

Dépannage de l'utilisation CPU élevée du commutateur Catalyst 3850

Table des matières

[Introduction](#)

[Informations générales](#)

[Étude de cas : Interruptions du protocole ARP \(Address Resolution Protocol\)](#)

[Étape 1 : Identifiez le processus qui consomme des cycles CPU](#)

[Étape 2 : Déterminez la file d'attente du processeur qui entraîne la condition d'utilisation élevée du processeur](#)

[Étape 3 : Videz le paquet envoyé au processeur](#)

[Étape 4 : Utiliser le suivi FED](#)

[Exemple de script EEM \(Embedded Event Manager\) pour le commutateur Cisco Catalyst 3850](#)

[Cisco IOS XE 16.x ou versions ultérieures](#)

[Informations connexes](#)

Introduction

Ce document décrit comment résoudre les problèmes d'utilisation du CPU, principalement dus à des interruptions, sur la nouvelle plate-forme Cisco IOS® XE.

Informations générales

Il est important de comprendre comment Cisco IOS® XE est conçu. Avec Cisco IOS® XE, Cisco est passé à un noyau Linux et tous les sous-systèmes ont été décomposés en processus. Tous les sous-systèmes qui étaient auparavant à l'intérieur de Cisco IOS, tels que les pilotes de modules, la haute disponibilité (HA), etc., s'exécutent désormais en tant que processus logiciels dans le système d'exploitation Linux. Cisco IOS lui-même fonctionne comme un démon dans le système d'exploitation Linux (IOSd). Cisco IOS® XE conserve non seulement la même apparence que le système d'exploitation classique Cisco IOS®, mais également son fonctionnement, sa prise en charge et sa gestion.

En outre, le document introduit plusieurs nouvelles commandes sur cette plate-forme qui sont intégrées afin de dépanner les problèmes d'utilisation du CPU.

Voici quelques définitions utiles :

- Pilote FED (Forwarding Engine Driver) : il s'agit du cœur du commutateur Cisco Catalyst 3850. Il est responsable de la programmation et du transfert de tout le matériel.
- Cisco IOSd : il s'agit du démon Cisco IOS® qui s'exécute sur le noyau Linux. Il est exécuté en tant que processus logiciel dans le noyau.

- Packet Delivery System (PDS) : il s'agit de l'architecture et du processus de livraison des paquets à destination et en provenance de divers sous-systèmes. Par exemple, il contrôle la manière dont les paquets sont livrés de la FED à l'IOSd et vice versa.
- Poignée : une poignée peut être considérée comme un pointeur. C'est un moyen de découvrir des informations plus détaillées sur des variables spécifiques qui sont utilisées dans les sorties que la boîte produit. Cela est similaire au concept d'index LTL (Local Target Logic) sur le commutateur Cisco Catalyst 6500.

Étude de cas : Interruptions du protocole ARP (Address Resolution Protocol)

Le processus de dépannage et de vérification de cette section peut être largement utilisé pour une utilisation CPU élevée due à des interruptions.

Étape 1 : Identifiez le processus qui consomme des cycles CPU

La commande `show process cpu` affiche naturellement l'apparence actuelle du processeur. Notez que le commutateur de la gamme Cisco Catalyst 3850 utilise quatre coeurs et que vous voyez l'utilisation du processeur répertoriée pour les quatre coeurs :

```
<#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
```

D'après le résultat, il est clair que le démon Cisco IOS® consomme une partie majeure du CPU avec le FED, qui est le coeur de cette boîte. Lorsque l'utilisation du CPU est élevée en raison d'interruptions, vous voyez que Cisco IOSd et FED utilisent une partie majeure du CPU, et ces sous-processus (ou un sous-ensemble de ceux-ci) utilisent le CPU :

- FED Punject TX
- FED Punject RX
- FED Punject replenish
- FED Punject TX terminé

Vous pouvez effectuer un zoom avant sur l'un de ces processus avec la commande `show process cpu detailed <process>`. Puisque Cisco IOSd est responsable de la majorité de l'utilisation du CPU, voici un examen plus approfondi.

```
<#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 sortie (sortie CPU Cisco IOSd) indique que la surveillance ARP, le processus de suivi d'hôte IP et l'entrée ARP sont élevés. Cela se produit généralement lorsque le processeur est interrompu en raison de paquets ARP.

Étape 2 : Déterminez la file d'attente du processeur qui entraîne la condition d'utilisation élevée du processeur

Le commutateur de la gamme Cisco Catalyst 3850 dispose d'un certain nombre de files d'attente qui prennent en charge différents types de paquets (le FED gère 32 files d'attente CPU RX, qui sont des files d'attente qui vont directement au CPU). Il est important de surveiller ces files d'attente afin de découvrir quels paquets sont envoyés au CPU et lesquels sont traités par Cisco IOSd. Ces files d'attente sont définies par ASIC.



Remarque : il existe deux ASIC : 0 et 1. Les ports 1 à 24 appartiennent à l'ASIC 0.

Afin de consulter les files d'attente, entrez la commande `show platform punt statistics port-asic <port-asic>cpuq <queue> direction erasecat4000_flash:`.

Dans la commande `show platform punt statistics port-asic 0 cpuq -1 direction rx`, l'argument -1 répertorie toutes les files d'attente. Par conséquent, cette commande répertorie toutes les files d'attente de réception pour Port-ASIC 0.

Vous devez maintenant identifier la file d'attente qui envoie un grand nombre de paquets à un débit élevé. Dans cet exemple, un examen des files d'attente a révélé ce coupable :

```
<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>
```

Le numéro de la file d'attente est 16 et son nom est CPU_Q_PROTO_SNOOPING.

Une autre façon de découvrir la file d'attente coupable est d'entrer la commande `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

```

Déterminez l'étiquette pour laquelle le plus grand nombre de paquets a été alloué. Dans cet exemple, il s'agit de 65561.

Entrez ensuite la commande suivante :

```
<#root>
```

```
3850-2#
```

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

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

Ce résultat montre que la file d'attente est Rx Proto Snoop.

Le s avant le 65561 dans le résultat de la commande show platform punt client signifie que le handle FED est suspendu et submergé par le nombre de paquets entrants. Si le s ne disparaît pas, cela signifie que la file d'attente est bloquée de façon permanente.

Étape 3 : Videz le paquet envoyé au processeur

Dans les résultats de la commande show pds tag all, notez qu'un handle, 7296672, est signalé à côté de la surveillance Proto Punt Rx.

Utilisez ce handle dans la commande show pds client <handle> packet last sink. Notez que vous devez activer debug pds pktbuf-last avant d'utiliser la commande. Sinon, vous rencontrez cette erreur :

```
<#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
```

Lorsque le débogage est activé, le résultat suivant s'affiche :

```
<#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 .....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 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 .....
```

Cette commande vide le dernier paquet reçu par le récepteur, qui est Cisco IOSd dans cet exemple. Cela montre qu'il vide l'en-tête et qu'il peut être décodé avec Terminal-based Wireshark (TShark). Les métadonnées sont destinées à une utilisation interne par le système, mais la sortie Données fournit des informations réelles sur les paquets. Les méta-données, cependant, restent extrêmement utiles.

Notez la ligne qui commence par 0070. Utilisez les 16 premiers bits suivants, comme illustré ci-dessous :

```
<#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

```

L'interface coupable est identifiée ici. Gig2/0/20 est l'emplacement où un générateur de trafic pompe le trafic ARP. Si vous arrêtez cette fonction, le problème sera résolu et l'utilisation du processeur sera réduite au minimum.

Étape 4 : Utiliser le suivi FED

Le seul inconvénient de la méthode décrite dans la dernière section est qu'elle ne fait que vider le dernier paquet qui va dans le récepteur, et qu'elle ne peut pas être le coupable.

Une meilleure façon de résoudre ce problème serait d'utiliser une fonctionnalité appelée suivi FED. Le traçage est une méthode de capture de paquets (à l'aide de différents filtres) qui sont envoyés par le FED au processeur. Cependant, le suivi FED n'est pas aussi simple que la fonctionnalité Netdr sur le commutateur de la gamme Cisco Catalyst 6500.

Ici, le processus est divisé en étapes :

1. Activer le suivi détaillé. Par défaut, le suivi des événements est activé. Vous devez activer le suivi détaillé afin de capturer les paquets réels :

```
<#root>
```

```
3850-2#
```

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

2. Ajustez la mémoire tampon de capture. Déterminez la profondeur de vos tampons pour le traçage des détails et augmentez-les si nécessaire.

```
<#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
```

Vous pouvez modifier la taille de la mémoire tampon avec cette commande :

```
<#root>
```

```
3850-2#
```

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

Les valeurs disponibles sont les suivantes :

```
<#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. Ajoutez des filtres de capture. Vous devez maintenant ajouter divers filtres pour la capture. Vous pouvez ajouter différents filtres et choisir de les faire correspondre tous ou de les faire correspondre à l'un d'entre eux pour votre capture.

Les filtres sont ajoutés avec cette commande :

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

Ces options sont actuellement disponibles :

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

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

Maintenant, vous devez relier les choses ensemble. Vous souvenez-vous de la file d'attente coupable identifiée à l'étape 2 de ce processus de dépannage ? Puisque la file d'attente 16 est la file d'attente qui pousse un grand nombre de paquets vers le CPU, il est logique de suivre cette file d'attente et de voir quels paquets sont envoyés au CPU par elle.

Vous pouvez choisir de tracer n'importe quelle file d'attente avec cette commande :

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

Voici la commande pour cet exemple :

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

Vous devez choisir une correspondance tout ou une correspondance quelconque pour vos filtres, puis activer le suivi :

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

4. Affiche les paquets filtrés. Vous pouvez afficher les paquets capturés avec la commande `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: 12_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>
```

Cette sortie fournit de nombreuses informations et peut généralement être suffisante pour découvrir d'où viennent les paquets et ce qu'ils contiennent.

La première partie du vidage d'en-tête est à nouveau constituée des métadonnées utilisées par le système. La deuxième partie est le paquet réel.

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

Vous pouvez choisir de tracer cette adresse MAC source afin de découvrir le port coupable (une fois que vous avez identifié qu'il s'agit de la majorité des paquets qui sont envoyés de la file d'attente 16 ; cette sortie n'affiche qu'une instance du paquet et l'autre sortie/les paquets sont écrêtés).

Cependant, il y a une meilleure façon. Notez que les journaux qui sont présents après les informations d'en-tête :

```
[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
```

Le premier journal vous indique clairement de quelle file d'attente et étiquette provient ce paquet. Si vous ne connaissiez pas la file d'attente précédemment, c'est un moyen facile d'identifier la file d'attente qu'elle était.

Le deuxième journal est encore plus utile, car il fournit l'ID d'interface physique IIF (Interface ID Factory) pour l'interface source. La valeur hexadécimale est un handle qui peut être utilisé afin de vider les informations sur ce 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
```

Vous avez une fois de plus identifié l'interface source et le coupable.

Le traçage est un outil puissant qui est essentiel pour dépanner les problèmes d'utilisation élevée du CPU et fournit beaucoup d'informations afin de résoudre avec succès une telle situation.

Exemple de script EEM (Embedded Event Manager) pour le commutateur Cisco Catalyst 3850

Utilisez cette commande afin de déclencher un journal à générer à un seuil spécifique :

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

```
interval
```

```
switch
```

Le journal généré avec la commande ressemble à ceci :

```
*Jan 13 00:03:00.271: %CPUMEM-5-RISING_THRESHOLD: 1 CPUMEMd[6300]: Threshold: : 50, Total CPU Utilization: 50/0
```


Le journal généré fournit les informations suivantes :

- Utilisation totale du processeur au moment du déclenchement. Ceci est identifié par Total CPU Utilization(total/Intr) : 50/0 dans cet exemple.
- Processus principaux : ceux-ci sont répertoriés au format PID/CPU%. Dans cet exemple, il s'agit de :

```
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.
```

Le script EEM est illustré ci-dessous :

```
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"
```

 Remarque : la commande process cpu threshold ne fonctionne pas actuellement dans le train 3.2.X. Un autre point à retenir est que cette commande examine l'utilisation moyenne du CPU parmi les quatre coeurs et génère un journal lorsque cette moyenne atteint le pourcentage qui a été défini dans la commande.

Cisco IOS XE 16.x ou versions ultérieures

Si vous avez des commutateurs Catalyst 3850 qui exécutent le logiciel Cisco IOS® XE Version 16.x ou ultérieure, consultez [Dépannage de l'utilisation élevée du CPU dans les plates-formes de commutateur Catalyst exécutant IOS-XE 16.x](#).

Informations connexes

- [Qu'est-ce que Cisco IOS XE ?](#)
- [Commutateurs Cisco Catalyst 3850 - Fiches techniques et documentation](#)
- [Assistance technique de Cisco et téléchargements](#)

À propos de cette traduction

Cisco a traduit ce document en traduction automatisée vérifiée par une personne dans le cadre d'un service mondial permettant à nos utilisateurs d'obtenir le contenu d'assistance dans leur propre langue.

Il convient cependant de noter que même la meilleure traduction automatisée ne sera pas aussi précise que celle fournie par un traducteur professionnel.