Troubleshoot Nexus Cheat Sheet for Beginners

Contents

Introduction

This document describes different tools available to troubleshoot Nexus products that you can utilize in order to diagnose and fix a problem.

Overview

It is important to understand what tools are available and in what scenario you would use them for maximum gain. In fact, sometimes a certain tool is not feasible simply because it is designed to work on something else.

This table compiles the various tools to troubleshoot on the Nexus platform and their capabilities. For details and CLI examples, refer to the section Nexus Tools.

TOOLS	FUNCTION	EXAMPLE USE CASES	PROS	CONS
Ethanalyzer	Capture traffic destined to or from the CPU.	Traffic slowness issues, latency and congestion	Excellent for slowness, congestion, and latency issues.	Typically only sees control plane traffic, rate limited.

SPAN	Capture and Mirror a bunch of packets.	Failed ping s, out-of-order packets, and so on.	Excellent for intermittent traffic loss.	Requires external device that runs sniffer software Requires TCAM resources.
DMirror	Capture traffic destined to or from the CPU for Broadcom Nexus devices only.	Traffic slowness issues, latency and congestion.	Excellent for slowness, congestion, and latency issues.	Only for Broadcom Nexus devices. Rate limited (CloudScale Nexu 9k does have SPAN-to-CPU)
ELAM	Captures a single packet that ingresses [or egresses, if Nexus 7K] the Nexus switch	Verify packet reaches the Nexus, check forwarding decisions, check packet for alterations, verify interface/VLAN of packet, and so on.	Excellent for packet flow and forwarding issues. Non-intrusive	Requires in-depth understanding of hardware. Utilizes unique trigger mechanisms that are Architecture specific. Useful only if you know what traffic you want to inspect.
Nexus 9k Packet Tracer	Detect path of the packet.	Connectivity issues and packet loss.	Provides counter for flow statistics useful for intermittent/complete loss. Perfect for line cards with no TCAM carvings.	Cannot capture ARP traffic. Only works for Nexus 9k.
Traceroute	Detect path of the packet with respect to L3 hops.	Failed pings, cannot reach host/destination/internet, and so on.	Detects the various hops in the path to isolate L3 failures.	Only identifies where the L3 boundary is broken (does not identify the issu itself).
Ping	Test connectivity between two points in a network.	Test reachability between devices.	A quick and simple tool to test connectivity.	Just identifies whether the host is reachable or not.
PACL/RACL/VACL	Capture	Intermittent packet loss	Excellent for	Requires TCAM

	traffic ingress/egress a certain port or VLAN	between hosts, confirm if packets arrive/leave at the Nexus, and so on.	intermittent traffic loss.	resources. For some modules manual TCAM carving is required.
LogFlash	Stores historical data for the switch globally such as logs accounts, crash files, and events regardless of device reload.	Sudden device reload/shutdown, anytime a device is reloaded, log flash data provides some information that can be helpful in analysis.	Information is retained on device reload (persistent storage).	External on Nexus 7K = Must be installed/integrated on the supervisor platform for these logs to be collected (con does not apply to 3K/9K since logflash is a partition of the internal storage device).
OBFL	Stores historical data a specific module such as failure and environmental information.	Sudden device reload/shutdown, anytime a device is reloaded, log flash data provides some information that can be helpful.	Information is retained on device reload (persistent storage).	Supports limited number of reads and writes.
Event-Histories	When you require information for a specific process that is currently running.	Every process in nexus have its own event histories such as CDP, STP, OSPF, EIGRP, BGP, vPC, LACP, and so on.	Troubleshoot a specific process running on Nexus.	Information is lost once the device is reloaded (non-persistent).
Debugs	When you require more granular real time/live information for a specific	Debug on every process in nexus can be done such as CDP, STP, OSPF, IGRP, BGP, vPC, LACP, and so on.	Troubleshoot a specific process running on Nexus in real time for more granularity.	Can impact network performance.

	process.			
GOLD	Provides Bootup, run time, and on-demand diagnostics on Hardware components (such as I/O and Supervisor Modules).	Test hardware such as USB, Bootflash, OBFL, ASIC memory, PCIE, Port loopback, NVRAM, and so on.	Can detect faults in the hardware and take necessary corrective actions only on release 6(2)8 and higher.	Only detects hardware issues.
EEM	Monitor events on the device and take necessary actions.	Any device activity which requires some action/workaround/notification such as interface shutdown, fan malfunction, CPU utilization, and so on.	Supports Python scripts.	Must have network admin privileges to configure EEM.

Nexus Tools

If you need more clarification on various commands and their syntax or options, refer to Cisco Nexus 9000 Series Switches - Command References - Cisco.

Ethanalyzer

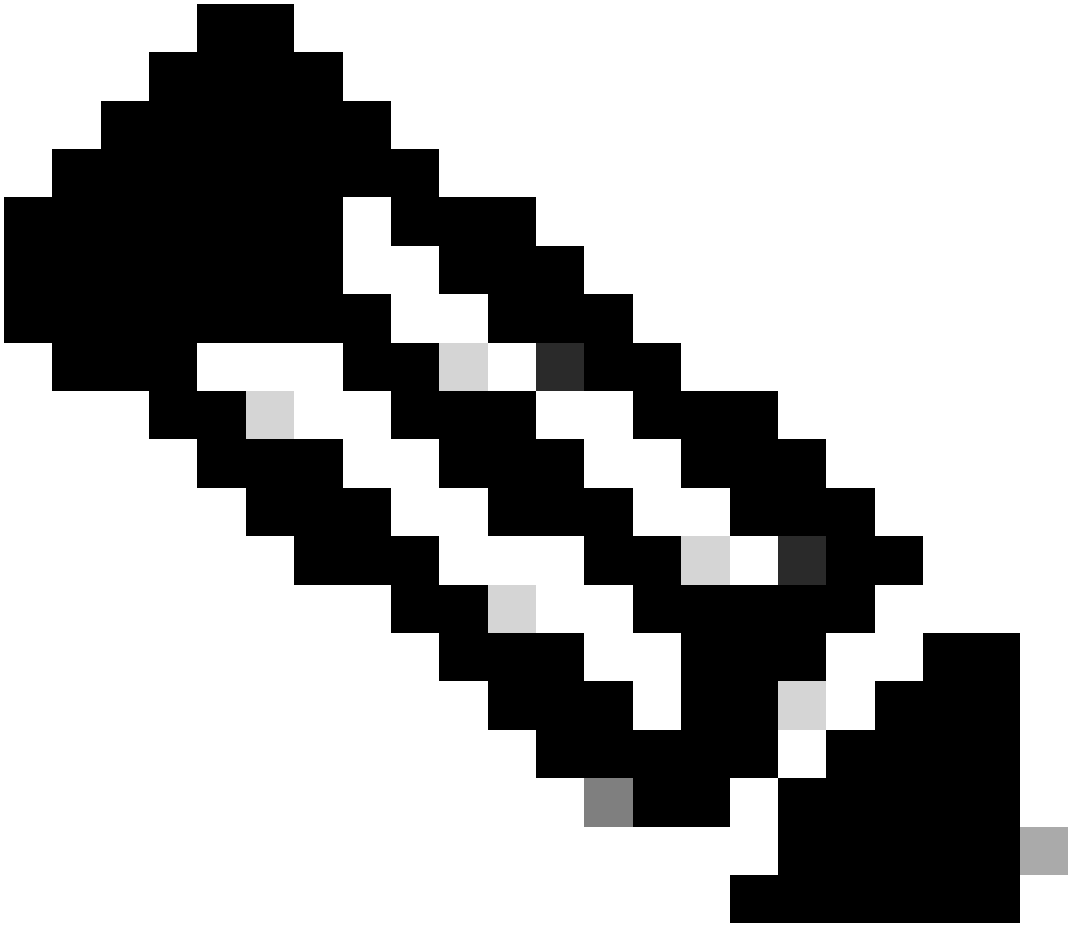
Ethanalyzer is a NX-OS tool that is designed to capture packets CPU traffic. Anything that hits the CPU whether ingress or egress can be captured with this tool. It is based on the widely-used open-source network protocol analyzer Wireshark. For more details on this tool, please refer to the Ethanalyzer on Nexus 7000 Troubleshooting Guide - Cisco

It is important to note that generally, Ethanalyzer captures all traffic to and from the supervisor. That is, it does not support interface specific captures. Specific interface enhancements are available for select platforms in more recent code points. Also, Ethanalyzer only captures traffic that is CPU switched, not hardware switched. For example, you can capture traffic on either the inband interface, the management interface or a front panel port (where supported):

```
Nexus9000_A(config-if-range)# ethanalyzer local interface inband
Capturing on inband
2020-02-18 01:40:55.183177 cc:98:91:fc:55:8b -> 01:80:c2:00:00:00 STP RST. Root = 32768/1/cc:98:91:fc:5
2020-02-18 01:40:55.184031 f8:b7:e2:49:2d:f2 -> 01:80:c2:00:00:0e LLC U, func=UI; SNAP, OUI 0x00000C (C
2020-02-18 01:40:55.184096 f8:b7:e2:49:2d:f5 -> 01:80:c2:00:00:0e LLC U, func=UI; SNAP, OUI 0x00000C (C
2020-02-18 01:40:55.184147 f8:b7:e2:49:2d:f4 -> 01:80:c2:00:00:0e LLC U, func=UI; SNAP, OUI 0x00000C (C
2020-02-18 01:40:55.184190 f8:b7:e2:49:2d:f3 -> 01:80:c2:00:00:0e LLC U, func=UI; SNAP, OUI 0x00000C (C
2020-02-18 01:40:55.493543 dc:f7:19:1b:f9:85 -> 01:80:c2:00:00:00 STP RST. Root = 32768/1/dc:f7:19:1b:f
2020-02-18 01:40:56.365722      0.0.0.0 -> 255.255.255.255 DHCP DHCP Discover - Transaction ID 0xc82a6d
2020-02-18 01:40:56.469094 f8:b7:e2:49:2d:b4 -> 01:80:c2:00:00:0e LLC U, func=UI; SNAP, OUI 0x00000C (C
2020-02-18 01:40:57.202658 cc:98:91:fc:55:8b -> 01:80:c2:00:00:00 STP RST. Root = 32768/1/cc:98:91:fc:5
```

```
2020-02-18 01:40:57.367890      0.0.0.0 -> 255.255.255.255 DHCP DHCP Discover - Transaction ID 0xc82a6d
10 packets captured

Nexus9000_A(config-if-range)# ethanalyzer local interface mgmt
Capturing on mgmt0
2020-02-18 01:53:07.055100 cc:98:91:fc:55:94 -> 01:80:c2:00:00:00 STP RST. Root = 32768/46/84:8a:8d:7d:
2020-02-18 01:53:09.061398 cc:98:91:fc:55:94 -> 01:80:c2:00:00:00 STP RST. Root = 32768/46/84:8a:8d:7d:
2020-02-18 01:53:11.081596 cc:98:91:fc:55:94 -> 01:80:c2:00:00:00 STP RST. Root = 32768/46/84:8a:8d:7d:
2020-02-18 01:53:13.080874 cc:98:91:fc:55:94 -> 01:80:c2:00:00:00 STP RST. Root = 32768/46/84:8a:8d:7d:
2020-02-18 01:53:15.087361 cc:98:91:fc:55:94 -> 01:80:c2:00:00:00 STP RST. Root = 32768/46/84:8a:8d:7d:
2020-02-18 01:53:17.090164 cc:98:91:fc:55:94 -> 01:80:c2:00:00:00 STP RST. Root = 32768/46/84:8a:8d:7d:
2020-02-18 01:53:19.096518 cc:98:91:fc:55:94 -> 01:80:c2:00:00:00 STP RST. Root = 32768/46/84:8a:8d:7d:
2020-02-18 01:53:20.391215 00:be:75:5b:d9:00 -> 01:00:0c:cc:cc:cc CDP Device ID: Nexus9000_A(FDO21512ZE
2020-02-18 01:53:21.119464 cc:98:91:fc:55:94 -> 01:80:c2:00:00:00 STP RST. Root = 32768/46/84:8a:8d:7d:
2020-02-18 01:53:23.126011 cc:98:91:fc:55:94 -> 01:80:c2:00:00:00 STP RST. Root = 32768/46/84:8a:8d:7d:
10 packets captured

Nexus9000-A# ethanalyzer local interface front-panel eth1/1
Capturing on 'Eth1-1'
1 2022-07-15 19:46:04.698201919 28:ac:9e:ad:5c:b8 → 01:80:c2:00:00:00 STP 53 RST. Root = 32768/1/28:ac:9e:ad
2 2022-07-15 19:46:04.698242879 28:ac:9e:ad:5c:b8 → 01:00:0c:cc:cc:cd STP 64 RST. Root = 32768/1/28:ac:9e:ad
3 2022-07-15 19:46:04.698314467 28:ac:9e:ad:5c:b8 → 01:00:0c:cc:cc:cd STP 64 RST. Root = 32768/10/28:ac:9e:a
4 2022-07-15 19:46:04.698386112 28:ac:9e:ad:5c:b8 → 01:00:0c:cc:cc:cd STP 64 RST. Root = 32768/20/28:ac:9e:a
5 2022-07-15 19:46:04.698481274 28:ac:9e:ad:5c:b8 → 01:00:0c:cc:cc:cd STP 64 RST. Root = 32768/30/28:ac:9e:a
6 2022-07-15 19:46:04.698555784 28:ac:9e:ad:5c:b8 → 01:00:0c:cc:cc:cd STP 64 RST. Root = 32768/40/28:ac:9e:a
7 2022-07-15 19:46:04.698627624 28:ac:9e:ad:5c:b8 → 01:00:0c:cc:cc:cd STP 64 RST. Root = 32768/50/28:ac:9e:a
```

This output shows a few of the messages that can be captured with Ethanalyzer.



Note: By default, Ethanalyzer only captures up to 10 packets. However, you can use this command to prompt the CLI to capture packets indefinitely. Use **CTRL+C** to exit the capture mode.

```
Nexus9000_A(config-if-range)# ethanalyzer local interface inband limit-captured-frames 0
Capturing on inband
2020-02-18 01:43:30.542588 f8:b7:e2:49:2d:f2 -> 01:80:c2:00:00:0e LLC U, func=UI; SNAP, OUI 0x00000C (C
2020-02-18 01:43:30.542626 f8:b7:e2:49:2d:f5 -> 01:80:c2:00:00:0e LLC U, func=UI; SNAP, OUI 0x00000C (C
2020-02-18 01:43:30.542873 f8:b7:e2:49:2d:f4 -> 01:80:c2:00:00:0e LLC U, func=UI; SNAP, OUI 0x00000C (C
2020-02-18 01:43:30.542892 f8:b7:e2:49:2d:f3 -> 01:80:c2:00:00:0e LLC U, func=UI; SNAP, OUI 0x00000C (C
2020-02-18 01:43:31.596841 dc:f7:19:1b:f9:85 -> 01:80:c2:00:00:00 STP RST. Root = 32768/1/dc:f7:19:1b:f
2020-02-18 01:43:31.661089 f8:b7:e2:49:2d:b2 -> 01:80:c2:00:00:0e LLC U, func=UI; SNAP, OUI 0x00000C (C
2020-02-18 01:43:31.661114 f8:b7:e2:49:2d:b3 -> 01:80:c2:00:00:0e LLC U, func=UI; SNAP, OUI 0x00000C (C
2020-02-18 01:43:31.661324 f8:b7:e2:49:2d:b5 -> 01:80:c2:00:00:0e LLC U, func=UI; SNAP, OUI 0x00000C (C
2020-02-18 01:43:31.776638 cc:98:91:fc:55:8b -> 01:80:c2:00:00:00 STP RST. Root = 32768/1/cc:98:91:fc:5
2020-02-18 01:43:33.143814 f8:b7:e2:49:2d:b4 -> 01:80:c2:00:00:0e LLC U, func=UI; SNAP, OUI 0x00000C (C
2020-02-18 01:43:33.596810 dc:f7:19:1b:f9:85 -> 01:80:c2:00:00:00 STP RST. Root = 32768/1/dc:f7:19:1b:f
2020-02-18 01:43:33.784099 cc:98:91:fc:55:8b -> 01:80:c2:00:00:00 STP RST. Root = 32768/1/cc:98:91:fc:5
2020-02-18 01:43:33.872280 f8:b7:e2:49:2d:f2 -> 01:80:c2:00:00:0e LLC U, func=UI; SNAP, OUI 0x00000C (C
2020-02-18 01:43:33.872504 f8:b7:e2:49:2d:f5 -> 01:80:c2:00:00:0e LLC U, func=UI; SNAP, OUI 0x00000C (C
2020-02-18 01:43:33.872521 f8:b7:e2:49:2d:f4 -> 01:80:c2:00:00:0e LLC U, func=UI; SNAP, OUI 0x00000C (C
15 packets captured
```

You can also use filters with Ethanalyzer to focus on specific traffic. There are two types of filters that you can use with ethanalzyer. They are known as Capture filters and Display filters. A capture filter only captures the traffic that matches the criteria defined in the capture filter. A display filter still captures all traffic, but only the traffic that matches the criteria defined in the display filter is shown.

```
Nexus9000_B# ping 10.82.140.106 source 10.82.140.107 vrf management count 2
PING 10.82.140.106 (10.82.140.106) from 10.82.140.107: 56 data bytes
64 bytes from 10.82.140.106: icmp_seq=0 ttl=254 time=0.924 ms
64 bytes from 10.82.140.106: icmp_seq=1 ttl=254 time=0.558 ms

Nexus9000_A(config-if-range)# ethanalyzer local interface mgmt display-filter icmp
Capturing on mgmt0
2020-02-18 01:58:04.403295 10.82.140.107 -> 10.82.140.106 ICMP Echo (ping) request
2020-02-18 01:58:04.403688 10.82.140.106 -> 10.82.140.107 ICMP Echo (ping) reply
2020-02-18 01:58:04.404122 10.82.140.107 -> 10.82.140.106 ICMP Echo (ping) request
2020-02-18 01:58:04.404328 10.82.140.106 -> 10.82.140.107 ICMP Echo (ping) reply

4 packets captured
```

You can also capture packets with the detail option and view them in your terminal, similar to how you would in Wireshark. This allows you to see the full header information based on the packet dissector result. For example, if a frame is encrypted, you would not be able to see the encrypted payload. See this example:

```
Nexus9000_A(config-if-range)# ethanalyzer local interface mgmt display-filter icmp detail
Capturing on mgmt0
Frame 2 (98 bytes on wire, 98 bytes captured)
    Arrival Time: Feb 18, 2020 02:02:17.569801000
    [Time delta from previous captured frame: 0.075295000 seconds]
    [Time delta from previous displayed frame: 0.075295000 seconds]
    [Time since reference or first frame: 0.075295000 seconds]
    Frame Number: 2
    Frame Length: 98 bytes
    Capture Length: 98 bytes
    [Frame is marked: False]
    [Protocols in frame: eth:ip:icmp:data]
Ethernet II, Src: 00:be:75:5b:de:00 (00:be:75:5b:de:00), Dst: 00:be:75:5b:d9:00 (00:be:75:5b:d9:00)
    Destination: 00:be:75:5b:d9:00 (00:be:75:5b:d9:00)
        Address: 00:be:75:5b:d9:00 (00:be:75:5b:d9:00)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
    Type: IP (0x0800)
>>>>>>>Output Clipped
```

With Ethanalyzer you can:

- Write the output (a PCAP file) to your specified file name on various target file systems: bootflash, logflash, USB, and so on.
- You could then transfer the saved file to outside of the device and view it in Wireshark, as needed.
- Read a file from bootflash and display on your terminal. Just like when you read from the CPU interface directly, you can also display the full packet info if you use the detail keyword.

See examples of this for various interface sources and output options:

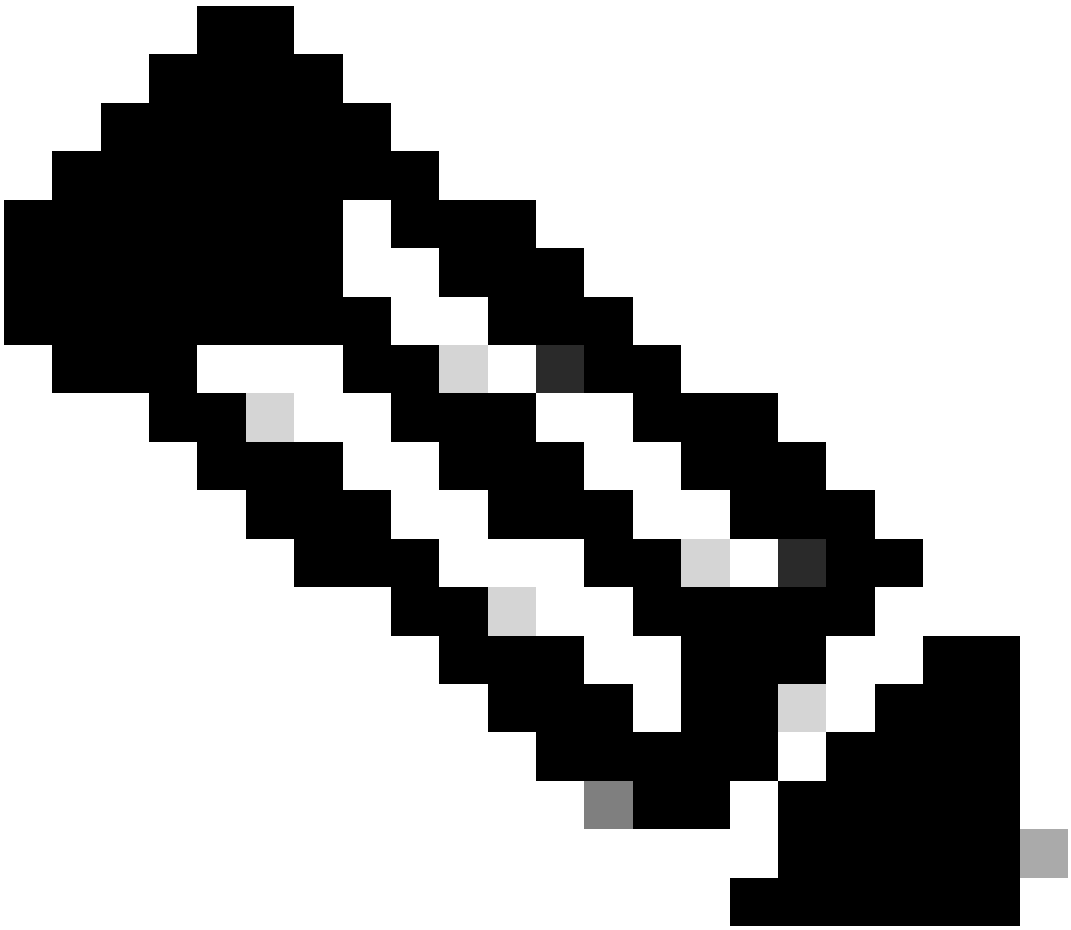
```
Nexus9000_A# ethanalyzer local interface mgmt capture-filter "host 10.82.140.107" write bootflash:TEST.
Capturing on mgmt0
10
Nexus9000_A# dir bootflash:
       4096    Feb 11 02:59:04 2020  .rpmstore/
       4096    Feb 12 02:57:36 2020  .swtam/
       2783    Feb 17 21:59:49 2020  09b0b204-a292-4f77-b479-1ca1c4359d6f.config
       1738    Feb 17 21:53:50 2020  20200217_215345_poap_4168_init.log
       7169    Mar 01 04:41:55 2019  686114680.bin
       4411    Nov 15 15:07:17 2018  EBC-SC02-M2_303_running_config.txt
   13562165    Oct 26 06:15:35 2019  GBGBLD4SL01DRE0001-CZ07-
        590    Jan 10 14:21:08 2019  MDS20190110082155835.lic
       1164    Feb 18 02:18:15 2020  TEST.PCAP
>>>>>>>Output Clipped

Nexus9000_A# copy bootflash: ftp:
Enter source filename: TEST.PCAP
Enter vrf (If no input, current vrf 'default' is considered): management
Enter hostname for the ftp server: 10.122.153.158
Enter username: calo
Password:
***** Transfer of file Completed Successfully *****
Copy complete, now saving to disk (please wait)...
Copy complete.

Nexus9000_A# ethanalyzer local read bootflash:TEST.PCAP
2020-02-18 02:18:03.140167 10.82.140.107 -> 10.82.140.106 ICMP Echo (ping) request
2020-02-18 02:18:03.140563 10.82.140.106 -> 10.82.140.107 ICMP Echo (ping) reply
2020-02-18 02:18:15.663901 10.82.140.107 -> 10.82.140.106 ICMP Echo (ping) request
2020-02-18 02:18:15.664303 10.82.140.106 -> 10.82.140.107 ICMP Echo (ping) reply
2020-02-18 02:18:15.664763 10.82.140.107 -> 10.82.140.106 ICMP Echo (ping) request
2020-02-18 02:18:15.664975 10.82.140.106 -> 10.82.140.107 ICMP Echo (ping) reply
2020-02-18 02:18:15.665338 10.82.140.107 -> 10.82.140.106 ICMP Echo (ping) request
2020-02-18 02:18:15.665536 10.82.140.106 -> 10.82.140.107 ICMP Echo (ping) reply
2020-02-18 02:18:15.665864 10.82.140.107 -> 10.82.140.106 ICMP Echo (ping) request
2020-02-18 02:18:15.666066 10.82.140.106 -> 10.82.140.107 ICMP Echo (ping) reply

RTP-SUG-BGW-1# ethanalyzer local interface front-panel eth1-1 write bootflash:e1-1.pcap
Capturing on 'Eth1-1'
10

RTP-SUG-BGW-1# ethanalyzer local read bootflash:e1-1.pcap detail
Frame 1: 53 bytes on wire (424 bits), 53 bytes captured (424 bits) on interface Eth1-1, id 0
    Interface id: 0 (Eth1-1)
        Interface name: Eth1-1
    Encapsulation type: Ethernet (1)
    Arrival Time: Jul 15, 2022 19:59:50.696219656 UTC
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1657915190.696219656 seconds
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 53 bytes (424 bits)
    Capture Length: 53 bytes (424 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:llc:stp]
```

SPAN

SwitchPort Analyzer (SPAN) is used to capture all the traffic from an interface and mirror that traffic to a destination port. The destination port typically connects to a network analyzer tool (such as a PC running Wireshark) that allows you to analyze the traffic that traverses through those ports. You can SPAN for traffic from a single port or multiple ports and VLANs.

SPAN sessions include a source port and a destination port. A Source Port can be an Ethernet Port (no sub-interfaces), Port Channels, Supervisor Inband Interfaces and can not be a destination port simultaneously. Moreover, for some devices like the 9300 and 9500 platform, Fabric Extender (FEX) ports are also supported. A Destination Port can be an Ethernet Port (Access or Trunk), Port Channel (Access or Trunk) and for some devices like the 9300 uplink ports are also supported while FEX ports are not supported. destination.

You can configure multiple SPAN sessions to be an ingress/egress/both. There is a limit to the total number of SPAN sessions that an individual device can support. For example, a Nexus 9000 can support up to 32 sessions while a Nexus 7000 can only support 16. You can check this on the CLI or refer to the SPAN configuration guides for the product you use.



Note: Each NX-OS release, the product type, the supported interfaces types, and functionality differ. Refer to the latest configuration guidelines and limitations for the product and version you use.

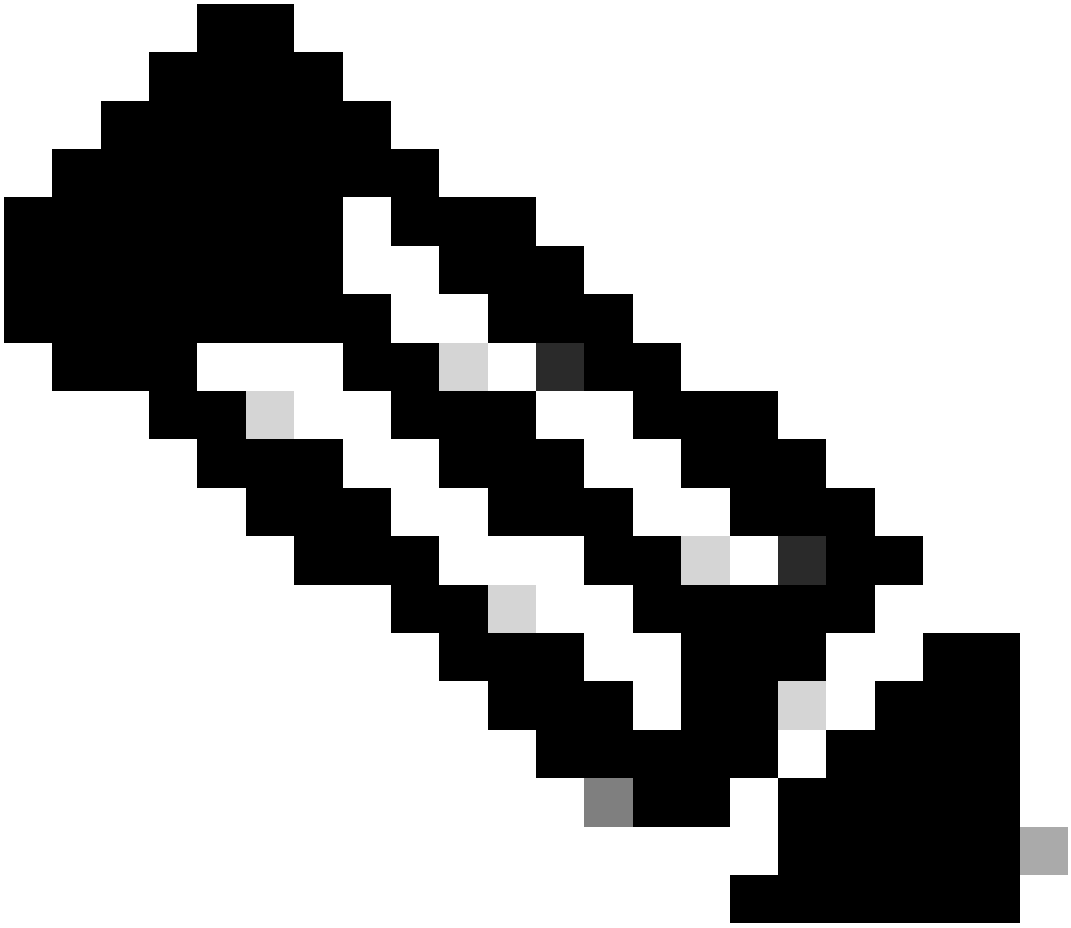
Here are the links for Nexus 9000 and Nexus 7000 respectively:

Cisco Nexus 9000 Series NX-OS System Management Configuration Guide, Release 9.3(x) - Configuring SPAN [Cisco Nexus 9000 Series Switches] - Cisco

Cisco Nexus 7000 Series NX-OS System Management Configuration Guide - Configuring SPAN [Cisco Nexus 7000 Series Switches] - Cisco

There are various types of SPAN sessions. Some of the more common types are listed here:

- Local SPAN: A type of SPAN session where the both the source and destination host are local to the switch. In other words, all of the configuration required to setup the SPAN session is applied to a single switch, the same switch where the source and destination host ports reside.
- Remote SPAN (RSPAN): A type of SPAN session where the source and destination host are not local to the switch. In other words, you configure source RSPAN sessions on one switch and destination RSPAN on the destination switch and extend the connectivity with RSPAN VLAN.



Note: RSPAN is not supported on Nexus.

- Extended Remote SPAN (ERSPAN): The switch encapsulates the copied frame with a Generic Routing Encapsulation (GRE) tunnel header and route the packet to the configured destination. You configure source and destination sessions on the encapsulation and decapsulation switches (two different devices). This gives you the ability to SPAN traffic over a layer 3 network.
- SPAN-to-CPU: A name given to a special type of SPAN session where your destination port is the supervisor or CPU. It is a form of local SPAN session and can be used in cases where you cannot utilize a standard SPAN session. Some of the common reasons are: no available or suitable SPAN destination ports, site not accessible, or unmanaged site, no device available which can connect to SPAN destination port, and so on. For details, refer to this link Nexus 9000 Cloud Scale ASIC NX-OS SPAN-to-CPU Procedure - Cisco. It is important to remember that SPAN-to CPU is rate limited by Control Plane Policing (CoPP), so sniffing one or more source interfaces which exceed the policer can result in drops for the SPAN to CPU session. If this happens, the data is not 100% reflective of what is on the wire, so SPAN to CPU is not always appropriate for troubleshooting scenarios with high data rate and/or intermittent loss. Once you configure a SPAN to CPU session and administratively enable it, you need to run Ethanalyzer to see the traffic that is sent to the CPU to perform analysis accordingly.

This is an example of how you can configure a simple local SPAN session on a Nexus 9000 switch:

```
Nexus9000_A(config-monitor)# monitor session ?

*** No matching command found in current mode, matching in (config) mode ***
  <1-32>
  all     All sessions

Nexus9000_A(config)# monitor session 10
Nexus9000_A(config-monitor)# ?
  description  Session description (max 32 characters)
  destination  Destination configuration
  filter       Filter configuration
  mtu          Set the MTU size for SPAN packets
  no           Negate a command or set its defaults
  show         Show running system information
  shut         Shut a monitor session
  source       Source configuration
  end          Go to exec mode
  exit         Exit from command interpreter
  pop          Pop mode from stack or restore from name
  push         Push current mode to stack or save it under name
  where        Shows the cli context you are in

Nexus9000_A(config-monitor)# description Monitor_Port_e1/1
Nexus9000_A(config-monitor)# source interface ethernet 1/1
Nexus9000_A(config-monitor)# destination interface ethernet 1/10
Nexus9000_A(config-monitor)# no shut
```

This example shows the configuration of a SPAN to CPU session that has been brought up, and then the use of Ethanalyzer to capture the traffic:

```
N9000-A# show run monitor

monitor session 1
source interface Ethernet1/7 rx
destination interface sup-eth0                  << this is what sends the traffic to CPU
no shut

RTP-SUG-BGW-1# ethanalyzer local interface inband mirror limit-c 0
Capturing on 'ps-inb'
2020-02-18 02:18:03.140167 10.82.140.107 -> 10.82.140.106 ICMP Echo (ping) request
2020-02-18 02:18:15.663901 10.82.140.107 -> 10.82.140.106 ICMP Echo (ping) request
```

Dmirror

Dmirror is a type of SPAN-TO-CPU session for Broadcom-based Nexus platforms. The concept is the same as it is for SPAN-to-CPU and is rate limited to 50 pps (packets per seconds). The feature was implemented to debug the internal data path with the bcm-shell CLI. Because of the associated limitations there is no NX-OS CLI to allow users to configure SPAN sessions to the Sup because it can affect control traffic and consume CoPP Classes.

ELAM

Embedded Logic Analyzer Module (ELAM) provides the ability to look into the ASIC and determine what

forwarding decisions are made for a SINGLE packet. So, with ELAM, you can identify whether the packet reach the Forwarding Engine and on what Ports/VLAN information. You can also check for L2 – L4 packet structure and whether any alterations were made to the packet or not.

It is important to understand that ELAM is architecture dependent and the procedure to capture a packet varies from platform to platform based upon the internal architecture. You must know the ASIC mappings of the hardware in order to correctly apply the tool. For Nexus 7000, two captures are taken for a single packet, one before the decision is made, Data BUS (DBUS), and the other after the decision has been made, Result BUS (RBUS). When you view the DBUS information you can see what/where the packet was received, as well as, the layer 2 to 4 information. Results in the RBUS can show you where the packet is forwarded to, and if the frame was altered. You need to set up triggers for DBUS and RBUS, ensure they are ready, and then try to capture the packet in real time. The procedures for various line cards is as follows:

For details on various ELAM procedures, please refer to the links in this table:

ELAM OVERVIEW	ELAM Overview - Cisco
Nexus 7K F1 Module	Nexus 7000 F1 Module ELAM Procedure - Cisco
Nexus 7K F2 Module	Nexus 7000 F2 Module ELAM Procedure - Cisco
Nexus 7K F3 Module	F3- ELAM Example
Nexus 7K M Module	Nexus 7000 M-Series Module ELAM Procedure - Cisco
Nexus 7K M1/M2 and F2 Module	Nexus 7K ELAM for M1/M2 and F2 and Ethanalyzer
Nexus 7K M3 Module	Nexus 7000 M3 Module ELAM Procedure - Cisco

ELAM for Nexus 7000 – M1/M2 (Eureka Platform)

- Check the module number with the command **show module**.
- Attach to the module with **attach module x**, where **x** is the module number.
- Check for internal ASIC mapping with the command **show hardware internal dev-port-map** and check for **L2LKP** and **L3LKP**.

<#root>

```
Nexus7000(config)#
```

show module

```
Mod  Ports  Module-Type                        Model              Status
---  -----  ---------------------------------- ------------------ ----------
1    0      Supervisor Module-2                N7K-SUP2E          active *
2    0      Supervisor Module-2                N7K-SUP2E          ha-standby
3    48     1/10 Gbps Ethernet Module          N7K-F248XP-25E     ok
4    24     10 Gbps Ethernet Module            N7K-M224XP-23L     ok


Nexus7000(config)#
```

attach module 4

```
Attaching to module 4 ...
To exit type 'exit', to abort type '$.'
Last login: Fri Feb 14 18:10:21 UTC 2020 from 127.1.1.1 on pts/0

module-4#
```

show hardware internal dev-port-map

```
--------------------------------------------------------------
CARD_TYPE:       24 port 10G
>Front Panel ports:24
--------------------------------------------------------------
 Device name             Dev role              Abbr num_inst:
--------------------------------------------------------------
> Skytrain               DEV_QUEUEING          QUEUE  4
> Valkyrie               DEV_REWRITE           RWR_0  4
> Eureka                 DEV_LAYER_2_LOOKUP    L2LKP  2
> Lamira                 DEV_LAYER_3_LOOKUP    L3LKP  2
> Garuda                 DEV_ETHERNET_MAC      MAC_0  2
> EDC                    DEV_PHY               PHYS   6
> Sacramento Xbar ASIC   DEV_SWITCH_FABRIC     SWICHF 1
+-----------------------------------------------------------------------+
+----------------+++FRONT PANEL PORT TO ASIC INSTANCE MAP+++------------+
+-----------------------------------------------------------------------+
FP port |  PHYS | SECUR | MAC_0 | RWR_0 | L2LKP | L3LKP | QUEUE |SWICHF
   1       0       0       0       0,1     0       0       0,1     0
   2       0       0       0       0,1     0       0       0,1     0
   3       0       0       0       0,1     0       0       0,1     0
   4       0       0       0       0,1     0       0       0,1     0
   5       1       0       0       0,1     0       0       0,1     0
   6       1       0       0       0,1     0       0       0,1     0
   7       1       0       0       0,1     0       0       0,1     0
   8       1       0       0       0,1     0       0       0,1     0
   9       2       0       0       0,1     0       0       0,1     0
   10      2       0       0       0,1     0       0       0,1     0
   11      2       0       0       0,1     0       0       0,1     0
   12      2       0       0       0,1     0       0       0,1     0
   13      3       1       1       2,3     1       1       2,3     0
   14      3       1       1       2,3     1       1       2,3     0
   15      3       1       1       2,3     1       1       2,3     0
   16      3       1       1       2,3     1       1       2,3     0
   17      4       1       1       2,3     1       1       2,3     0
   18      4       1       1       2,3     1       1       2,3     0
   19      4       1       1       2,3     1       1       2,3     0
   20      4       1       1       2,3     1       1       2,3     0
   21      5       1       1       2,3     1       1       2,3     0
   22      5       1       1       2,3     1       1       2,3     0
   23      5       1       1       2,3     1       1       2,3     0
   24      5       1       1       2,3     1       1       2,3     0
+-----------------------------------------------------------------------+
```

```
+------------------------------------------------------------------------+
```

- First, you capture the packet in L2 and see if the forwarding decision is right. To do this, look into the L2LKP mappings column and identify the ASIC instance # that corresponds to the port.
- Next, you run ELAM on this instance with the command **elam asic eureka instance x**where **x** is the ASIC instance number and configure our triggers for DBUS and RBUS. Check the status of the triggers with the command **status** and confirm the triggers have been configured.

<#root>

```
module-4(eureka-elam)#
```

```
trigger dbus dbi ingress ipv4 if source-ipv4-address 192.0.2.2 destination-ipv4-address 192.0.2.4 rbi-co
```

```
module-4(eureka-elam)#
```

```
trigger rbus rbi pb1 ip if cap2 1
```

```
module-4(eureka-elam)# status

Slot: 4, Instance: 1
EU-DBUS: Configured
trigger dbus dbi ingress ipv4 if source-ipv4-address 192.168.10.1
EU-RBUS: Configured
trigger rbus rbi pb1 ip if cap2 1
```

- Activate the triggers with the command start, and verify that the status of the triggers with the command status to confirm that the triggers are armed.

```
module-4(eureka-elam)# start
module-4(eureka-elam)# status

Slot: 4, Instance: 1
EU-DBUS: Armed     <<<<<<<<<<
trigger dbus dbi ingress ipv4 if source-ipv4-address 192.168.10.1
EU-RBUS: Armed     <<<<<<<<<<
trigger rbus rbi pb1 ip if cap2 1
```

- Once the status shows that the triggers are armed, they are ready to capture. At this time, you must send the traffic through and check the status again to see if your triggers were actually triggered.

<#root>

```
module-4(eureka-elam)#
```

```
status
```

```
Slot: 4, Instance: 1
EU-DBUS:
```

```
Triggered     <<<<<<<<<<
```

```
trigger dbus dbi ingress ipv4 if source-ipv4-address 192.168.10.1
EU-RBUS:

Triggered     <<<<<<<<<<

trigger rbus rbi pb1 ip if cap2 1
```

- Once triggered, check the packet sequence number for both rbus and dbus in order to confirm that they both have captured the same packet. This can be done with the command **show dbus | i seq ; show rbus | i seq**. If the sequence number matches, you can view the contents of the dbus and rbus. If not, re-run the capture until you are able to capture the same packet.

Note: For more accuracy, always run ELAM multiple times to confirm forwarding issues.

- You can view the content of rbus and dbus with the commands **show dbus** and **show rbus**. The important thing in the capture is the sequence # and the source/destination index. Dbus shows you the source index which tells you the port it received the packet on. Rbus shows you the destination index of the port to which the packet is forwarded to. Additionally, you can also look at source and destination IP/MAC addresses as well as VLAN information.
- With the source and destination index (also known as LTL index), you can check the the associated front panel port with the command **show system internal pixm info ltl #**.

ELAM for Nexus 7000 – M1/M2 (Lamira Platform)

The procedure is the same for Lamira platform as well, however, there are a few differences:

- You run ELAM with keyword Lamiraelam asic lamira instance x.
- The commands to trigger the ELAM are:

<#root>

```
module-4(lamira-elam)#

trigger dbus ipv4 if source-ipv4-address 192.0.2.2 destination-ipv4-address 192.0.2.4

module-4(lamira-elam)#

trigger rbus <ife|ofe> ip if elam-match 1
```

- You check for status with the **status** command and ensure they are Armed before you send traffic, and triggered, after you capture it.
- You can then interpret the outputs of dbus and show bus in similar fashion as shown for Eureka.

ELAM for Nexus 7000 – F2/F2E (Clipper Platform)

Again, the procedure is similar, only the triggers are different. The few differences are as follows:

- You run **ELAM** with keyword Clipperelam asic clipper instance x and specify **Layer 2** or **Layer 3** mode.

<#root>

```
module-4# elam asic  clipper instance 1
```

```
module-4(clipper-elam)# <layer2/Layer3>
```

- The commands to trigger the ELAM are as follows:

<#root>

```
module-4(clipper-l2-elam)#
```

```
trigger dbus ipv4 ingress if source-ipv4-address 192.0.2.3 destination-ipv4-address 192.0.2.2
```

```
module-4(clipper-l2-elam)#
```

```
trigger rbus ingress if trig
```

- You check for status with the **status** command, and ensure they are Armed before you send traffic, and triggered, after you capture it.
- You can then interpret the outputs of dbus and show bus in similar fashion as shown for Eureka.

ELAM for Nexus 7000 – F3 (Flanker Platform)

Again, the procedure is similar, only triggers are different. The few differences are as follows:

- You run **ELAM** with keyword Flanker **elam asic flanker instance x** and specify **Layer 2** or **Layer 3** mode.

```
module-4# elam asic flanker instance 1
module-4(flanker-elam)# <layer2/Layer3>
```

- The commands to trigger the ELAM are as follows:

<#root>

```
module-9(fln-l2-elam)#
```

```
trigger dbus ipv4 if destination-ipv4-address 10.1.1.2
```

```
module-9(fln-l2-elam)#
```

```
trigger rbus ingress if trig
```

- You check for status with the **status** command, and ensure they are Armed before you send traffic, and triggered, after you capture it.
- You can then interpret the outputs of dbus and rbus in similar fashion as shown for Eureka.

ELAM for Nexus 9000 (Tahoe Platform)

In Nexus 9000, the procedure is a little different than Nexus 7000. For Nexus 9000, refer to the link [Nexus 9000 Cloud Scale ASIC (Tahoe) NX-OS ELAM - Cisco]

- First, check for interface mapping with the command **show hardware internal tah interface #**. The most important information in this output is the ASIC#, Slice #, and source ID (srcid) #.

- Additionally, you can also double check this information with the command **show system internal ethpm info interface # | i i src**. The important thing here, in addition to what was listed previously, is the dpid and dmod values.
- Check the module number with the command **show module**.
- Attach to the module with **attach module x**, where **x** is the module number.
- Run ELAM on the module with the command **module-1# debug platform internal tah elam asic #**.
- Configure your inner or outer trigger based on the kind of traffic you want to capture (L2, L3, encapsulated traffic such as GRE or VXLAN, and so on):

<#root>

Nexus9000(config)#

attach module 1

module-1#

debug platform internal tah elam asic 0

module-1(TAH-elam)#

trigger init asic # slice #

lu-a2d 1 in-select 6 out-select 0 use-src-id #

module-1(TAH-elam-insel6)#

reset

module-1(TAH-elam-insel6)#

set outer ipv4 dst_ip 192.0.2.1 src_ip 192.0.2.2

- Once triggers are set, start ELAM with the command **start**, send traffic and view the output with the command **report**. The output of report shows you the outgoing and incoming interfaces along with the vlan ID, source and destination IP/MAC address.

<#root>

SUGARBOWL ELAM REPORT SUMMARY
slot - 1, asic - 1, slice - 1
==========

Incoming Interface:

Eth1/49

Src Idx : 0xd, Src BD :

10

Outgoing Interface Info:

dmod 1, dpid 14

```
Dst Idx : 0x602, Dst BD : 10

Packet Type: IPv4
Dst MAC address: CC:46:D6:6E:28:DB
Src MAC address: 00:FE:C8:0E:27:15
.1q Tag0 VLAN:
```

10

```
, cos = 0x0
Dst IPv4 address: 192.0.2.1
Src IPv4 address: 192.0.2.2

Ver     =  4, DSCP    =    0, Don't Fragment = 0
Proto   =  1, TTL     =   64, More Fragments = 0
Hdr len = 20, Pkt len =   84, Checksum       = 0x667f
```

ELAM for Nexus 9000 (NorthStar Platform)

The procedure for NorthStar platform is the same as Tahoe platform, the only difference is that the keyword **ns** is used instead of **tah** when ELAM mode is entered:

<#root>

module-1# debug platform internal ns elam asic 0

N9K Packet Tracer

Nexus 9000 packet tracer tool can be used to track the path of the packet, and with its built-in counters for flow statistics, makes it a valuable tool for intermittent/complete traffic loss scenarios. It would be very useful where TCAM resources are limited, or not available, to run other tools. Additionally, this tool cannot capture ARP traffic and does not display details of the packets content like Wireshark.

To configure packet tracer, use these commands:

<#root>

```
N9K-9508#
```

test packet-tracer src_ip <src_ip> dst_ip <dst_ip>

```
       <==== provide your src and dst ip
N9K-9508#
```

test packet-tracer start

```
                                <==== Start packet tracer
N9K-9508#
```

test packet-tracer stop

```
                                 <==== Stop packet tracer
N9K-9508#
```

test packet-tracer show

```
                                        <==== Check for packet matches
```

For details, refer to the link [Nexus 9000: Packet Tracer tool explained - Cisco]

Traceroute and Pings

These commands are the two most useful commands that allows you to quickly identify connectivity issues.

Ping uses Internet Control Message Protocol (ICMP) to send ICMP echo messages to the specific destination and waits for ICMP echo replies from that destination. If the path between the host works fine with no issues, you can see the replies come back and pings are successful. The ping command by default sends 5x ICMP echo messages (equal size in both directions) and if everything works fine, you can see 5x ICMP echo replies. Sometimes, the initial echo request fails when switches learn the MAC address during the Address Resolution Protocol (ARP) request. If you run the ping again straight after, there is no initial ping loss. Moreover, you can also set the number of pings, packet-size, source, source interface, and time-out intervals with these keywords:

<#root>

```
F241.04.25-N9K-C9318O-1#
```

```
ping 10.82.139.39 vrf management
```

```
PING 10.82.139.39 (10.82.139.39): 56 data bytes
36 bytes from 10.82.139.38: Destination Host Unreachable
Request 0 timed out
64 bytes from 10.82.139.39: icmp_seq=1 ttl=254 time=23.714 ms
64 bytes from 10.82.139.39: icmp_seq=2 ttl=254 time=0.622 ms
64 bytes from 10.82.139.39: icmp_seq=3 ttl=254 time=0.55 ms
64 bytes from 10.82.139.39: icmp_seq=4 ttl=254 time=0.598 ms
```

```
F241.04.25-N9K-C9318O-1#
```

```
ping 10.82.139.39 ?
```

```
  <CR>
  count             Number of pings to send
  df-bit            Enable do not fragment bit in IP header
  interval          Wait interval seconds between sending each packet
  packet-size       Packet size to send
  source            Source IP address to use
  source-interface  Select source interface
  timeout           Specify timeout interval
  vrf               Display per-VRF information
```

Traceroute is used to identify the various hops a packet takes before it reaches its destination. It is a very important tool because it helps to identify the L3 boundary where failure happens. You can also use the port, source, and source interface with these keywords:

<#root>

```
F241.04.25-N9K-C9318O-1#
```

```
traceroute 10.82.139.39 ?
```

```
  <CR>
  port              Set destination port
  source            Set source address in IP header
  source-interface  Select source interface
  vrf               Display per-VRF information


Nexus_1(config)#
```

```
traceroute 192.0.2.1
```

```
traceroute to 192.0.2.1 (192.0.2.1), 30 hops max, 40 byte packets
 1 198.51.100.3 (198.51.100.3)  1.017 ms  0.655 ms  0.648 ms
 2 203.0.113.2 (203.0.113.2)  0.826 ms  0.898 ms  0.82 ms
 3 192.0.2.1 (192.0.2.1)  0.962 ms  0.765 ms  0.776 ms
```

PACL/RACL/VACL

Access Control List (ACL) is an important tool that allows you to filter traffic based on a relevant defined criterion. Once the ACL is filled with entries for match criteria, it can be applied to capture either inbound or outbound traffic. An important aspect of ACL is its ability to provide counters for flow statistics. The terms PACL/RACL/VACL refer to various implementations of these ACLs that allows you to use ACL as a powerful troubleshooting tool especially for intermittent traffic loss. These terms are described briefly here:

- Port Access Control List (PACL): When you apply an access list to a L2 switchport/interface, that access list is known as PACL.
- Router Access Control List (RACL): When you apply an access list to a L3 routed port/interface, that access list is known as RACL.
- VLAN Access Control List (VACL): You can configure VACLs to apply to all packets that are routed into or out of a VLAN or are bridged within a VLAN. VACLs are strictly for security packet filters and to redirect traffic to specific physical interfaces. VACLs are not defined by direction (ingress or egress).

This table provides a comparison between the versions of ACLs.

ACL TYPE	**PACL**	**RACL**	**VACL**
FUNCTION	Filter traffic received on an L2 interface.	Filter traffic received on an L3 interface.	Filter vLAN traffic.
APPLIED ON	- L2 interfaces/ports. - L2 port-channel interfaces. - If applied on a trunk port, ACL filters traffic on all VLANs allowed on that trunk port.	- VLAN interfaces. - Physical L3 interfaces. - L3 sub-interfaces. - L3 port-channel interfaces. - Management interfaces.	Once enabled, the ACL is applied to all ports in that VLAN (includes trunk ports).

APPLIED DIRECTION	Inbound only.	Inbound or Outbound	-

Here is an example of how you can configure an access list. For details, refer to the link Cisco Nexus 9000 Series NX-OS Security Configuration Guide, Release 9.3(x) - Configuring IP ACLs [Cisco Nexus 9000 Series Switches] - Cisco

<#root>

```
Nexus93180(config)#

ip access-list <Name_of_ACL>

Nexus93180(config-acl)# ?
  <1-4294967295>  Sequence number
  deny            Specify packets to reject
  fragments       Optimize fragments rule installation
  no              Negate a command or set its defaults
  permit          Specify packets to forward
  remark          Access list entry comment
  show            Show running system information
  statistics      Enable per-entry statistics for the ACL
  end             Go to exec mode
  exit            Exit from command interpreter
  pop             Pop mode from stack or restore from name
  push            Push current mode to stack or save it under name
  where           Shows the cli context you are in

Nexus93180(config)# int e1/1

Nexus93180(config-if)# ip port access-group <NAME_of_ACL> ? >>>>>> When you configure ACL like this, it
  in  Inbound packets

Nexus93180(config-if)# ip access-group <NAME_of_ACL> ?      >>>>>> When you configure ACL like this, it
  in   Inbound packets
  out  Outbound packets
```

LOGFLASH

LogFlash is a type of persistent storage available on Nexus platforms as either an external compact flash, a USB device, or an embedded disk in the supervisor. If removed from the switch, the system periodically notifies the user that LogFlash is missing. Logflash is installed on the supervisor and holds historical data such as accounting logs, syslog messages, debugs and Embedded Event Manager (EEM) outputs. EEM is discussed later in this article. You can check the contents of the LogFlash with this command:

<#root>

```
Nexus93180(config)#

dir logflash:

          0    Nov 14 04:13:21 2019  .gmr6_plus
      20480    Feb 18 13:35:07 2020  ISSU_debug_logs/
         24    Feb 20 20:43:24 2019  arp.pcap
         24    Feb 20 20:36:52 2019  capture_SYBO10L2289.pcap
```

```
       4096    Feb 18 17:24:53 2020  command/
       4096    Sep 11 01:39:04 2018  controller/
       4096    Aug 15 03:28:05 2019  core/
       4096    Feb 02 05:21:47 2018  debug/
    1323008    Feb 18 19:20:46 2020  debug_logs/
       4096    Feb 17 06:35:36 2020  evt_log_snapshot/
       4096    Feb 02 05:21:47 2018  generic/
       1024    Oct 30 17:27:49 2019  icamsql_1_1.db
      32768    Jan 17 11:53:23 2020  icamsql_1_1.db-shm
     129984    Jan 17 11:53:23 2020  icamsql_1_1.db-wal
       4096    Feb 14 13:44:00 2020  log/
      16384    Feb 02 05:21:44 2018  lost+found/
       4096    Aug 09 20:38:22 2019  old_upgrade/
       4096    Feb 18 13:40:36 2020  vdc_1/

Usage for logflash://sup-local
 1103396864 bytes used
 7217504256 bytes free
 8320901120 bytes total
```

In the event that a user reloaded the device, or it suddenly reloaded on its own due to an event, all of the log information would be lost. In such scenarios, LogFlash can provide historical data that can be reviewed to identify a probable cause of the issue. Off course, further due diligence is required to identify the root cause which provides you with hints on what to look for in case this event happens again.

For information about how to install logflash on the device, refer to the link [Nexus 7000 Logging Capabilities - Cisco].

OBFL

On Board Failure Logging (OBFL) is a type of persistent storage available to both Nexus Top of Rack and Modular switches. Just like the LogFlash, the information is retained once the device is reloaded. OBFL stores information such as failures and environmental data. The information varies for each platform and module, however, here is a sample output of module 1 of Nexus 93108 platform (that is a fixed chassis with one module only):

<#root>

```
Nexus93180(config)#
```

`show logging onboard module 1 ?`

```
*** No matching command found in current mode, matching in (exec) mode ***
  <CR>
  >                     Redirect it to a file
  >>                    Redirect it to a file in append mode
  boot-uptime           Boot-uptime
  card-boot-history     Show card boot history
  card-first-power-on   Show card first power on information
  counter-stats         Show OBFL counter statistics
  device-version        Device-version
  endtime               Show OBFL logs till end time mm/dd/yy-HH:MM:SS
  environmental-history Environmental-history
  error-stats           Show OBFL error statistics
  exception-log         Exception-log
  internal              Show Logging Onboard Internal
  interrupt-stats       Interrupt-stats
  obfl-history          Obfl-history
```

```
  stack-trace            Stack-trace
  starttime              Show OBFL logs from start time mm/dd/yy-HH:MM:SS
  status                 Status
  |                      Pipe command output to filter


Nexus93180(config)#
```

show logging onboard module 1 status

```
----------------------------
OBFL Status
----------------------------
    Switch OBFL Log:                                      Enabled
    Module:  1 OBFL Log:                                  Enabled
    card-boot-history                                     Enabled
    card-first-power-on                                   Enabled
    cpu-hog                                               Enabled
    environmental-history                                 Enabled
    error-stats                                           Enabled
    exception-log                                         Enabled
    interrupt-stats                                       Enabled
    mem-leak                                              Enabled
    miscellaneous-error                                   Enabled
    obfl-log (boot-uptime/device-version/obfl-history)    Enabled
    register-log                                          Enabled
    system-health                                         Enabled
    temp Error                                            Enabled
    stack-trace                                           Enabled
```

Again, this information is useful in the event of a device that is reloaded either purposely by the user, or due to an event that triggered a reload. In this case, OBFL information can help to identify what went wrong from a Line Card's perspective. The command **show logging onboard** is a good place to start. Remember that you must capture from inside the module context to get everything you need. Ensure you use **show logging onboard module x** or **attach mod x ; show logging onboard**.

Event-Histories

Event-histories are one of the powerful tools that can provide you with information about various events that take place for a process that runs on Nexus. In other words, every process that runs on a Nexus platform has event-histories that run in the background and store information about various events of that process (think of them as debugs that run constantly). These event histories are non-persistent and all the information stored is lost upon device reload. These are very useful when you have identified a problem with a certain process, and would like to troubleshoot that process. For example, if your OSPF routing protocol does not work properly, you can use event-histories associated with OSPF to identify where the OSPF process fails. You can find event histories associated with almost every process on the Nexus platform such as CDP/STP, UDLD, LACP/OSPF, EIGRP/BGP, and so on.

This is how you would typically check for event histories for a process with reference examples. Every process has multiple options so use **?** to check for various options available under a process.

<#root>

```
Nexus93180(config)#
```

show <Process> internal event-history ?

```
Nexus93180# show ip ospf event-history ?
  adjacency       Adjacency formation logs
  cli             Cli logs
  event           Internal event logs
  flooding        LSA flooding logs
  ha              HA and GR logs
  hello           Hello related logs
  ldp             LDP related logs
  lsa             LSA generation and databse logs
  msgs            IPC logs
  objstore        DME OBJSTORE related logs
  redistribution  Redistribution logs
  rib             RIB related logs
  segrt           Segment Routing logs
  spf             SPF calculation logs
  spf-trigger     SPF TRIGGER related logs
  statistics      Show the state and size of the buffers
  te              MPLS TE related logs

Nexus93180#
```

```
show spanning-tree internal event-history ?

  all      Show all event historys
  deleted  Show event history of deleted trees and ports
  errors   Show error logs of STP
  msgs     Show various message logs of STP
  tree     Show spanning tree instance info
  vpc      Show virtual Port-channel event logs
```

Debugs

Debugs are powerful tools within NX-OS that allow you to run real-time troubleshooting events and log them to a file or display in CLI. It is highly recommended to log the debug outputs on a file since they do impact CPU performance. Use caution before you run a debug directly on the CLI.

Debugs are usually run only when you have identified a problem to be a single process and would like to check how this process behaves in real-time with real traffic in the network. You need to enable a debug feature based on the user account privileges defined.

Just like event-histories, you can run debugs for every process on a Nexus device such as CDP/STP, UDLD, LACP/OSPF, EIGRP/BGP, and so on.

This is how you would typically run a debug for a process. Every process has multiple options, so use **?** to check for various options available under a process.

<#root>

```
Nexus93180# debug <Process> ?
```

```
Nexus93180# debug spanning-tree ?
  all      Configure all debug flags of stp
  bpdu_rx  Configure debugging of stp bpdu rx
  bpdu_tx  Configure debugging of stp bpdu tx
  error    Configure debugging of stp error
  event    Configure debugging of Events
  ha       Configure debugging of stp HA
```

```
  mcs      Configure debugging of stp MCS
  mstp     Configure debugging of MSTP
  pss      Configure debugging of PSS
  rstp     Configure debugging of RSTP
  sps      Configure debugging of Set Port state batching
  timer    Configure debugging of stp Timer events
  trace    Configure debugging of stp trace
  warning  Configure debugging of stp warning

Nexus93180#
```

```
debug ip ospf ?

  adjacency               Adjacency events
  all                     All OSPF debugging
  database                OSPF LSDB changes
  database-timers         OSPF LSDB timers
  events                  OSPF related events
  flooding                LSA flooding
  graceful-restart        OSPF graceful restart related debugs
  ha                      OSPF HA related events
  hello                   Hello packets and DR elections
  lsa-generation          Local OSPF LSA generation
  lsa-throttling          Local OSPF LSA throttling
  mpls                    OSPF MPLS
  objectstore             Objectstore Events
  packets                 OSPF packets
  policy                  OSPF RPM policy debug information
  redist                  OSPF redistribution
  retransmission          OSPF retransmission events
  rib                     Sending routes to the URIB
  segrt                   Segment Routing Events
  snmp                    SNMP traps and request-response related events
  spf                     SPF calculations
  spf-trigger             Show SPF triggers
```

GOLD

Generic OnLine Diagnostics (GOLD), as the name suggests, these tests are generally used as a system health check and are used to check or verify the hardware in question. There are various online tests that are carried out and based on the platform in use, some of these tests are disruptive while some are non-disruptive. These online tests can be categorized as follows:

- Boot-Up Diagnostics: These tests are the tests that run when the device is booting up. They also check for connectivity between the supervisor and the modules which includes connectivity between data and control plane for all ASICs. Tests such as ManagementPortLoopback and EOBCLoopback are disruptive while tests for OBFL and USB are non-disruptive.
- Run-Time or Health Monitoring Diagnostics: These tests provide information about the health of the device. These tests are non-disruptive and run in the background to ensure stability of hardware. You can enable/disable these tests as needed or for troubleshooting purposes.
- On-demand Diagnostics: All the tests mentioned can be re-run on-demand basis in order to localize an issue.

You can check for the various types of online tests available for your switch with this command:

```
Nexus93180(config)# show diagnostic content module all
Diagnostics test suite attributes:
```

```
B/C/* - Bypass bootup level test / Complete bootup level test / NA
P/*   - Per port test / NA
M/S/* - Only applicable to active / standby unit / NA
D/N/* - Disruptive test / Non-disruptive test / NA
H/O/* - Always enabled monitoring test / Conditionally enabled test / NA
F/*   - Fixed monitoring interval test / NA
X/*   - Not a health monitoring test / NA
E/*   - Sup to line card test / NA
L/*   - Exclusively run this test / NA
T/*   - Not an ondemand test / NA
A/I/* - Monitoring is active / Monitoring is inactive / NA

Module 1: 48x10/25G + 6x40/100G Ethernet Module (Active)

                                                    Testing Interval
ID     Name                                Attributes     (hh:mm:ss)
____   _____________________________       ___________    _____________
 1)    USB--------------------------->     C**N**X**T*     -NA-
 2)    NVRAM------------------------->     ***N******A     00:05:00
 3)    RealTimeClock----------------->     ***N******A     00:05:00
 4)    PrimaryBootROM---------------->     ***N******A     00:30:00
 5)    SecondaryBootROM-------------->     ***N******A     00:30:00
 6)    BootFlash--------------------->     ***N******A     00:30:00
 7)    SystemMgmtBus----------------->     **MN******A     00:00:30
 8)    OBFL-------------------------->     C**N**X**T*     -NA-
 9)    ACT2-------------------------->     ***N******A     00:30:00
10)    Console----------------------->     ***N******A     00:00:30
11)    FpgaRegTest------------------->     ***N******A     00:00:30
12)    Mce--------------------------->     ***N******A     01:00:00
13)    AsicMemory-------------------->     C**D**X**T*     -NA-
14)    Pcie-------------------------->     C**N**X**T*     -NA-
15)    PortLoopback------------------>     *P*N**XE***     -NA-
16)    L2ACLRedirect----------------->     *P*N***E**A     00:01:00
17)    BootupPortLoopback------------>     CP*N**XE*T*     -NA-
```

To display what each of the 17 mentioned tests do, you can use this command:

<#root>

```
Nexus93180(config)#
```

show diagnostic description module 1 test all

```
USB :
        A bootup test that checks the USB controller initialization
        on the module.

NVRAM :
        A health monitoring test, enabled by default that checks the
        sanity of the NVRAM device on the module.

RealTimeClock :
        A health monitoring test, enabled by default that verifies
        the real time clock on the module.

PrimaryBootROM :
        A health monitoring  test that verifies the primary BootROM
        on the module.

SecondaryBootROM :
```

```
        A health monitoring  test that verifies the secondary BootROM
        on the module.

BootFlash :
        A Health monitoring test, enabled by default, that verifies
        access to the internal compactflash devices.

SystemMgmtBus :
        A Health monitoring test, enabled by default, that verifies
        the standby System Bus.

OBFL :
        A bootup test that checks the onboard flash used for failure
        logging (OBFL) device initialization on the module.

ACT2 :
        A Health monitoring test, enabled by default, that verifies
        access to the ACT2 device.

Console :
        A health monitoring test,enabled by default that checks health
        of console device.

FpgaRegTest :
        A health monitoring test,enabled by default that checks read/write
        access to FPGA scratch registers on the module.

Mce :
        A Health monitoring test, enabled by default, that check for
        machine errors on sup.

AsicMemory :
        A bootup test that checks the asic memory.

Pcie :
        A bootup test that tests pcie bus of the module

PortLoopback :
        A health monitoring test that tests the packet path from
        the Supervisor card to the physical port in ADMIN DOWN state
        on Linecards.

L2ACLRedirect :
        A health monitoring test, enabled by default, that does a
        non disruptive loopback for TAHOE asics to check the ACL Sup
        redirect with the CPU port.

BootupPortLoopback :
        A Bootup test that tests the packet path from the Supervisor
        card to all of the physical ports at boot time.
```

EEM

Embedded Event Manager (EEM) is a powerful tool that allows you to program your device to perform specific tasks in case a certain event happens. It monitors various events on the device and then takes necessary action to troubleshoot the issue and possibly recover. EEM consists of three major components, each of which is briefly described here:

- Event Statement: These are the events that you want to monitor and want Nexus to perform a certain

action such as do a workaround or simply notify an SNMP server or display a CLI log, and so on.

- Action Statements: These would be the steps that EEM would take once an event is triggered. These actions could be simply to disable an interface or execute some show commands and copy outputs to a file on ftp server, send an e-mail, and so on.
- Policies: It is basically an event in combination with one or more action statements that you can configure on the supervisor via CLI or a bash script. You can also invoke EEM with a python script. Once the policy is defined on the supervisor, it then pushes the policy to the relevant module.

For details about EEM, refer to the link Cisco Nexus 9000 Series NX-OS System Management Configuration Guide, Release 9.2(x) - Configuring the Embedded Event Manager [Cisco Nexus 9000 Series Switches] - Cisco.