XC™ Series Aries Network Resiliency Guide (CLE 6.0.UP04) S-0041

(CLE 6.0.UP04)

S-0041

Contents

1 About the XC Series Aries Network Resiliency Guide (S-0041)

The *XC™ Series Aries® Network Resiliency Guide* (S-0041) describes the link resiliency and congestion protection features of the Cray® XC™ system. This guide summarizes the automatic responses to failures and describes how to identify and manage issues with routing and network congestion.

Release CLE 6.0.UP04

This version of the publication supports Cray software release 6.0.UP04 for Cray XC systems. This release includes the following new feature:

- The cable warmswap feature supports adding or removing multiple cables with a single command; see *Add or Remove High-speed Network Cables from Service* on page 12.

Title Change

This publication was previously titled *CLE XC System Network Resiliency Guide (CLE 6.0.UP01)* and *Network Resiliency for Cray XC Systems (S-0041)*.

Scope and Audience

This publication is written for experienced Cray system administrators and service personnel. For application-level information, see the *Aries™ Network Tech Note - Application Changes to Avoid Network Congestion* (S-0048).

Typographic Conventions

`Monospace` Indicates program code, reserved words, library functions, command-line prompts, screen output, file/path names, and other software constructs.

`Monospaced Bold` Indicates commands that must be entered on a command line or in response to an interactive prompt.

`Oblique` or *`Italics`* Indicates user-supplied values in commands or syntax definitions.

Proportional Bold Indicates a **GUI Window**, **GUI element**, cascading menu (**Ctrl**→**Alt**→**Delete**), or key strokes (press **Enter**).

\ (backslash) At the end of a command line, indicates the Linux® shell line continuation character (lines joined by a backslash are parsed as a single line).

Trademarks

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, Urika-GX, and YARCDATA. The following are trademarks of Cray Inc.: APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYDOC, CRAYPAT, CRAYPORT, DATAWARP, ECOPHLEX, LIBSCI,

2 About Link Resiliency

The Aries™ high-speed network (HSN) consists of lanes and channels. A lane provides bi-directional communication between two ports and a channel is three lanes. A Link Control Block (LCB) facilitates the transport of data across asynchronous chip boundaries by providing buffering and handshake support logic between higher-level network core logic and lowest-level channel protocol.

Cray XC series systems, which use the Aries™ interconnect technology, have hardware and software support that allows the system to handle certain types of hardware failures without requiring a system reboot. In addition, the same technology allows for the removal and replacement of compute blades without a system reboot. These features contribute to a reduction in both planned and unplanned system downtime.

In the case of loss of power to an Aries™ Network Card (ANC), blade, set of blades, or the warm swap out of a compute blade, applications running on the affected blades will either be killed, or in the case of a warm swap out, be allowed to complete.

Warm swap of service blades is not supported.

Recovery from link failure is handled identically for compute blades and for service blades.

There are several components to Aries™ link resiliency:

- Hardware design that permits failed link detection and corrective action
- Software (`bcnwd`) on each blade controller (BC) that detects failed links and power loss to ANCs.
- A daemon on the System Management Workstation (SMW) (`xtnlrd`) that coordinates the system response to failures.
- Another daemon on the SMW (`xthwerrlogd`) that logs hardware errors.
- An administrative command on the SMW (`xtwarmswap`) that facilitates warm swap of blades.

For more information, see the `xthwerrlogd`, `xtnlrd`, and `xtwarmswap` man pages.

When the Cray system is booted using the `xtbootsys` command, the `xtnlrd` and `xthwerrlogd` daemons are started on the SMW. Link monitoring on each BC is also enabled at this time, and link failures are logged by `xthwerrlogd` and responded to by `xtnlrd` rerouting the HSN around the failures.

2.1 Automatic Response to Failures

The following failures are handled automatically by the hardware and software supporting the Aries™ interconnect technology.

- *Automatic Recovery of a Failed Single HSN Channel*
- *Automatic Recovery of a Failed High-speed Network Cable*
- *Automatic Recovery of Power Loss to an Aries™ Network Card on a Blade*
- *Automatic Recovery of Power Loss to a Blade*

- *Automatic Recovery of Power Loss to a Cabinet*
- *Automatic xtnlrd Actions to Respond to Critical Aries™ ASIC Errors*

2.1.1 Automatic Recovery of a Failed Single HSN Channel

When a single Aries™ high-speed network (HSN) channel fails, two link control blocks (LCBs) are reported as failed by `bcnwd` on the blade controllers (BCs) at each end of the channel. The failures are reported in the `xtnlrd` log file; for example:

```
2012-11-18 23:45:51 smw 8609 cb_hw_error: failed_component c0-0c0s4a0l30, type 37,
error_code 0x6218, error_category 0x0080
2012-11-18 23:45:51 smw 8609 cb_hw_error: failed_component c0-0c0s5a0l30, type 37,
error_code 0x620a, error_category 0x0080
2012-11-18 23:45:51 smw 8609 cb_hw_error: failed_component c0-0c0s9a0l45, type 37,
error_code 0x6218, error_category 0x0080
2012-11-18 23:45:51 smw 8609 cb_hw_error: failed_component c0-0c0s9a0l45, type 37,
error_code 0x620a, error_category 0x0080
```

These failures are followed by a series of steps (shown below) as the automatic recovery actions are performed. The automatic recovery steps typically complete in about 30 seconds. Each link endpoint with a fatal error has an alert flag set to indicate that the link is not available and should be bypassed.

The automatic recovery steps are visible in the `xtnlrd` log file. In summary, these steps are:

- `initial`: Waits for failures
- `aggregate_failures`: Waits 10 seconds by default for any more links to fail
- `link_failed`: Begins to process the failed links
- `alive`: Determines which blades are alive

  When power loss to a blade occurs, no response to the `alive` request is received; instead a 30-second timeout occurs for the blade that lost power.

  When power loss to a cabinet occurs, no response to the `alive` request is received; instead cabinet power loss results in a maximum of 960 LCB failures being reported to `xtnlrd`. The exact number depends on the topology class.
- `route_compute`: Computes and stages new routes to the BCs
- `quiesce`: Stops all high-speed network traffic temporarily
- `switch_netwatch`: Causes the `netwatch` daemon (`bcnwd`) on the blades to use the new routes file
- `down_unused_links`: Causes links that are up but unused to be taken down
- `route_install`: Asserts new routes in the Aries™ chips
- `unquiesce`: Resumes all high-speed network traffic
- `finish`: Performs final cleanup
- `initial`: Waits for failures (process restarts)

2.1.2 Automatic Recovery of a Failed High-speed Network Cable

Aries™ network connections are divided into three groups: green links (15 per Aries ASIC), black links (15 per Aries ASIC), and blue links (10 per Aries ASIC). Green links are electrical connections over the backplane to other Aries in the same chassis. Black links are electrical connections over cables to other chassis within the same cabinet group. Blue links are optical connections over fiber to other cabinet groups.

Loss of a black network cable results in the failure of at most 6 LCBs, whereas loss of a blue network cable results in the failure of at most 8 LCBs. These appear in the `xtnlrd` log file as two entries per failed LCB, and each of the failed LCBs has an alert flag set. These failures are followed by a series of steps (shown below) as the automatic recovery actions are performed. The automatic recovery steps typically complete in about 30 seconds.

The automatic recovery steps are visible in the `xtnlrd` log file. In summary, these steps are:

- `initial`: Waits for failures
- `aggregate_failures`: Waits 10 seconds by default for any more links to fail
- `link_failed`: Begins to process the failed links
- `alive`: Determines which blades are alive

  When power loss to a blade occurs, no response to the `alive` request is received; instead a 30-second timeout occurs for the blade that lost power.

  When power loss to a cabinet occurs, no response to the `alive` request is received; instead cabinet power loss results in a maximum of 960 LCB failures being reported to `xtnlrd`. The exact number depends on the topology class.
- `route_compute`: Computes and stages new routes to the BCs
- `quiesce`: Stops all high-speed network traffic temporarily
- `switch_netwatch`: Causes the `netwatch` daemon (`bcnwd`) on the blades to use the new routes file
- `down_unused_links`: Causes links that are up but unused to be taken down
- `route_install`: Asserts new routes in the Aries™ chips
- `unquiesce`: Resumes all high-speed network traffic
- `finish`: Performs final cleanup
- `initial`: Waits for failures (process restarts)

2.1.3 Automatic Recovery of Power Loss to an Aries™ Network Card on a Blade

Loss of power to an Aries™ Network Card (ANC) results in LCB failures (at most 25 for a class 0, or mini, topology, or at most 40 for a class 2 topology) being reported by `xtnlrd`. Although the endpoints on the blade whose ANC lost power are not reported as failed, the ANC on that blade is reported and alert flags are set on both the ANC and all LCBs that were reported as failed. This results in the entire blade being routed around.

The following output appears in the `xtnlrd` log file, including a report of the blade that lost ANC power.

```
2012-10-26 16:20:21 smw 52404 ***** dispatch: current_state alive *****
2012-10-26 16:20:21 smw 52404 INFO: 47 out of 48 L0s are alive
2012-10-26 16:20:21 smw 52404 Beginning to wait for response(s)
2012-10-26 16:20:21 smw 52404 svid 0:c0-0c0s8 (131282: Mezzanine Voltage Fault)
2012-10-26 16:20:21 smw 52404 Received 47 of 47 responses
2012-10-26 16:20:21 smw 52404 ***** dispatch: current_state check_alive *****
2012-10-26 16:20:21 smw 52404 MODULE ERROR: c0-0c0s8 - 210 - Mezzanine Voltage
Fault
2012-10-26 16:20:21 smw 52404 add_blade_to_list: adding blade c0-0c0s8 to
bladefailed list
```

The steps that are automatically performed to reroute the HSN are provided below. The automatic recovery steps typically complete in about 30 seconds.

The automatic recovery steps are visible in the `xtnlrd` log file. In summary, these steps are:

- `initial`: Waits for failures
- `aggregate_failures`: Waits 10 seconds by default for any more links to fail
- `link_failed`: Begins to process the failed links
- `alive`: Determines which blades are alive

  When power loss to a blade occurs, no response to the `alive` request is received; instead a 30-second timeout occurs for the blade that lost power.

  When power loss to a cabinet occurs, no response to the `alive` request is received; instead cabinet power loss results in a maximum of 960 LCB failures being reported to `xtnlrd`. The exact number depends on the topology class.
- `route_compute`: Computes and stages new routes to the BCs
- `quiesce`: Stops all high-speed network traffic temporarily
- `switch_netwatch`: Causes the `netwatch` daemon (`bcnwd`) on the blades to use the new routes file
- `down_unused_links`: Causes links that are up but unused to be taken down
- `route_install`: Asserts new routes in the Aries™ chips
- `unquiesce`: Resumes all high-speed network traffic
- `finish`: Performs final cleanup
- `initial`: Waits for failures (process restarts)

2.1.4 Automatic Recovery of Power Loss to a Blade

Blade power loss appears very similar to an Aries™ Network Card (ANC) power loss, except that no response to the `alive` request is received; instead a 30-second timeout occurs for the blade that lost power. The `alive` stage of the recovery process shows a timeout for the blade that lost power, such as:

```
2012-10-26 15:48:10 smw 52404 generic_rsp_timeout: ERROR: Did not receive responses
from
the following L0s: c0-0c0s9
2012-10-26 15:48:10 smw 52404 ***** dispatch: current_state check_alive *****
2012-10-26 15:48:10 smw 52404 add_blade_to_failed_list: adding blade c0-0c0s9 to
bladefailed list
```

The automatic recovery steps that are performed are provided below. Due to the timeout, which is 30 seconds by default, recovery from a failed blade typically takes about 60 seconds.

The automatic recovery steps are visible in the `xtnlrd` log file. In summary, these steps are:

- `initial`: Waits for failures
- `aggregate_failures`: Waits 10 seconds by default for any more links to fail
- `link_failed`: Begins to process the failed links
- `alive`: Determines which blades are alive

  When power loss to a blade occurs, no response to the `alive` request is received; instead a 30-second timeout occurs for the blade that lost power.

  When power loss to a cabinet occurs, no response to the `alive` request is received; instead cabinet power loss results in a maximum of 960 LCB failures being reported to `xtnlrd`. The exact number depends on the topology class.
- `route_compute`: Computes and stages new routes to the BCs

- quiesce: Stops all high-speed network traffic temporarily
- switch_netwatch: Causes the netwatch daemon (bcnwd) on the blades to use the new routes file
- down_unused_links: Causes links that are up but unused to be taken down
- route_install: Asserts new routes in the Aries™ chips
- unquiesce: Resumes all high-speed network traffic
- finish: Performs final cleanup
- initial: Waits for failures (process restarts)

2.1.5 Automatic Recovery of Power Loss to a Cabinet

Cabinet power loss is the most widespread single high-speed network event that is handled by xtnlrd. This case is very similar to *Automatic Recovery of Power Loss to a Blade* on page 8, but rather than a single blade, cabinet power loss results in a maximum of 960 link control block (LCB) failures being reported to xtnlrd. The exact number depends on the topology class.

Each of these LCBs is marked with an alert flag as part of the recovery process. In addition, the Aries™ chip on each blade in the failed cabinet is marked with an alert flag. During the recovery process, all blades in the failed cabinet time out and are removed from routing.

The automatic recovery steps that are performed are provided below. Due to the timeout, which is 30 seconds by default, recovery from a failed cabinet typically takes about 60 seconds.

The automatic recovery steps are visible in the xtnlrd log file. In summary, these steps are:

- initial: Waits for failures
- aggregate_failures: Waits 10 seconds by default for any more links to fail
- link_failed: Begins to process the failed links
- alive: Determines which blades are alive

  When power loss to a blade occurs, no response to the alive request is received; instead a 30-second timeout occurs for the blade that lost power.

  When power loss to a cabinet occurs, no response to the alive request is received; instead cabinet power loss results in a maximum of 960 LCB failures being reported to xtnlrd. The exact number depends on the topology class.
- route_compute: Computes and stages new routes to the BCs
- quiesce: Stops all high-speed network traffic temporarily
- switch_netwatch: Causes the netwatch daemon (bcnwd) on the blades to use the new routes file
- down_unused_links: Causes links that are up but unused to be taken down
- route_install: Asserts new routes in the Aries™ chips
- unquiesce: Resumes all high-speed network traffic
- finish: Performs final cleanup
- initial: Waits for failures (process restarts)

2.1.6 Automatic xtnlrd Actions to Respond to Critical Aries™ ASIC Errors

In addition to link failure due to a bad Link Control Block (LCB), cable, Aries™ Network Card (ANC), blade, or cabinet, critical errors on the Aries ASIC are also monitored by `bcnwd` and responded to by `xtnlrd`. Depending on which logic block in the Aries reported the critical error, different actions are taken by `xtnlrd`. The following table explains the various actions:

Table 1. Automatic xtnlrd Actions to Respond to Critical Aries™ ASIC Errors

Block	Component	Action	Note
NIC	`rt_node`	Alert on node	
PI	`rt_node`	Alert on node	
LB_NIC	`rt_node`	Alert on node	Some LB errors related to a specific NIC or PI, so for these, an alert is set on the node.
LB	`rt_node`	Route around Aries	Other LB errors are Aries-wide
NL	`rt_node`	Route around Aries	Netlink errors can affect the whole Aries, but significant error analysis needs to be done to determine the scope of each NL error.
NT	`rt_aries_lcb`	Route around failed link	Do not consult the critical error table.
PT	`rt_aries_lcb`	Route around Aries	Ptile errors are not necessarily internal to that group of Ptiles. Need to distinguish between a critical Ntile error (row 0-4) where we just route around a bad link, and a critical Ptile error (row 5) where we route around the bad Aries.
RTR	`rt_aries`	Route around Aries	Router errors affect the whole Aries.
PCIe	`rt_node` `rt_aries_nic`	Alert on node	

2.2 Identify Lane Degrades

Each High Speed Network (HSN) link is comprised of three SerDes lanes in each direction. Under normal circumstances, all three lanes are operational and running cleanly. The Aries™ SerDes and LCB hardware do a very good job of keeping the lanes and links alive, even in the presence of soft errors. There can, however, be cases where lane hardware is faulty, such as a bad cable. In those cases, one or more lanes can give up and shut down due to excessive errors. This is due to bad signal paths in the HSN hardware. The link itself will continue to run at reduced bandwidth as long as there is at least one lane running. If a lane degrades, the hardware involved needs to be serviced.

Use the `xtnetwatch` log to identify degraded lanes. In this example, `lanemask=6`, which means that lane 0 is down.

```
130625 09:41:35  ##############  ##############  ###############  ################
130625 09:41:35  LCB ID          Peer LCB        Category         Description
130625 09:41:35  ##############  ##############  ###############  ##############
130625 13:30:18  c0-0c2s10a0l23  c1-0c1s10a0l22  Info             RX Lane 
Degrade, 
lanemask: 6
130625 13:30:18  c0-0c2s10a0l23  c1-0c1s10a0l22  Info             TX Lane 
Degrade, 
lanemask: 6
```

2.3 Update the System Configuration While the System Is Booted

To change the system configuration physically while the system is booted, use the `xtwarmswap` command to remove or add one or more blades or high-speed network (HSN) cables.

⚠ **CAUTION:** When reserving nodes for maintenance, an `admindown` of any node in use by a current batch job can cause a subsequent `aprun` in the job to fail. Instead, it is recommended that a batch subsystem be used to first reserve nodes for maintenance, and then verify that a node is not in use by a batch job prior to setting a node to `admindown`. Contact a Cray service representative to reserve nodes for maintenance.

The `xtwarmswap` command runs on the SMW and coordinates with the `xtnlrd` daemon to take the necessary steps to perform warm swap operations. For additional information, see the `xtwarmswap(8)` man page.

2.3.1 Reuse One or More Previously-failed HSN Links

About this task

To integrate failed links back into the HSN configuration, the `xtwarmswap` command may be invoked with one of the following:

- `-s` *`LCB, ...`*, specifying the list of LCBs to bring back up
- `-s all`, to bring in all available LCBs
- `-s none`, to cause a reroute without changing the LCBs that are in use

Procedure

1. Execute an `xtwarmswap -s` *`LCB_names`* `-p` *`partition_name`* to tell the system to reroute the HSN using the specified set of LCBs in addition to those that are currently in use.

   Doing so will clear the alert flags on the specified LCBs automatically. If the warm swap fails, the alert flag will be restored to the specified LCBs.

2. Execute an `xtwarmswap -s all -p` *`partition_name`* command to tell the system to reroute the HSN using all available links.

The xtwarmswap command results in xtnlrd performing the same link recovery steps as for a failed link, but with two differences: no alert flags are set, and an init_new_links and a reset_new_links step are performed to initialize both ends of any links to be used, before new routes are asserted into the Aries™ routing tables.

The elapsed time for the warm swap synchronization operation is typically about 30 seconds.

2.3.2 Reuse One or More Previously-failed Blades, ANCs, or Cabinets

About this task

Failed blades have alert flags set on the ASICs and the LCBs. These alert flags must be cleared before the blade, ANCs, or cabinets can be reused.

Perform an xtwarmswap --add operation to bring the blades back into the HSN configuration. Doing so clears any alert flags on the added blades and LCBs relating to those blades, initializes the blades, runs the BIOS on the nodes, and initializes the links to the blades.

Procedure

1. Ensure that blades/ANCs/cabinets have power.

2. Ensure that an xtalive command to all required blades succeeds.

3. Add the blade(s) to the HSN by executing the xtwarmswap --add *blade_ID*,... command. Note that this command automatically executes a mini-xtdiscover command after the warm swap steps have completed successfully. No manual invocation of xtdiscover, which gets the new hardware attributes from the added blades, is necessary.

   Because the xtwarmswap --add command initializes the added blades, the time to return the blades back to service is about 10 minutes, including the time to initialize the blades, run the BIOS on the nodes, and initialize the links to the blades.

4. Boot the nodes on the blade(s) by executing the xtcli boot CNL0 *blade_ID,...* command on the SMW.

2.3.3 Add or Remove High-speed Network Cables from Service

To specify one or more high-speed network (HSN) cables to add or remove from service, use the xtwarmswap --add-cable command or the xtwarmswap --remove-cable command, respectively. These options provide the ability to replace one or more cables without removing blades or shutting down the system. The routing of the Cray HSN will be updated to route around the removed cable or cables.

To add or remove a single HSN cable, specify one *cable* argument as in this --add-cable example:

```
# xtwarmswap --add-cable cable
```

To add or remove multiple HSN cables, specify a comma-separated list of cables, as in this --remove-cable example:

```
# xtwarmswap --remove-cable cable1,cable2,...,cableN
```

The `--add-cable` and `--remove-cable` options are not supported if more than a single active partition exists in the system. Do not specify the `-p|--partition` option when using these options. In addition, do not use the `--linktune` option when using the `--remove-cable` option.

2.3.4 Remove a Compute Blade from Service While the System is Running

About this task

A compute blade can be physically removed for maintenance or replacement while the system is running; however, the applications using the nodes on the blade to be removed must be allowed to drain, or be killed beforehand.

Before starting this warm swap procedure, verify that the proposed system configuration is routable. Doing this in advance of idling the nodes on the blades to be removed provides assurance that a valid set of nodes is being taken out of service before affecting the system. Log on to the SMW as `crayadm` and execute the following command, where p*N* is the partition from which the blades are being removed:

```
smw:~> rtr -S --id test --remove=blade_ID pN
```

CAUTION: This procedure warm swaps a compute blade from service while the system is running. Do not warm swap service blades, unless the blade is an I/O base blade (IBB) that has InfiniBand cards and is an LNET blade. Before attempting to warm swap any service blade, it is advisable to consult with a Cray service representative.

Procedure

1. Log on to the login node as `root`.

2. Ensure that the batch system or Slurm marks the blade as unavailable for scheduling.

3. Execute the following command to mark the nodes on the compute blade as `admindown`. This tells ALPS not to launch new applications onto them. (This command may also be executed from the boot node as user `root`.)

   ```
   login:~ # xtprocadmin -k s admindown -n blade_ID
   ```

   The arguments to the `-n` option should be the NID values for the nodes on the blade being removed, as shown by executing `xtprocadmin | grep` *bladename*.

   For example, to find the NID values for the nodes on the blade `c0-0c0s2` being removed:

   ```
   login:~ # xtprocadmin | grep c0-0c0s2
        8      0x8  c0-0c0s2n0  compute      up       batch
        9      0x9  c0-0c0s2n1  compute      up       batch
       10      0xa  c0-0c0s2n2  compute      up       batch
       11      0xb  c0-0c0s2n3  compute      up       batch
   ```

4. From the login node, execute the `apstat -n` command or the appropriate Slurm command to determine if any applications are running on the node marked `admindown`. This example shows that `apid 675722` is running on all nodes of blade `c0-0c0s2`.

```
login:~ # apstat -n | egrep -w 'NID|8|9|10|11
  8   XT UP  B 32 32  1    4K 16777216 8388608  262144    1 675722
  9   XT UP  B 32 32  1    4K 16777216 8388608  262144    1 675722
 10   XT UP  B 32 32  1    4K 16777216 8388608  262144    1 675722
 11   XT UP  B 32 32  1    4K 16777216 8388608  262144    1 675722
```

5. Wait until the applications using the nodes on the blade finish or use the `apkill` *`apid`* command to kill the application.

6. Log on to the SMW as `crayadm`.

7. Execute the `xtcli halt` *`blade_ID`* command to halt the blade.

   ```
   smw:~> xtcli halt blade_ID
   ```

8. Execute the `xtwarmswap --remove` *`blade_ID`* command to remove the compute blade from service. The routing of the Cray HSN will be updated to route around the removed blade.

   The `--remove` stage of the `xtwarmswap` process uses the Aries™ resiliency infrastructure and takes about 30 seconds to complete.

   ```
   smw:~> xtwarmswap --remove blade_ID
   ```

9. Execute the `xtcli power down` *`blade_ID`* command, which helps to identify which blade to pull (all lights are off on the blade).

   ```
   smw:~> xtcli power down blade_ID
   ```

10. Physically remove the blade, if desired. To complete this step, see the hardware maintenance and replacement procedures documentation for the Cray system, or contact a Cray Service representative.

    CAUTION: If a blade cannot be reinstalled in the empty slot within 2 minutes, install a filler blade assembly in the empty slot; failure to do so can cause other blades in the system to overheat.

2.3.5 Return a Compute Blade into Service

About this task

After a blade has been repaired or when a replacement blade is available, use the following procedure to return the blade into service.

Procedure

1. Physically insert the blade into the slot. To complete this step, see the hardware maintenance and replacement procedures documentation for the Cray system, or contact a Cray Service representative.

2. On the SMW, execute the `xtcli power up` *`blade_ID`* command.

   ```
   smw:~> xtcli power up blade_ID
   ```

3. Ensure that the blade is ready by entering the following command, and wait until the command returns the correct response:

```
smw:~> xtalive blade_ID
The expected response was received.
```

4. Verify the status of the blade controller to ensure that its "Comp state" is "up" and that there are no flags set.

   ```
   smw:~> xtcli status -t bc blade_ID
   ```

5. Bounce the blade.

   ```
   smw:~> xtbounce blade_ID
   ```

6. If the blade or PDC type is different, `su` to `root`, execute the `xtdiscover` command, and then exit `root`. Otherwise, skip this step.

   ```
   smw:~> su - root
   smw:~> xtdiscover
   smw:~> exit
   smw:~>
   ```

7. Execute the `xtzap --blade` command to update the BC BIOS, node BIOS, microcontroller, and FPGAs as required.

   ```
   smw:~> xtzap --blade blade_cname
   ```

8. Execute the `xtbounce --linkdown` *`blade_ID`* command to prepare the blade for the warm swap (takes down all HSN links on the blade).

   ```
   smw:~> xtbounce --linkdown blade_ID
   ```

9. Add the blade(s) to the HSN by executing the `xtwarmswap --add` *`blade_ID`*`,...` command. This command activates routing on the newly installed blade and automatically executes a mini-`xtdiscover` command once the warm swap steps have completed successfully. No additional manual invocation of `xtdiscover`, which gets the new hardware attributes from the added blades, is necessary.

   ```
   smw:~> xtwarmswap --add blade_ID
   ```

   Because the `xtwarmswap --add` command initializes the added blades, the time to return the blades back to service is about 10 minutes, including the time to initialize the blades, run the BIOS on the nodes, and initialize the links to the blades.

10. Boot the nodes on the blade(s) by executing the `xtcli boot CNL0` *`blade_ID`*`,...` command on the SMW.

    ```
    smw:~> xtcli boot CNL0 blade_ID
    ```

11. As `root` on the login node, execute the following command to mark the nodes on the compute blade as up. This tells ALPS that new applications may be launched onto those nodes. (This command may also be executed from the boot node as user `root`.)

    ```
    login:~ # xtprocadmin -k s up -n blade_ID
    ```

12. Verify that the blade is up.

    ```
    login:~ # xtprocadmin | grep blade_ID
    ```

13. Ensure that the batch system or Slurm marks the blade as available for scheduling.

2.4 Identify Unrouteable Configurations

Cray XC series systems have a dragonfly network topology that provides for less restrictive connectivity than a torus based network topology. Because of this, it is expected that there will also be fewer unrouteable configurations. The main requirement for any Cray XC series system configuration to be routeable is that it must be fully connected. Another characteristic of the configuration that is not strictly required, but is highly desirable, is that every group is connected to every other group. If this is not the case, performance between some pairs of groups can be greatly impacted.

In other cases however, the configuration per se is fine, but one blade is unable to complete the request due to some unexpected failure specific to the blade (or rarely, to multiple blades). Such blades are typically operating fine otherwise, and may be actively running jobs, but cannot complete the new routing operation. This can happen with any routing attempt; that is, it is not limited to the removal of components.

If a warmswap operation fails during the test reroute stage, then something is preventing the operation from completing due to a failure to route the proposed new (that is, post-warmswap) configuration, and the following output is displayed, as in this example of a warmswap remove operation. Note that if a large number of blades encounter problems, the warmswap output might display only an overall error count.

```
smw:~# xtwarmswap --remove c0-0c2s15
Adding 0 blades and removing 1 blade in partition p0
Removing blades:
   c0-0c2s15

Sending command to xtnlrd
17:43:35(T+00:00) Warm swap beginning
17:43:35(T+00:00) Setting alert flag on blades being removed
17:43:35(T+00:00) Turning link monitoring off for blades being removed
17:43:35(T+00:00)     Link monitoring turned off for blades being removed
17:43:35(T+00:00) Testing for routeability
17:43:36(T+00:01)     Test reroute proceeding...
17:43:46(T+00:11)     Test reroute proceeding...
17:43:56(T+00:21)     Test reroute proceeding...
17:44:06(T+00:31)     Test reroute proceeding...
17:44:16(T+00:41)     Test reroute proceeding...
17:44:26(T+00:51)     Test reroute proceeding...
17:44:35(T+01:00)     ERROR: test route exited with an error status
17:44:35(T+01:00)     Timeouts were reported on the following blade(s):
17:44:35(T+01:00)         c0-0c2s13
17:44:35(T+01:00) Clearing alert flags for blades that could not be removed
17:44:35(T+01:00)     Clearing alert flags on blades that failed to be removed
17:44:35(T+01:00) Turning link monitoring on for blades that could not be removed
17:44:35(T+01:00) Cleaning up

Daemon reported errno 11 (Resource temporarily unavailable)
Error text:
Error during route computation
1 of 45 blades reported timeouts during routing.
You may need to reboot these blade controllers, or warmswap the blades out,
before trying again.

FAILURE: Warm swap command failed.
```

A failure could also occur during a warmswap add operation; for example:

```
smw:~# xtwarmswap --add c0-0c2s15
Adding 1 blade and removing 0 blades in partition p0
Adding blades:
    c0-0c2s15

Sending command to xtnlrd
17:48:59(T+00:00) Warm swap beginning
17:48:59(T+00:00) Clearing alert flags on blades being added
17:49:00(T+00:01)     Finished clearing link alerts
17:49:00(T+00:01) Testing for routeability
17:49:01(T+00:01)     ERROR: test route exited with an error status
17:49:01(T+00:01)     Failures were reported on the following blade(s):
17:49:01(T+00:01)         c0-0c2s13
17:49:01(T+00:01) Marking HSN links down on blades that could not be added
17:49:01(T+00:01) Correcting alert flags for blades or LCBs that could not be added
or removed
17:49:01(T+00:01)     Restoring alert flags for blades that failed to be added
17:49:01(T+00:01) Cleaning up

Daemon reported errno 11 (Resource temporarily unavailable)
Error text:
Error during route computation
1 of 45 blades reported errors during routing.
You may need to reboot these blade controllers, or warmswap the blades out,
before trying again.

FAILURE: Warm swap command failed.
```

Usually, this type of failure will result in the blades being automatically removed from the system if a critical resiliency operation is triggered to address a failed component. However, this is not the case for warmswap operations, as manual recovery may be possible without the blade's removal, allowing running jobs to be preserved and the blade to be left in the system. For the blade or blades that were reported as having failed or timed out, see *Recover a Blade That Failed to Respond to the Reroute Request* on page 17.

Note that these types of failures are normally expected for one or a very small number of blades. If large numbers of blades are indicated to have failed, then a more general routing issue may be the ultimate cause instead. This may be an indirect indication that the new configuration is not routable or may indicate a more general system issue.

If a routing issue appears to have occurred, additional information may be found in the output of the `rtr` command corresponding to the failed test reroute. This output is recorded in the `xtnlrd` (`nlrd-`*`YYYYMMDD`*) log. Use the following command to display the relevant log entries:

```
smw:~# grep "CMD: rtr:" nlrd-YYYYMMDD
CMD: rtr: WARNING: c0-0c2s15: ERROR: <error-string>; procedure aborted
CMD: rtr: ERROR: 1 errors, rtr failed
CMD: rtr: WARNING: c0-0c0s4: bcrtr process died due to signal 6"
CMD: rtr: ERROR: 1 errors, rtr failed
CMD: rtr: WARNING: c0-0c0s1: bcrtr process exited abnormally with status 14"
CMD: rtr: ERROR: 1 errors, rtr failed
CMD: rtr: ERROR: Timeout while executing bcrtr -f /var/tmp/rtr-cfg.test -S --id
test --instance=0x82e5, didn't hear from 1 BC: c0-0c2s6
```

2.4.1 Recover a Blade That Failed to Respond to the Reroute Request

Prerequisites

- The xtwarmswap command was executed and failed during the test reroute stage, and an otherwise routable individual blade (or blades) fails to respond to the reroute request.
- The failed blade was identified in the output of the failing xtwarmswap command.

About this task

For warmswap operations, a manual recovery of a blade may be possible without the blade's removal, allowing running jobs to be preserved and the blade to be left in the system.

Procedure

1. Reboot the blade controllers for the blade or blades reporting problems.

   ```
   smw:~# xtrsh -m s -l root -s reboot
   comma-separated list of blade controller names
   ```

2. Rerun the previously-failing xtwarmswap command.

   ```
   smw:~# xtwarmswap --remove c0-0c2s15
   ```

3. If the command fails a second time due to a routing failure, note the still-failing blade or blades reported in the xtwarmswap output. Then remove these blades with xtwarmswap --remove before reattempting the original warmswap operation.

2.5 Link Resiliency Parameters

Parameters in the xtnlrd.ini file control the system response to failures of link resiliency on the system.

aggregate_timeout

Seconds to wait after a link failure occurs before taking action to route around it, to allow any other link failures to arrive (default is 10 seconds)

alive_timeout

Seconds to wait for alive responses from blades (default is 30 seconds)

init_new_blades_timeout

Seconds to wait for blade initialization to finish (default is 1200 seconds)

init_new_links_timeout

Seconds to wait for link initialization to finish (default is 600 seconds)

quiesce_timeout

Seconds to wait for quiesce to finish (default is 120 seconds)

reroute_l0_timeout

Seconds to wait for route requests to BCs to time out (default is 60 seconds) `reroute_retries`

Number of route command retries on error (default is `1`)

`reroute_retry_wait`

Seconds between route command retries on failure (default is `5` seconds)

`reroute_smw_timeout`

Seconds to wait for route requests to the SMW to time out (default is `15` seconds)

`route_compute_timeout`

Seconds to wait for route computation to finish (default is `300` seconds)

`route_install_timeout`

Seconds to wait for route installation to finish (default is `60` seconds)

`unquiesce_timeout`

Seconds to wait for unquiesce to finish (default is `120` seconds)

`userexit_on_failure`

Calls user exit script `/opt/cray/hss/default/etc/xtnlrd.userexit` if link recovery fails (default is `y`)

3 About Network Congestion

Network congestion is a condition that occurs when the volume of the traffic on the high-speed network (HSN) exceeds the network's capacity to handle it. This causes traffic in the network to stall and make extremely slow forward progress. In and of itself, network congestion is not a problem and is not unique to any particular network. However, in the rare case of extreme congestion, problems can arise that may affect system performance.

On the Aries™ network, packets must reach their destinations quickly enough to avoid various higher level timeouts and to enable reasonable transit latencies. The Outstanding Request Buffer (ORB) is a logic block in the Aries ASIC that tracks sent packets. The ORB waits for response packets to match the extant entries. If the sending ORB does not receive a response packet within some number of seconds then a timeout occurs. Under normal conditions, this implies a lost packet that is then reported to higher-level clients for handling. If a timeout occurs, the Hardware Supervisory System (HSS) on the blade detects the condition and executes an algorithm to safely scrub the timed-out ORB entries. The scrub algorithm includes a 10-second quiesce of the Network Interface Card (NIC) whose ORB indicated a timeout condition. When the HSN experiences congestion, however, a timeout may not actually reflect a lost packet. Instead, in a congested network, the response packet's latency will increase, which is the motivation for quiescing the NIC to ensure only truly lost packets get reported.

3.1 Network Congestion Causes

Congestion on the Aries™ network may result from a variety of causes. Congestion occurs most often when application programmers use one-sided programming models such as PGAS (Partitioned Global Address Space) and Cray SHMEM with applications that perform many- or all-to-one communication. Missing or compromised lanes, routes, links and channels on the HSN can also cause congestion. It is possible that the Lustre Network Driver (`kgnilnd`) or any of the higher level software that relies on it (e.g., Lustre and Data Virtualization Services) can cause congestion.

3.2 Congestion Management Software Components

To manage congestion on the Aries™ network, HSS software monitors and throttles traffic injected into the network when necessary. Congestion protection is implemented by two daemons: one on the SMW, called `xtnlrd`, and one on the BC controllers, called `bcbwtd`. It limits the aggregate injection bandwidth across all compute nodes to less than the ejection bandwidth of a single node. This alleviates congestion in the system, which in turn has the effect of reducing network latency. This trade-off is acceptable because Congestion Protection is active only in cases of extreme congestion.

3.3 How Congestion Management Software Detects Network Congestion

The `bcbwtd` daemon running on the BC monitors the percentage of time that traffic trying to enter the network from the nodes attached to the BC is stalled and the percentage of time network tiles are stalled. When these percentages cross a high water mark threshold, `bcbwtd` communicates that information to the `xtnlrd` daemon running on the SMW. After the congestion subsides, `bcbwtd` again relays this information to the SMW. When the SMW receives a number of notifications of congestion sufficient to cross another high water mark threshold, Congestion Protection is enabled.

Congestion protection remains active until congestion subsides and the number of congested tiles and nodes drops below their respective low water mark thresholds. If an application (or combination of applications) persistently congests the network, Congestion Protection is reapplied and released periodically until the application terminates.

3.4 Congestion Management Log File and Messages

The daemon handling congestion management on the SMW, called `xtnlrd`, produces a log file that includes:

- Following an increase in overall system congestion, a chronological, component-based history of each congestion event is generated, ranging from the first detection of congestion to the time total system congestion again returns to zero (non-increases are omitted to help readability; that is, only up-ticks are logged).
- Because once congestion begins, it can spread to its neighbors quickly, this history is displayed in the order in which `xtnlrd` receives the congestion events from the individual blades. The congestion history is logged for all incidents of detected congestion, even those that do not reach the threshold for throttling the system. This history includes the network ASIC reporting the congestion, how much was reported, and which NICs were congested, as well as the resulting effect on the overall system tile and NIC congestion. The congestion history is displayed following the system's return to zero overall reported congestion.
- Events received from the Application Level Placement Scheduler (ALPS) when an application starts and ends.
- A periodic update showing all applications that are running, repeated every five minutes, or whenever a congestion protection event occurs.
- A list of the top 10 applications by aggregate ejection bandwidth (e.g., the per-node ejection bandwidth summed across all nodes in the application) whenever a congestion protection event occurs.
- A list of the top 10 nodes by ejection flit counts whenever a congestion protection event occurs, which is useful for identifying certain congesting many-to-few communications patterns.
- The applications running on those nodes, including APID, number of nodes, the user ID, and the application name.

The `xtnlrd` log file is located in `/var/opt/cray/log/`*session-id*`/nlrd.`*session-id*.

The following log messages are produced in the `xtnlrd` log file when a throttle event is triggered and additional information is gathered about the event:

```
2012-02-09 19:51:01 smw check_for_throttle_action: tile_weight 32 nic_weight 1
2012-02-09 19:51:01 smw do_throttle: Telling all blades to throttle network
bandwidth
2012-02-09 19:51:01 smw _do_throttle: sending throttle mask 0xf to 1632 blades
2012-02-09 19:51:06 smw Executing: Bandwidth data gathering
```

```
2012-02-09 19:51:11 smw Command succeeded.
2012-02-09 19:51:11 smw Executing: Unthrottle service blades only
2012-02-09 19:51:13 smw Command succeeded.
```

In addition, any time a congestion protection event occurs, a message like the following is inserted by `aprun` into `/var/opt/cray/log/p`*`n`*`-current/messages-`*`YYYYMMDD`*. A similar message is also written to the user's standard out:

```
Feb 9 11:32:40 nid00002 aprun[14718]: apid=6204, Network throttled,
user=10320, batch_id=unknown, nodes_throttled=20, node_seconds=00:01:50
```

3.5 Send an Event to ALPS to Terminate the Top Network Congestion Candidate Application

Use `xtnlrd`, which is the daemon on the SMW that handles congestion management, to send an event to ALPS to terminate the top congestion candidate based on the worst (individual) node's ejection flit counts after the application appears *`N`* consecutive times at the top of the list of nodes. However, this feature carries the risk that innocent applications may be terminated before ultimately killing the congesting application, due to the possibility of other congestion patterns that may not be unambiguously detected by this scheme. Consequently, the default value is n (no).

To use this feature, change the `bw_congest_kill_top_app` parameter in the `/opt/cray/hss/default/etc/xtnlrd.ini` file to y (yes) and restart `xtnlrd` to enable it. `xtnlrd` can be restarted by killing all `xtnlrd` processes and executing `xtbootsys --start-daemons pN`, where *`N`* is the partition number (see *SMW Network Congestion Parameters* on page 27). This functionality does not involve the top 10 applications by aggregate bandwidth, only the top 10 by ejection flit count. The former is a completely separate metric.

Example: `xtnlrd` messages showing top congested nodes

The following messages provide a sample snapshot of the 10 nodes where congestion appears to be highest, based on the assumption that the congestion has been caused by a many-to-few communications pattern, including the application at the top of the list. More information about interpreting the flit count values found in this listing is provided in *Identify the User Application that Caused Excessive Network Congestion* on page 24.

```
2012-02-09 19:51:13 smw responder_work_func: Top 10 nodes involved with network congestion
2012-02-09 19:51:13 smw Congestion candidate node 1: c8-1c0s7n1 (1237956 flits/sec)
(nid 1871; apid 384488 userid 51668 numnids 1024 apname xyz)
2012-02-09 19:51:13 smw Congestion candidate node 2: c3-0c0s6n2 (89834 flits/sec)
(nid 6060; apid 384498 userid 35032 numnids 128 apname pqr)
2012-02-09 19:51:13 smw Congestion candidate node 3: c3-0c0s6n3 (74578 flits/sec)
(nid 6061; apid 384498 userid 35032 numnids 128 apname pqr)
2012-02-09 19:51:13 smw Congestion candidate node 4: c1-0c0s6n2 (66725 flits/sec)
(nid 6226; apid 384498 userid 35032 numnids 128 apname pqr)
2012-02-09 19:51:13 smw Congestion candidate node 5: c3-0c0s6n1 (64722 flits/sec)
(nid 6131; apid 384498 userid 35032 numnids 128 apname pqr)
2012-02-09 19:51:13 smw Congestion candidate node 6: c8-1c0s7n0 (61527 flits/sec)
(nid 1870; apid 384486 userid 51668 numnids 1024 apname xyz)
2012-02-09 19:51:13 smw Congestion candidate node 7: c3-0c1s7n2 (50087 flits/sec)
(nid 6064; apid 384498 userid 35032 numnids 128 apname pqr)
2012-02-09 19:51:13 smw Congestion candidate node 8: c1-0c0s6n0 (46784 flits/sec)
(nid 6156; apid 384498 userid 35032 numnids 128 apname pqr)
2012-02-09 19:51:13 smw Congestion candidate node 9: c1-0c0s6n1 (41545 flits/sec)
(nid 6157; apid 384498 userid 35032 numnids 128 apname pqr)
2012-02-09 19:51:13 smw Congestion candidate node 10: c3-0c0s6n0 (41137 flits/sec)
(nid 6130; apid 384498 userid 35032 numnids 128 apname pqr)
2012-02-09 19:51:13 smw responder_work_func: found new apid 384488 userid 51668
numnids 1024 apname xyz at top of congestion list since last time we throttled
```

Example: xtnlrd messages when an application is killed

The following messages demonstrate xtnlrd sending an event to ALPS to kill an application after it appeared at the top of the congestion list two consecutive times.

```
2012-02-09 19:51:53 smw Saw apid 384488 userid 51668 numnids 1024 apname xyz at
top
of congestion list for 2 consecutive times; killing application by sending event
to
ALPS for node c8-1c0s7n1 (nid 1871)!
```

How apevent communicates application start and end events

To help with congestion management, the apevent daemon running on the boot node is used to send application start and end information to HSS software on the SMW. It does this by connecting with the apsched daemon and converting apsched messages into events that are consumed by HSS software. The event includes the application ID (apid), the user ID, the command name, the batch system ID, and the list of NIDs where the application is running. When apevent first makes a connection with apsched, it requests a sync message, which will contain information about all running applications. Following that, apsched sends a message to apevent before sending a (successful) placement reply message back to aprun and sends another message after it receives an exit message from the application. To make sure that ALPS software and the HSS software are synchronized (even in the event of daemons starting/stopping), apevent periodically requests another sync message from apsched. The time between sync messages is configurable by using the apevent -r option.

3.6 Find Throttled Applications in the xtnlrd Log

When the system initiates Congestion Protection software, xtnlrd outputs a list of applications and their APIDs subject to system throttling. An example of output from xtnlrd:

```
2012-08-31 16:49:39 host1 53037 responder_work_func: List of all currently running applications:
2012-08-31 16:49:39 host1 53037 APPS_WHILE_THROTTLED: APID 3107982 USERID 31633 NIDS 32 NAME bnt
2012-08-31 16:49:39 host1 53037 APPS_WHILE_THROTTLED: APID 3107988 USERID 31633 NIDS 22 NAME
AlltoAll_allput
2012-08-31 16:49:39 host1 53037 APPS_WHILE_THROTTLED: APID 3107967 USERID 31633 NIDS 19 NAME
fma_multiple_pu
2012-08-31 16:49:39 host1 53037 APPS_WHILE_THROTTLED: APID 3107975 USERID 31633 NIDS 19 NAME bnt3
2012-08-31 16:49:39 host1 53037 APPS_WHILE_THROTTLED: APID 3107977 USERID 31633 NIDS 19 NAME
AlltoAll_allput
2012-08-31 16:49:39 host1 53037 APPS_WHILE_THROTTLED: APID 3107917 USERID 31633 NIDS 16 NAME
AlltoAll_allput
2012-08-31 16:49:39 host1 53037 APPS_WHILE_THROTTLED: APID 3107989 USERID 31633 NIDS 9 NAME
fma_simple_put_
2012-08-31 16:49:39 host1 53037 APPS_WHILE_THROTTLED: APID 3107985 USERID 31633 NIDS 2 NAME
AlltoAll_allput
```

In addition, an application list sorted by aggregate throttled ejection bandwidth is output to the xtnlrd log. This list identifies the applications generating the most HSN traffic at the time of the throttling incident; however, it does not necessarily include the application responsible for the congestion that triggered the throttling. A well behaved application run on many nodes can generate a very large volume of traffic without causing congestion. Nevertheless, in the absence of an obvious congestion candidate within the list of individual nodes with the highest throttled ejection bandwidths (see the example in *Send an Event to ALPS to Terminate the Top Network Congestion Candidate Application* on page 22), the applications shown in this aggregate throttled ejection bandwidth list may be considered for closer scrutiny. For example:

```
TOP_BANDWIDTH_APPS: 0: apid 3145021 userid 12795 numnids 16 apname bnt Kflits/sec: Total 486  Per-node:
30
TOP_BANDWIDTH_APPS: 1: apid 3145020 userid 12795 numnids 16 apname bnt Kflits/sec: Total 481  Per-node:
```

```
30
Done with apps; all other bandwidth values are zero
```

3.7 Identify the User Application that Caused Excessive Network Congestion

Prerequisites

All of the congestion protection event information provided in the `xtnlrd` log file for the boot session in which the event occurred is gathered from `/var/opt/cray/log/session-id/nlrd.session-id`. This is the starting point for further analysis. See *Congestion Management Log File and Messages* on page 21 for information related to congestion protection that can be found in this file.

About this task

Use this procedure as a guideline to determine which user application caused the network congestion.

Procedure

1. If the `congestion-tools` module is available on the SMW, it is highly recommended to load it and use the `xtcpreport` tool that it provides to perform the initial analysis of the `xtnlrd` log file.

```
smw:~> module load congestion-tools
smw:~> xtcpreport /var/opt/cray/log/session-id/nlrd.session-id
```

This command provides a helpful at-a-glance summary of each congestion incident contained within the log; a quick scan can potentially reveal patterns in the applications involved with a congestion protection event. The generated report details all congestion protection incidents over the period of time covered by the `nlrd` log file, typically the time since the machine was booted.

The first four lines produced by the report provide an overall summary of the congestion protection activity that has occurred, if any. If and when congestion protection is invoked, each incident is grouped into a series made up of one or more throttles occurring in succession. This log-wide summary will list the total number of throttles and the total number of series over which they occurred, along with the time covered by the `nlrd` log provided, and the overall peak congestion value that was measured. These peak numbers are given on a tile and NIC basis, representing the number of each that was considered to be in a congested state in the system when the measurement occurred.

Following this summary, the report then provides zero or more detailed reports corresponding to each individual congestion protection series. These will include the list of applications that were running during the congestion protection series, the length of that series, and the peak amount of congestion that was observed.

To help identify the responsible application, all of the applications running during at least part of the congestion protection series are listed with their start and end times in descending order according to their node count. This makes it easy to see at a glance if any applications began running around the time the congestion series started, or if any completed around the time it ended. Note that `xtcpreport` can be executed on a live system; therefore, applications still in progress will be listed without an end time.

In addition, the user id (UID) for each application is given, along with the ALPS application ID (APID), so that each application can be traced back to a specific job invocation. Finally, the `Max Flits/s` column presents the bandwidth data found in the top 10 worst nodes by ejection bandwidth list (mentioned in the previous section) intended to detect many-to-few patterns. If multiple node measurements are associated with a single application, only the overall worst is listed.

An example report produced by `xtcpreport` is:

```
Congestion Protection Report for 2012-10-10T23:02:37-5 through
2012-10-12T10:02:58-5
Peak Tile:    139         Peak NIC:     19
Total Series:   1           Total Throttles:  1

Information for Throttle Series  1 of  1            2012-10-24
    Series Start:    19:19:08     Last Throttle Start: 19:19:08
    Series End:      19:19:28     Throttling Duration: 00:00:20
    Series Duration: 00:00:20     Throttled 1 times
    Series Peak Tile:     139     Series Peak NIC:       19

                        Max
APID    Name    Nodes Flits/s   UID    Start      End
-----------------------------------------------------------------
10114   all2all  9728     --    1450   18:41:28   19:24:23
10147   wlfc     4560   23419   5961   19:09:06   19:19:27
10114   qmc      1536     --    9549   18:41:33
10147   ptest      23     --    1450   19:09:01   19:09:30
10147   ptest      23     --    1450   19:09:08   19:09:37
10146   P-G.pgi    16     --    7921   19:08:18   19:40:41
```

2. Alternately, administrators can also directly get the complete list of throttled applications in the `xtnlrd` log file; the list of these applications is logged with every congestion protection event. For sample output, see *Find Throttled Applications in the xtnlrd Log* on page 23.

3. With the proper data in hand, begin isolating the application that caused the congestion event.

The clearest evidence, provided enough data exists, comes from establishing a pattern based on the past behavior of the set of applications involved with the current congestion protection event. This includes whether they ran previously and if so, whether they were associated with any previous congestion protection events. Recall that the `xtnlrd` log file records all applications and their node counts, as well as all those that are associated with a congestion event; therefore, these logs may be useful for cross-referencing the applications currently under consideration. Regardless of the method used, the node counts of the applications must be a significant consideration, because the communication patterns of many applications may only become problematic from a congestion standpoint when they are run at a sufficiently large node count.

Although this method may suffice if new applications are relatively infrequent, it is clearly suboptimal to simply wait for such patterns to become apparent.

Therefore, the following guidelines are frequently helpful in determining whether a given application is likely to be the cause of a congestion event:

- With the exception of truly huge applications using perhaps thousands of nodes, congestion-causing applications will almost never be pure MPI codes; they typically use PGAS programming models (CAF and UPC), SHMEM, or uGNI-MPI hybrid approaches.
- Therefore, large non-MPI applications should always be considered the primary suspects until proven otherwise. In addition, smaller non-MPI applications should also be considered suspects if very large MPI applications are not present on the system.
- The most unambiguous evidence that a particular application is to blame for network congestions is when the `Max Flits/s` value for the (single) worst node on the top-10 congestion candidates list is in the millions, as this indicates an extremely high likelihood that a large many-to-few pattern is responsible for

the congestion protection event. It is also the most likely scenario in which congestion protection will be invoked on small- to moderate-sized systems. Unfortunately, the reverse is not implied if the worst individual node flit count value does not reach this level. In that case, a many-to-few pattern could still be the culprit in certain instances.

- In contrast to smaller systems, many-to-few patterns may be harder to detect on large, multi-row systems. Smaller many-to-few applications can cause congestion events that would not be possible on lower-dimension networks, and more transient patterns that might otherwise escape trouble can cause just enough congestion to be an issue.
- Finally, once a culprit application has been identified, work with the user to avoid future congestion-protection incidents. Application users may interact with congestion management software as it provides congestion protection in an attempt to curb congestion system-wide. Upon the termination of a throttled application, a message like the following appears in `stderr`:

```
Application apid network throttled: nodecount nodes throttled, time node-
seconds
```

This message shows the number of nodes that were throttled and the duration of throttling in node-seconds. A node-second is one wall clock second on one node.

ALPS logs and `aprun` output contain additional information about congestion protection-related activity.

All users running applications while the system is throttled will receive a message upon application completion indicating this throttling event. Therefore, receipt of this message alone does not necessarily indicate that a user's application has caused the throttling event; however, all applications running on the system are affected when throttling occurs.

For more information about how to avoid network congestion in a user's program, see *Aries™ Network Tech Note - Application Changes to Avoid Network Congestion (S-0048)*.

3.8 Possible Critical Errors Resulting from Network Congestion

In the rare event that the software is unable to successfully protect the system, a delayed response for a timed-out packet can cause an ORB critical error event. An ORB critical error event will abort the application and provide messages for the user and administrator. The following message will appear on the application's `stderr` during ORB error events:

```
Cray HSN detected critical error 0xNNNptag N.
```

where the particular HSN critical errors (`0xNNN`) that may result from excessive network congestion are:

- `0x4801: ORB Request with No Entry` - This indicates that there was no matching entry for the response packet within the ORB.
- `0x4802: ORB Command Mismatch` - This indicates that the command stored in the ORB entry is different than the command indicated by the response packet.
- `0x480a: ORB DWORD Flit Mismatch` - This indicates that the response packet size is different than the corresponding entry in the ORB.

CAUTION: If administrators or application programmers receive reports of the above errors, contact a Cray service representative for more information.

To learn how application programmers can modify their programs to prevent these types of errors on Cray XC series systems, see *Aries™ Network Tech Note - Application Changes to Avoid Network Congestion (S-0048)*.

3.9 SMW Network Congestion Parameters

Parameters in the xtnlrd.ini file control the system response to failures of network congestion on the system.

bw_congest_kill_top_app

Controls whether xtnlrd should tell ALPS to kill the top congesting application after bw_congest_max_consecutive_hits times at the top of the list (default is n)

bw_congest_max_consecutive_hits

Number of times an APID has to be at the top of the congestion list to be eligible to be killed (default is 2)

bw_congest_throttle_delay

Seconds to wait before the parent xtnlrd gathers ejection bandwidth data (default is 5)

bw_congest_top_nodes

Number of non-service nodes to log as having high ejection bandwidth (default is 10)

bw_throttle_nic_enable

Throttle based on NIC congestion (default is y)

bw_throttle_nic_hi_num

Total number of congested NICs to initiate throttling (default is 10)

bw_throttle_nic_lo_num

Total number of congested NICs before unthrottling is done (default is 6)

bw_throttle_tile_enable

Throttle based on tile congestion (default is y)

bw_throttle_tile_hi_num

Total number of congested tiles to initiate throttling (default is 20)

bw_throttle_tile_lo_num

Total number of congested tiles before unthrottling is done (default is 12)

bw_unthrottle_interval

Minimum seconds for a throttle to persist (default is 20 seconds)

3.10 Blade Controller Network Congestion Parameters

Parameters in the xtnlrd.ini file control the system response to failures of network congestion on the system.

bw_em_sample_interval_sec

Seconds between ejection bandwidth samples on a blade controller (BC); the default is 4

bw_qos_rate_b1=511999

Do not change; this parameter is related to the unthrottled and throttled Aries™ traffic shaping control settings `bw_qos_rate_b2=5119`

Do not change; this parameter is related to the unthrottled and throttled Aries™ traffic shaping control settings

`bw_qos_size_b1=17`

Do not change; this parameter is related to the unthrottled and throttled Aries™ traffic shaping control settings

`bw_qos_size_b2=1`

Do not change; this parameter is related to the unthrottled and throttled Aries™ traffic shaping control settings

`bw_throttle_heartbeat_misses`

Number of missed `xtnlrd` heartbeats before `gmbwtd` locally throttles that blade. Receipt of a heartbeat while locally throttled causes an unthrottle; the default is `3`

`bw_throttle_heartbeat_sec`

Time in seconds between heartbeats from `xtnlrd` to `gmbwtd` to indicate the SMW and HSS network is still alive. These heartbeats are intended to protect the system from HSN congestion if the HSS system is dead (`0 == off`); the default is `3`

`bw_throttle_nic_history`

Number of sample periods in a row before sending a congestion notification (`ec_l0_bw_throttle`); the default is `3`

`bw_throttle_node_stall_thresh`

Ration of stalls to forwarded flits in the NIC; the default is `200000`

`bw_throttle_sample_usec`

Sample rate, in microseconds, to sample MMRs for congestion detection; the default is `500000`

`bw_throttle_tile_vc0_stall_thresh`

Ratio of stalls to forwarded flits on the request channel; the default is `200000`

`bw_throttle_tile_history`

Number of sample periods in a row before sending a congestion notification (`ec_l0_bw_throttle`); the default is `3`