

Troubleshoot the Corrupted Ethernet Packet on Cisco Nexus 9000

Contents

[Introduction](#)

[Background Information](#)

[How is a Packet Processed by a Switch](#)

[Padding Modified with Tagged VLANs when Traffic traverses N9K](#)

[Solution](#)

Introduction

This document describes how to troubleshoot the corrupted ethernet packet on Cisco Nexus 9000 when a padding information is corrupted or malformed.

Background Information

The minimal size of an Ethernet frame is 64 bytes, no matter VLAN tag is present there or not.

The minimal Ethernet payload size is:

- 46 bytes if the VLAN tag is absent.
- 42 bytes if the VLAN tag is present.

You can verify this fact:

- On Wikipedia, section **Payload**: https://en.wikipedia.org/wiki/Ethernet_frame
- On the IEEE 802.3 standard (http://people.ee.duke.edu/~mbrooke/EE164.02/Spring_2004/group_2/index_files/8023.pdf), where the MAC frame format (without VLAN) is defined in section 3.1.1, page 39, and the elements of a tagged MAC frame is defined on page 43 section 3.5.

The minimal size of an ethernet packet is 64 bytes, no matter VLAN header is present there or not. The server is allowed to send a 64 bytes long packet that contains a VLAN, which you should accept and process correctly.

Note: This behaviour is correctly handled by a catalyst 4500x not by Nexus 9k.

How is a Packet Processed by a Switch

Step 1. Receive a **VALID** 64 bytes Ethernet frame.

Step 2. Remove the Frame Check Sequence (FCS), so the packet becomes 60 bytes long.

Step 3. Remove the VLAN tag, so the packet becomes 56 bytes long.

Step 4. Add padding to make the packet 60 bytes long.

Step 5. It adds the FCS, making the packet 64 bytes long.

Padding should not get modified when a packet goes through cut-through switch.

Padding Modified with Tagged VLANs when Traffic traverses N9K

Instead of padding with zeroes, the packet is padded with garbage characters, in most of the cases it has no impact because checksums are not modified and so nobody uses these data. However, if customers have a special usage and need to recompute the checksums, these garbage data leads to corruption of checksums in the end (other appliances, like NAT/load-balancers might see the issue too).

Device is a N9K 93120TX (was initially detected on a 9372TX though), version is latest NXOS 7.0(3)I2(2a).

Use Linux hosts with directly connected hardware to the N9K (no virtualization of any kind) here (1000base-T links).

Use this configuration:

```
interface Ethernet1/59
    switchport mode trunk
!
interface Ethernet1/60
    switchport mode trunk
```

linux configurations:

```
inet 10.2.1.1/24 brd 10.2.1.255 scope global eth1 <= native vlan
inet 10.1.1.1/24 brd 10.1.1.255 scope global eth1.100 <= tagged vlan 100
```

or

Just connect the windows host and send the tagged frames, they should trigger the problem. Moreover, ensure that the Network Interface Card (NIC) has the ability to tag the packet.

Switch adds the non-zero padding to the frames that passes through.

For Example: Host — [Trunk] N9K [Trunk] — Host

You can use netcat to send and receive the packets.

As shown in the image, it sends Side (VLAN 100 tagged), port e1/59 on the switch.

```
6: eth1.100@eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc noqueue state UP group default
link/ether 44:a8:42:2c:5f:c4 brd ff:ff:ff:ff:ff:ff
inet 10.1.1.1/24 brd 10.1.1.255 scope global eth1.100
    valid_lft forever preferred_lft forever
inet6 fe80::46a8:42ff:fe2c:5fc4/64 scope link
    valid_lft forever preferred_lft forever
```

```
root@s35-c2-0:~# nc 10.1.1.2 3002 -u
a
^C
root@s35-c2-0:~#
```

It receives Side (VLAN 100 tagged), port e1/60 on the switch, as shown in the image:

```
7: eth1.100@eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc noqueue state UP group default
link/ether 44:a8:42:2c:63:d1 brd ff:ff:ff:ff:ff:ff
inet 10.1.1.2/24 brd 10.1.1.255 scope global eth1.100
    valid_lft forever preferred_lft forever
inet6 fe80::46a8:42ff:fe2c:63d1/64 scope link
    valid_lft forever preferred_lft forever
```

```
root@s35-c2:~# nc -l -u -p 3002
a
^C
root@s35-c2:~#
```

As shown in the image, the packet is transmitted.

```
root@s35-c2-0:~# tcpdump -i eth1.100 -nvex
tcpdump: listening on eth1.100, link-type EN10MB (Ethernet), capture size 65535 bytes
10:42:20.953994 44:a8:42:2c:5f:c4 > 44:a8:42:2c:63:d1, ethertype IPv4 (0x0800), length 44: (tos 0x0, ttl 64, id 64283, offset 0, flags [DF], proto UDP (17), length 30)
    10.1.1.1.41675 > 10.1.1.2.3002: UDP, length 2
    0x0000: 4500 001e fb1b 4000 4011 29af 0a01 0101
    0x0010: 0a01 0102 a2cb 0bba 000a 1620 610a
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@s35-c2-0:~#
```

The packet is received, as shown in the image:

```
10:43:12.665897 44:a8:42:2c:5f:c4 > 44:a8:42:2c:63:d1, ethertype IPv4 (0x0800), length 60: (tos 0x0, ttl 64, id 64283, offset 0, flags [DF], proto UDP (17), length 30)
    10.1.1.1.41675 > 10.1.1.2.3002: UDP, length 2
    0x0000: 4500 001e fb1b 4000 4011 29af 0a01 0101
    0x0010: 0a01 0102 a2cb 0bba 000a dc45 610a 0000
    0x0020: 0000 0000 0000 0000 0000 7562 710e
^C
7 packets captured
7 packets received by filter
0 packets dropped by kernel
root@s35-c2:~#
```

As shown in the image, the wrong padding is highlighted.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.1.1.1	10.1.1.2	UDP	60	Source port: 40849 Destination port: 3002

▶ Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)

▼ Ethernet II, Src: Dell_2c:5f:c4 (44:a8:42:2c:5f:c4), Dst: Dell_2c:63:d1 (44:a8:42:2c:63:d1)

- ▶ Destination: Dell_2c:63:d1 (44:a8:42:2c:63:d1)
- ▶ Source: Dell_2c:5f:c4 (44:a8:42:2c:5f:c4)
- Type: IP (0x0800)

Padding: 000000000000000000000000f1b7bc5c

▼ Internet Protocol Version 4, Src: 10.1.1.1 (10.1.1.1), Dst: 10.1.1.2 (10.1.1.2)

- 0100 = Version: 4
- 0101 = Header Length: 20 bytes
- ▶ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
- Total Length: 30
- Identification: 0xfbd (64285)
- ▶ Flags: 0x02 (Don't Fragment)
- Fragment offset: 0
- Time to live: 64
- Protocol: UDP (17)
- ▶ Header checksum: 0x29ad [validation disabled]
- Source: 10.1.1.1 (10.1.1.1)
- Destination: 10.1.1.2 (10.1.1.2)
- [Source GeoIP: Unknown]
- [Destination GeoIP: Unknown]

▼ User Datagram Protocol, Src Port: 40849 (40849), Dst Port: 3002 (3002)

- Source Port: 40849 (40849)
- Destination Port: 3002 (3002)
- Length: 10
- ▼ Checksum: 0xdd7f [validation disabled]
- [Good Checksum: False]
- [Bad Checksum: False]
- [Stream index: 0]

▼ Data (2 bytes)

- Data: 610a
- [Length: 2]

0000	44 a8 42 2c 63 d1 44 a8 42 2c 5f c4 08 00 45 00	D,B,c,D, B,....E.
0010	00 1e fb 1d 40 00 40 11 29 ad 0a 01 01 01 0a 01	...@.@.).....
0020	01 02 9f 91 0b ba 00 0a dd 7f 61 0a 00 00 00 00a.....
0030	00 00 00 00 00 00 00 00 f1 b7 bc 5c\

This is also displayed with a packet analyzer (in another packet, the data is different from the previous screenshots but test and bug is identical),

Solution

The workaround is to disable [buffer-boost](#) on interface where we have this server connected.

```
C9396PX-1(config)# int et 1/7
C9396PX-1(config-if)# no buffer-boost
```

Related defect:

[CSCva46849](#) 60 Byte Frame with dot1q Header L2 switching on N9k