Troubleshooting 3650/3850 reloads by stack manager through a system report

Introduction

Troubleshooting stack reloads through a system report in the absence of a crash is commonly done on NGWC switching platforms using stackwise technology. The current documentation is limited on the uses of a system report and this guide is being written to explain how you can leverage these reports to diagnose problems typically found with stacking issues. This guide is particularly geared for the Catalyst 3650/3850 switching architectures running IOS-XE that supports stacking capabilities.

The majority of issues with stackwise technology stem from a communication problem between the members within a stack. Any inconsistency of information between the members or loss of connectivity can result in a problem that permeates through the entire stack ultimately leading to a reset with stack manager. This document will highlight some of the common types of failures seen with stack-manager reloads, uses of a system report, and relevant CLIs available to diagnose and different types of problems.

Background information:

System report versus switch reports

A system-report is a comprehensive report of the member from how it perceives the state of the stack. This is not a crashinfo (which will dump out memory for further debugging), but instead, is a report that has logs and debugging information for various services running under IOS-XE that would be useful for development to track the state of that service. A system-report can be generated when the switch is reloaded by stack manager, a process crash has occurred, or manually generated by the user during live operation.

In many cases, there are situations in which a single switch might fail in a stack but the rest of the members may remain intact. To gather as information on the state of the stack at the given time, switch_reports were introduced so that remaining members will generate one when it detects that a member has gone down. The switch_report will be a local report of how that member perceives the current state of the stack.

Note: These reports are written and compressed so they cannot be printed to the terminal using ‘more’. They will need to be transferred off the switch and decompressed to view the log.
Where to gather system/switch reports

System reports will typically be written in the crashinfo: directory of the member in that stack. For instance, if there’s an x-member switch stack, then each switch will have their own crashinfo directory which can be accessible using “dir crashinfo-x” where ‘x’ corresponds to that member within the stack.

3850#dir crashinfo-1:

Directory of crashinfo:/

11  -rwx         355  Aug 14 2015 07:48:17 -04:00 last_systemreport_log

12  -rwx      724015  Oct 15 2014 07:14:32 -04:00 system-report_1_20141015-111342-UTC.gz

3850#dir crashinfo-2:

Directory of crashinfo-2:/

11  -rwx         357  Aug 14 2015 07:50:49 -04:00 last_systemreport_log

12  -rwx      751340  Oct 15 2014 06:41:12 -04:00 system-report_1_20141015-104022-UTC.gz

Note: Be sure to gather the output for “dir crashinfo-x:” for every switch in that stack because the ‘show tech’ will not list out the available file systems or the crashinfo files. It is important that you have the entire picture of each and every member in that stack. Update: As of newer IOS-XE releases >3.6E, the show tech will reflect the "dir crashinfo:" + 'show file systems' output. See CSCun50428.

Interesting sections in the system report

From a TAC perspective, these are some of the more commonly viewed entries within the system report that can help diagnose events of a specific issue. There are other logs from other services contained in here that development may find want to review.
log file: /mnt/pss/sup_sysmgr_reset.log

This is a short output to very generically understand why a reset was seen. See the below types of failures section to look the meaning and context in how these reasons will vary.

log file: IOS

This is the log buffer maintained from within IOSd. Any commands that were issued by the user or syslogs generated within IOSd will be found in this section. Most recent logs are towards the end of this output.

Trace Buffer: stack-mgr-events

Keeps track of events seen from stack manager which will include when other members are joining/removed from the stack or which stack port the messages come in through.

Trace Buffer: redundancy-timer-ha_mgr

Displays keep alive events between switches in the stack. The timestamps can help determine when the breakdown in communication started.

Types of failures

This section will highlight some commonly seen resets from a system report which are typically invoked by the stack manager process. Stack manager is a linux process that manages the members in the stack and will oversee any changes in roles between switches in the stack. If stack manager detects a problem during initialization or role election, it will send a reload signal to individual switches in order for the stack to reset. Below will also list known bugs that have been associated to the respective failure type.

Note: Not all stack-manager reloads are attributed to a software problem. In fact, it is more common to see these problems manifest as a secondary/victim issue to an underlying hardware problem.

Reset Reason: Reset/Reload requested by [stack-manager]. [ISSU Incompatibility]

You might see this type of reset when there's a bulk sync failure while trying to synchronize the configuration on the master between all the members in the stack. Checking the logfile: IOS or the
the logs from the active switch might highlight the configurations that contributed to this reset.


This seen when the switch crashes in IOSd process. Looking at the crashinfo directories for any crashinfo files + core dumps can be used to debug this failure further.

**hap_sup_reset:** Reset Reason:Reset/Reload requested by [stack-manager]. [stack merge]

A stack merge is seen when there are two or more switches that believe they are the stack master. This can be seen when there’s a break in the ring of a stack (i.e. two cables are disconnected from the stack) so both the active and standby loses communication to the other members. The addition of an already powered switch to an existing stack can cause a stack merge as there will be two active switches in the stack.

[CSCuh58098] - 3850 stack may reload when stack cabling issues are present
[CSCui56058] - Enabling debounce logic for stack cable
[CSCup53338] - 3850 IOSD crash | Signal=SIGSEGV(11) @ pm_port_data_from_swidb

**hap_sup_reset:** Reason Code:[4] Reset Reason:Reset/Reload requested by [stack-manager]. [stack merge due to incompatibility]

This has been seen in situations when an active and standby switch exists in the stack. If the active switch loses communication to the standby, the standby will attempt to take over as the active even though the active is still up.

[CSCuo49555] /CSCup58016 - 3850/3650 crashes due to unicast flood on mgmt port
[CSCur07909] - Stack merge due to active and standby lost connectivity

**Reset Reason:** Reset/Reload requested by [stack-manager]. [Wrong neighbor encountered after ASIC ballot]

Switches participate in an ASIC ballot during boot up to determine its neighboring switches within
the stack. This reset can be seen when a timer expires for a neighbor to declare itself or if there’s a logic error during the nbrcast conversation. This has been seen in context of faulty stack cables which have been resolved through replacement.

**CSCun60777** - Switch reloaded due to Wrong neighbor encountered after ASIC ballot

**CSCud93761** - Switch reloaded due to Wrong neighbor encountered after ASIC ballot

**hap_sup_reset: Reason Code:[4] Reset Reason:Reset/Reload requested by [stack-manager]. [lost both active and standby]**

This is typically seen from a member on the stack that is not in an either an active or standby role. When the active fails, if there is no standby switch to assume the active role for the stack, then the entire stack will reset. If the stack is an unstable state or redundancy policy have not synced yet, this can be seen. This is likely a victim of the why the active/standby switches went down or potentially an indication that redundancy is not syncing correctly. This can also be seen in when stacks are configured in a half-ring setup.

**CSCup53882** - Member switches in a 3850 stack reboot - [lost both active and standby]

**hap_sup_reset: Reason Code:[1] Reset Reason:Reset/Reload requested by [stack-manager]. [Keepalive_Timeout]**

Seen when keep alive messages are not received from the switch in the stack. Looking at “Trace Buffer: redundancy-timer-ha_mgr” should show the exchange of keep alive messages and provide a perspective of time for when the breakdown in communication began. Gathering switch reports from the rest of the stack and looking at logs during the time frame may help here.

**hap_sup_reset: Reset Reason:Reset/Reload requested by [stack-manager]. [Reload command]**

This is a pretty intuitive reset reason – this is seen when stack-manager receives a reload request which could be invoked through CLI or externally via management device (SNMP). In cases of **CSCuj17317**, this will also show up as a ‘reload command’ issued as well. From the log file: IOS you can see:
CMD: 'reload'
%SYS-5-RELOAD: Reload requested by console. Reload Reason: Reload command.
%STACKMGR-1-RELOAD_REQUEST: 1 stack-mgr: Received reload request for all switches, reason Reload command
%STACKMGR-1-RELOAD: 1 stack-mgr: Reloading due to reason Reload command

**Relevant Bugs**

[CSCur76872](#) - Stack manager goes down when the system runs out of SDP buffer.

[CSCup49704](#) - 3850 FED Crash - Waiting for SPI channels FED_SPI_FLCD,FED_SPI_FAST ...

**Diagnosing a potential stack cabling/port issue**

Symptom 1) Any signs of a stack cabling issue will be apparent by any flapping of the stack port prior to the reset. Looking at the “logfile: IOS” report prior to a reset is typically a good place to start. Here’s an example of where you see flapping of the stack port which is registered on both the current SW2 and the standby SW1. This same stack port was flapping each in each instance of the reset and was resolved by replacing the stack cable:

```
---------------------- log file: IOS ----------------------

Aug 8 21:40:14.532 UTC: %STACKMGR-1-STACK_LINK_CHANGE: STANDBY:1 stack-mgr: Stack port 1 on switch 1 is down (SW1-1)
Aug 8 21:40:17.242 UTC: %STACKMGR-1-STACK_LINK_CHANGE: STANDBY:1 stack-mgr: Stack port 1 on switch 1 is up (SW1-1)
Aug 8 21:46:11.194 UTC: %STACKMGR-1-STACK_LINK_CHANGE: 2 stack-mgr: Stack port 2 on switch 2 is down
Aug 8 21:46:12.937 UTC: %STACKMGR-1-STACK_LINK_CHANGE: 2 stack-mgr: Stack port 2 on switch 2 is up
Aug 8 21:48:23.063 UTC: %STACKMGR-1-STACK_LINK_CHANGE: 2 stack-mgr: Stack port 2 on switch 2 is down
Aug 8 21:48:24.558 UTC: %STACKMGR-1-STACK_LINK_CHANGE: 2 stack-mgr: Stack port 2 on switch 2 is up
Aug 8 21:50:40.666 UTC: %STACKMGR-6-SWITCH_REMOVED: 2 stack-mgr: Switch 1 has been removed from the stack.
Aug 8 21:50:40.671 UTC: Starting SWITCH-DELETE sequence, switch 1
```

Symptom 2) Depending on the stackwise setup is used (180, 480, plus), the number of transmission rings per port ASIC will vary. These commands will poll global registers that maintain a running total of how many read errors are seen for each transmission ring. ‘Port-asic 0’ corresponds to stack port 1 and ‘port-asic 1’ corresponds to stack port 2. This should be issued for
every switch and any signs of incrementing counts can isolate whether there maybe a problem at
the port or with the stack cable.

These can be collected several times over a few minutes to compare the deltas in the count:

show platform port-asic <0-1> read register SifRacDataCrcErrorCnt switch <switch#>

- Segment with data CRC error

show platform port-asic <0-1> read register SifRacRwCrcErrorCnt switch <switch#>

- Incremented on any failed CRC check

show platform port-asic <0-1> read register SifRacPcsCodeWordErrorCnt switch <switch#>

- Incremented on invalid PCS code, unknown PCS codeword, running disparity error is
detected

show platform port-asic <0-1> read register SifRacInvalidRingWordCnt switch <switch#>

- Bit error on stack caused ringword CRC error

The following is an example where you had a stack merge event seen both members of a 2-
member stack without any signs of a flapping stack port. You see ring[0] incrementing with CRCs
on stack port-1 of switch 1 and ended up replacing the stack cable to get past this issue.

```
3850#$show platform port-asic 0 read register SifRacRwCrcErrorCnt switch 1
Load for five secs: 11%/4%; one minute: 11%; five minutes: 12%
Time source is NTP, 14:02:49.119 EDT Thu Aug 20 2015

For asic 0

SifRacRwCrcErrorCnt on  Asic 0
[0]    count 0x00000031 <<
[1]    count 0x00000001
[2]    count 0x00000000
[3]    count 0x00000000
[4]```
Note: Depending on the register that is being looked at, the mask maybe different in each case. In the above example, the mask will wrap around on the last 14 bits. Thus, when the counter reaches 0x00003FFF, it will wrap back to 0x00000000.

Additional tips

1. Archiving Crashinfo Directories

More switches in the stack means that there will be more report files to be collected. It is easy to get overwhelmed by the number of reports that are generated. Organization is vital to isolating a failure. Find a consistency using timestamps of when each switch wrote report file for a given incident if possible. From there, ask for those very specific reports from those given switches so the customer does not upload several files. The crashinfo directory can also be archived so the customer may send a single archive containing the interested reports. The following will create an archive named 'crashinfo-archive.tar' in the flash directory:

```
F340.03.10-3800-1#archive tar /create ?
WORD Tar filename

F340.03.10-3800-1#archive tar /create crashinfo-archive.tar ?
WORD Dir to archive files from

F340.03.10-3800-1#archive tar /create crashinfo-archive.tar crashinfo ?
WORD File or Dir
<cr>

F340.03.10-3800-1#archive tar /create crashinfo-archive.tar crashinfo:
```
2. Recovering an Unstable Stack

There may be some situations where you see a several members in a stack reloading during boot up after the stack election process takes place. If a reloaded switch believes itself to be the stack master then this can often lead to a stack merge event and will enter into a boot loop state. In this situation, it may be advisable to ask the customer:

- Power down the entire stack and reseat all the stack cables firmly.
- Power-on each member switch in the stack one by one until all members have converged to its expected state.
- In cases where a member fails to join the stack, remove this from the stack and try booting this individual as a standalone to troubleshoot further.

3. Generate System Reports Manually

Manually creating system reports requires ‘service internal’ to be enabled. This will write a system report as a text file which can be done per switch basis.

```
3800-1#conf t
Enter configuration commands, one per line. End with CNTL/Z.
3800-1(config)#service internal
3800-1(config)#exit
3800-1#resource create_system_report ?
WORD system report filename
3800-1#resource create_system_report sysreport.txt ?
switch Switch number
<cr>
3800-1#resource create_system_report sysreport.txt switch ?
<1-1> Switch number
3800-1#resource create_system_report sysreport.txt switch 1 ?
<cr>
```