Troubleshoot Catalyst 3850 Series Switch High CPU Usage

Contents

Introduction

Background Information

Case Study: Address Resolution Protocol (ARP) Interrupts

- Step 1. Identify the Process that Consumes CPU Cycles
- Step 2. Determine the CPU Queue that Causes the High CPU Usage Condition
- Step 3. Dump the Packet Sent to the CPU
- Step 4. Use FED Tracing
- Sample Embedded Event Manager (EEM) Script for the Cisco Catalyst 3850 Series Switch

Cisco IOS XE 16.x or Later Releases

Related Information

Introduction

This document describes how to troubleshoot CPU usage concerns, primarily due to interrupts, on the new Cisco IOS® XE platform.

Background Information

It is important to understand how Cisco IOS® XE is built. With Cisco IOS® XE, Cisco has moved to a Linux kernel and all of the subsystems have been broken down into processes. All of the subsystems that were inside Cisco IOS before such as the modules drivers, High Availability (HA), and so on now run as software processes within the Linux Operating System (OS). Cisco IOS itself runs as a daemon within the Linux OS (IOSd). Cisco IOS® XE retains not only the same look and feel of the classic Cisco IOS®, but also its operation, support, and management.

Additionally, the document introduces several new commands on this platform that are integral in order to troubleshoot CPU usage problems.

Here are some useful definitions:

- Forwarding Engine Driver (FED): This is the heart of the Cisco Catalyst 3850 Series Switch and is responsible for all hardware programming/forwarding.
- Cisco IOSd: This is the Cisco IOS® daemon that runs on the Linux kernel. It is run as a software process within the kernel.
- Cisco IOS Packet Delivery System (PDS): This is the architecture and process of how packets are delivered to and from various subsystems. As an example, it controls how packets are delivered from the FED to the IOSd and the other way around.
- Handle: A handle can be thought of as a pointer. It is a means to discover more detailed information about specific variables that are used in the outputs that the box produces. This is similar to the

concept of Local Target Logic (LTL) indices on the Cisco Catalyst 6500 Series Switch.

Case Study: Address Resolution Protocol (ARP) Interrupts

The troubleshoot and verification process in this section can be broadly used for high CPU usage due to interrupts.

Step 1. Identify the Process that Consumes CPU Cycles

The **show process cpu** command naturally displays how the CPU currently looks. Notice that the Cisco Catalyst 3850 Series Switch uses four cores, and you see the CPU usage listed for all four cores:

```
<#root>

3850-2#

show processes cpu sorted | exclude 0.0


Core 0: CPU utilization for five seconds: 53%; one minute: 39%;  five minutes: 41%
Core 1: CPU utilization for five seconds: 43%; one minute: 57%;  five minutes: 54%
Core 2: CPU utilization for five seconds: 95%; one minute: 60%;  five minutes: 58%
Core 3: CPU utilization for five seconds: 32%; one minute: 31%;  five minutes: 29%
PID    Runtime(ms) Invoked  uSecs  5Sec     1Min     5Min     TTY   Process
8525   472560      2345554  7525   31.37    30.84    30.83    0     iosd
5661   2157452     9234031  698    13.17    12.56    12.54    1088  fed
6206   19630       74895    262    1.83     0.43     0.10     0     eicored
6197   725760      11967089 60     1.41     1.38     1.47     0     pdsd
```

From the output, it is clear that the Cisco IOS® daemon consumes a major portion of the CPU along with the FED, which is the heart of this box. When CPU usage is high due to interrupts, you see that Cisco IOSd and FED use a major portion of the CPU, and these subprocesses (or a subset of these) use the CPU:

- FED Punject TX
- FED Punject RX
- FED Punject replenish
- FED Punject TX complete

You can zoom into any of these processes with the **show process cpu detailed <process>** command. Since Cisco IOSd is responsible for the majority of the CPU usage, here is a closer look into that.

```
<#root>

3850-2#

show processes cpu detailed process iosd sorted | ex 0.0


Core 0: CPU utilization for five seconds: 36%; one minute: 39%; five minutes: 40%
Core 1: CPU utilization for five seconds: 73%; one minute: 52%; five minutes: 53%
Core 2: CPU utilization for five seconds: 22%; one minute: 56%; five minutes: 58%
Core 3: CPU utilization for five seconds: 46%; one minute: 40%; five minutes: 31%
PID    T C  TID   Runtime(ms)Invoked uSecs 5Sec   1Min   5Min  TTY  Process
                                          (%)    (%)    (%)
8525   L          556160     2356540 7526  30.42   30.77  30.83  0   iosd
```

```
8525   L 1  8525  712558     284117  0     23.14   23.33  23.38  0   iosd
59     I          1115452    4168181 0     42.22   39.55  39.33  0   ARP Snoop
198    I          3442960    4168186 0     25.33   24.22  24.77  0   IP Host Track Proce
30     I          3802130    4168183 0     24.66   27.88  27.66  0   ARP Input
283    I          574800     3225649 0     4.33    4.00   4.11   0   DAI Packet Process

3850-2#
```

show processes cpu detailed process fed sorted | ex 0.0

```
Core 0: CPU utilization for five seconds: 45%; one minute: 44%; five minutes: 44%
Core 1: CPU utilization for five seconds: 38%; one minute: 44%; five minutes: 45%
Core 2: CPU utilization for five seconds: 42%; one minute: 41%; five minutes: 40%
Core 3: CPU utilization for five seconds: 32%; one minute: 30%; five minutes: 31%
PID    T C  TID   Runtime(ms)Invoked uSecs 5Sec   1Min   5Min   TTY  Process
                                          (%)    (%)    (%)
5638   L          612840     1143306 536   13.22  12.90  12.93  1088  fed
5638   L 3  8998  396500     602433  0     9.87   9.63   9.61   0     PunjectTx
5638   L 3  8997  159890     66051   0     2.70   2.70   2.74   0     PunjectRx
```

The output (Cisco IOSd CPU output) shows that ARP Snoop, IP Host Track Process, and ARP Input are high. This is commonly seen when the CPU is interrupted due to ARP packets.

Step 2. Determine the CPU Queue that Causes the High CPU Usage Condition

The Cisco Catalyst 3850 Series Switch has a number of queues that cater to different types of packets (the FED maintains 32 RX CPU queues, which are queues that go directly to the CPU). It is important to monitor these queues in order to discover which packets are punted to the CPU and which are processed by the Cisco IOSd. These queues are per ASIC.

Note: There are two ASICs: 0 and 1. Ports 1 through 24 belong to ASIC 0.

In order to look at the queues, enter the **show platform punt statistics port-asic <port-asic>cpuq <queue> direction** <rx|tx> command.

In the **show platform punt statistics port-asic 0 cpuq -1 direction rx** command, the -1 argument lists all of the queues. Therefore, this command lists all receive queues for Port-ASIC 0.

Now, you must identify which queue pushes a large number of packets at a high rate. In this example, an examination of the queues revealed this culprit:

```
<snip>
RX (ASIC2CPU) Stats (asic 0 qn 16 lqn 16):
RXQ 16: CPU_Q_PROTO_SNOOPING
---------------------------------------
Packets received from ASIC     : 79099152
Send to IOSd total attempts    : 79099152
Send to IOSd failed count      : 1240331
RX suspend count               : 1240331
RX unsuspend count             : 1240330
RX unsuspend send count        : 1240330
RX unsuspend send failed count : 0
RX dropped count               : 0
RX conversion failure dropped  : 0
```

```
RX pkt_hdr allocation failure  : 0
RX INTACK count                : 0
RX packets dq'd after intack   : 0
Active RxQ event               : 9906280
RX spurious interrupt          : 0
<snip>
```

The queue number is 16 and the queue name is CPU_Q_PROTO_SNOOPING.

Another way to discover the culprit queue is to enter the **show platform punt client** command:

<#root>

3850-2#

show platform punt client

```
  tag      buffer        jumbo    fallback     packets   received   failures
                                              alloc   free  bytes    conv  buf
 27       0/1024/2048     0/5       0/5         0     0          0     0     0
 65536    0/1024/1600     0/0       0/512       0     0          0     0     0
 65537    0/ 512/1600     0/0       0/512    1530  1530     244061     0     0
 65538    0/   5/5        0/0       0/5         0     0          0     0     0
 65539    0/2048/1600     0/16      0/512       0     0          0     0     0
 65540    0/ 128/1600     0/8       0/0         0     0          0     0     0
 65541    0/ 128/1600     0/16      0/32        0     0          0     0     0
 65542    0/ 768/1600     0/4       0/0         0     0          0     0     0
 65544    0/  96/1600     0/4       0/0         0     0          0     0     0
 65545    0/  96/1600     0/8       0/32        0     0          0     0     0
 65546    0/ 512/1600     0/32      0/512       0     0          0     0     0
 65547    0/  96/1600     0/8       0/32        0     0          0     0     0
 65548    0/ 512/1600     0/32      0/256       0     0          0     0     0
 65551    0/ 512/1600     0/0       0/256       0     0          0     0     0
 65556    0/  16/1600     0/4       0/0         0     0          0     0     0
 65557    0/  16/1600     0/4       0/0         0     0          0     0     0
 65558    0/  16/1600     0/4       0/0         0     0          0     0     0
 65559    0/  16/1600     0/4       0/0         0     0          0     0     0
 65560    0/  16/1600     0/4       0/0         0     0          0     0     0
s65561  421/ 512/1600     0/0       0/128  79565859 131644697  478984244     0 37467
 65563    0/ 512/1600     0/16      0/256       0     0          0     0     0
 65564    0/ 512/1600     0/16      0/256       0     0          0     0     0
 65565    0/ 512/1600     0/16      0/256       0     0          0     0     0
 65566    0/ 512/1600     0/16      0/256       0     0          0     0     0
 65581    0/   1/1        0/0       0/0         0     0          0     0     0
 131071   0/  96/1600     0/4       0/0         0     0          0     0     0
fallback pool: 98/1500/1600
jumbo pool:    0/128/9300
```

Determine the tag for which the most packets have been allocated. In this example, it is 65561.

Then, enter this command:

<#root>

3850-2#

```
show pds tag all | in Active|Tags|65561
```

```
   Active   Client Client
     Tags   Handle Name                  TDA       SDA         FDA   TBufD       TBytD
    65561  7296672 Punt Rx Proto Snoop  79821397  79821397    0  79821397   494316524
```

This output shows that the queue is Rx Proto Snoop.

The **s** before the 65561 in the output of the **show platform punt client** command means that the FED handle is suspended and overwhelmed by the number of incoming packets. If the **s** does not vanish, it means the queue is stuck permanently.

Step 3. Dump the Packet Sent to the CPU

In the results of the **show pds tag all** command, notice a handle, 7296672, is reported next to the Punt Rx Proto Snoop.

Use this handle in the **show pds client <handle> packet last sink** command. Notice that you must enable debug pds pktbuf-last before you use the command. Otherwise, you encounter this error:

<#root>

3850-2#

show pds client 7296672 packet last sink

```
% switch-2:pdsd:This command works in debug mode only. Enable debug using
"debug pds pktbuf-last" command
```

With the debug enabled, you see this output:

<#root>

3850-2#

show pds client 7296672 packet last sink

```
Dumping Packet(54528) # 0 of Length 60
----------------------------------------
Meta-data
0000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0010 00 00 16 1d 00 00 00 00 00 00 00 00 55 5a 57 f0  ............UZW.
0020 00 00 00 00 fd 01 10 df 00 5b 70 00 00 10 43 00  .........[p...C.
0030 00 10 43 00 00 41 fd 00 00 41 fd 00 00 00 00 00  ..C..A...A......
0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0050 00 00 00 3c 00 00 00 00 00 01 00 19 00 00 00 00  ...<............
0060 01 01 b6 80 00 00 00 4f 00 00 00 00 00 00 00 00  .......O........
0070 01 04 d8 80 00 00 00 33 00 00 00 00 00 00 00 00  .......3........
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0090 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00a0 00 00 00 00 00 00 00 02 00 00 00 00 00 00 00 00  ................
Data
0000 ff ff ff ff ff ff aa bb cc dd 00 00 08 06 00 01  ................
```

```
0010 08 00 06 04 00 01 aa bb cc dd 00 00 c0 a8 01 0a  ................
0020 ff ff ff ff ff ff c0 a8 01 14 00 01 02 03 04 05  ................
0030 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11              ............
```

This command dumps the last packet received by the sink, which is Cisco IOSd in this example. This shows that it dumps the header and it can be decoded with Terminal-based Wireshark (TShark). The meta-data is for internal use by the system, but the Data output provides actual packet information. The meta-data, however, remains extremely useful.

Notice the line that starts with 0070. Use the first 16 bits after that as shown here:

<#root>

```
3850-2#
```

show platform port-asic ifm iif-id 0x0104d88000000033

```
Interface Table
Interface IIF-ID        : 0x0104d88000000033
Interface Name          : Gi2/0/20
Interface Block Pointer : 0x514d2f70
Interface State         : READY
Interface Stauts        : IFM-ADD-RCVD, FFM-ADD-RCVD
Interface Ref-Cnt       : 6
Interface Epoch         : 0
Interface Type          : ETHER
        Port Type         : SWITCH PORT
        Port Location     : LOCAL
Slot              : 2
        Unit              : 20
        Slot Unit         : 20
        Acitve            : Y
        SNMP IF Index     : 22
        GPN               : 84
        EC Channel        : 0
        EC Index          : 0
        ASIC

              :

0
        ASIC Port         : 14
        Port LE Handle    : 0x514cd990
Non Zero Feature Ref Counts
        FID : 48(AL_FID_L2_PM), Ref Count : 1
        FID : 77(AL_FID_STATS), Ref Count : 1
        FID : 51(AL_FID_L2_MATM), Ref Count : 1
        FID : 13(AL_FID_SC), Ref Count : 1
        FID : 26(AL_FID_QOS), Ref Count : 1
Sub block information
        FID : 48(AL_FID_L2_PM), Private Data &colon; 0x54072618
        FID : 26(AL_FID_QOS), Private Data &colon; 0x514d31b8
```

The culprit interface is identified here. Gig2/0/20 is where there is a traffic generator that pumps ARP traffic. If you shut this down, then it would resolve the problem and minimize the CPU usage.

Step 4. Use FED Tracing

The only drawback with the method discussed in the last section is that it only dumps the last packet that goes into the sink, and it cannot be the culprit.

A better way to troubleshoot this would be to use a feature called FED tracing. Tracing is a packet capture method (using various filters) that are pushed by the FED to the CPU. FED tracing is not as simple as the Netdr feature on the Cisco Catalyst 6500 Series Switch, however.

Here the process is broken into steps:

1. Enable detail tracking. By default, event tracing is on. You must enable detail tracing in order to capture the actual packets:

   ```
   <#root>

   3850-2#

   set trace control fed-punject-detail enable
   ```

2. Fine-tune the capture buffer. Determine how deep your buffers are for detail tracing and increase as needed:

   ```
   <#root>

   3850-2#

   show mgmt-infra trace settings fed-punject-detail


   One shot Trace Settings:

   Buffer Name: fed-punject-detail
   Default Size: 32768
   Current Size: 32768
   Traces Dropped due to internal error: No
   Total Entries Written: 0
   One shot mode: No
   One shot and full: No
   Disabled: False
   ```

   You can change the buffer size with this command:

   ```
   <#root>

   3850-2#

   set trace control fed-punject-detail buffer-size <buffer size>
   ```

   The values available to you are:

```
<#root>

3850-2#

set trace control fed-punject-detail buffer-size ?


  <8192-67108864>  The new desired buffer size, in bytes
  default          Reset trace buffer size to default
```

3. Add capture filters. You now need to add various filters for the capture. You can add different filters and either choose to match all or match any of those for your capture.

   Filters are added with this command:

```
<#root>

3850-2#

set trace fed-punject-detail direction rx filter_add <filter>
```

   These options are currently available:

```
<#root>

3850-2#

set trace fed-punject-detail direction rx filter_add ?


  cpu-queue  rxq 0..31
  field      field
  offset     offset
```

   Now you must link things together. Remember the culprit queue that was identified in Step 2 of this troubleshoot process? Since queue 16 is the queue that pushes a large number of packets towards the CPU, it makes sense to trace this queue and see what packets are punted to the CPU by it.

   You can choose to trace any queue with this command:

```
<#root>

 3850-2#

set trace fed-punject-detail direction rx filter_add cpu-queue


<start queue>


 <end queue>
```

Here is the command for this example:

```
<#root>

3850-2#

set trace fed-punject-detail direction rx filter_add cpu-queue 16 16
```

You must choose either a match all or a match any for your filters and then enable the trace:

```
<#root>

3850-2#

set trace fed-punject-detail direction rx match_all


3850-2#

set trace fed-punject-detail direction rx filter_enable
```

4. Display filtered packets. You can display the packets captured with the **show mgmt-infra trace messages fed-punject-detail** command:

```
<#root>

3850-2#

show mgmt-infra trace messages fed-punject-detail


[11/25/13 07:05:53.814 UTC 2eb0c9 5661]
00 00 00 00 00 4e 00 40 07 00 02 08 00 00 51 3b
00 00 00 00 00 01 00 00 03 00 00 00 00 00 00 01
00 00 00 00 20 00 00 0e 00 00 00 00 00 01 00 74
00 00 00 04 00 54 41 02 00 00 00 00 00 00 00 00

[11/25/13 07:05:53.814 UTC 2eb0ca 5661]
ff ff ff ff ff ff aa bb cc dd 00 00 08 06 00 01
08 00 06 04 00 01 aa bb cc dd 00 00 c0 a8 01 0a
ff ff ff ff ff ff c0 a8 01 14 00 01 02 03 04 05
06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 f6 b9 10 32
[11/25/13 07:05:53.814 UTC 2eb0cb 5661] Frame descriptors:
[11/25/13 07:05:53.814 UTC 2eb0cc 5661]
=========
fdFormat=0x4    systemTtl=0xe
loadBalHash1=0x8        loadBalHash2=0x8
spanSessionMap=0x0      forwardingMode=0x0
destModIndex=0x0        skipIdIndex=0x4
srcGpn=0x54     qosLabel=0x41
srcCos=0x0      ingressTranslatedVlan=0x3
bpdu=0x0        spanHistory=0x0
sgt=0x0 fpeFirstHeaderType=0x0
srcVlan=0x1     rcpServiceId=0x2
```

```
wccpSkip=0x0    srcPortLeIndex=0xe
cryptoProtocol=0x0      debugTagId=0x0
vrfId=0x0       saIndex=0x0
pendingAfdLabel=0x0     destClient=0x1
appId=0x0       finalStationIndex=0x74
decryptSuccess=0x0      encryptSuccess=0x0
rcpMiscResults=0x0      stackedFdPresent=0x0
spanDirection=0x0       egressRedirect=0x0
redirectIndex=0x0       exceptionLabel=0x0
destGpn=0x0     inlineFd=0x0
suppressRefPtrUpdate=0x0        suppressRewriteSideEfects=0x0
cmi2=0x0        currentRi=0x1
currentDi=0x513b        dropIpUnreachable=0x0
srcZoneId=0x0   srcAsicId=0x0
originalDi=0x0  originalRi=0x0
srcL3IfIndex=0x2        dstL3IfIndex=0x0
dstVlan=0x0     frameLength=0x40
fdCrc=0x7       tunnelSpokeId=0x0

=========
[11/25/13 07:05:53.814 UTC 2eb0cd 5661]
[11/25/13 07:05:53.814 UTC 2eb0ce 5661] PUNT PATH (fed_punject_rx_process_packet:
830):RX: Q: 16, Tag: 65561
[11/25/13 07:05:53.814 UTC 2eb0cf 5661] PUNT PATH (fed_punject_get_physical_iif:
579):RX: Physical IIF-id 0x104d88000000033
[11/25/13 07:05:53.814 UTC 2eb0d0 5661] PUNT PATH (fed_punject_get_src_l3if_index:
434):RX: L3 IIF-id 0x101b6800000004f
[11/25/13 07:05:53.814 UTC 2eb0d1 5661] PUNT PATH (fed_punject_fd_2_pds_md:478):
RX: l2_logical_if = 0x0
[11/25/13 07:05:53.814 UTC 2eb0d2 5661] PUNT PATH (fed_punject_get_source_cos:638):
RX: Source Cos 0
[11/25/13 07:05:53.814 UTC 2eb0d3 5661] PUNT PATH (fed_punject_get_vrf_id:653):
RX: VRF-id 0
[11/25/13 07:05:53.814 UTC 2eb0d4 5661] PUNT PATH (fed_punject_get_src_zoneid:667):
RX: Zone-id 0
[11/25/13 07:05:53.814 UTC 2eb0d5 5661] PUNT PATH (fed_punject_fd_2_pds_md:518):
RX: get_src_zoneid failed
[11/25/13 07:05:53.814 UTC 2eb0d6 5661] PUNT PATH (fed_punject_get_acl_log_direction:
695): RX: : Invalid CMI2
[11/25/13 07:05:53.814 UTC 2eb0d7 5661] PUNT PATH (fed_punject_fd_2_pds_md:541):RX:
 get_acl_log_direction failed
[11/25/13 07:05:53.814 UTC 2eb0d8 5661] PUNT PATH (fed_punject_get_acl_full_direction:
724):RX: DI 0x513b ACL Full Direction 1
[11/25/13 07:05:53.814 UTC 2eb0d9 5661] PUNT PATH (fed_punject_get_source_sgt:446):
RX: Source SGT 0
[11/25/13 07:05:53.814 UTC 2eb0da 5661] PUNT PATH (fed_punject_get_first_header_type:680):
RX: FirstHeaderType 0
[11/25/13 07:05:53.814 UTC 2eb0db 5661] PUNT PATH (fed_punject_rx_process_packet:916):
RX: fed_punject_pds_send packet 0x1f00 to IOSd with tag 65561
[11/25/13 07:05:53.814 UTC 2eb0dc 5661] PUNT PATH (fed_punject_rx_process_packet:744):
RX: **** RX packet 0x2360 on qn 16, len 128 ****
[11/25/13 07:05:53.814 UTC 2eb0dd 5661]
buf_no 0 buf_len 128

<snip>
```

This output provides plenty of information and can typically be enough to discover where the packets come from and what is contained in them.

The first part of the header dump is again the meta-data that is used by the system. The second part is

the actual packet:

```
ff ff ff ff ff ff - destination MAC address
aa bb cc dd 00 00 - source MAC address
```

You can choose to trace this source MAC address in order to discover the culprit port (once you have identified that this is the majority of the packets that are punted from queue 16; this output only shows one instance of the packet and the other output/packets are clipped).

However, there is a better way. Notice that logs that are present after the header information:

```
[11/25/13 07:05:53.814 UTC 2eb0ce 5661] PUNT PATH (fed_punject_rx_process_packet:
830):RX: Q: 16, Tag: 65561
[11/25/13 07:05:53.814 UTC 2eb0cf 5661] PUNT PATH (fed_punject_get_physical_iif:
579):RX: Physical IIF-id 0x104d88000000033
```

The first log clearly tells you from which queue and tag this packet comes. If you were not aware of the queue earlier, this is a easy way to identify which queue it was.

The second log is even more useful because it provides the physical Interface ID Factory (IIF)-ID for the source interface. The hex value is a handle that can be used in order to dump information about that port:

<#root>

```
3850-2#
```

`show platform port-asic ifm iif-id 0x0104d88000000033`

```
Interface Table
Interface IIF-ID        : 0x0104d88000000033
Interface Name          : Gi2/0/20
Interface Block Pointer : 0x514d2f70
Interface State         : READY
Interface Stauts        : IFM-ADD-RCVD, FFM-ADD-RCVD
Interface Ref-Cnt       : 6
Interface Epoch         : 0
Interface Type          : ETHER
        Port Type         : SWITCH PORT
        Port Location     : LOCAL
        Slot              : 2
        Unit              : 20
        Slot Unit         : 20
        Active            : Y
        SNMP IF Index     : 22
        GPN               : 84
        EC Channel        : 0
        EC Index          : 0
        ASIC              : 0
        ASIC Port         : 14
        Port LE Handle    : 0x514cd990
```

```
     Non Zero Feature Ref Counts
             FID : 48(AL_FID_L2_PM), Ref Count : 1
             FID : 77(AL_FID_STATS), Ref Count : 1
             FID : 51(AL_FID_L2_MATM), Ref Count : 1
             FID : 13(AL_FID_SC), Ref Count : 1
             FID : 26(AL_FID_QOS), Ref Count : 1
     Sub block information
             FID : 48(AL_FID_L2_PM), Private Data &colon; 0x54072618
             FID : 26(AL_FID_QOS), Private Data &colon; 0x514d31b8
```

You have once again identified the source interface and culprit.

Tracing is a powerful tool that is critical in order to troubleshoot high CPU usage problems. Tracing provides plenty of information in order to successfully resolve such a situation.

Sample Embedded Event Manager (EEM) Script for the Cisco Catalyst 3850 Series Switch

Use this command in order to trigger a log to be generated at a specific threshold:

```
process cpu threshold type total rising <CPU %> interval <interval in seconds>
switch <switch number>
```

The log generated with the command looks like this:

```
*Jan 13 00:03:00.271: %CPUMEM-5-RISING_THRESHOLD: 1 CPUMEMd[6300]:  Threshold: : 50, Total CPU Utilzati
```

The log generated provides this information:

- The total CPU utilization at the time of the trigger. This is identified by Total CPU Utilization(total/Intr) :50/0 in this example.
- Top processes - these are listed in the format of PID/CPU%. So, in this example, these are:

```
    8622/25 - 8622 is PID for IOSd and 25 implies that this process is using 25% CPU.
    5753/12 - 5733 is PID for FED and 12 implies that this process is using 12% CPU.
```

The EEM script is shown here:

```
event manager applet highcpu
event syslog pattern "%CPUMEM-5-RISING_THRESHOLD"
 action 0.1 syslog msg "high CPU detected"
 action 0.2 cli command "enable"
 action 0.3 cli command "show process cpu sorted | append nvram:<filename>.txt"
 action 0.4 cli command "show process cpu detailed process <process name|process ID>
 sorted | nvram:<filename>.txt"
```

```
action 0.5 cli command "show platform punt statistics port-asic 0 cpuq -1
direction rx | append nvram:<filename>.txt"
action 0.6 cli command "show platform punt statistics port-asic 1 cpuq -1
direction rx | append nvram:<filename>.txt"
action 0.7 cli command "conf t"
action 0.8 cli command "no event manager applet highcpu"
```

Note: The **process cpu threshold** command does not currently work in the 3.2.X train. Another point to remember is that this command looks at the average CPU utilization among the four cores and generates a log when this average reaches the percentage that has been defined in the command.

Cisco IOS XE 16.x or Later Releases

If you have Catalyst 3850 switches that run Cisco IOS® XE Software Release 16.x or later, see Troubleshoot High CPU Usage in Catalyst Switch Platforms Running IOS-XE 16.x.

Related Information

- What is Cisco IOS XE?
- Cisco Catalyst 3850 Switches - Data Sheets and Literature
- Cisco Technical Support & Downloads