instance. `# solar_capture help shm_fanout`
postrotate_command	Commands to be executed on files closed on rotation. See Rotate Capture Files on page 42 for examples. `# solar_capture help postrotate_command`

© Copyright 2017-2019 Xilinx Inc

Table 3: Command Line Options (continued)

Option	Description

partial_suffix	Suffix to add to capture file allowing the file to be read while still being written to.
	`partial_suffix=`
	`partial_suffix=<string>`
	See also write_mode above for note on concurrent file access.

<extension>	add optional processing step into pipeline.

1. One method of ensuring that unique filenames are created each time the capture file is rotated is to use the Linux date formatting escapes when naming the file:

   ```
   solar_capture rotate_seconds=10 eth2=eth2%Y%m%d-%H:%M:%S.pcap
   ```

   See Rotate Capture Files on page 42 for examples.

2. When the output filename includes the '$i', the filename must be quoted to prevent the shell from substituting the '$i' resulting in unexpected output filenames. To ensure an incrementing number value is applied as expected, the filename should be quoted in single quotation marks, or the '$i' should be escaped, for example:

   ```
   output='ethX_$i.pcap'
   ```

8.4 Capturing packets with solar_capture

Identify the physical network interface receiving the packets to be captured; for example *eth2*. The following command captures all packets arriving at eth2, and writes them to eth2.pcap.

```
solar_capture interface=eth2 output=eth2.pcap
```

To get the highest level of performance from SolarCapture, and to achieve consistently accurate software timestamps, it is best to assign SolarCapture threads to individual CPU cores as follows:

```
solar_capture interface=eth2 output=eth2.pcap capture_cores=1 writeout_core=2
```

The example above assigns the capture thread to core 1 and the write-out thread to core 2. For best performance it is best to select two cores on the same processor. Refer to Tuning Guide on page 92 for further tuning options.

© Copyright 2017-2019 Xilinx Inc

8.5 Command line options

The command line consists of options and capture instances. For example `interface=eth2` starts a capture instance.

```
solar_capture [global-options] interface=eth2 output=eth2.pcap [instance-options]
```

Options appearing before a capture instance apply as defaults for all instances. Options appearing after a capture instance apply only to that instance, and override defaults.

```
solar_capture snap=0 interface=eth2 output=eth2.pcap \
                     interface=eth3 output=eth3.pcap snap=60
```

In the above example, the `snap=60` option overrides the `snap=0` option for interface eth3. Therefore complete packets are written to eth2.pcap, but only the first 60 bytes of each packet are written to eth3.pcap.

8.6 Selecting Streams to Capture

By default `solar_capture` captures all packets on the specified interface. The `streams` option can be used to capture a subset of packets, with other packets being delivered to the network stack as normal. Streams of TCP and UDP packets can be specified by IP address and port number, and streams can also be specified by destination Ethernet MAC address with optional VLAN. Here are some examples:

- Capture TCP packets with destination IP 123.1.2.3 and port 1111:

  ```
  streams=tcp:123.1.2.3:1111
  ```

- Capture UDP packets with destination IP 123.1.2.3 and port 1111 and destination IP 123.4.5.6 and port 2222 (note the semi-colon separators and escape sequence):

  ```
  streams="udp:123.1.2.3:1111;udp:123.4.5.6:2222"
  ```

- Capture TCP packet on a single connection specifying both source and destination parameters:

  ```
  streams="tcp:123.1.2.3:1111,120.3.2.1:222"
  ```

- Capture all packets to a specific destination MAC address and all broadcasts:

  ```
  streams="eth:00:0F:53:16:04:74;eth:ff:ff:ff:ff:ff:ff"
  ```

- Capture all broadcast packets on VLAN 3:

  ```
  streams="eth:vid=3,ff:ff:ff:ff:ff:ff"
  ```

For further information enter the following command:

```
# solar_capture help streams
```


© Copyright 2017-2019 Xilinx Inc

8.7 Join Multicast Groups

Most networks perform multicast filtering in the switches so that multicast packets are only delivered to hosts subscribed to the corresponding groups. In such environments it may be necessary to join multicast groups to be captured. The `join_mcast` option instructs SolarCapture to join the given groups via the IGMP protocol. For example:

```
join_mcasts="224.1.1.1;239.2.2.2"
```

To join a multicast group on a particular VLAN - add the VLAN ID:

```
join_mcasts="239.0.0.1;239.0.0.2;vid=77,239.1.2.3"
```

It is common to want to capture a multicast stream and join the corresponding group, so a shortcut is provided to make this easy: The join_streams option combines the functions of the `streams=` and `join_mcasts=` options. These two examples are equivalent:

```
solar_capture interface=eth2 output=eth2.pcap \
              streams=udp:224.1.2.3:21002 join_mcasts=224.1.2.3

solar_capture interface=eth2 output=eth2.pcap \
              join_streams=udp:224.1.2.3:21002
```

For further information enter the following command:

```
# solar_capture help join_streams
```

8.8 Setting thread affinity

As noted above, assigning SolarCapture threads to individual CPU cores delivers the best level of performance and software timestamp accuracy.

The `writeout_core` option sets the CPU core to use for a write-out thread.

The `capture_cores` option sets the CPU core or cores used for packet capture and timestamping. If a single core is given, then a single thread is used to capture packets. If a list of cores is given, then packet capture is distributed over multiple threads using receive-side scaling (RSS). This algorithm distributes received packets over the cores according to a hash on addresses in the packet headers. This ensures that packets within any given stream will be consistently delivered to the same capture thread, and so will be processed in order.

NOTE: It only makes sense to use RSS if the capture instance is configured to capture multiple independent streams of packets. There is no guarantee that the load will be spread evenly over the available capture cores.

Below are some examples:

- Create a single capture thread, assigned to CPU core 2:
  `capture_cores=2`
- Create 2 capture threads, assigned to CPU cores 4 and 8:
  `capture_cores=4,8`

© Copyright 2017-2019 Xilinx Inc

- Create 2 capture threads, not affinitized to any particular CPU core:

  ```
  capture_cores=-1,-1
  ```

- Create a single thread with 2 receive queues:

  ```
  capture_cores=1,1
  ```

- To capture data from multiple streams, using multiple capture-cores and multiple writeout cores, run a command similar to the following:

  ```
  solar_capture \
      format=pcap capture_buffer=49152000 snap=0 \
      rx_ring_low=40 rx_ring_high=60 rx_refill_batch_low=64 rx_ring_max=4095 \
      eth1=file1 join_streams="<list 1 of streams>" capture_cores=1,2 writeout_core=3 \
      eth2=file2 join_streams="<list 2 of streams>" capture_cores=4,5 writeout_core=6 \
      eth3=file3 join_streams="<list 3 of streams>" capture_cores=7,8 writeout_core=9
  ```

  In the above example streams are grouped with each group of streams using a different writeout core. By default SolarCapture creates a single capture thread, and a single writeout thread and does not set the core affinity.

8.9 Command Configuration File

The command configuration file is an alternative mechanism to configure SolarCapture. Instead of specifying capture instances and options directly on the command line, these can be placed in one or more command configuration files.

A separate command configuration file can be created for each capture instance, or a single file can configure all options requested for a `solar_capture` instance.

Specifying arguments in a command configuration file is exactly the same as the standard command line with each argument appearing on a separate line.

Per Capture Instance # 1

```
solar_capture snap=60 eth2=/tmp/eth2.pcap streams=tcp:192.168.1.1:5001
```

This command line can be placed in a configuration file e.g ‘`sc_eth2.cfg`’

```
snap=60
eth2=/tmp/eth2.pcap
streams=tcp:192.168.1.1:5001
```

NOTE: Each command line argument appears on a separate line and the capture instance `eth2=/tmp/eth2.pcap` is specified **in the configuration file**.

The command configuration file can then be identified when starting solar_capture e.g

```
solar_capture config_file=sc_eth2.cfg
```

A separate command configuration file can be created for each capture instance.

© Copyright 2017-2019 Xilinx Inc

Per Capture Instance # 2

```
solar_capture snap=60 eth2=/tmp/eth2.pcap streams=tcp:192.168.1.1:5001 \
              streams=udp:127.0.0.100:8080
```

This command line could be placed in a configuration file e.g ‘sc_eth2.cfg’

```
snap=60
streams=tcp:192.168.1.1:5001
streams=udp:127.0.0.100:8080
```

NOTE: Each command line argument appears on a separate line and the capture instance `eth2=/tmp/eth2.pcap` is identified **on the command line**.

The command configuration file can then be identified when starting solar_capture e.g

```
solar_capture config_file=sc_eth2.cfg eth2=/tmp/eth2.pcap
```

A separate command configuration file can be created for each capture instance.

Per Capture Instance # 3

```
solar_capture eth2=/tmp/eth2.pcap streams=tcp:192.168.1.1:5001 snap=60 \
                capture_cores=3 writeout_core=5 \
              eth3=/tmp/eth3.pcap streams=udp:127.0.0.100:8080 snap=120 \
                capture_cores=7 writeout_core=9
```

This command line could be placed in TWO configuration files e.g ‘sc_eth2.cfg’ and ‘sc_eth3.cfg’

```
streams=tcp:192.168.1.1:5001
snap=60
capture_cores=3
writeout_core=5
```

```
streams=udp:127.0.0.100:8080
snap=120
capture_cores=7
writeout_core=9
```

NOTE: Each command line argument appears on a separate line and the capture instances `eth2=/tmp/eth2.pcap` and `eth3=/tmp/eth3.pcap` are identified **on the command line**.

The command configuration files can then be identified when starting solar_capture e.g

```
solar_capture config_file=sc_eth2.cfg eth2=/tmp/eth2.pcap \
              config_file=sc_eth3.cfg eth3=/tmp/eth3.pcap
```

A separate command configuration file can be created for each capture instance.

© Copyright 2017-2019 Xilinx Inc

Single File

```
solar_capture snap=0 \
              eth2=/tmp/eth2.pcap streams=tcp:192.168.1.1:5001 \
                capture_cores=1 writeout_core=3 \
              eth3=/tmp/eth3.pcap \ streams=udp:127.0.0.1:8080 snap=60 \
                capture_cores=5,7 writeout_core=9 \
              eth4=/tmp/eth4.pcap join_mcasts="224.1.1.100;239.2.2.200" \
                capture_cores=2,4 writeout_core=6 snap=160
```

This command line can be placed in a single file:

```
snap=0
eth2=/tmp/eth2.pcap
streams=tcp:192.168.1.1:5001
capture_cores=1
writeout_core=3
eth3=/tmp/eth3.pcap
streams=udp:127.0.0.1:8080
snap=60
capture_cores=5,7
writeout_core=9
eth4=/tmp/eth4.pcap
join_mcasts="224.1.1.100;239.2.2.200"
capture_cores=2,4
writeout_core=6
snap=160
```

NOTE: Each command line argument appears on a separate line. The command configuration file can then be identified when starting solar_capture e.g.

```
solar_capture config_file=serverX.cfg
```

For the standard command line, options appearing before any capture instance are global options. Options appearing after a capture instance affect only that instance.

8.10 Port Aggregation/Merging

Using SolarCapture, the captured traffic from one or more interfaces - from the same or from different adapters can be aggregated into a single receive stream. **Packets captured will be written to the same pcap file**.

The following example demonstrates the correct method to capture from two interfaces to the same capture file:

```
solar_capture eth1=/tmp/capturefile.pcap join_streams=udp:224.1.2.3:319" \
              eth2=/tmp/capturefile.pcap join_streams=udp:225.1.2.3:8100
```


© Copyright 2017-2019 Xilinx Inc

8.11 Multiple Ports/Multiple Files

The captured traffic from one or more interfaces – from the same or from different adapters – can be captured to different capture files as follows:

```
solar_capture eth1=/tmp/cap1.pcap join_streams=udp:224.1.2.3:319" \
              eth2=/tmp/cap2.pcap join_streams=udp:225.1.2.3:8100
```

or like this:

```
solar_capture eth1=/tmp/cap1.pcap join_streams=udp:224.1.2.3:319"
```

```
solar_capture eth2=/tmp/cap2.pcap join_streams=udp:225.1.2.3:8100"
```

8.12 Software Based Filtering

Software Filtering allows the user to create user-level filters using the Berkeley Packet Filter (BPF) format. Software filters can be created when using the SolarCapture Pro API or can be specified from the solar_capture command line.

To create software filters on the command line, use the command line `filter` option e.g.

```
$ solar_capture eth2=<capture_file> filter="port 80"
```

Users interested in BPF format software based filtering should refer to:

http://www.tcpdump.org/papers/bpf-usenix93.pdf.

See also Sniff Mode on page 27 for examples of bpf filter syntax.

8.13 Capture using VLAN Identifier

SolarCapture can only capture from a physical interface. The following syntax is wrong and will not capture any packets from the VLAN interface:

```
solar_capture snap=0 eth2.117=<capture_file>
```

To capture from VLAN eth2.117 use the following syntax (examples):

```
solar_capture snap=0 \
              eth2=<capture_file> streams="eth:vid=117, 00:0F:53:16:04:78"
```

OR

```
solar_capture eth2=<capture_file> join_streams=udp:224.1.2.3:8001;vid=117
```


© Copyright 2017-2019 Xilinx Inc

8.14 Rotate Capture Files

The following config file examples demonstrates how to ensure capture files have a unique name and how to apply post rotate commands.

```
enp5s0f0=%Y%m%d-%H:%M:%S.pcap
streams=udp:224.1.2.3:8001
rotate_seconds=60
postrotate_command=gzip "$F"

The example above produces gzip capture files with each having a unique
data/time name e.g.

1816 Nov 27 11:46 20171127-11:46:00.pcap.gz
5315 Nov 27 11:47 20171127-11:47:00.pcap.gz
5320 Nov 27 11:48 20171127-11:48:00.pcap.gz
```

Rotate multiple files:

In the following example, two streams are captured from the same interface to different capture files.

The rotate file size is specified as a global value. In the following example file1 is rotated when it reaches a size of 20000 bytes, file2 when it reaches 40000 bytes.

```
solar_capture rotate_file_size=20000 enp4s0f0=/temp/file1.\$i.pcap
join_streams="udp:224.0.1.129:319"   enp4s0f0=/temp/file2.\$i.pcap
join_streams="udp:224.0.1.129:320" rotate_file_size=40000
```

In the following example, two streams are captured from different interfaces to the same capture file.

The rotate_file_size is specified as a global variable as there is only one capture file.

```
solar_capture rotate_file_size=5000 enp4s0f0=/temp/file1.\$i.pcap
join_streams="udp:224.0.1.129:319"  enp4s0f1=/temp/file1.\$i.pcap
join_streams="udp:224.0.1.129:320"
```


© Copyright 2017-2019 Xilinx Inc

8.15 Command Line Examples

- Capture all traffic from a single interface. Packet capture will be handled on CPU core 8 and written to file using CPU core 12:

Capture interface	eth2
Capture file	capfile
capture_cores	8
writeout_core	12

```
solar_capture eth2=capfile capture_cores=8 writeout_core=12
```

- Capture all traffic from multiple interfaces:

Capture interface	eth2 and eth3
Capture file	eth2file.pcap and eth3file.pcap

```
solar_capture interface=eth2 output=eth2file.pcap \
              interface=eth3 output=eth3file.pcap
```

- Capture a single UDP stream:

Protocol	udp
Capture interface	eth2
Destination address	123.0.1.129
Destination port	319
Source address	172.16.128.28
Source port	319
Capture file	capfile

```
solar_capture streams="udp:123.0.1.129:319,172.16.128.28:319" eth2=capfile
```

- Capture all traffic arriving at specific MAC address:

local MAC address	00:0F:53:16:04:74
Capture interface	eth2 - the interface corresponding to the MAC address
Capture file	capfile

```
solar_capture streams=eth:00:0F:53:16:04:74 eth2=capfile
```


© Copyright 2017-2019 Xilinx Inc

For a complete listing and description of command line options, use command:

```
solar_capture help all
```

or refer to Table 3 on page 31.

8.16 User Privileges

SolarCapture supports the ‘user’ command line option - which functions like the tcpdump -Z argument. If SolarCapture is started by root, following startup, the user ID is changed allowing the lower privileged user to access to the capture files and to terminate the SolarCapture application.

```
# solar_capture user=1234
# solar_capture user=jdoe
```


© Copyright 2017-2019 Xilinx Inc

9 Application Clustering

9.1 Application Clusters

The application clustering feature allows the captured traffic load, from one or more physical interfaces to be spread amongst multiple receiving applications.

Using a configuration file, the user defines one or more ‘clusters’. Each cluster identifies a capture interface, a set of hardware filters to define a subset of traffic to capture and includes a number of Virtual Interfaces (VI) over which to spread this traffic. The virtual interface is a receive channel between the adapter hardware and the software application.

Application clustering uses hardware filters to describe the traffic to be captured within each cluster. It then uses RSS to spread this traffic over the configured number of VIs within the cluster.

Multiple instances of `solar_capture` are run with each instance receiving traffic from one or more VIs. Each solar_capture instance captures packets to a separate capture (.pcap) file. Multiple instances of `solar_capture` can use the same cluster ID to receive a portion of the total received traffic from the `CaptureInterface` identified in the cluster definition. No two applications will receive replicated or duplicated traffic.

© Copyright 2017-2019 Xilinx Inc

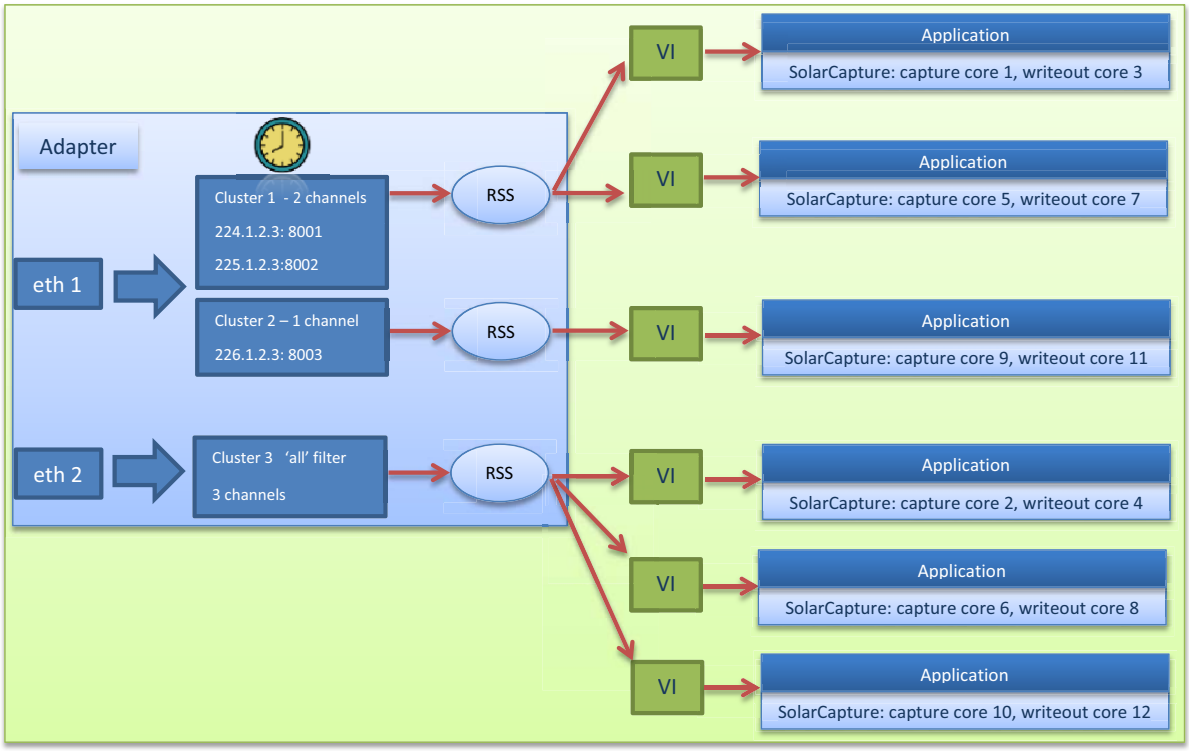


Figure 6: Application Clustering with SolarCapture

Figure 6 above shows an example configuration of application clustering. Hardware filters are used to capture a subset of the traffic received on eth1 into two clusters.

The first cluster uses RSS to spread this traffic over two VIs. In contrast, ALL traffic received on eth2 is captured into a single cluster which is then spread over three VIs.

The number of receive channels created per cluster is configured with the `NumChannels` option in the configuration file. The number of channels can be any value. However, if there are less receiving applications than there are `NumChannels`, then some of the received traffic will be lost.

9.2 Configuration Sequence

The following steps provide the configuration file and `solar_capture` command lines required for the example scenario shown in Figure 6 above.

1 Configuration File

The configuration file can be placed anywhere on the host server and have any name/any extension - .cfg is recommended. The following is an example configuration file and associated command lines.

Capture streams can use 4-tuple dhost,dport,shost,sport or 2-tuple dhost,dport descriptions.

© Copyright 2017-2019 Xilinx Inc

```
[Cluster cluster1]
CaptureInterface = eth1
CaptureMode = steal
NumChannels = 2
CaptureStream = udp,dhost=224.1.2.3,dport=8001,shost=172.16.131.156,sport=3010
CaptureStream = udp,dhost=225.1.2.3,dport=8002,shost=171.10.111.150,sport=3144

[Cluster cluster2]
CaptureInterface = eth1
CaptureMode = steal
NumChannels = 1
CaptureStream = udp,dhost=226.1.2.3,dport=8003

[Cluster cluster3]
CaptureInterface = eth2
CaptureMode = steal
NumChannels = 3
CaptureStream = all
```

NOTE: Values should NOT be encased in quotes.

2 Run solar_clusterd daemon

```
$ solar_clusterd -f <path to config file>
```

This starts the `solar_clusterd` daemon and establishes the `solar_capture` environment. The -f option ensures that the `solar_clusterd` daemon will run in the foreground.

solar_clusterd can also be run as a Linux service using the standard start, stop, restart and status commands.

3 Run solar_capture

For each channel defined in the config file, start an instance of `solar_capture` to capture a portion of the traffic.

For each cluster the number of `solar_capture` instances running must be equal to the `NumChannels` otherwise a portion of the total received traffic will be lost.

```
$ solar_capture eth1=/tmp/cap11.pcap capture_cores=1 writeout_core=3 \
                 cluster=cluster1
$ solar_capture eth1=/tmp/cap12.pcap capture_cores=5 writeout_core=7 \
                 cluster=cluster1
$ solar_capture eth1=/tmp/cap13.pcap capture_cores=9 writeout_core=11 \
                 cluster=cluster2
$ solar_capture eth2=/tmp/cap21.pcap capture_cores=2 writeout_core=4 \
                 cluster=cluster3
$ solar_capture eth2=/tmp/cap22.pcap capture_cores=6 writeout_core=8 \
                 cluster=cluster3
$ solar_capture eth2=/tmp/cap23.pcap capture_cores=10 writeout_core=12 \
                 cluster=cluster3
```


© Copyright 2017-2019 Xilinx Inc

9.3 Snort Example

The following example uses the Snort network intrusion prevention and intrusion detection application with libpcap and SolarCapture Pro application clustering. For more information about Snort see the following link http://www.snort.org/.

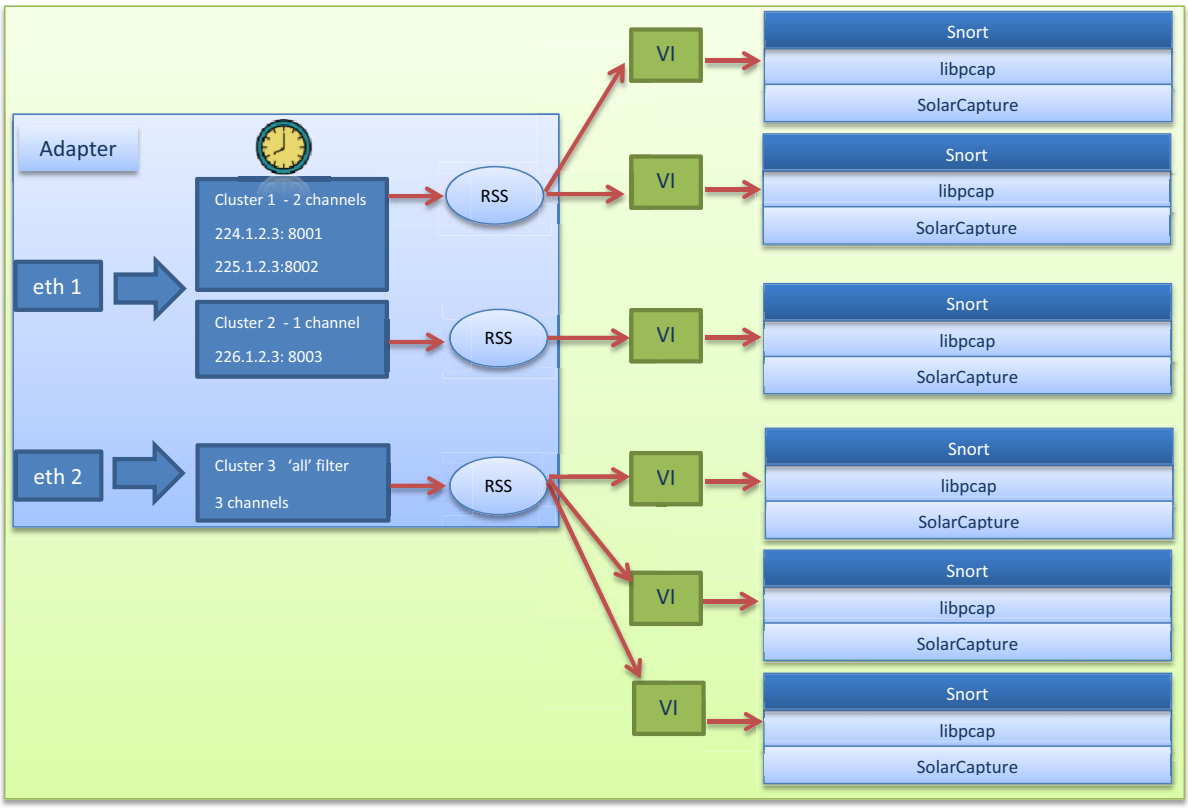


Figure 7: Application Clustering with libpcap application

The configuration sequence for the scenario shown in Figure 7 uses the same configuration file and captures the same traffic as demonstrated in the previous example(Figure 6 on page 46). Each instance of the Snort application uses the SolarCapture enabled libpcap library to gain SolarCapture functionality.

© Copyright 2017-2019 Xilinx Inc

9.4 Configuration Sequence - Snort

The following steps provide the configuration file and command lines required for the scenario shown in Figure 7 above.

1 Configuration File

The configuration file can be placed anywhere on the host server and have any name/any extension - .cfg is recommended. The following is an example configuration file and associated command lines.

Capture streams can use 4-tuple dhost,dport,shost,sport or 2-tuple dhost,dport descriptions.

```
[Cluster cluster1]
CaptureInterface = eth1
CaptureMode = steal
NumChannels = 2
CaptureStream =
udp,dhost=224.1.2.3,dport=8001,shost=172.16.131.156,sport=3010
CaptureStream =
udp,dhost=225.1.2.3,dport=8002,shost=171.10.111.150,sport=3144

[Cluster cluster2]
CaptureInterface = eth1
CaptureMode = steal
NumChannels = 1
CaptureStream = udp,dhost=226.1.2.3,dport=8003

[Cluster cluster3]
CaptureInterface = eth2
CaptureMode = steal
NumChannels = 3
CaptureStream = all
```

NOTE: Values should NOT be encased in quotes.

2 Run solar_clusterd daemon

```
$ solar_clusterd -f <path to config file>
```

3 Run Snort Applications

```
$ SC_PCAP_THREAD="eth1=1" SC_ATTR="cluster=cluster1" solar_libpcap snort \
  <snort command line>
```

`solar_libpcap` is used to cause Snort to link to the SolarCapture enabled libpcap library to gain SolarCapture functionality.

The `SC_PCAP_THREAD` environment variable instructs the SolarCapture libpcap library to create an additional thread for packet processing outside of the main Snort application threads. In the above command line, this thread is affinitized to CPU core **1**.

If `SC_PCAP_THREAD` is not defined, all capture processing is instead performed in the context of the Snort libpcap API calls.

The cluster used by this instance of Snort is identified using the `SC_ATTR` environment variable.

© Copyright 2017-2019 Xilinx Inc

This process is repeated for each instance of snort as follows:

```
$ SC_PCAP_THREAD="eth1=3" SC_ATTR="cluster=cluster1" solar_libpcap snort \
  <snort command line>
$ SC_PCAP_THREAD="eth1=5" SC_ATTR="cluster=cluster2" solar_libpcap snort \
  <snort command line>
$ SC_PCAP_THREAD="eth2=2" SC_ATTR="cluster=cluster3" solar_libpcap snort \
  <snort command line>
$ SC_PCAP_THREAD="eth2=4" SC_ATTR="cluster=cluster3" solar_libpcap snort \
  <snort command line>
$ SC_PCAP_THREAD="eth2=6" SC_ATTR="cluster=cluster3" solar_libpcap snort \
  <snort command line>
```

NOTE: Specifying an affinity value of **-1** with SC_PCAP_THREAD, i.e. eth3=**-1**, creates a thread but does not affinitize this thread to any core.

9.5 solar_clusterd Configuration File

The `solar_clusterd` configuration file used by clustering applications is a text file conforming to the File Structure Conventions on page 104.

The configuration file contains one or more `[Cluster]` sections and a number of per-cluster properties identified in the following table.

Table 4: [Cluster]

Property	Description
CaptureInterface	The interface to capture traffic from. This should be specified only once for each cluster.
NumChannels	The number of receive channels to create. If there are less receiving applications than receive channels, a portion of the captured traffic will be lost. Default value is 1.
CaptureStream	Identify the streams to capture traffic from. Each stream should be prefixed with the CaptureStream property. And each stream should appear on a separate line.
CaptureMode	sniff|steal The default capture mode is steal.
ProtectionMode	Protection Domain Mode. If not specified the default value is EF_PD_DEFAULT. For other values see the file: `/src/include/etherfabric/pd.h`

© Copyright 2017-2019 Xilinx Inc

9.6 Running solar_clusterd as a Linux service

The `solar_clusterd` daemon can also be run as a Linux service using the standard Linux service commands, start, stop, restart.

© Copyright 2017-2019 Xilinx Inc

10 Libpcap Support

10.1 Introduction

The Solarflare enabled libpcap library `solar_libpcap` allows existing applications that are dynamically linked with the standard libpcap to be accelerated and gain SolarCapture functionality. Recompilation is not required.

The `solar_libpcap` library is supported in SolarCapture Pro. The library is located in the following directory:

```
/usr/lib64/solar_capture/libpcap
```

10.2 Usage

By default, applications will continue to dynamically link to the standard libpcap library. The following command example will cause runtime linking to the `solar_libpcap` library.

```
solar_libpcap tcpdump -i eth2
```

Thereafter, pcap API calls made by the application are processed by the SolarCapture libpcap library.

To check that the application is dynamically linked with `solar_libpcap`, use the Linux ldd command:

```
# solar_libpcap ldd /usr/sbin/tcpdump

  linux-vdso.so.1 =>  (0x00007fff49dff000)
  libcrypto.so.10 => /usr/lib64/libcrypto.so.10 (0x0000003c74600000)
  libpcap.so.1 => /usr/lib64/solar_capture/libpcap/libpcap.so.1 (0x00007fafd8f27000)
  libc.so.6 => /lib64/libc.so.6 (0x0000003c66600000)
  libdl.so.2 => /lib64/libdl.so.2 (0x0000003c66200000)
  libz.so.1 => /lib64/libz.so.1 (0x0000003c67200000)
  librt.so.1 => /lib64/librt.so.1 (0x0000003c67600000)
  libm.so.6 => /lib64/libm.so.6 (0x0000003c66e00000)
  libaio.so.1 => /lib64/libaio.so.1 (0x00007fafd8d24000)
  /lib64/ld-linux-x86-64.so.2 (0x0000003c65e00000)
  libpthread.so.0 => /lib64/libpthread.so.0 (0x0000003c66a00000)
```

Refer to Snort Example on page 48 for an example of using Snort with `solar_libpcap`.

© Copyright 2017-2019 Xilinx Inc

10.3 Device names

The device name passed by the application to `pcap_open_live()` or similar is typically:

- The name of a network interface.

  ```
  solar_libpcap tcpdump -i eth4
  ```

Alternatively it can take one of the following forms:

- The name of an application cluster.

  ```
  solar_libpcap tcpdump -i cluster1
  ```

- A specific channel of an application cluster:

  ```
  solar_libpcap tcpdump -i 2@cluster1
  ```

- A shared memory channel:

  ```
  solar_libpcap tcdpump -i scshm:/dev/hugepages/eth6
  ```

  The above example sources packets from a shared memory channel.

- A node specification:

  ```
  solar_libpcap tcdpump -i sc:sc_shm_import:path=/dev/hugepages/eth6
  ```

  The above line instructs the libpcap bindings to create a node of type `sc_shm_import`, with the `path` argument set. Arguments are separated by semicolons. Packets emitted by that node are passed to `tcpdump` through the libpcap interface. This example has the same effect as using `scshm`.

  Node specifications can be used to customize the packet stream exposed by the libpcap bindings. The node that is instantiated can be one of the builtin nodes (as above) or a custom node written using the C bindings.

10.4 Using libpcap with solar_clusterd

It is possible to use solar_clusterd together with the libpcap bindings to spread packets over multiple application instances or threads. This can also be useful when not spreading load in order to customize capture parameters. For example, you can use the solar_clusterd configuration file to set the `CaptureStream` or `CaptureMode`.

Refer to Application Clustering on page 45 for more details regarding applications clustering.

In the following example, solar_clusterd is used to configure the libpcap bindings to use `sniff` mode in order to capture a copy of the packets received by the network interface.

1 Create the solar_clusterd config file (name it `conf1.cfg`):

```
[Cluster cluster1]
CaptureInterface = eth6
CaptureMode = sniff
NumChannels = 1
CaptureStream = all
```


© Copyright 2017-2019 Xilinx Inc

2 Start solar_clusterd daemon – specifying the name of the config file:

```
# solar_clusterd conf1.cfg
solar_clusterd version: 201811
Cluster cluster1: eth6, 1 channels, EF_PD_DEFAULT
```

3 Start the application using solar_libpcap – but also identify the cluster to be used so that it picks up the config:

```
# solar_libpcap tcpdump -i cluster1 -w eth6.pcap
```

The example uses tcpdump using solar_libpcap. In sniff mode network traffic is delivered to the end consumer application and also captured to the tcpdump pcap file.

10.5 Configuration

The following tunable environment variables are available when using SolarCapture libpcap bindings. Use the export command to set variables in the application environment, for example,

```
export SC_PCAP_LOG_FILE=/tmp/filename
```

SC_ variables can also be set using the SC_ATTR environment variable - refer to SolarCapture Attributes on page 106 for details.

SC_PCAP_THREAD

Range: CPU core
Default: none

By default, SolarCapture creates no additional processing threads. All capture processing is performed in the context of the libpcap API calls. Alternatively, `solar_libpcap` can use a dedicated thread for processing received packets delivered by the adapter.

The SC_PCAP_THREAD environment variable instructs `solar_libpcap` to create an additional thread for packet processing outside of the linked application threads.

The value specified can be a single integer to identify the core to which the thread is affinitized:

```
export SC_PCAP_THREAD=2
```

A value of -1 means the thread is created, but not affinitized to any particular CPU core:

```
export SC_PCAP_THREAD=-1
```

A comma separated list identifies each capture interface and a CPU core on which to process packets captured from the interface.

```
export SC_PCAP_THREAD="eth1=3,eth2=5"
```

If SC_PCAP_THREAD is not defined, all capture processing is instead performed in the context of the application->libpcap API calls.

© Copyright 2017-2019 Xilinx Inc

SC_PCAP_RECV_BATCH

Range: 1 -
Default: 4 (normal mode), 10 (packed-stream mode)

Controls how often SolarCapture polls the adapter for new packets when the packet arrival rate exceeds the processing rate.

Set to 1 means to poll the adapter for each received packet. Set to the default value 4, means poll the adapter every 4 packets received.

A lower value means more CPU time is given to the capture thread, reducing the risk of loss under bursty traffic conditions.

A higher value means more CPU time is given to the application processing, so processing efficiency is higher.

NOTE: SC_PCAP_RECV_BATCH has no effect if a dedicated capture thread has been created using the SC_PCAP_THREAD option.

SC_PCAP_SOURCE_$devname

This allows you to replace the device name specified in `pcap_open_live()` or similar with an alternative device name or specification. For example:

```
export SC_PCAP_SOURCE_eth4=cluster1
solar_libpcap tcpdump -i eth4
```

In this example `tcpdump` requests to capture from interface `eth4`, but the libpcap bindings will instead use interface `cluster1`, which might be an application cluster.

SC_PCAP_LOG_LEVEL

Range: 0 - 3
Default: 1 (errors only)

Specify the logging level of output produced by SolarCapture.

SC_PCAP_LOG_FILE

Range: filename
Default: none

Identify the file for logging output.

© Copyright 2017-2019 Xilinx Inc

SC_PCAP_NANOSEC

Range: 0 - 1
Default: 0

Specify SolarCapture timestamp resolution. 0 is microseconds, 1 is nanoseconds.

The environment variable is an alternative to the `pcap_set_tstamp_precision()` function call. An application linked to `solar_libpcap` must be separately configured to expect nanosecond precision.

© Copyright 2017-2019 Xilinx Inc

11 Data Acquisition Module

11.1 Introduction

Solarflare `daq_sfsc` is a library module developed for the Snort Data Acquisition Module (DAQ) framework.

The Solarflare DAQ is installed to:

```
/usr/lib64/solar_capture/daq/daq_sfsc.so
```

The snort DAQ supports the *read-file*, *passive* and *inline* modes.

Using *read-file* mode, packets are read from a pcap file and passed to Snort. In *passive* mode the DAQ will send all captured packets to Snort before silently discarding them. In *inline* mode the DAQ will send all captured packets to Snort, but can forward any packets not rejected by Snort rules to a Solarflare interface.

For more information about Snort and the DAQ see https://www.snort.org/

11.2 Usage

The DAQ can be identified from the command line:

```
snort --daq-dir /usr/lib64/solar_capture/daq --daq sfsc
```

Alternatively the DAQ can be identified by adding the following lines to the Snort configuration file:

```
config daq_dir: /usr/lib64/solar_capture/daq
config daq: sfsc
```

11.3 Read File Mode

In read file mode packets read from a pcap file are passed to Snort via the DAQ. Packets are not available to be forwarded again to a Solarflare interface.

Configure read-file mode on the Snort command line:

```
snort -r <pcap_file>
```

or in the Snort configuration file:

```
config daq_mode:  read-file
config interface: <pcap-file>
```


© Copyright 2017-2019 Xilinx Inc

11.4 Passive Mode

In passive mode, the DAQ passes all captured packets to Snort before silently discarding them.

Configure passive mode on the Snort command line:

```
snort -i <capture-interface>
```

or in the Snort configuration file:

```
config daq_mode:  passive
config interface: <capture-interface>
```

11.5 Inline Mode

In Inline mode, the DAQ passes all captured packets to Snort, but will also forward packets not rejected by Snort rules to a Solarflare interface. The forwarding interface may be the capture interface or it may be a different Solarflare interface.

Configure inline mode on the Snort command line:

```
snort -Q -i <capture-interface>:<forward-interface>
```

or in the Snort configuration file:

```
config daq_mode:  inline
config interface: <capture-interface>:<forward-interface>
```

11.6 Configuration

Solarflare DAQ configuration parameters are specified on the command line using the following syntax:

```
--daq-var key=value
```

Parameters can also be placed in the Snort configuration file:

```
config daq_var: key=value
```

ns_timestamps

Range: 0 - 1
Default: 0

Specify the timestamp resolution of packets passed to Snort. Set to 0 for microseconds. Set to 1 for nanosecond resolution.

To ensure timestamps are reported correctly, the output plugin must be configured to expect nanoseconds values in the `ts.tv_usec` field of the `DAQ_PktHdr_t` structure.

© Copyright 2017-2019 Xilinx Inc

buffer_size

Range: none
Default: 65536 (bytes)

Identify the maximum buffer size (bytes) to receive packets into the DAQ in a single read operation.

capture_core

Range: CPU core
Default: none

By default packet capture is performed in the context of the DAQ/snort libpcap API calls. Alternatively, a dedicated thread can be selected for processing received packets.

The following example will creates a thread and affinitize this to the specified core:

```
capture_core=2
```

A value of -1 means the thread is created, but not affinitized to any particular CPU core:

```
capture_core=-1
```

If `capture_core` is not defined, all capture processing is instead performed in the context of the application->libpcap API calls.

recv_batch

Range: 0 - 32
Default: 4

Controls how often SolarCapture polls the adapter for new packets when the packet arrival rate exceeds the processing rate.

Set to 1 means to poll the adapter for each received packet. Set to the default value 4, means poll the adapter every 4 packets received.

A lower value means more CPU time is given to the capture thread, reducing the risk of loss under bursty traffic conditions.

A higher value means more CPU time is given to the packet processing, so processing efficiency is higher.

NOTE: The recv_batch value has no effect if a dedicated capture_core has been specified using the capture_core option.

© Copyright 2017-2019 Xilinx Inc

streams

Range: none
Default: all

Identify a subset of received packets to be captured. If no streams are specified, all received packets are captured.

Refer to Selecting Streams to Capture on page 36 for streams examples and also use the following command for further streams syntax:

```
solar_capture help streams
```

When adapters are not able to filter TCP and UDP streams by VLAN-ID, the VLAN specification is ignored for capture, but is used for the purposes of joining multicast groups (join_streams).

See Filtering on VLAN on page 89 for limitations detail.

11.7 Limitations

Refer to Known Issues and Limitations on page 87.

 © Copyright 2017-2019 Xilinx Inc

12 SolarReplay

12.1 Introduction

SolarReplay allows packets captured to file in libpcap format to be replayed through a Solarflare adapter interface.

The packets could be first captured using SolarCapture, or from any other source e.g. tcpdump or wireshark.

Command line options provide flexible control over replay speed and bandwidth whilst preserving inter-packet pacing.

Packets can be replayed between two ports of the same adapter, between two adapters in the same host or replayed from a source server to a remote destination server where the network path may include network switches.

12.2 How it Works

Packets from the capture file are read from the capture file through a reader node and then to the injector node before passing directly to the Solarflare adapter to be replayed through the specified interface.

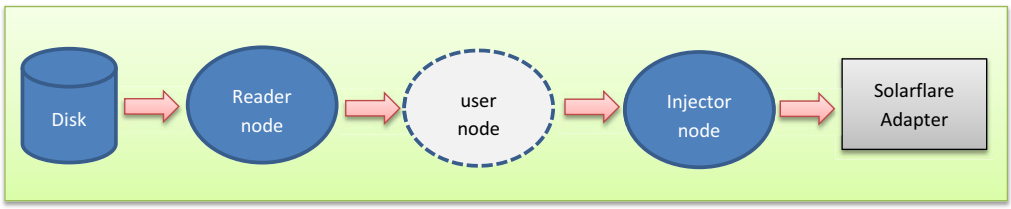


Figure 8: SolarReplay Sequence

The (optional) node command line option provides extra flexibility allowing the user to insert customized nodes into the replay path and perform pre-processing of packets, for example, filters could be applied, source and destination addresses, port number could be altered to match network requirements.

If the prebuffer option is specified on the command line, packets from the capture file will first be buffered by SolarReplay before insertion into the replay path. This is useful and may be necessary to facilitate reading the capture file from slower storage devices.

© Copyright 2017-2019 Xilinx Inc

12.3 Getting Help

To display all SolarReplay command line options:

```
# solar_replay --help
```

For a detailed description of a specific option:

```
# solar_replay help <option>
```

12.4 Replay Command Line

Syntax

```
solar_replay [global-opts] <interface>=<pcap> [stream-opts]
```

Global options apply to all streams and may be overridden by stream options which apply only to the interface immediately preceding them.

Options

Table 5: SolarReplay Options

Option	Description
bpf	Replay only packets from the capture file matching the given Berkeley Packet Filter (BPF) specification. `bpf="port 80"` See Sniff Mode on page 27 for further BPF syntax examples.

bw	Specify maximum bandwidth bits-per-second rate when replaying the pcap file. The suffixes `k`, `M` and `G` can be used to specify higher rates, and are interpreted as powers of 10.
	`bw=1000`
	`bw=2M`

ctl_script	Specify a script of SolarReplay options in a single script file. This allows the user to simulate different traffic patterns and traffic burst periods by varying options such as the speedup, bandwidth, pause, packet-per-second and repeat options. Use the following command for more details: `solar_replay help ctl_script`

© Copyright 2017-2019 Xilinx Inc

Table 5: SolarReplay Options (continued)

Option	Description
injector_core	Identify the CPU core processing packets through the injector node. If you assign two injectors to the same core, they will run in the same thread. `injector_core=5` CPU cores can be assigned per stream when more than a single stream is identified on the command line. Default: each injector has its own non-affinitized thread.
input	Specify the input file to read from.
interactive	Connects stdin to the playback process allowing the user to control replay by executing command line options such as pause, speedup and bandwidth. Use the following command for more details: `solar_replay help interactive`

node	Customized node(s) to be inserted into the replay path allow pre-processing of captured packets before replay.
	Example:
	`node_name:[node_args]`
	`./solar_replay my_node:range=1-100,bw=1000`

num_repeats	Set to loop back round to the first packet once all packets have been replayed, up to the specified number of repeats.
	This option requires at least as many buffers as there are packets in the input file. Additional buffers can be allocated using the SC_ATTR environment variable:
	`SC_ATTR="n_bufs_tx=1024" solar_capture...`
	or
	`export SC_ATTR="n_bufs_tx=2048"`
	Example:
	`num_repeats=5`

© Copyright 2017-2019 Xilinx Inc

Table 5: SolarReplay Options (continued)

Option	Description
packet_range	Select a subset of the input file for transmit, based on packet index within the file. Specify a comma-separated list of indexes or ranges: • ranges may be open or closed. • the specified ranges must be non-overlapping and in order • ranges are inclusive at both ends • Indexes are 0-based. For example: • `packet_range=1-100` all packets 1-100 • `packet_range=1-10,200-250` packets 1-10 then 200-250 • `packet_range=1,3,5` packets 1, 3 and 5 only • `packet_range=100-` from packet 100 onwards
pause	Pause the given number of seconds at startup before replaying the first packet. This option is ignored in interactive mode, which does not replay any packets at startup until it receives a command to do so. Example: `pause=10`

pps	Replay packets at a fixed rate, transmitting the given number of packets per second. The suffixes `k`, `M` and `G` can be used to specify higher rates, and are interpreted as powers of 10.
	Example:
	`pps=1.5e6`
	`pps=1000`
	`pps=2M`

© Copyright 2017-2019 Xilinx Inc

Table 5: SolarReplay Options (continued)

Option	Description

prebuffer	Buffer the full input file before sending any packets. If there are not enough buffers for the entire file, solar_replay will keep reading until it runs out of buffers, and then start sending.
	This is particularly useful when the capture file has to be read from a slow storage device, but must be replayed at high speed.
	Each packet is read into a single buffer from the default allocation of 512 buffers. Additional buffers can be allocated using the SC_ATTR environment variable:
	`SC_ATTR="n_bufs_tx=1024" solar_capture...`
	or
	`export SC_ATTR="n_bufs_tx=2048"`
	Note: Reading the capture file from slow storage devices may cause the replay to slow down when packets cannot be read into buffers quick enough to keep pace with the replay speed, bw or pps parameters.

reader_core	Identify the CPU core processing packets through the reader node. If you assign multiple readers to the same core, they will still run in separate threads. Example: `reader_core=3` Default: each reader has its own non-affinitized thread.

repeat	Set to loop back round to the first packet once all packets have been replayed, and continue to do so until killed.
	This option requires at least as many buffers there are packets in the input file. Additional buffers can be allocated using the SC_ATTR environment variable:
	`SC_ATTR="n_bufs_tx=1024" solar_capture...`
	or
	`export SC_ATTR="n_bufs_tx=2048"`
	Example:
	`repeat=1`

© Copyright 2017-2019 Xilinx Inc

Table 5: SolarReplay Options (continued)

Option	Description
speedup	Speed up (or slow down) packet replay by the given factor, while preserving packet pacing dictated by the packet timestamps. This can be useful to simulate differing traffic patterns and burst conditions: • `speedup=10` replay is ten times faster • `speedup=0.1` replay is 10 times slower The speedup option is incompatible with either the pps or bw options.

time_range	Select a subset of the input file for transmit, based on packet timestamp.
	Use the following command for details of timestamp formats:
	`solar_replay help time_range`
	`Examples:`
	`From 3 mins into capture file until 4 mins into capture file:`
	`time_range=3m-4m`
	`From 10 mins into capture file for 30 seconds:`
	`time_range=10m-+30s`
	`Packets with timestamps between specific times:`
	`time_range=15:10:00-15:25:00`
	`From a specific capture time:`
	`time_range=15:24:00-`

Examples

```
solar_replay pps=1000 interface=eth2 input=example.pcap

solar_replay interface=eth2 input=play1.pcap speedup=10 \
             interface=eth3 input=play2.pcap
```


© Copyright 2017-2019 Xilinx Inc

12.5 Replay Script File

A control script can be used to simulate traffic patterns using SolarReplay options to control packet send rates.

The control script is a text file having any name and any extension. To view an example script with detailed syntax information and list commands which can be included, use the following command:

```
solar_replay help ctl_script
```

The following is an example control script file:

```
pps 100
for 1s
pps 200
for 1s
pps 400
for 1s
pps 800
for 1s
pps 1600
for 1s
stop
```

The final line in the control script should always be the '**stop**' label which will cause the playback to terminate.

Identify the control_script on the SolarReplay command line:

```
solar_replay interface=eth4 input=eth4.pcap ctl_script=playctl.txt
```


© Copyright 2017-2019 Xilinx Inc

13 SolarCapture Monitor

13.1 Introduction

The `solar_capture_monitor` utility reports configuration and runtime data for instances of SolarCapture.

SolarCapture exports runtime data via shared memory-mapped files, and `solar_capture_monitor` maps and reads those files to provide visibility of SolarCapture operation. This mechanism ensures that `solar_capture_monitor` has very little impact on SolarCapture performance.

13.2 Getting Help

```
$ solar_capture_monitor --help
```

13.3 Monitor Command Line

Syntax

```
solar_capture_monitor [options] [sessions] [commands]
```

Commands

Command	Description
dump	Dump complete state of session
list	List pid and user-id of instance
nodes	Dump table of nodes with packet counts
nodes_rate	Continuously updated table of nodes with packet rates
line_rate	Line-by-line output with packet rate and bandwidth
line_total	Line-by-line output with packet and byte counts
poke obj.attr=val	Overwrite an object attribute
dot [mailboxes] [free_path]	Output topology graph in graphviz format, optionally showing mailboxes and free path

© Copyright 2017-2019 Xilinx Inc

Sessions

Session identifier	Description
pid	All sessions for the given process
pid/session_id	Specific session from the given process
directory	Log directory for a session

If no sessions are specified, then all running sessions belonging to the user are selected.

Options

Option	Description
-h, --help	Show help message and exit
--running	Select running sessions (default)
--stopped	Select stopped sessions
--all	Select running and stopped sessions
--user=USER	Select sessions owned by this (trusted) user
--interval=INTERVAL	Time interval in between output updates
--localtime	Use local time (default is UTC)
--strftime=STRFTIME	Specify format string for timestamps
--base-dir=BASE_DIR	Location of stats directory
--debug	Show source of errors

© Copyright 2017-2019 Xilinx Inc

Examples

- To list running processes using solar_capture:

```
$ solar_capture_monitor
#pid   user-id log-directory
22175  root    /var/tmp/solar_capture_22175
22177  root    /var/tmp/solar_capture_22177
```

- To display full output from all running instances of solar_capture:

```
$ solar_capture_monitor dump
```

- To display full output from a specific instance of solar_capture:

```
$ solar_capture_monitor <pid> dump
```

  See Output on page 71 for details of the output from the dump command.

- To display counts of packets captured:

```
$ solar_capture_monitor line_total
                  time  eth4-cap-pkts   eth4-cap-bytes  eth4-0-write-pkts
  20140702-09:34:55.033        1398427         83905620            1398362
  20140702-09:34:56.033        1413417         84805020            1413415
  20140702-09:34:57.033        1428394         85703640            1428393
  20140702-09:34:58.033        1443372         86602320            1443369
  20140702-09:34:59.033        1458349         87500940            1458348
```

- To display capture rate and bandwidth:

```
$ solar_capture_monitor line_rate
                  time  eth4-cap-rate  eth4-cap-mbps  eth4-0-write-rate
  20140702-09:36:38.459              0              0                  0
  20140702-09:36:39.459          14976        6.85574              15055
  20140702-09:36:40.459          14977        6.85607              14976
  20140702-09:36:41.459          14977        6.85637              14976
  20140702-09:36:42.459          14977        6.85595              14978
```

  - cap-rate is the packets per second capture rate.
  - cap-mbps is the mega bits per second capture rate.
  - write-rate is the per second rate at which packets are passed to the writeout-core.

- To output a topology graph using Graphviz freeware:

```
$ solar_capture_monitor dot | dot –Tpng > /tmp/graph.png
```

  Graphviz is an optional component in many Linux distributions. It can also be downloaded from http://www.graphviz.org.

© Copyright 2017-2019 Xilinx Inc

Output

Per VI counters are generated by the solar_capture_monitor dump:

- Some fields contain the values of configuration attributes, and are described in detail by the `solar_capture_doc` command. For example:

  ```
  solar_capture_doc attr rx_refill_batch_low
  ```

Field	Description
`name`	sc_vi(0,eth<N>), visible in log messages
`id`	Identifier of this VI
`thread_id`	Identifier of the thread using this VI
`vi_group_id`	RSS group identifier
`pool_id`	Packet pool identifier
`interface_id`	The interface this VI is capturing packets from
`recv_node_id`	Receiver node identifier
`n_rxq_low`	Estimation of number of times the capture thread fell behind See Notes On Monitor Output on page 74.
`n_free_pool_empty`	Estimation of number of times the writeout thread fell behind
`n_rx_csum_bad`	Number of packets that failed with bad TCP, UDP or IP checksum reported by the network adapter
`n_rx_crc_bad`	Number of packets with bad Ethernet CRC reported by the network adapter
`n_rx_trunc`	Number of packets truncated due to network adapter fifo overflow
`n_rx_mcast_mismatch`	Number of unsolicited multicast packets received
`n_rx_ucast_mismatch`	Number of unsolicited unicast packets received
`n_rx_no_desc_trunc`	Number of large packets discarded part way through due to the descriptor ring going empty
`rx_refill_batch_low`	Batch size of packets buffers re-assigned to the rx_ring on each refill after the `rx_ring_low_level` threshold is reached

© Copyright 2017-2019 Xilinx Inc

Field	Description
rx_refill_batch_high	Batch size of packets buffers re-assigned to the rx_ring on each refill after the rx_ring_high_level threshold is reached
poll_batch	Number of packets recovered from the receive queue on each poll
n_bufs_rx_req	Number of packet buffers available to this VI
n_bufs_rx_min	Minimum number of packet buffers that will be reserved for this VI
evq_size	Size of the event queue in this VI
tx_ring_max	Max number of packet buffers that can be held in the tx_ring
rx_ring_max	Max number of packet buffers that can be held in the rx_ring
rx_ring_size	Size of the receive queue in this VI (buffers)
rx_ring_low_level	rx_ring low watermark (number of buffers available)
rx_ring_high_level	rx_ring high watermark (number of available buffers is considered to be sufficient if above this value)
discard_mask	Mask of error packets to discard
hw_timestamps	Whether hardware timestamping is used
n_rx_pkts	Number of packets received
n_rx_bytes	Number of bytes received
n_bufs_out_of_order	Number of times buffers could not be reclaimed in the order of their allocation
packed_stream_mode	1 if using packed_stream mode, else 0

© Copyright 2017-2019 Xilinx Inc

Field	Description
packed_stream_flush_nanos	The attribute batch_timeout_nanos sets the timeout for various forms of preemptive batching operations. Settings this attribute to a higher value will increase latency on the paths where it is used, but may improve efficiency by encouraging batching. Batching can be enabled/ disabled with the rx_batch_nanos attribute. When the packed_stream_flush_nanos timer expires, packed stream buffers are flushed Default: 100000 nanoseconds
packed_backlog	Number of packets in backlog until more packet buffers become available to deliver them into
packed_backlog_max	Backlog fill level (max reached)
n_packed_backlog_enter	Increments when backlog transitions from empty to non-empty
n_wakes	Increments when VI becomes readable after sleep
n_total_ev	Number of TX, RX or RX_DISCARD events
n_tx_ev	Number of completed TX events
n_tx_doorbell	Number of tx_doorbells
tx_backlog	Number of packets to be transmitted
n_tx_backlog_enter	Increments when tx backlog transitions from empty to non-empty
group_name	Object group name visible in log/monitor only
interface	higher level interface tag i.e. bond0, VLAN2
real_interface	real interface identifier

© Copyright 2017-2019 Xilinx Inc

13.4 Debug Level

- When using the SolarCapture API the debug logging level can be set as follows:

  ```
  export SC_ATTR="log_level=6"
  ```

  or

  ```
  SC_ATTR="log_level=6" solar_capture interface=eth1 output=eth1.pcap
  ```

- When using the SolarCapture command line the debug logging level can be set as follows:

  ```
  solar_debug[-l 6] solar_capture ...
  ```

Debug levels are in the range 0 (silent) - 6 (verbose). The default level is 3.

13.5 Notes On Monitor Output

n_rxq_low

Observing `n_rxq_low` events does not necessarily mean that packets are being dropped. This is a fairly conservative warning which is raised when the RX ring drops below 60% fill level. The `n_rxq_low counter` is not relevant when using the `capture-packed-stream` firmware variant.

Identify Asynchronous Writer Mode

Using the following command, SolarCapture is using the asynchronous writer mode is the async_mode is set to 1. Set to 0 the writer mode is synchronous.

```
solar_capture_monitor dump | grep async_mode
async_mode   1
```

Running low on buffers

The pcap packer node `sc_pcap_packer` has an explicit counter, `buffer_low`, which will increment when the pool of pcap buffers runs low.

Use the following command to identify if SolarCapture is running low on buffers:

```
solar_capture_monitor dump | grep buffer_low
buffer_low   0
```

A value of 0 means SolarCapture has sufficient buffers, otherwise the value indicates the number of times SolarCapture has run low on buffers.

© Copyright 2017-2019 Xilinx Inc

Writer backpressure

If the writing thread cannot output the captured packets fast enough, this creates backpressure. Buffering fills up, initially at the output nodes, then progressing back towards the adapters.

The symptoms are increases in the following counters:

- `n_bufs_out_of_order`, reported by `solar_capture_monitor dump`
- `n_free_pool_empty`, reported by `solar_capture_monitor dump`
- `rx_nodesc_drops`, reported by `ethtool -S <interface>`

The issue can be addressed by some or all of the following measures:

- allocate more packet buffers
- use asynchronous writing
- use a different writeout_core for each capture, if running multiple captures with multiple available cores
- write to faster media such as a RAM disk.

See Tuning Guide on page 92 for more detailed tuning instructions.

© Copyright 2017-2019 Xilinx Inc

14 Arista Timestamps

14.1 Using Arista timestamps with SolarCapture

SolarCapture has been developed to take advantage of the Arista 7150 and Arista 7280 switch hardware timestamping features.

The `arista_ts` node supports options to replace the solar_capture timestamp with a timestamp decoded from the Arista switch.

For inline descriptions of all `arista_ts` node options, use the following command:

```
# solar_capture help arista_ts
```

For additional information on Arista switch timestamps, refer to the following white paper:

https://www.arista.com/assets/data/pdf/Whitepapers/Overview_Arista_Timestamps.pdf

Arista 7150

The Arista 7150 switch generates a timestamp for a packet on the ingress port and appends the timestamp to the packet on the egress port. The timestamp is a 32bit tick counter (a tick = ~2.857ns) which is appended to the payload immediately before the FCS or is written in place of the FCS.

Using the replace FCS method, the Arista switch will overwrite the FCS with the timestamp. There is no latency impact and the original frame size is unchanged. Downstream switches will recognize the FCS as invalid (now a timestamp). Cut-through switches will forward the frame but may increment checksum errors on egress ports.

Using the append payload method, the switch will append the timestamp as the last 32 bits of the payload. The FCS is recalculated.

The 32bit timestamp provides the high resolution part of the complete 64bit timestamp.

The 7150 also generates keyframes which provide the low resolution part of the 64bit timestamp plus a mapping between the timestamp and UTC.

SolarCapture uses the keyframes and timestamp field to convert a received packet software timestamp (arrival at the host) to a hardware timestamp (arrival at the switch) to deliver greater accuracy in received packet timestamps.

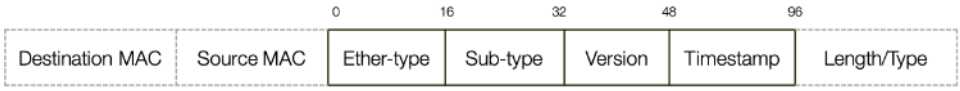
Timestamps are UTC time at the switch.

© Copyright 2017-2019 Xilinx Inc

Arista 7280 - 48bit Timestamp

Encapsulated

By default the Arista 7280 switch encapsulates the 48bit timestamp in the L2 header adding an additional 12 bytes to the packet.



The timestamp is identifiable by the Arista Ethertype **0xd28b** and consists of:

- 2-byte protocol subtype 0x1
- 2-byte protocol version:
  - 0x20 = TAI timestamp, 0x120 = UTC timestamp
- 6-byte low-order 48bits of the timestamp

Overwrite source MAC

The switch can also overwrite the L2 header source MAC address with the 48bit timestamp. This does not increase the packet size.

A receiver running PTP can calculate and append the remaining high-order 16bits. Keyframes are not required when using 7280 timestamps.

Arista 7280 - 64bit Timestamp

The Arista 7280 switch encapsulates the 64bit timestamp in a VLAN type header at the start of each packet.

Inserted in the Ethernet header directly following the original MAC address, the timestamp is identifiable by the Arista Ethertype **0xd28b** and consists of:

- 2-byte protocol subtype 0x1
- 2-byte protocol version:
  - 0x10 = TAI timestamp, 0x110 = UTC timestamp
- 8-byte UTC timestamp (32bit seconds field and 32bit nanoseconds part)

NOTE: For the 7280 switch, the 64bit timestamp format is deprecated in SolarCapture v1.6.6. The 48bit timestamp format is the default decode mode if ts_format is not specified.

© Copyright 2017-2019 Xilinx Inc

14.2 The arista_ts Node

The `arista_ts` node specification supports the following optional arguments. All arguments must be separated by semicolons.

Name	Type	Description	7150	7280
`kf_ip_dest`	val	Dot notation destination IP address for keyframes	Y	
`kf_eth_dhost`	val	Destination MAC address for keyframes	Y	
`kf_ip_proto`	int	The IP protocol used to send key frames. Default: 253	Y	
`log_level`	str	The logging level of the node, must be set to one of “silent”, “errors”, “setup”, “sync” or “verbose”. Default: “sync”	Y	Y
`filter_oui`	str	Filter out timestamps with this OUI.	Y	Y
`kf_device`	str	Filter keyframes by device field.	Y	
`tick_freq`	int	Expected frequency in Hz of the switch tick. Default: 350000000	Y	Y
`max_freq_error_ppm`	int	Max ppm between expected and observed frequency before entering no sync state. Default: 20000	Y	
`lost_sync_ms`	int	Time after last keyframe to enter lost sync state. Default: 20000	Y	
`no_sync_ms`	int	Time after last keyframe to enter no sync state. Default: 60000	Y	

© Copyright 2017-2019 Xilinx Inc

Name	Type	Description	7150	7280
`no_sync_drop`	int	Toggle sync drop, set to 1 for on 0 for off. Default: 0	Y	
`drop_sync_on_skew`	int	Set to 1 to drop synchronization on bad timestamp.	Y	
`strip_ticks`	int	Toggle the option for the node to strip switch timestamps. Set to 0 for off and 1 for on. Default: 1	Y	Y
`switch_model`	int	7150 | 7280	Y	Y
`rollover_window_ms`	int	Window to check for 7280 (64 bit) rollover bug. See Arista documentation.		Y

`ts_format`	str	Identify timestamp format to be decoded:
		`ts_format=48bit`
		`ts_format=64bit`
		48bit is the default value from the 7280 switch
		64bit timestamps are deprecated in SolarCapture 1.6.6.
				Y

`ts_src_mac`	int	Set to 1 if the **48bit** timestamp is located in the L2 header source MAC field. Default=0		Y
`replacement_src_mac`	str	MAC address to replace the **48bit** timestamp which has been received in the L2 header source MAC field. (if ts_src_mac=1)		Y

14.3 Time Synchronization

The clocks on the SolarCapture server and Arista switch must be synchronized to the same time source. High accuracy is not required, but the clocks should be within one second of one another.

© Copyright 2017-2019 Xilinx Inc

14.4 Configuration

Configure Arista timestamps: 7150

Refer to https://eos.arista.com/timestamping-on-the-7150-series/ for guidance on configuring the Arista 7150 switch for packet timestamping.

SolarCapture supports the Arista switch timestamp modes (FCS type):

- 0 timestamping disabled
- 1 timestamp appended to payload
- 2 timestamp overwrites the FCS

This feature can be used with the `solar_capture` command line tool as follows:

```
solar_capture interface=eth2 output=./eth2.pcap \
              arista_ts="kf_ip_dest=255.255.255.255;kf_eth_dhost=ff:ff:ff:ff:ff:ff"
```

or explicitly identify the 7150 switch mode:

```
solar_capture interface=eth2 output=./eth2.pcap \
arista_ts="switch_model=7150;kf_ip_dest=255.255.255.255;kf_eth_dhost=ff:ff:ff:ff:ff:ff"
```

The corresponding configuration for the connected **egress port** on the Arista switch is:

```
switch(config)# platform fm6000 keyframe kf1 int et1 255.255.255.255 ff:ff:ff:ff:ff:ff
switch(config)# int et1
switch(config-if-Et1)# mac timestamp before-fcs
```

If the switch is configured to replace the FCS with the timestamp (replace-fcs), the user should also set the solar_capture strip_fcs=0 option to prevent removal of the FCS.

Configure Arista timestamps: 7280

Use the switch_model option to identify the switch type:

```
solar_capture interface=eth2 arista_ts="switch_model=7280;strip_ticks=0" output=./
eth2.pcap
```

48bit timestamp format is assumed if the `ts_format` option is not used.

Keyframes are not generated or used when using the 7280 switch timestamps.

- *example - default behaviour:*

```
arista_ts="switch_model=7280;ts_format=48bit;ts_src_mac=1;strip_ticks=0"
```

- *example - overwrite source MAC:*

```
arista_ts="switch_model=7280;ts_format=48bit;ts_src_mac=1;strip_ticks=0;\
replacement_src_mac=aa:bb:cc:dd:ee:ff"
```


© Copyright 2017-2019 Xilinx Inc

When the Arista switch has replaced the source MAC field with a timestamp, the receiver can optionally use the `replacement_src_mac` option to specify a MAC address to re-insert into the packet header.

14.5 Using Arista 7150 timestamps

With the Arista 7150 switch configured as described in Configuration on page 80, running SolarCapture will produce output similar to the following to show that keyframes are being received from the switch:

```
[root@server]# solar_capture interface=eth4 output=./eth4.pcap \
"arista_ts=kf_ip_dest=255.255.255.255;kf_eth_dhost=ff:ff:ff:ff:ff:ff"
SolarCapture 1.6.0.35.  Copyright 2016 Solarflare Communications, Inc.
arista_ts: kf_eth_dhost=ff:ff:ff:ff:ff:ff
arista_ts: kf_ip_proto=253
arista_ts: kf_ip_dest=255.255.255.255
arista_ts: strip_ticks=1
arista_ts: tick_freq=0.000000
arista_ts: lost_sync_gap=10000
arista_ts: no_sync_gap=60000
arista_ts: max_freq_error=20000
arista_ts: KF: state=no_sync utc=1376928723.298764705
host=1376928723.510198000 ticks=15487b6950a drops=0
arista_ts: no_sync => sync1
arista_ts: KF: state=sync1 utc=1376928724.300648689
host=1376928724.510208000 ticks=1549c9d432c drops=0
arista_ts: KF: delta=1.001883983 tick_freq=350002779
arista_ts: sync1 => sync2
arista_ts: KF: state=sync2 utc=1376928725.302667379
host=1376928725.510214000 ticks=154b184a24e drops=0
arista_ts: KF: delta=1.002018690 tick_freq=350000947
```

Once in sync (state=sync2), the received Arista keyframes continue to update and are displayed on the console every second.

When the SolarCapture pcap file is viewed in Wireshark, the hardware timestamps are visible in the Time field as shown below.

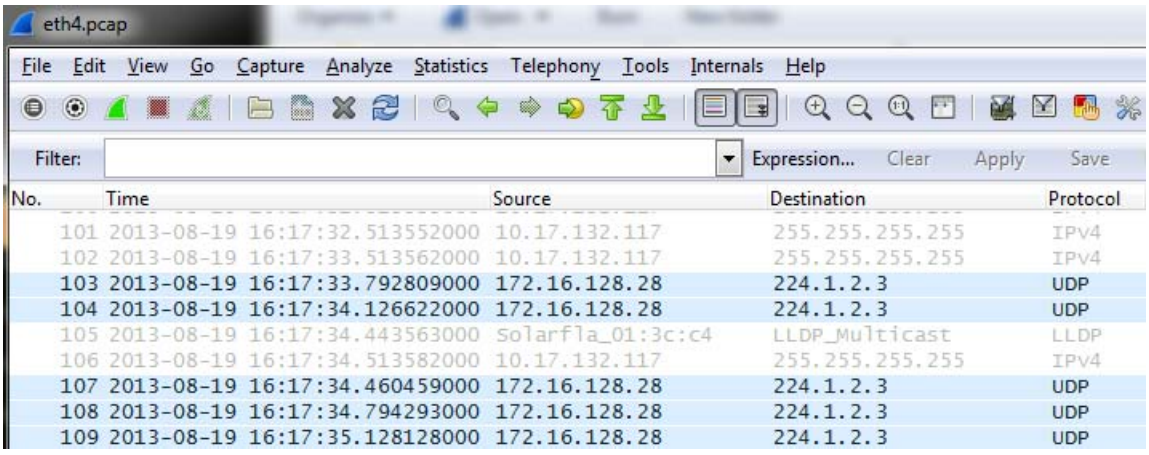


Figure 9: Hardware timestamps viewed in Wireshark

© Copyright 2017-2019 Xilinx Inc

Demonstration

The following two screen shots show the same packet received by SolarCapture on a single server from an Arista 7150 switch.

Figure 10 shows packet 3 received by SolarCapture which is not configured to convert the Arista hardware timestamps. Time displayed is the software timestamp packet arrival at the host.

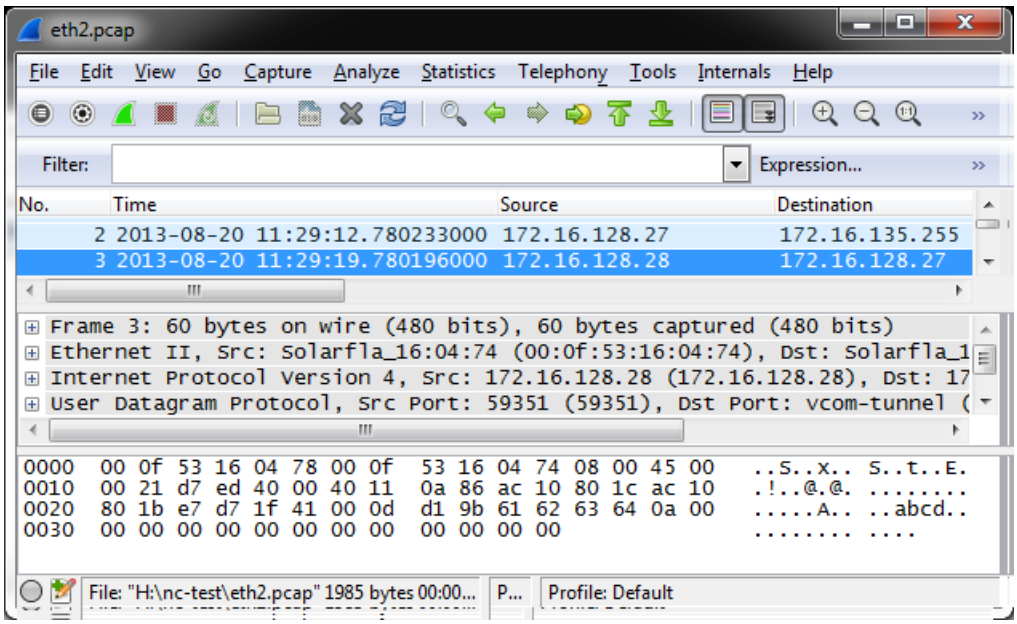


Figure 10: SolarCapture - Timestamp arrival at the host

© Copyright 2017-2019 Xilinx Inc

Figure 11 shows packet 17 (which is the same as packet 3 above) received by SolarCapture configured to receive Arista keyframes and to use the hardware timestamp packet arrival at the switch.

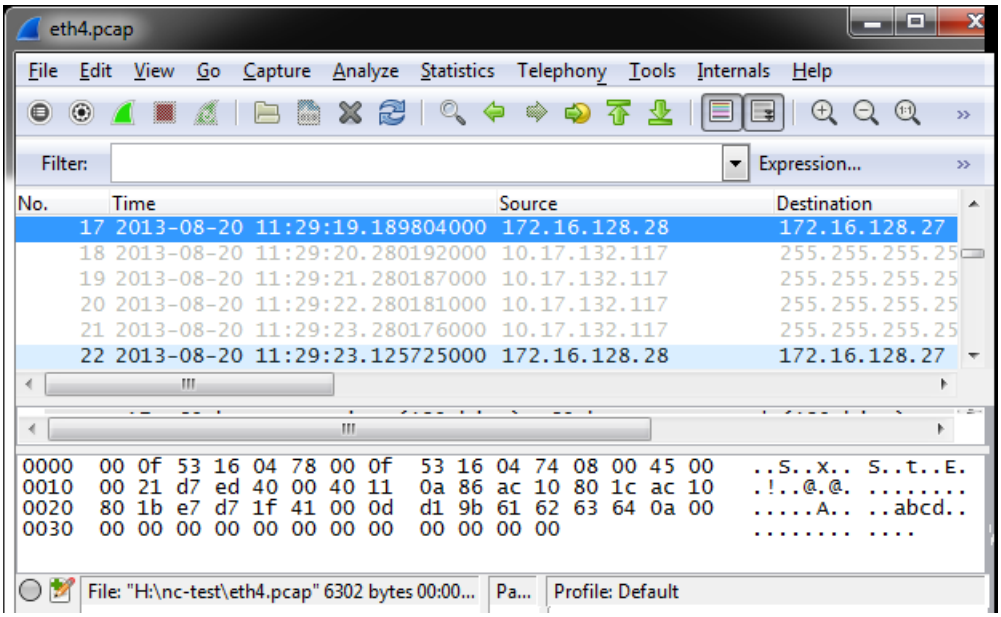


Figure 11: SolarCapture - Timestamp arrival at the switch

14.6 cpacket Timestamps

When inter-working with an Arista-Metamko switch, SolarCapture v1.6.6 includes the `sc_cpacket_ts` node used to replace the NIC arrival timestamp with a cpacket trailer timestamp.

For inline-help, use the following command:

```
# solar_capture help cpacket_ts
```

example:

```
cpacket_ts="is_metamako=1" strip_fcs=0 /
cpacket_ts="keep_cpacket_footer=1;is_metamako=1"
```


© Copyright 2017-2019 Xilinx Inc

The sc_cpacket_ts node includes the following options:

Name	Type	Description
`is_metamako`	int	Default=0 Set to 1 if the cpacket footer has Arista-Metamako TLV extensions.
`strip_fcs`	int	cpacket decode respects the strip_fcs option. Each cpacket footer includes the old FCS of the packet it is applied to - so the FCS can be preserved independently of the cpacket footer. Set to 1 to remove FCS from captured packets, set to 0 to capture packets with FCS intact.
`keep_cpacket_footer`	int	Default=0, do not include the cpacket footer from the capture file. Set to 1 to include the entire cpacket footer in the capture file.

© Copyright 2017-2019 Xilinx Inc

15 Embedding SolarCapture

The SolarCapture distribution includes C and Python bindings, allowing SolarCapture to be embedded in applications.

For more information please refer to the following:

- The *SolarCapture C Bindings User Guide* (SF-115721-CD).

  This guide covers all publicly available nodes, and all the exposed API. It is generated from the C header files.

  An HTML version is supplied with the SolarCapture SDK product, and is installed from the `solar_capture-python` package. A PDF version is available from https://support.solarflare.com.

- The example code.

  Each example is summarized in the *SolarCapture C Bindings User Guide*, and documented inline.

  The example code is supplied with the SolarCapture SDK product, and is installed from the `solar_capture-python` package.

© Copyright 2017-2019 Xilinx Inc

16 Extending SolarCapture

SolarCapture defines a coherent API allowing applications to be constructed from reusable components known as nodes. The core SolarCapture functionality can be extended by implementing new types of nodes in C.

For more information please refer to the following:

- The *SolarCapture C Bindings User Guide* (SF-115721-CD).

  This guide covers all publicly available nodes, and all the exposed API. It is generated from the C header files.

  An HTML version is supplied with the SolarCapture SDK product, and is installed from the `solar_capture-python` package. A PDF version is available from https://support.solarflare.com.

- The example of how to define a new node type.

  This is documented inline.

  It is supplied with the SolarCapture SDK product, and is installed from the `solar_capture-python` package.

© Copyright 2017-2019 Xilinx Inc

17 Known Issues and Limitations

17.1 Captured packets

Packets captured by SolarCapture are not available to the kernel stack, OpenOnload or any other process. Therefore if the SolarCapture version does not support 'sniff' mode, it is not possible to use SolarCapture to monitor streams that are consumed by applications on the same server. For unicast streams, Solarflare recommends using SolarCapture with mirror/span switch ports.

Unexpected behavior might be observed when Onloaded applications and SolarCapture run on the same server. This is due to the priorities of different filters created by the two applications which direct packets to one application but not both.

SolarCapture Pro supports *steal* and *sniff* modes.

17.2 Software Timestamp accuracy

Solarflare adapters will generate a hardware timestamp for each captured packet as it is received by the adapter. Timestamp resolution on the adapter is <8 nanoseconds and **hardware timestamps are not subject to jitter**.

17.3 Capture performance

The capture performance depends on many factors. In most deployments the sustained capture rate is limited by storage performance:

- The SolarCapture™ products described in this Guide *do not* guarantee any specific level of performance for storage to disk, since this depends on factors outside of Solarflare's control.
- Solarflare's **SolarCapture Appliance** and **SolarCapture System** products support guaranteed capture rates to disk, as well as many more advanced features. Please contact your sales representative for more information about these products or refer to http://www.solarflare.com/solarcapture

Other factors that affect capture performance include:

- Storage technology, raid and filesystem type.
- The I/O performance of the server.
- The size of the internal packet buffer pool.
- Spreading of load using receive-side scaling.

© Copyright 2017-2019 Xilinx Inc

See also Line Rate Packet Capture on page 23.

17.4 Stopping SolarCapture

The `solar_capture` command line tool can be stopped by sending it a `SIGINT` signal, for example by typing *Ctrl-C*.

The C bindings currently lack a way to free the resources used by a SolarCapture session, but packet processing can be stopped with the `sc_session_pause()` call.

17.5 Allocation of Packet Buffers

SolarCapture allocates pools of buffers into which captured data is saved before writing out to the capture file. By default, about 20,000 packet buffers are allocated to each interface.

An error similar to the following indicates insufficient packet buffers available for each capture interface.

```
ERROR: Unable to allocate sufficient buffers for VI (wanted=20480 min=8192 got=0)
```

To overcome this the user can:

- reduce the amount of packet buffering per interface using the command line `capture_buffer` parameter
- use huge pages (see Allocating Huge Pages on page 97)
- use packed stream firmware (this uses huge pages).

17.6 Limited Availability of Hardware Timestamping VIs

Solarflare adapters can support a maximum 256 VIs where hardware timestamping is enabled.

When the number of capture cores is set as a global value and a number of capture streams are created, the number of VIs created will be:

```
number of capture cores * number of capture instances
```

```
for example:
```

```
solar_capture capture_cores=1,2,3,4,5
        eth2=cap_1.pcap join_streams=udp:239.100.10.1:1001
        eth2=cap_2.pcap join_streams=udp:239.100.10.2:1001
        eth2=cap_3.pcap join_streams=udp:239.100.10.3:1001
        eth2=cap_4.pcap join_streams=udp:239.100.10.4:1001
        eth2=cap_5.pcap join_streams=udp:239.100.10.5:1001
        eth2=cap_6.pcap join_streams=udp:239.100.10.6:1001
```

Will create (5*6) = 30 VIs.

When the number of capture cores is set on a per stream basis, the number of VIs created will be equal to the number of instances, for example:

© Copyright 2017-2019 Xilinx Inc

```
solar_capture capture_cores=1,2,3,4,5
        eth2=cap_1.pcap join_streams=udp:239.100.10.1:1001 capture_cores=1
        eth2=cap_2.pcap join_streams=udp:239.100.10.2:1001 capture_cores=2
        eth2=cap_3.pcap join_streams=udp:239.100.10.3:1001 capture_cores=3
        eth2=cap_4.pcap join_streams=udp:239.100.10.4:1001 capture_cores=4
        eth2=cap_5.pcap join_streams=udp:239.100.10.5:1001 capture_cores=5
        eth2=cap_6.pcap join_streams=udp:239.100.10.6:1001 capture_cores=6
```

Will create 6 VIs.

17.7 Solarflare DAQ for Snort

If not using packed-stream mode, then the Solarflare Data Acquisition Module (DAQ) for use with Snort does not support packets larger than around 1778 bytes. Larger packets are truncated.

When using packed-stream mode all packet sizes are supported.

17.8 Filtering on VLAN

Solarflare adapters are not able to filter TCP and UDP streams by VLAN-ID when configured with low-latency or capture-packed-stream firmware variants.

The VLAN specification is ignored for capture, but is used for the purposes of joining multicast groups (join_streams).

Solarflare adapters configured to use the full-feature firmware variant are able to filter TCP and UDP streams by VLAN-ID.

17.9 PTP - Hybrid Mode

When capturing from an interface also being used to send/receive PTP messages, PTP hybrid mode will not function correctly as SolarCapture consumes the ARP response messages. This prevents the unicast Delay_Request messages being sent from a PTP slave. PTP in multicast mode is not affected and users are advised to select multicast mode in the sfptpd ptp_slave.cfg file:

```
ptp-network-mode multicast
```

Solarflare aim to address this issue in a future release of SolarCapture.

17.10 Onload and Line Rate Packet Capture

It is not possible at this time to accelerate applications with OpenOnload or EnterpriseOnload when the adapter is using the `capture-packed-stream` firmware version. This functionality will be available in a future release. Users who need Onload should first change the adapter firmware variant configuration.

© Copyright 2017-2019 Xilinx Inc

17.11 Sniff Mode in Packed Stream Firmware

Sniff mode is not supported when using the `capture-packed-stream` firmware variant and will result in the following type or errors:

```
# solar_capture mode=sniff eth2=/dev/null
SolarCapture 0.3.0.31.  Copyright (c) 2012-2014 Solarflare Communications, Inc.
SolarCapture session=19901/0 log=/var/tmp/solar_capture_root_19901/0
ERROR: errno=22 from core/sc_stream.c:615 in sc_stream_add():
ERROR: Bad stream: specified stream is not supported on this NIC
ERROR: Bad stream: specified stream is not supported on this NIC
```

or

```
sc_stream_add: ERROR: unable to add stream type 'sniff' (Operation not supported)
```

Configure the adapter to the full-feature firmware variant.

17.12 Sniff Mode Transmitted PTP Packets

Due to the specific nature of the transmit path for PTP packets, when hardware timestamping is enabled on the adapter, the following transmitted PTP packets will not be captured:

- Sync

Sync packets are generated by the PTP host configured in PTP master mode.

- Delay-Request

Delay-Request packets are generated by the PTP host configured in PTP slave mode.

17.13 Sniff Mode TX Packet Timestamp - Constant Offset

When a transmitted packet is hardware timestamped, a ‘sniffed’ transmitted packet will always have a timestamp which is early by a constant offset of 358 nanoseconds. This will result in larger than expected values when calculating on-the-wire latency.

An additional 358 nanoseconds should be added to each transmit packet timestamp when calculating on-the-wire latency.

© Copyright 2017-2019 Xilinx Inc

17.14 Packet Filters

VLAN only Filter

When using a vlan only filter:

- Multicast packets that match the filter are captured and also delivered to the kernel (if there are kernel subscribers).
- Unicast packets with destination MAC equal to the port's MAC address are not captured.

EtherType Filter

EtherType filters do NOT match IPv4 or IPv6 protocols.

IP Protocol Filter

IP protocol filters do NOT match UDP or TCP protocols.

© Copyright 2017-2019 Xilinx Inc

18 Tuning Guide

18.1 Introduction

This chapter details tuning options and recommendations to maximize performance of the `solar_capture` command line tool. Some of these tuning practices also apply to other SolarCapture applications.

Firmware Variants

Solarflare adapters support configurable firmware variants. Refer to Firmware Variants on page 12 for details.

Packet Buffers

Incoming packets are captured by a capture thread before being handed over to a write-out thread and written to disk. Packets are captured into *packet buffers*.

Using the capture-packed-stream firmware variant, the size of a packet buffer is a 64KB.

Using other firmware variants, the size of a packet buffer is a 2KB and each buffer can store up to 1792 bytes + 256 bytes of internal meta data. Larger packets are split over several buffers.

Before writing to disk, packet buffers are copied into a *pcap buffers* in the writer thread. A pcap buffer is, by default, 32k in size – but can be set to different sizes. Individual packets are bundled together in a pcap buffer to increase disk write efficiency. When a pcap buffer is filled, the contents are written to disk.

PCAP Buffers

PCAP buffers are the “blocks” of data written to the storage medium. Buffers can be configured using SolarCapture attributes using the `SC_ATTR` environment variable - refer to SolarCapture Attributes on page 106 for details.

The attribute `buf_size_pcap` specifies the size (in bytes) of the pcap buffers available to a SolarCapture instance. **CRITICAL: This must be a multiple of 4KiB**.

The attributes `n_bufs_pcap` specifies the number of pcap buffers available to a SolarCapture instance.

© Copyright 2017-2019 Xilinx Inc

By default, `buf_size_pcap` is set to 32k. Selecting an optimum value for this will depend on both the disk and the RAID setup (i.e tuning will be required). On some systems, the optimal value can be determined by inspection of the value in the following file:

```
/sys/block/<disk>/queue/optimal_io_size
```

Another option is to set `buf_size_pcap` to the stripe size of the RAID.

18.2 File System Tuning

Solarflare recommend the ext4 file system and it has been observed that other file systems can incur performance penalties. For example, the async writer in combination with xfs and some types of RAID controllers causes xfs to block on I/O whereas ext4 delivers better performance with less blocking on I/O. Additionally, if the RAID system consists of SSDs the file system should be trimmed periodically to maintain optimal performance. Trimming wipes blocks on an SSD which are no longer in use.

Mount Options

File I/O performance is influenced by the underlying disk partition alignment and file-system mount options. Users are advised to consult the system RAID documentation and the OS mount options for recommended settings for optimal write performance.

The following example identifies some `fstab` tuning parameters that can be adjusted to improve I/O performance on a standard Linux system.

```
rw,data=writeback,nobarrier,commit=3000,journal_ioprio=6
```

rw	read/write access.
data=writeback	set the journaling mode for file data to writeback, for maximum throughput.
nobarrier	disable the use of write barriers. A write barrier is a mechanism for enforcing a particular ordering in a sequence of writes to the filesystem.
commit=3000	sync all data and metadata every 3000 seconds. Writeout performance is improved when syncs are delayed.
journal_ioprio=6	Set the I/O priority for kjournald2 during a commit operation. This option can take values from 0 (highest priority) to 7 (lowest priority).

© Copyright 2017-2019 Xilinx Inc

Scheduling Options

These options are for block device scheduling and only apply to a device dedicated to packet capture.

Operating systems have different scheduling options, for example, on RedHat Enterprise 6.4 (64 bit), the default policy is CFQ (Completely Fair Queuing). It has been observed that CFQ is suboptimal for the SolarCapture use case and the recommendation is to use the deadline scheduler instead:

```
echo “deadline” > /sys/block/<device>/queue/scheduler
```

A further recommendation is to set the write expire option to a large value:

```
echo “50000” > /sys/block/sdb/queue/iosched/write_expire
```

If concurrent reading from capture files is required, the user should also experiment by increasing the value for the `read_expire` option.

Synchronous vs. Asynchronous writer

Standard `write`/`writev` calls are synchronous such that data to be written is copied into the page cache and buffers can be reused.

Asynchronous writes greatly improve performance in most cases, but require that sufficient file space be pre-allocated before submitting writes.

The asynchronous writer only works on file systems which support `fallocate`. This can be verified if the following command succeeds or fails - and if it fails the writer will fall back to a synchronous mode:

```
fallocate -l <length> <destination file on relevant partition>
```

The `solar_capture_monitor` utility will also identify when the asynchronous writer is being used. In the output from solar_capture_monitor, if `async_mode` in the node with `node_type_name` set to `sc_disk_writer` is set to 1, then the writer is using the asynchronous method. Otherwise it will use the synchronous writer.

Using the asynchronous writer, storage space for the pcap file is pre-allocated in 64MB chunks. Therefore, during normal operation of SolarCapture, the pcap file size is not an accurate indicator of the number or size of packets captured. On shutdown, SolarCapture will close the file and set the end point of the file correctly.

With `async` writes, performance is sensitive to the I/O scheduling policy used. The default policy on RedHat Linux is CFQ (Completely Fair Queuing). However, it has been found that performance can be improved by changing the scheduling policy to deadline. Refer to Scheduling Options above for details.

With `sync` writes tuning the pcap buffer size is less critical than when using the asynchronous writer. The sync writer may require a larger buffer pool as packet buffers are consumed while the writer is blocked during a sync write.

If the sync writer is to be used, then a key performance bottleneck will be the virtual memory manager. Refer to Virtual Memory Tuning for more details.

© Copyright 2017-2019 Xilinx Inc

NOTE: When the synchronous writer is being used, increasing the number of pcap buffers may not be sufficient to prevent a slow write from causing drops.

This occurs because the writer thread is responsible for filling the pcap buffers and for writing the data to disk and will block the filling of pcap buffers until a write has completed. Under these conditions the number of packet buffers available to SolarCapture should be increased.

18.3 RAID Controller Tuning

- Interrupts

  Interrupts serving the RAID controller should be pinned away from cores being used for the capture threads.

- Partitions and File System

  Disk partitions should be aligned to the stripe size of the RAID array. On Red Hat Enterprise Linux, the alignment parameters for a RAID system can be determined by inspection of the following file:

  ```
  cat /sys/block/<disk>/queue/optimal_io_size
  ```

- File System

  When configuring the server file system, ensure that the file system is correctly configured for the specified stripe size. The RAID vendor documentation should also be consulted for system-specific recommendations.

18.4 Virtual Memory Tuning

When the synchronous writer is being used, performance gains can be achieved by tuning the virtual memory manager. This section can be ignored when not using the synchronous writer.

Kernel tunable parameters in `/proc/sys/vm` can be adjusted to optimize the virtual memory manager for high performance disk write workloads:

- Set `dirty_background_ratios` (`dirty_background_bytes`) to a small value, no more than 5.
- Set the `dirty_ratio` (`dirty_bytes`) values between 80 and 90. It has been observed that values larger than 90 do not necessarily offer improved performance.
- Set `dirty_writeback_centisecs` value to a large number. This setting is advised only on machines dedicated to packet capture and setting this parameter increases the risk of data loss in the case of an unclean shutdown. User who wish to experiment with this value should set it to a value significantly larger than the default value of 500.

© Copyright 2017-2019 Xilinx Inc

18.5 Capture Thread Tuning

A capture thread operates in different modes and SolarCapture will select the best performing option available:

- Packed Stream: This mode provides best capture performance only available when the adapter is using the `capture-packed-stream` firmware variant.
- Normal: This mode is used when packed stream mode is not applied.

Packed Stream Mode

Using the `capture-packed-stream` firmware variant, buffers used by the adapter to push packets to the host are 1MB in size and can contain multiple packets.

By default, the number of buffers is 256. Increasing this can provide greater resilience to bursty traffic. Having more than 2000 buffers provides limited additional benefit, and can impact performance.

The 1MB buffers need to be kept in contiguous memory, so it is important to pre-allocate sufficient huge pages `((1/2)*(number of buffers) + 1)` for these buffers before startup. If insufficient huge pages have been allocated, SolarCapture will fail on startup.

Normal Mode:

The number of available packet buffers depends on a number of factors:

- The amount of physical memory in the system.
- The amount of free memory in the system, and whether the system is heavily loaded or not.
- The kernel version being used (2.6.32 and later, or older versions).

The theoretical maximum for the number of buffers available is between 120 thousand (requiring 240MB of system memory) and 30 million (requiring 60GB of system memory), depending on the level of fragmentation of system memory.

The reason for this variability is as follows: The adapter has a hard limit for the amount of memory locations it can index (around 60 thousand).

On modern kernels (newer than 2.6.32), the adapter tries to allocate memory in 2MB contiguous chunks (i.e. groups of 512 4k pages) using transparent huge pages. If all memory is allocated in this way, then the maximum of 30 million packets can be allocated. If, however, if the system memory is badly fragmented (due to heavy use), then only the minimum of 120 thousand packets will be attained. On older kernels, transparent huge pages are not available and explicit huge pages should be enabled to allow the adapter to allocate memory in 2MB contiguous chunks.

Memory fragmentation can be avoided by using huge pages. SolarCapture supports two attributes which can be set to use huge pages:

© Copyright 2017-2019 Xilinx Inc

- The attribute `request_huge_pages`, if set, will cause the packet pool to use explicitly allocated huge pages, if available. Note that, even if this attribute is not set, transparent huge pages may be used, if they are supported on the system.
- The attribute `require_huge_pages`, if set, will cause the packet pool to only use explicitly allocated huge pages. If enough huge pages are not available, then the buffer allocation mechanism will fail.

If the system supports transparent huge pages, the recommendations are:

1. Packet allocation should be increased to 1GB of packet buffers, using the `capture_buffer` option in SolarCapture - see example in below.
2. Experiment with adjusting the size and number of buffers available for the pcap buffer pool (pcap buffers) using the SC_ATTR settings `buf_size_pcap` and `n_bufs_pcap` – see the example in below.

18.6 Allocating Huge Pages

The current hugepage allocation can be checked by inspection of `/proc/meminfo`

```
cat /proc/meminfo | grep Huge
```

This should return something similar to

```
AnonHugePages:      2048 kB
HugePages_Total:    2050
HugePages_Free:     2050
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
```

The total number of hugepages available on the system is the value `HugePages_Total`

The following command can be used to dynamically set and/or change the number of huge pages allocated on a system to (<N> is a non-negative integer, and <size> is the size of huge page to which the setting applies):

```
echo <N> > /sys/kernel/mm/hugepages/hugepages-<size>/nr_hugepages
```

or, using a less recent interface:

```
echo <N> > /proc/sys/vm/nr_hugepages
```

On a NUMA platform, the kernel will attempt to distribute the huge page pool over the set of all allowed nodes specified by the NUMA memory policy of the task that modifies `nr_hugepages`. The following command can be used to check the per node distribution of huge pages in a NUMA system:

```
cat /sys/devices/system/node/node*/meminfo | grep Huge
```

Huge pages can also be allocated on a per-NUMA node basis (rather than have the hugepages allocated across multiple NUMA nodes). The following command can be used to allocate <N> hugepages on NUMA node <M>:

© Copyright 2017-2019 Xilinx Inc

```
echo <N> > /sys/devices/system/node/node<M>/hugepages/hugepages-2048kB/ \ nr_hugepages
```

Users should also refer to file backed memory pools text in current release notes.

18.7 NUMA Binding

NUMA binding should follow from core allocations. Ensure that the cores being used are those which are local to the adapter. The adapter NUMA node can be determined by inspection of the value in the following file:

```
/sys/class/net/eth<N>/device/numa_node
```

The NUMA ID for each core can be determined by inspection of the output from the command:

```
numactl –hardware
```

The following example shows the output from a system with 2 NUMA nodes:

```
# numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 2 4 6 8 10 12 14
node 0 size: 3059 MB
node 0 free: 1492 MB
node 1 cpus: 1 3 5 7 9 11 13 15
node 1 size: 3071 MB
node 1 free: 1798 MB
node distances:
node   0   1
  0:  10  20
  1:  20  10
```

In this system, there are 2 NUMA nodes, each with 8 cores. The system has a total of 6GB of memory, split evenly between the two nodes (3GB each). The node distances indicates the relative distance between the nodes.

18.8 C-States

When SolarCapture is spinning (which is the default mode), these tuning options will have limited impact, but are most relevant if SolarCapture is running in interrupt driven mode (when the attribute `capture_busy_wait` is set). See Polling vs Interrupt Mode on page 25 for further details.

There are a number of tuning parameters which can be added to the kernel command line in `/boot/grub/grub.conf` file:

- Disable the Intel driver. Setting this to 0 disables `intel_idle` driver and falls back to the `acpi_idle` driver:

  ```
  intel_idle.max_cstate=0
  ```

- Set processor `max_state`. Note that even if this is set, the kernel actually silently sets it to 1 (see file processor_idle.c in the kernel source code):

  ```
  processor.max_cstate=0
  ```


© Copyright 2017-2019 Xilinx Inc

- Set the `idle` parameter to poll. Setting this to poll forces a polling idle loop, which can improve performance, but at the expense of power:

  ```
  idle=poll
  ```

Thus, a full, aggressive squeeze out as much latency as possible approach would be to set all of the following:

```
intel_idle.max_cstate=0 processor.max_cstate=0 idle=poll
```

18.9 Isolate CPU Cores

There are a number of ways to isolate CPU cores on a system to ensure that only specified tasks run on them. One of these is the grub configuration option `isolcpus` (another method is to use `csets`).

The command `isolcpus` will cause the specified CPUs (defined by the cpu_number value) to be removed from kernel SMP balancing and scheduler algorithms. Once isolated, the only way to move a userland process on or off an isolated CPU is to manually specify the process affinity, for example via taskset.

The `isolcpus` can be used by adding the following command to the `/boot/grub/grub.conf` kernel command line options:

```
isolcpus=<CPU ID 1>,<CPU ID 2>,...,<CPU ID k>
```

Note that **CPU numbers start at 0**. For example, the directive:

```
isolcpus=1,2
```

will isolate the 2nd and 3rd CPUs on a system.

It is recommended not to use the first CPU core on each CPU socket. The kernel typically uses these cores for routine housekeeping tasks (see Limitations below).

18.10 Memory Usage

The amount of memory used by SolarCapture is dependent on:

- The size of packet buffers (set via the command line option `capture_buffer`) being used.
- The number of writer threads being called.
- The size (`buf_size_pcap`) and number (`n_bufs_pcap`) of pcap buffers being used.

The calculation is:

```
buf_size_pcap * n_bufs_pcap + capture_buffer + <overhead ~16MB>
```

In the following example the total memory required is 1GB.

```
SC_ATTR = "buf_size_pcap=4194304 ; n_bufs_pcap=16" \
solar_capture eth4=/dev/null capture_buffer=1024000000 \
capture_cores=3 writeout_core=5 rx_ring_max=4095
```


© Copyright 2017-2019 Xilinx Inc

18.11 Packet Pool Limitations

SolarCapture Pro supports at most 64 packet pools. If this limit is exceeded, the mechanism by which these pools are refilled can fail, resulting in some pools running dry. To avoid this issue, the user should create no more than 60 writeout files per instance of SolarCapture.

SolarCapture has two types of packet:

- packet buffers, where incoming packets are stored;
- pcap buffers, where packets are copied to prior to being written to disk by the disk writer.

When a VI or a disk writer node is created, it is assigned a packet pool, which provides a store of buffers (packet or pcap) for use by the node.

- When not using `capture-packed-stream` firmware, each capture core has a pool.
- Using `capture-packed-stream`, each capture core has 2 pools.
- Each file being written out has a pool.
- Additionally, SolarCapture has a private pool, which is used for internal housekeeping tasks.

Example:

```
solar_capture capture_cores=1 writeout_core=3 \
eth2=/tmp/file1.pcap stream="udp:239.100.10.1" \
eth2=/tmp/file2.pcap stream="udp:239.100.10.2" \
eth2=/tmp/file3.pcap stream="udp:239.100.10.3"
```

This will require a total of:

- 5 (1 capture, 3 writers, 1 private) pools in (`not capture-packed-stream`) mode.
- 6 (2 capture, 3 writers, 1 private) pools in (`capture-packed-stream`) mode.

The number of packet pools can be identified from the output from `solar_capture_monitor dump`:

```
solar_capture_monitor dump | grep  sc_pool | wc -l
```

18.12 RXQ size on Packed Stream Firmware

The `capture-packed-stream` firmware variant imposes a limitation of 2048 descriptors in the receive queue. Setting this to a larger value will cause SolarCapture to fail to start. By default, this value is set to 512 which maintains a smaller working set.

© Copyright 2017-2019 Xilinx Inc

18.13 Kernel Services

Terminate the following OS services:

```
service cpuspeed stop
service iptables stop
```

18.14 Interrupt Moderation

- Disable interrupt moderation:

  ```
  ethtool -C eth<N>  rx-usecs 0 rx-frames 0 adaptive-rx off
  ```

- When using interrupt-driven mode, it may be beneficial to experiment with the interrupt moderation interval to establish the optimum setting.

  Increasing the interrupt moderation interval:

  - increase latency
  - reduce CPU utilization
  - improve peak throughput

  Decreasing the interrupt moderation interval:

  - decrease latency
  - increase CPU utilization
  - reduce throughput

18.15 SolarCapture Configuration

Set the following options on the SolarCapture command line:

```
capture_buffer=225280000
rx_ring_low=40
rx_ring_high=60
rx_refill_batch_low=64
rx_ring_max=4095
```

If physical addressing mode is being used, a greater amount of packet buffers may be allocated.

NOTE: The above settings are not compatible with capture-packed-stream mode - customers using this firmware variant may contact support@solarflare.com for advice.

© Copyright 2017-2019 Xilinx Inc

18.16 RSS

When multiple capture-cores are identified for a capture instance, Receive Side Scaling (RSS) is used to spread the received traffic load over these cores.

The sfc driver module option can be enabled to restrict RSS to only use cores on the NUMA node local to the Solarflare adapter. Driver module options can be set in a file e.g. sfc.conf, in the `/etc/modprobe.d` directory

```
options sfc rss_numa_local=1
```

This option is relevant only when using SolarCapture in the interrupt-driven mode (`capture_busy_wait=0`). When SolarCapture is being used in the default 'polling' mode (`capture_busy_wait=1`) few interrupts, if any, will be generated therefore it is not necessary to set this option.

18.17 Maximum multicast group membership

The number of multicast groups (per interface) that can be joined is governed by the kernel `igmp_max_memberships` parameter. Its default value is 20 in Linux, and the server will revert to the default following reboot.

Attempts to join a multicast group when this maximum has been reached result in an error such as the following:

```
ERROR: errno=105 from core/sc_misc.c:103 in sc_join_mcast_group():
ERROR: sc_join_mcast_group: Failed to join multicast group '233.200.79.146' on eth0
       (hit kernel igmp_max_memberships limit)
```

(Error number 105 is `ENOBUFS`.)

To check the current value of the kernel `igmp_max_memberships` parameter, use the following command:

```
# cat /proc/sys/net/ipv4/igmp_max_memberships
```

The value can be increased to ensure it is large enough for all the multicast groups you wish to join, for example to 50:

- To change the value non persistently (and revert to the default on reboot):

  ```
  # echo 50 > /proc/sys/net/ipv4/igmp_max_memberships
  ```

- To make the change persistent across reboots, add the following lines to the `/etc/sysctl` file:

  ```
  # Controls the number of multicast groups that can be joined per interface:
  net.ipv4.igmp_max_memberships = 50
  ```


© Copyright 2017-2019 Xilinx Inc

18.18 Statistics files

SolarCapture stores per-thread statistics information on disk. For increased resilience of these files in production, it might be worth putting them in `tmpfs` rather than be disk backed. This can be done using one of the following:

- Mount `tmpfs` at the default path, beneath `/var/tmp/solar_capture_<user>_<pid>`
- Set the `log_base_dir` attribute to override the location.

  This will require `solar_capture_monitor` to be invoked with the `base-dir` argument.

© Copyright 2017-2019 Xilinx Inc

A Configuration File Structure

This appendix identifies the required INI file format of configuration files used by SolarCapture applications.

A configuration file can have any name and any extension e.g <name>.ini, <name>.cfg, <name>.txt.

```
$ solar_capture --config /<path to config file>/filename.ini
```

A.1 File Structure Conventions

Configuration files follow standard INI file formats and conventions.

A.2 Properties

The basic construct in an INI file is a property. Each property is a name and value pair delimited by an equals character(=), e.g.

```
name=value
```

Property names are case-insensitive and any leading or trailing white space around the property name and property value will be ignored making all the following equivalent:

```
name=value Name=value name = value naMe =value name= value
```

Property declarations with duplicate names, or names that only differ in case, will override earlier occurrences.

A.3 Multi-Value Properties

White space within multi-value properties will be treated as a delimiter. Quoting with either double quotes (") or single quotes (') should be used for values where white space is significant:

```
name = value1 value2 value3
name = "value1" "value2" "value3"
name = 'value1 ' 'value2 ' 'value3 '
name = 'multi-word value' '  leading white space' 'trailing white space  '
name = 'a value containing "quotes"'
```

A backslash (\) immediately followed by an end-of-line causes the end-of-line to be ignored allowing multiple lines to be concatenated together - useful for properties with many values.

© Copyright 2017-2019 Xilinx Inc

A.4 Sections

INI files support properties grouped into sections. The section name should be enclosed in square brackets [] and appear on a line by itself. All properties defined after the section declaration are associated with that section.

A section ends when another section is started or the end-of-file is reached. Section names are case-insensitive and may contain white space:

```
[section]
name=value

[another section]
name=value
name2=value
```

A.5 Comments

Comments start with semi-colon (;) or hash (#) symbols. The comment symbol can occur at any point on a line and any characters following a comment symbol up to the end of the line are considered a comment.

Full line comments should appear on a line by themselves. A comment which includes a backslash character (\) immediately before the end of the line causes the end-of-line to be ignored allowing multiple lines to be concatenated.

A.6 Blank Lines

The INI file can contain blank lines which are ignored.

A.7 Character set and encoding

The INI file should only use 7-bit ASCII characters. Spaces and horizontal tabs (HT) are considered white space. Lines are terminated by either a carriage return (CR) or linefeed (LF) character. When CR and LF appear together they are treated as a single line terminator.

© Copyright 2017-2019 Xilinx Inc

B SolarCapture Attributes

SolarCapture includes the `solar_capture_doc` command providing detailed information for all attributes which can be set on the command line or exported to the solar_capture environment.

B.1 Getting Help

```
$ solar_capture_doc --help
```

B.2 Doc Command Line

Syntax

```
solar_capture_doc [options] <command>
```

Commands

Command	Description
`list_attr`	List names of all attributes
list_attr <obj>	List names of attributes that apply to one of the following types of object: • session • thread • vi • node • mailbox • pool

© Copyright 2017-2019 Xilinx Inc

Command	Description
attr	Document all attributes
attr <name>	Document named attribute
attr <obj>	Document attributes that apply to one of the following types of object: • session • thread • vi • node • mailbox • pool

Options

Option	Description
--help	Show help message and exit
--all	Do not hide deprecated and unstable features

B.3 To set attributes

In C:

```
rc = sc_attr_set_int(attr, "n_bufs_pcap", 4);
```

In python:

```
thread.new_node("sc_writer", args=args, attr=dict(n_bufs_pcap=4))
```

As environment variable:

```
export SC_ATTR=n_bufs_pcap=4
```

```
# export SC_ATTR="log_level=6;force_sync_writer=1"
# solar_capture interface=eth1 output=eth1.pcap
```

When setting attributes in the environment, multiple attributes should be separated by semi-colons. A semi-colon is not required after the final attribute, or when setting a single attribute.

© Copyright 2017-2019 Xilinx Inc

C Change History

This Chapter provides a brief history of changes, additions and removals to SolarCapture releases.

The tables below list only major features. For a complete summary of changes, users should refer to the SolarCapture distribution release notes.

C.1 Features

Feature	Version	Description/Notes
X2 series support	1.6.6	X2522 at 10G and 25G X2541 at 100G, 2x50G, 1x50G, 1X40G and 4x25G
Arista 48bit timestamp format	1.6.6	Arista 7280 switch 48bit timestamp decode
Arista-Metamako TLVs in cpacket footer	1.6.6	Arista timestamp from cpacket packet trailer
sc_session_destroy() sc_session_run() sc_session_stop()	1.6.6	SolarCapure extensions - Refer to SF-115721-CD for details.
RX packet batching	1.6.6	Enable/disable batching with the rx_batch_nanos attribute.
Bug fix release	1.6.5	Bug fix release - no new features.
Arista 7280 switch support	1.6.4	The arista_ts field ‘switch_model’ identifies the Arista timestamping protocol to decode. Currently supports Arista 7150 and 7280 models.
SFN8000 series support	1.6.3	Support SFN8000 series adapters.
Signed RPMs	1.6.3	Digitally signed RPM packages.
Expanded C bindings documentation	1.6.0	Improved documentation covers all C bindings, public APIs and built-in nodes.
Shared memory channel	1.6.0	Allows transfer of packets between SolarCapture processes.
TCP tunnels	1.6.0	Allows transfer of packets between SolarCapture processes via a TCP socket.

© Copyright 2017-2019 Xilinx Inc

Feature	Version	Description/Notes
Load balancing	1.6.0	solar_balancer utility provides flow-aware load balancing to distribute flows to consumers over shared memory channels.
Decoding of cPacket trailer	1.6.0	Packets with cPacket trailer are decoded with the cpacket_ts option or sc_cpacket_ts node.
Post rotate command	1.6.0	The postrotate_command allows a command to be executed after file rotation.
Improved filters	1.6.0	New filters on Ethertype and IP protocol.

© Copyright 2017-2019 Xilinx Inc

D Third Party Software

SolarCapture includes binary code from libpcap; the following declaration applies:

Copyright © 1992, 1993, 1994, 1995, 1996 The Regents of the University of California. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.