

Catalyst 3850系列交換器高CPU使用率疑難排解

目錄

[簡介](#)

[背景資訊](#)

[案例研究：地址解析協定\(ARP\)中斷](#)

[第1步：確定消耗CPU週期的進程](#)

[第2步：確定導致CPU使用率過高的CPU隊列](#)

[第3步：轉儲傳送到CPU的資料包](#)

[第4步：使用FED跟蹤](#)

[Cisco Catalyst 3850系列交換機的嵌入式事件管理器\(EEM\)指令碼示例](#)

[Cisco IOS XE 16.x或更高版本](#)

[相關資訊](#)

簡介

本檔案介紹如何在新的Cisco IOS® XE平台上排除CPU使用問題（主要由於中斷）的故障。

背景資訊

瞭解Cisco IOS® XE的構建方式很重要。藉助Cisco IOS® XE，思科已遷移到Linux核心，並且所有子系統都已拆分為多個進程。以前在Cisco IOS中的所有子系統(如模組驅動程式、高可用性(HA)等)現在都作為Linux作業系統(OS)中的軟體進程運行。Cisco IOS本身在Linux OS(IOSd)中作為後台程式運行。Cisco IOS® XE不僅保留傳統Cisco IOS®的外觀和感覺，而且保留其操作、支援和管理。

此外，本文檔還介紹了此平台上用於解決CPU使用問題的幾個整合新命令。

以下是一些有用的定義：

- 轉發引擎驅動程式(FED)：這是Cisco Catalyst 3850系列交換機的核心，負責所有硬體程式設計/轉發。
- Cisco IOSd：這是在Linux核心上運行的Cisco IOS®守護程式。它在核心中作為軟體進程運行。
- 封包交付系統(PDS)：這是將封包傳送到各個子系統和從各個子系統傳送封包的架構和程式。例如，它控制資料包如何從FED傳送到IOSd，反之亦然。
- 控制代碼：可以將控制代碼視為指標。該工具用於查詢有關在該框生成的輸出中使用的特定變數的更詳細資訊。這類似於Cisco Catalyst 6500系列交換機上的本地目標邏輯(LTL)索引的概念。

案例研究：地址解析協定(ARP)中斷

本節中的故障排除和驗證過程可廣泛用於由於中斷而導致CPU使用率較高的情況。

第1步：確定消耗CPU週期的進程

show process cpu命令自然顯示CPU當前的顯示方式。請注意，Cisco Catalyst 3850系列交換機使用四個核心，並且您會看到所有四個核心的CPU使用率：

```
<#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
```

從輸出中，很明顯Cisco IOS®後台程式會與FED一起佔用大部分CPU，而FED是此框的核心。當CPU使用率因中斷而高時，您會看到Cisco IOSd和FED使用CPU的主要部分，並且這些子進程（或這些子進程的子集）使用CPU：

- FED Punject TX
- FED Punject RX
- FED Punject補充
- FED Punject TX完成

您可以使用show process cpu detailed <process>命令放大這些進程中的任何一個。由於Cisco IOSd負責大部分CPU使用，因此下面更深入瞭解一下這一點。

```
<#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

```

輸出 (Cisco IOSd CPU輸出) 顯示ARP監聽、IP主機跟蹤進程和ARP輸入為高。當CPU因ARP資料包而中斷時，通常會出現這種情況。

第2步：確定導致CPU使用率過高的CPU隊列

Cisco Catalyst 3850系列交換器具有許多佇列，可滿足不同型別的封包 (FED維護32個RX CPU佇列，這些佇列是直接前往CPU的佇列)。監控這些隊列很重要，以便發現哪些資料包被轉發到CPU以及哪些資料包由Cisco IOSd處理。這些隊列是按ASIC劃分的。

 註：有兩個ASIC:0和1。埠1到24屬於ASIC 0。

要檢視隊列，請輸入 `show platform punt statistics port-asic <port-asic>cpuq <queue>direction` 指令。

在 `show platform punt statistics port-asic 0 cpuq -1 direction rx` 命令中，-1引數列出了所有隊列。因此，此命令列出埠ASIC 0的所有接收隊列。

現在，您必須確定哪個隊列以高速推入大量資料包。在本例中，對隊列的檢查揭示了這一罪魁禍首：

```

<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>

```

隊列編號為16，隊列名稱為CPU_Q_PROTO_SNOOPING。

發現罪魁禍首隊列的另一種方法是輸入show platform punt client命令。

```
<#root>
```

```
3850-2#
```

```
show platform punt client
```

tag	buffer	jumbo	fallback	packets		received bytes	failures	
				alloc	free		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

```

確定為其分配了最多資料包的標籤。在本例中，它是65561。

然後輸入以下命令：

<#root>

3850-2#

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

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

此輸出顯示隊列是Rx Proto Snoop。

show platform punt client65561令輸出中的s before表示已掛起FED控制代碼，且傳入封包數量過多。如果s沒有消失，則表示隊列永久停滯不前。

第3步：轉儲傳送到CPU的資料包

在show pds tag all命令的結果中，請注意Punt Rx Proto Snoop旁邊會報告控制代碼7296672。

在show pds client <handle> packet last sink命令中使用此控制代碼。請注意，使用命令之前，必須啟用debug pds pktbuf-last。否則您會遇到以下錯誤：

<#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
```

啟用偵錯後，您會看到以下輸出：

<#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 .....
0010 00 00 16 1d 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 .....
0050 00 00 00 3c 00 00 00 00 00 01 00 19 00 00 00 ...<.....
0060 01 01 b6 80 00 00 00 4f 00 00 00 00 00 00 00 .....0.....
0070 01 04 d8 80 00 00 00 33 00 00 00 00 00 00 00 .....3.....
0080 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 .....
00a0 00 00 00 00 00 00 00 02 00 00 00 00 00 00 00 .....
Data
0000 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 c0 a8 01 14 00 01 02 03 04 05 .....
0030 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 .....

```

此命令轉儲接收方接收的最後一個資料包，在本例中為Cisco IOSd。這表示會轉儲標頭，且可以使用終端型Wireshark(TShark)進行解碼。後設資料供系統內部使用，但資料輸出提供實際資料包資訊。然而，後設資料仍然非常有用。

請注意以0070開頭的行。使用後面的16位，如下所示：

```
<#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

```

此處指明了罪魁禍首。Gig2/0/20是一個流量生成器，用於泵ARP流量。如果關閉此功能，則會解決問題並最小化CPU使用率。

第4步：使用FED跟蹤

在上一節中討論的方法的唯一缺點是，它只會轉儲進入接收器的最後一個資料包，而不可能是原因。

對此進行故障排除的更好方法是使用名為FED跟蹤的功能。跟蹤是FED推送到CPU的資料包捕獲方法（使用各種過濾器）。但是，FED跟蹤並不像Cisco Catalyst 6500系列交換機上的Netdr功能那樣簡單。

此過程分為幾個步驟：

1. 啟用詳細資訊跟蹤。預設情況下，事件跟蹤處於開啟狀態。您必須啟用詳細資料追蹤，才能擷取實際封包：

```
<#root>
3850-2#
set trace control fed-punject-detail enable
```

2. 微調捕獲緩衝區。確定緩衝區的深度以便進行詳細跟蹤並根據需要增加。

```
<#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
```

您可以使用以下命令變更緩衝區大小：

```
<#root>
```

3850-2#

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

您可以使用的值包括：

<#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. 新增捕獲篩選器。現在，您需要為擷取新增各種篩選條件。您可以新增不同的過濾器，並選擇匹配所有過濾器或匹配任何用於捕獲的過濾器。

過濾器使用以下命令新增：

<#root>

3850-2#

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

目前提供以下選項：

<#root>

3850-2#

```
set trace fed-punject-detail direction rx filter_add ?
```

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


現在你必須把東西連在一起還記得此故障排除過程步驟2中確定的責任隊列嗎？由於隊列16是將大量資料包推向CPU的隊列，因此有必要跟蹤此隊列並檢視它向CPU推入哪些資料包。

您可以選擇使用以下命令追蹤任何佇列：

```
<#root>
3850-2#
set trace fed-punject-detail direction rx filter_add cpu-queue
```

以下是此範例的命令：

```
<#root>
3850-2#
set trace fed-punject-detail direction rx filter_add cpu-queue 16 16
```

必須為過濾器選擇match all或match any，然後啟用跟蹤：

```
<#root>
3850-2#
set trace fed-punject-detail direction rx match_all

3850-2#
set trace fed-punject-detail direction rx filter_enable
```

4. 顯示過濾的資料包。您可以顯示使用show mgmt-infra trace messages fed-punject-detail命令捕獲的資料包。

```
<#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      suppressRewriteSideEffects=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>

```

此輸出提供大量資訊，通常足以發現資料包的來源及其所包含的內容。

報頭轉儲的第一部分也是系統使用的後設資料。第二部分為實際資料包。

```

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

```

您可以選擇追蹤此來源MAC位址，以探索元兇port（一旦您確定這是從佇列16中捨棄的大部分封包；此輸出只會顯示封包的一個執行個體，且其他輸出/封包已捨棄）。

不過，還有更好的辦法。請注意，標頭資訊後顯示的日誌：

```

[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

第一個日誌清楚地告訴您此資料包來自哪個隊列和標籤。如果您之前沒有意識到該隊列，則這是一種簡單的方法，可以輕鬆識別該隊列是哪個隊列。

第二個日誌甚至更加有用，因為它為源介面提供了物理介面ID工廠(IIF)-ID。十六進位制值是一個控制代碼，可用於轉儲有關該埠的資訊：

<#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 Status     : 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
```

您再次確定了源介面和罪魁禍首。

跟蹤是一個功能強大的工具，對於解決高CPU使用率問題至關重要，它提供了大量資訊以便成功解決此類問題。

Cisco Catalyst 3850系列交換機的嵌入式事件管理器(EEM)指令碼示例

使用以下命令以觸發要在特定閾值處生成的日誌：

```
process cpu threshold type total rising
```

```
interval
```

```
switch
```

使用該命令生成的日誌如下所示：

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


生成的日誌提供以下資訊：

- 觸發時的CPU總使用率。在本例中，這是由Total CPU Utilization(total/Intr):50/0標識的。
- 排名靠前的進程 — 這些進程以PID/CPU%的格式列出。在本例中，這些是：

```
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.
```

EEM指令碼如下所示：

```
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"
```

 註:process cpu threshold 命令當前在3.2.X系列中不起作用。需要記住的另外一點是，此命令檢視四個核心之間的平均CPU利用率，並在該平均值達到命令中定義的百分比時生成日誌。

Cisco IOS XE 16.x或更高版本

如果您的Catalyst 3850交換器執行Cisco IOS® XE軟體版本16.x或更新版本，請參閱[對執行IOS-XE 16.x的Catalyst交換器平台中的高CPU使用率進行疑難排解](#)。

相關資訊

- [什麼是Cisco IOS XE?](#)
- [Cisco Catalyst 3850交換器 — 產品手冊和文獻](#)
- [思科技術支援與下載](#)

關於此翻譯

思科已使用電腦和人工技術翻譯本文件，讓全世界的使用者能夠以自己的語言理解支援內容。請注意，即使是最佳機器翻譯，也不如專業譯者翻譯的內容準確。Cisco Systems, Inc. 對這些翻譯的準確度概不負責，並建議一律查看原始英文文件（提供連結）。