

# Resolución de problemas de uso elevado de la CPU del switch Catalyst serie 3850

## Contenido

---

[Introducción](#)

[Antecedentes](#)

[Caso práctico: Interrupciones del protocolo de resolución de direcciones \(ARP\)](#)

[Paso 1: Identificar el Proceso que Consume los Ciclos de CPU](#)

[Paso 2: Determine la cola de CPU que causa la condición de uso elevado de la CPU](#)

[Paso 3: Volcar el paquete enviado a la CPU](#)

[Paso 4: Utilizar el seguimiento de FED](#)

[Ejemplo de script de Embedded Event Manager \(EEM\) para el switch Catalyst de Cisco serie 3850](#)

[Cisco IOS XE versión 16.x o posterior](#)

[Información Relacionada](#)

---

## Introducción

Este documento describe cómo resolver problemas de uso de CPU, principalmente debido a interrupciones, en la nueva plataforma Cisco IOS® XE.

## Antecedentes

Es importante comprender cómo se crea Cisco IOS® XE. Con Cisco IOS® XE, Cisco se ha pasado a un kernel de Linux y todos los subsistemas se han dividido en procesos. Todos los subsistemas que estaban dentro de Cisco IOS antes, como los controladores de módulos, la alta disponibilidad (HA), etc., ahora se ejecutan como procesos de software dentro del sistema operativo (OS) Linux. El propio Cisco IOS se ejecuta como un demonio dentro del sistema operativo Linux (IOSd). Cisco IOS® XE conserva no solo el mismo aspecto del Cisco IOS® clásico, sino también su funcionamiento, soporte y gestión.

Además, el documento presenta varios comandos nuevos en esta plataforma que son integrales para resolver problemas de uso de la CPU.

Estas son algunas definiciones útiles:

- Controlador de motor de reenvío (FED): se trata del núcleo del switch Catalyst de Cisco serie 3850 y es responsable de toda la programación y retransmisión del hardware.
- Cisco IOSd: Este es el demonio Cisco IOS® que se ejecuta en el kernel de Linux. Se ejecuta como un proceso de software dentro del núcleo.

- Sistema de entrega de paquetes (PDS): arquitectura y proceso de cómo se entregan los paquetes desde y hacia varios subsistemas. Como ejemplo, controla cómo se entregan los paquetes desde la FED al IOSd y viceversa.
- Mango: Un mango se puede considerar como un puntero. Es un medio para descubrir información más detallada sobre variables específicas que se utilizan en las salidas que produce la caja. Esto es similar al concepto de los índices de lógica de destino local (LTL) en el switch Catalyst de Cisco serie 6500.

## Caso práctico: Interrupciones del protocolo de resolución de direcciones (ARP)

El proceso de resolución de problemas y verificación de esta sección se puede utilizar ampliamente para un uso elevado de la CPU debido a interrupciones.

### Paso 1: Identificar el Proceso que Consume los Ciclos de CPU

El comando `show process cpu` naturalmente muestra cómo se ve actualmente la CPU. Observe que el switch Catalyst de Cisco serie 3850 utiliza cuatro núcleos y verá el uso de la CPU enumerado para los cuatro núcleos:

```
<#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
```

De la salida, queda claro que el daemon de Cisco IOS® consume una parte importante de la CPU junto con la FED, que es el corazón de esta caja. Cuando el uso de la CPU es alto debido a interrupciones, verá que Cisco IOSd y FED utilizan una parte importante de la CPU, y estos subprocesos (o un subconjunto de estos) utilizan la CPU:

- FED Punject TX
- FED Punject RX
- Reposición de FED Punject
- FED Punject TX completado

Puede ampliar cualquiera de estos procesos con el comando `show process cpu detailed <process>`. Dado que Cisco IOSd es responsable de la mayor parte del uso de la CPU, he aquí un análisis más detallado de esto.

```
<#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
```

La salida (salida de CPU de Cisco IOSd) muestra que ARP Snoop, IP Host Track Process y ARP Input son altos. Esto se observa comúnmente cuando la CPU se interrumpe debido a los paquetes ARP.

## Paso 2: Determine la cola de CPU que causa la condición de uso elevado de la CPU

El switch Catalyst de Cisco serie 3850 tiene varias colas que atienden a diferentes tipos de paquetes (la FED mantiene 32 colas de CPU RX, que son colas que van directamente a la CPU). Es importante monitorear estas colas para descubrir qué paquetes se envían a la CPU y cuáles son procesados por el IOSd de Cisco. Estas colas son por ASIC.

---

 Nota: Hay dos ASIC: 0 y 1. Los puertos 1 a 24 pertenecen a ASIC 0.

---

Para ver las colas, ingrese el show platform punt statistics port-asic <port-asic>cpuq <queue> direction comando.

En el comando show platform punt statistics port-asic 0 cpuq -1 direction rx, el argumento -1 enumera todas las colas. Por lo tanto, este comando enumera todas las colas de recepción para Port-ASIC 0.

Ahora, debe identificar qué cola envía una gran cantidad de paquetes a una velocidad alta. En este ejemplo, un examen de las colas reveló a este culpable:

```
<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>
```

El número de cola es 16 y el nombre de cola es CPU\_Q\_PROTO\_SNOOPING.

Otra manera de descubrir la cola culpable es ingresar el comando show platform punt client.

<#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

```

Determine la etiqueta para la cual se han asignado la mayoría de los paquetes. En este ejemplo, es 65561.

A continuación, introduzca este comando:

```
<#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

```

Esta salida muestra que la cola es Rx Proto Snoop.

La s antes del 65561 en la salida del comando show platform punt client significa que el identificador FED está suspendido y saturado por el número de paquetes entrantes. Si la s no se desvanece, significa que la cola está atascada permanentemente.

### Paso 3: Volcar el paquete enviado a la CPU

En los resultados del comando show pds tag all, observe un identificador, 7296672, que se informa junto a Punt Rx Proto Snoop.

Utilice este identificador en el comando show pds client <handle> packet last sink. Observe que debe habilitar debug pds pktbuf-last antes de utilizar el comando. De lo contrario, se producirá este 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

Con la depuración habilitada, verá este resultado:

<#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 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 00 ...<.....  
0060 01 01 b6 80 00 00 00 4f 00 00 00 00 00 00 00 00 .....0.....  
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 .....  
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 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 .....  
.....
```

Este comando vuelca el último paquete recibido por el receptor, que es Cisco IOSd en este ejemplo. Esto muestra que vuelca el encabezado y puede ser decodificado con Wireshark basado en Terminal (TShark). Los metadatos son para uso interno del sistema, pero la salida de datos proporciona información real del paquete. Sin embargo, los metadatos siguen siendo extremadamente útiles.

Observe la línea que comienza con 0070. Use los primeros 16 bits después de esto como se muestra aquí:

<#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

```

Aquí se identifica la interfaz culpable. Gig2/0/20 es donde hay un generador de tráfico que bombea el tráfico ARP. Si apaga esto, entonces resolvería el problema y minimizaría el uso de la CPU.

#### Paso 4: Utilizar el seguimiento de FED

El único inconveniente del método descrito en la última sección es que sólo vuelca el último paquete que entra en el receptor, y no puede ser el culpable.

Una mejor manera de solucionar este problema sería utilizar una función llamada seguimiento FED. El seguimiento es un método de captura de paquetes (que utiliza varios filtros) que la FED envía a la CPU. Sin embargo, el seguimiento FED no es tan sencillo como la función Netdr del switch Catalyst de Cisco serie 6500.

En este caso, el proceso se divide en pasos:

1. Habilitar seguimiento de detalles. De forma predeterminada, el seguimiento de eventos está activado. Debe habilitar el seguimiento detallado para capturar los paquetes reales:

```
<#root>
```

```
3850-2#
```

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

2. Ajuste del búfer de captura. Determine la profundidad de las memorias intermedias para el seguimiento detallado y aumente según sea necesario.

```
<#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

Puede cambiar el tamaño del búfer con este comando:

```
<#root>
```

```
3850-2#
```

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

Los valores disponibles son:

```
<#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. Agregue filtros de captura. Ahora necesita agregar varios filtros para la captura. Puede agregar filtros diferentes y elegir si desea que coincidan todos los filtros o cualquiera de ellos para la captura.

Los filtros se agregan con este comando:

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

Estas opciones están disponibles actualmente:

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

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

Ahora debe vincular las cosas. ¿Recuerda la cola de culpables que se identificó en el paso 2 de este proceso de solución de problemas? Dado que la cola 16 es la cola que empuja una gran cantidad de paquetes hacia la CPU, tiene sentido rastrear esta cola y ver qué paquetes son impulsados a la CPU por ella.

Puede elegir rastrear cualquier cola con este comando:

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

Este es el comando para este ejemplo:

```
<#root>
```

```
3850-2#
```

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

Debe elegir una coincidencia total o una coincidencia cualquiera para los filtros y, a continuación, habilitar el seguimiento:

```
<#root>
```

```
3850-2#
```

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

```
3850-2#
```

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

4. Muestra los paquetes filtrados. Puede mostrar los paquetes capturados con el comando `show mgmt-infra trace messages fed-punject-detail`.

```
<#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>
```

Este resultado proporciona mucha información y normalmente puede ser suficiente para descubrir de dónde vienen los paquetes y qué contienen.

La primera parte del vaciado de encabezados es nuevamente el Meta-data que es utilizado por el sistema. La segunda parte es el paquete real.

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

Puede elegir rastrear esta dirección MAC de origen para descubrir el puerto culpable (una vez que haya identificado que esta es la mayoría de los paquetes que se impulsan desde la cola 16; esta salida solo muestra una instancia del paquete y los otros paquetes/salida se recortan).

Sin embargo, hay una manera mejor. Observe que los registros que están presentes después de la información de encabezado:

```
[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
```

El primer registro le indica claramente de qué cola y etiqueta proviene este paquete. Si no conocía la cola anteriormente, esta es una manera fácil de identificar qué cola era.

El segundo registro es aún más útil porque proporciona el ID de fábrica de ID de interfaz (IIF) físico para la interfaz de origen. El valor hexadecimal es un identificador que se puede utilizar para volcar información sobre ese puerto:

```
<#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
```

Una vez más ha identificado la interfaz de origen y el culpable.

El seguimiento es una herramienta poderosa que es crítica para resolver problemas de uso elevado de la CPU y proporciona abundante información para resolver con éxito tal situación.

### Ejemplo de script de Embedded Event Manager (EEM) para el switch Catalyst de Cisco serie 3850

Utilice este comando para activar un registro que se generará en un umbral específico:

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

```
interval
```

```
switch
```

El registro generado con el comando tiene el siguiente aspecto:

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

El registro generado proporciona esta información:

- El uso total de la CPU en el momento del desencadenador. Esto se identifica mediante la utilización total de la CPU (total/Intr) :50/0 en este ejemplo.
- Procesos principales: se enumeran en el formato PID/CPU%. En este ejemplo, estos son:

```
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.
```

La secuencia de comandos de EEM se muestra aquí:


```
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"
```



Nota: El comando process cpu threshold no funciona actualmente en el tren 3.2.X. Otro

---

---

 punto a recordar es que este comando observa el uso promedio de la CPU entre los cuatro núcleos y genera un registro cuando este promedio alcanza el porcentaje que se ha definido en el comando.

---

## Cisco IOS XE versión 16.x o posterior

Si tiene switches Catalyst 3850 que ejecutan la versión 16.x o posterior del software Cisco IOS® XE, consulte [Resolución de problemas de uso elevado de la CPU en plataformas de switches Catalyst que ejecutan IOS-XE 16.x](#).

## Información Relacionada

- [¿Qué es Cisco IOS XE?](#)
- [Switches Cisco Catalyst 3850 - Hojas de datos y documentación](#)
- [Soporte técnico y descargas de Cisco](#)

## Acerca de esta traducción

Cisco ha traducido este documento combinando la traducción automática y los recursos humanos a fin de ofrecer a nuestros usuarios en todo el mundo contenido en su propio idioma.

Tenga en cuenta que incluso la mejor traducción automática podría no ser tan precisa como la proporcionada por un traductor profesional.

Cisco Systems, Inc. no asume ninguna responsabilidad por la precisión de estas traducciones y recomienda remitirse siempre al documento original escrito en inglés (insertar vínculo URL).