Catalyst 3850 Series probleemoplossing bij gebruik met hoge CPU's

Inhoud

Inleiding

Dit document beschrijft hoe u problemen met uw CPU-gebruik kunt oplossen, vooral door onderbrekingen, op het nieuwe Cisco IOS®-XE-platform. Daarnaast voert het document verschillende nieuwe opdrachten uit op dit platform die integraal zijn om dergelijke problemen op te lossen.

Achtergrondinformatie

Het is belangrijk om te begrijpen hoe Cisco IOS-XE wordt gebouwd. Met Cisco IOS-XE is Cisco naar een Linux-kern verplaatst en alle subsystemen zijn gesplitst in processen. Alle subsystemen die al eerder in Cisco IOS waren - zoals de modemstuurprogramma's, hoge beschikbaarheid (HA) en dergelijke - worden nu gebruikt als softwareprocessen binnen het Linux Operating System (OS). Cisco IOS zelf functioneert als een daemon binnen het Linux OS (IOSd). Cisco IOS-XE behoudt niet alleen de zelfde kijk en het zelfde gevoel van de klassieke Cisco IOS, maar ook de werking, de ondersteuning en het beheer.

Hier zijn een paar nuttige definities:

- **Stuurprogramma voor doorsturen van motoren (FED)**: Dit is het hart van de Cisco Catalyst 3850 Series switch en is verantwoordelijk voor alle hardware programmering/doorsturen.
- **IOSd**: Dit is de Cisco IOS-telefoon die op de Linux-kern draait. Het wordt in de vorm van een softwareproces uitgevoerd in de kern.
- **Packet Delivery System (PDS)**: Dit is de architectuur en het proces van de manier waarop pakketten aan en van verschillende subsystemen worden geleverd. Als voorbeeld controleert het hoe de pakketten van FED aan IOSd worden geleverd en vice versa.

- **Handle**: Een hendel kan worden gezien als een muisaanwijzer. Het is een middel om meer gedetailleerde informatie te ontdekken over specifieke variabelen die worden gebruikt in de output die de doos produceert. Dit is gelijk aan het concept van Local Target Logic-indices (LTL) op Cisco Catalyst 6500 Series switch.

Case Study: Protocol voor adresoplossing (ARP) - onderbreken

Het proces voor probleemoplossing en verificatie in deze sectie kan breed worden gebruikt voor gebruik met een hoge CPU-functie door onderbrekingen.

Stap 1: Identificeer het proces dat CPU-cycli verbruikt

Het **cpu-**opdracht **tonen** geeft uiteraard weer hoe de CPU er momenteel uitziet. Merk op dat de Cisco Catalyst 3850 Series-switch vier kernen gebruikt en u ziet het CPU-gebruik dat voor alle vier kernen is vermeld:

```
3850-2#show processes cpu sort | 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
```

Van de output is het duidelijk dat de Cisco IOS daemon een groot deel van de CPU samen met FED verwerkt, wat het hart van dit vakje is. Wanneer het CPU-gebruik hoog is door onderbrekingen, ziet u dat IOSd en FED een groot deel van de CPU gebruiken en dat deze subprocessen (of een subset daarvan) de CPU's gebruiken:

- FED-punt TX
- FED punt RX
- Aanvullen van FED-punten
- FED punt TX voltooid

U kunt in een van deze processen inzoomen met de **gedetailleerde *<proces>*- *opdracht van het show-proces-cpu***. Aangezien IOSd verantwoordelijk is voor het grootste deel van het CPU-gebruik, is hier een nadere analyse daarvan.

```
3850-2#show processes cpu detailed process iosd sort | 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
```

De output (IOSd CPU-uitvoer) toont dat ARP-sleuven, IP-hosttrajecten en ARP-ingangen hoog zijn. Dit wordt vaak gezien wanneer de CPU wordt onderbroken door ARP-pakketten.

Stap 2: Bepaal de CPU-wachtrij die de gebruiksconditie met hoge CPU's veroorzaakt

De Cisco Catalyst 3850 Series-switch heeft een aantal wachtrijen die betrekking hebben op verschillende typen pakketten (de FED handhaaft 32 RX CPU-wachtrijen, die rijen zijn die rechtstreeks naar de CPU gaan). Het is belangrijk om deze rijen te controleren om te ontdekken welke pakketten aan de CPU worden gestraft en door IOSd worden verwerkt. Deze rijen zijn per ASIC.

> Opmerking: Er zijn twee ASIC' s: 0 en 1. De poorten 1 tot en met 24 behoren tot ASIC 0.

Om de wachtrijen te bekijken, voert u de **basisstatistieken van het showplatform punt in *<poort-basis>*CAQ *<wachtrij>* richting *<rx|tx>*** opdracht.

In het **show platform punt statistiek port-basis 0 cpuq -1 richting rx** opdracht, **-1** argument maakt een lijst van alle wachtrijen. Om deze reden, deze opdracht lijsten alle ontvangen wachtrijen voor Port-ASIC 0.

Nu moet u identificeren welke rij een groot aantal pakketten tegen een hoog tempo duwt. In dit voorbeeld onthulde een onderzoek van de rijen deze schuldige:

```
<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>
```

Het wachtrijnummer is 16 en de wachtrijnaam is **CPU_Q_PROTO_SNOOPING**.

Een andere manier om de belastingsrij te ontdekken is de opdracht van de **showplatform punt client** in te voeren.

```
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
```

Bepaal de tag waarvoor de meeste pakketten zijn toegewezen. In dit voorbeeld is het **65561**.

Typ vervolgens deze opdracht:

```
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
```

Deze output laat zien dat de rij **Rx Proto Snoop** is.

De opdracht vóór de **65561** in de uitvoer van de opdracht van de **showplatform punt client** betekent dat de FED handle is opgeschort en overweldigd door het aantal inkomende pakketten. Als de **s** niet verdwijnt betekent dit dat de rij permanent overeind blijft.

Stap 3: Pomp het pakket dat naar de CPU is verzonden af

In de resultaten van de **show pds tag all** opdracht, merk een handle op, **7296672**, wordt gerapporteerd naast de **Punt Rx Proto Snoop**.

Gebruik deze handgreep in de *opdracht tonen* Pds **client<*handle*> pakketverlies** opdracht. Merk op dat u **debug pktbuf-last** moet inschakelen voordat u de opdracht gebruikt. Anders ontmoet u deze fout:

```
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
```

Als debug ingeschakeld is, ziet u deze uitvoer:

```
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              ............
```

Deze opdracht dumpt het laatste pakket dat door de gootsteen wordt ontvangen, wat IOSd in dit voorbeeld is. Dit toont aan dat de kop is vastgedraaid en dat de kop kan worden gedecodeerd met Wireshark (TShark) op terminaal basis. De **Meta-gegevens** zijn bedoeld voor intern gebruik door het systeem, maar de **Data**-output geeft feitelijke pakketinformatie. De **Meta-data** blijven echter extreem nuttig.

Let op de lijn die begint met **0070**. Gebruik de eerste 16 bits daarna, zoals hier wordt getoond:

```
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
```

De schuldige interface wordt hier geïdentificeerd. **Gig2/0/20** is waar er een verkeersgenerator is die ARP-verkeer pompt. Als u dit activeert, wordt het probleem opgelost en het CPU-gebruik tot een minimum beperkt.

Stap 4: FED-overtrekken gebruiken

Het enige nadeel met de methode die in het laatste deel is besproken is dat het alleen het laatste pakje dumpt dat in de gootsteen gaat, en dat het dan niet de schuldige is.

Een betere manier om dit probleem op te lossen is door gebruik te maken van een optie die **FED-tracering** wordt genoemd. Tracing is een methode van de pakketvastlegging (met verschillende filters) die door de FED naar de CPU wordt geduwd. FED-overtrekken is niet zo eenvoudig als de NetFlow-functie in Cisco Catalyst 6500 Series switch, maar. Hier wordt het proces in stappen verdeeld:

1. Voer informatie in bij het volgen. Standaard is de gebeurtenis-overtrekken **ingeschakeld**. U moet het overtrekken van informatie toestaan om de eigenlijke pakketten op te nemen:

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

2. Stel de opnamefbuffer fijnaf. Bepaal hoe diep uw buffers voor detail het vinden zijn en vermeerder zoals nodig.

```
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
```

U kunt de buffergrootte met deze opdracht wijzigen:

```
3850-2#set trace control fed-punject-detail buffer-size
```

De waarden die u beschikbaar zijn zijn:

```
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. Opname filters toevoegen. U moet nu verschillende filters voor de opname toevoegen. U kunt verschillende filters toevoegen en of kiezen om **alle filters** aan te **passen** of **aan een** van deze voor uw opname **aan te** passen.

   Filters worden toegevoegd aan deze opdracht:

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

   Deze opties zijn momenteel beschikbaar:

```
3850-2#set trace fed-punject-detail direction rx filter_add ?
 cpu-queue  rxq 0..31
 field      field
 offset     offset
```

   Nu moet je dingen met elkaar verbinden. Herinner je de culturen die in Stap 2 van dit proces van probleemoplossing werden geïdentificeerd? Aangezien wachtrij 16 de wachtrij is die een groot aantal pakketten naar de CPU aanstuurt, is het verstandig deze wachtrij te overtrekken en te zien welke pakketten bij de CPU worden gestraft.

   U kunt ervoor kiezen om een wachtrij met deze opdracht te overtrekken:

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

   Hier is de opdracht voor dit voorbeeld:

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

   U moet kiezen **voor alle** filters **overeenkomende producten** of een **overeenkomend product** voor uw filters en vervolgens het overtrekken inschakelen:

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

4. Vertoon gefilterde pakketten. U kunt de pakketten weergeven die zijn opgenomen met de opdracht **Show Sprint-Inra-overtrekken naar meldingen met** gedetailleerde **informatie-punjectie**.

```
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>
```

Deze uitvoer biedt veel informatie en zou normaal genoeg moeten zijn om te ontdekken waar de pakketten vandaan komen en wat in hen zit.

Het eerste deel van het header-dumpen is opnieuw de **Meta-data** die gebruikt worden door het systeem. Het tweede deel is het echte pakje.

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

U kunt ervoor kiezen om dit bron-MAC-adres te overtrekken om de schuldige poort te ontdekken (zodra u hebt geïdentificeerd dat dit de meerderheid is van de pakketten die uit rij 16 worden gestraft; deze uitvoer toont slechts één exemplaar van het pakket en de andere uitvoer/pakketten worden geknipt).

Er is echter een betere manier. Merk op dat blogs die aanwezig zijn na de veldnameninformatie:

```
[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
```

Het eerste logbestand vertelt duidelijk vanaf welke rij en tag dit pakket komt. Als je niet op de hoogte was van de rijouder, dan is dit een makkelijke manier om in de rij te zoeken.

Het tweede logbestand is zelfs nog nuttiger, omdat het de fysieke interface-ID Factory (IIF)-ID biedt voor de broninterface. De hex-waarde is een handle die gebruikt kan worden om

informatie over die poort te dumpen:

```
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
```

U hebt opnieuw de bron-interface en de schuldige geïdentificeerd.

Tracing is een krachtig gereedschap dat van cruciaal belang is voor het oplossen van grote problemen in het CPU-gebruik en biedt voldoende informatie om een dergelijke situatie met succes op te lossen.

Monster Embedded Event Manager (EEM) scripts voor Cisco Catalyst 3850 Series switch

Gebruik deze opdracht om een logbestand te activeren dat op een specifieke drempel gegenereerd wordt:

```
process cpu threshold type total rising interval
switch
```

Het logbestand dat met de opdracht gegenereerd wordt, ziet er als volgt uit:

```
*Jan 13 00:03:00.271: %CPUMEM-5-RISING_THRESHOLD: 1 CPUMEMd[6300]:  Threshold: :
50, Total CPU Utilzation(total/Intr) :50/0, Top 3 processes(Pid/Util) : 8622/25,
```

5753/12, 9663/0

Het weblog dat wordt gegenereerd geeft deze informatie:

- Het totale CPU-gebruik ten tijde van de trigger. Dit wordt in dit voorbeeld geïdentificeerd door **totaal CPU-gebruik (totaal/Inter):50/0**.
- Belangrijkste processen - deze worden weergegeven in het formaat **PID/CPU%**. In dit voorbeeld zijn dit:

```
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.
```

Het EEM-script wordt hier getoond:

```
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"
```

> Opmerking: De **procescpu drempelopdracht** werkt momenteel niet in de 3.2.X-trein. Een ander punt om te onthouden is dat deze opdracht de gemiddelde CPU-benutting tussen de vier kernen bekijkt en een logbestand genereert wanneer dit gemiddelde het percentage bereikt dat in de opdracht is gedefinieerd.

Cisco IOS-XE 16.x of latere releases

Als u Catalyst 3850-switches hebt die Cisco IOS-XE softwarerelease 16.x of hoger uitvoeren, zie Gebruik van probleemoplossing in Catalyst-switchplatforms die IOS-XE 16.x uitvoeren.

Gerelateerde informatie

- Wat is Cisco IOS XE?
- Cisco Catalyst 3850-switches - Datasheets en documentatie
- Technische ondersteuning en documentatie – Cisco Systems