



VERITAS NerveCenter: Downstream Alarm Suppression

The new NerveCenter downstream alarm suppression behavior model monitors nodes in a complex network. Using topology information—either from HP OpenView or TME 10 NetView or from a file that you provide—the model uses the relationships between nodes to determine the status of those nodes accurately. You can also use the model to log data to the database for outage and availability reports.

This white paper describes how the model works, how to test the model, and the technical details of the model.

The latest downstream alarm suppression model, `nodestatus_dwnstrm.mod`, is included with the current release of VERITAS NerveCenter. You can also get them from the VERITAS NerveCenter site at <http://www.veritas.com>.

This document includes the following sections:

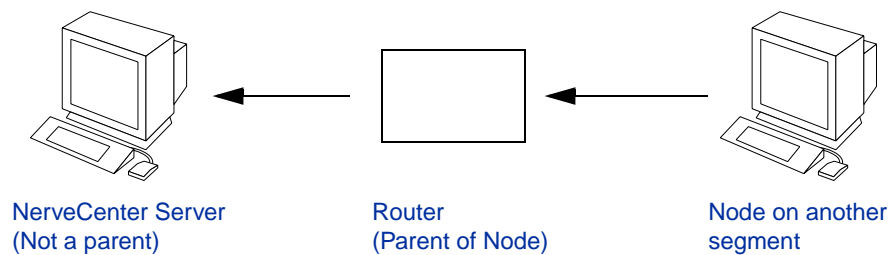
- ◆ “[Understanding How the Model Works](#)” on page 2
- ◆ “[Testing the Model](#)” on page 6
- ◆ “[Understanding the Technical Details](#)” on page 13
- ◆ “[Glossary](#)” on page 31

Understanding How the Model Works

The first downstream alarm suppression model (which included DSCollectRoutes, DSicmpStatus, and DSSnmpStatus), used information about local routers to determine the status of an unreachable node. If a route existed for the node, the node was assumed to be down; otherwise, it was marked as unreachable. In either case, the node was suppressed. For simple networks that consisted of nodes behind routers, this model was adequate. However, for more complex networks with multiple routers, switches, and hubs, and for certain routing protocols, the new model provides a more accurate determination of a node's status.

What is a complex network, as opposed to a simple network? A simple network might include single parent-child relationships. Nodes that are dependent on other nodes for a route to the NerveCenter server are *child* nodes. Nodes on which other nodes are dependent are *parent* nodes.

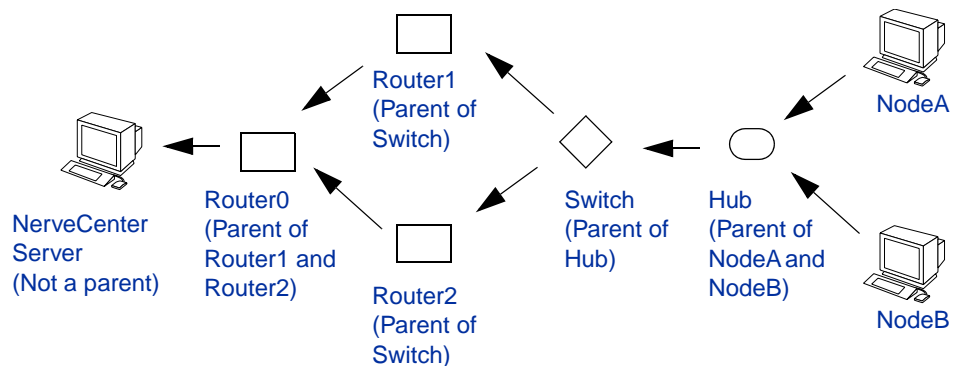
Figure 1. A Simple Network



Note The NerveCenter Server is not a parent node, nor does it have a parent. Logically, then, no node on the same segment as the NerveCenter Server has a parent.

A more complex network might include nodes with multiple parents and nodes that are themselves parents to other nodes.

Figure 2. A More Complex Network



The new model uses the status of devices between NerveCenter and managed nodes in the network to make real-time determinations about whether nodes are up, down, or unreachable. NerveCenter can then take appropriate actions based on the statuses of those nodes. For example, NerveCenter is monitoring 1000 nodes and 300 nodes behind a router stop responding to polls. NerveCenter can use the status of the router and any intermediate devices to determine whether the nodes are down or unreachable. If the nodes are actually down, NerveCenter forwards the appropriate alarms to the network management platform; however, if they are unreachable,

NerveCenter just forwards one critical alarm for the router and uses built-in VERITAS NerveCenter Smart Polling technology to stop suppressible polls for those nodes until they are available again.

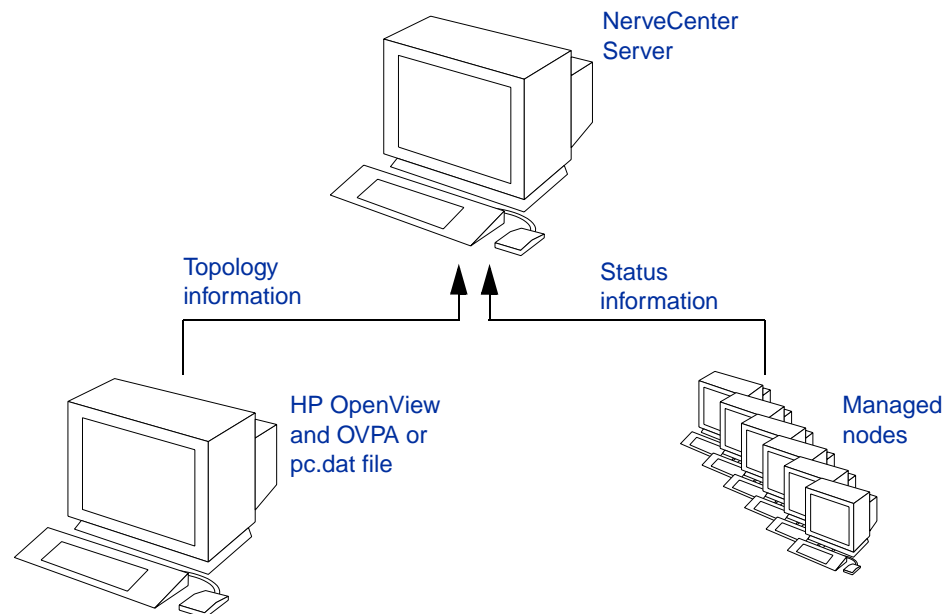
NerveCenter can get information about the nodes in the following ways:

- ◆ The OpenView Platform Adaptor (OVPA) extracts the topology information from HP OpenView or TME 10 NetView and stores the relationship information for each node in the NerveCenter database.
- ◆ OVPA extracts the topology information from HP OpenView or TME 10 NetView and creates a text file that contains the relationship information. NerveCenter loads that information by using a Perl subroutine that you define.
- ◆ You create a text file that defines the relationships of the nodes on your network. Then, you create an alarm that uses a Perl subroutine with some built-in functions to load that information into the database.

Once NerveCenter has that relationship information, the DwnStrmSnmpStatus and DwnStrmIcmpStatus alarms monitor nodes and maintain their statuses in the NerveCenter database.

Note The accuracy of the decisions NerveCenter makes does depend on the accuracy of the topology information it receives. Note that if you export node information, which includes parent information, from one server to another on a different segment, the parent data might not be accurate because the topology perspective will be different.

Figure 3. NerveCenter Maintains Parent-Child Relationship and Status Information



The following table lists exceptions that apply to HP OpenView:

Table 1. Special Considerations When Getting Parenting Information from HP OpenView

If you are getting parenting information from HP OpenView...
<ul style="list-style-type: none"> ◆ Any interface of a device in HP OpenView that has a segment ID of 0 is not used as a parent because it cannot be determined whether that interface shares a segment with any other device.
<ul style="list-style-type: none"> ◆ NerveCenter does not list any parents or children for nodes that are only displayed on isolated subnetworks because HP OpenView is not able to determine how that node is connected to the network.
<ul style="list-style-type: none"> ◆ Because HP OpenView does not provide enough information to determine whether a switch is a parent, switches are treated as though they were ordinary nodes. In other words, a switch can have parents, but no child nodes can list a switch as their parent. As a result, nodes that are only connected to a network by a switch appear to have no parents.

Nodes can have the following statuses: up, testing, down, and unreachable. Any node that responds has a status of up. The first time a node does not respond, its status is set to testing. While a node is in testing, its status is not updated again until NerveCenter determines that the node is up, down, or unreachable.

NerveCenter decides whether the node is down or unreachable based on whether the node has parents, whether the parents' statuses are more current than the node's last status update, and what those statuses are.

Note The current downstream alarm suppression model evaluates parent status at the node level, not the subobject or interface level.

The model uses the following logic:

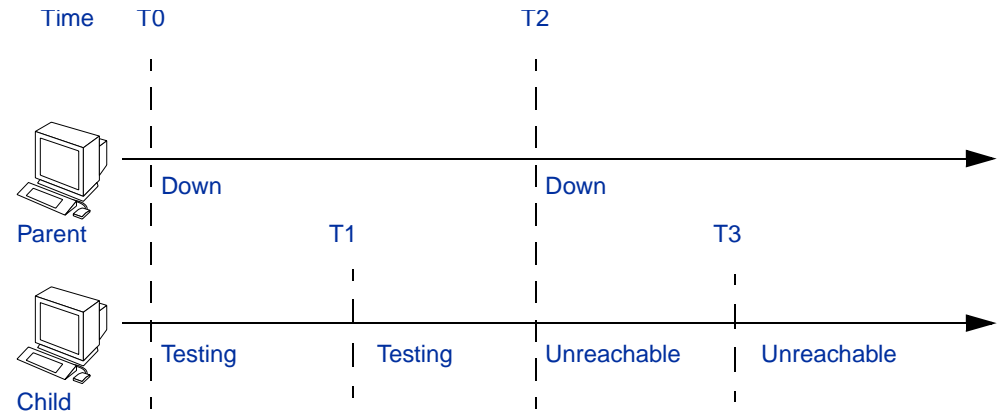
- ◆ The node is set to unreachable if the following condition is true: all parents have more current statuses and no parents are up or in testing.
- ◆ The node is set to down if one of the following conditions is true:
 - ◆ At least one parent has a more current status than the node and is up
 - ◆ The node has no parents
- ◆ The status of the node does not change as long as one of the following is true:
 - ◆ No parents have a more current status than the node
 - ◆ One or more parents have a more current status than the node but are not up

The key is to only update a node's status when NerveCenter can make a definitive decision about the status based on real-time data, which can only happen when the parents' status is more current than the node's status.

If the node does go to down or unreachable, NerveCenter continues to monitor the node and its parents to determine if the node is available again, if the parents' statuses have affected the status of the node, or if there has been no change.

For example, the following figure shows a node that has one parent. At T_0 , the node does not respond to an SNMP poll, so the alarm transitions from ground to error and the node's status is updated to testing. If the node does not respond to a second poll at T_1 , the alarm transitions from error to testing but the node's status is not updated. On a circular transition that loops back to the testing state, a Perl subroutine checks—and continues to check—the parent's status. At T_2 , the parent's status has been updated. Since the parent's status is more current than the node's status, the alarm transitions to unreachable and the node's status is set to unreachable. At T_3 , the parent's status has not changed, so the node's status is not updated.

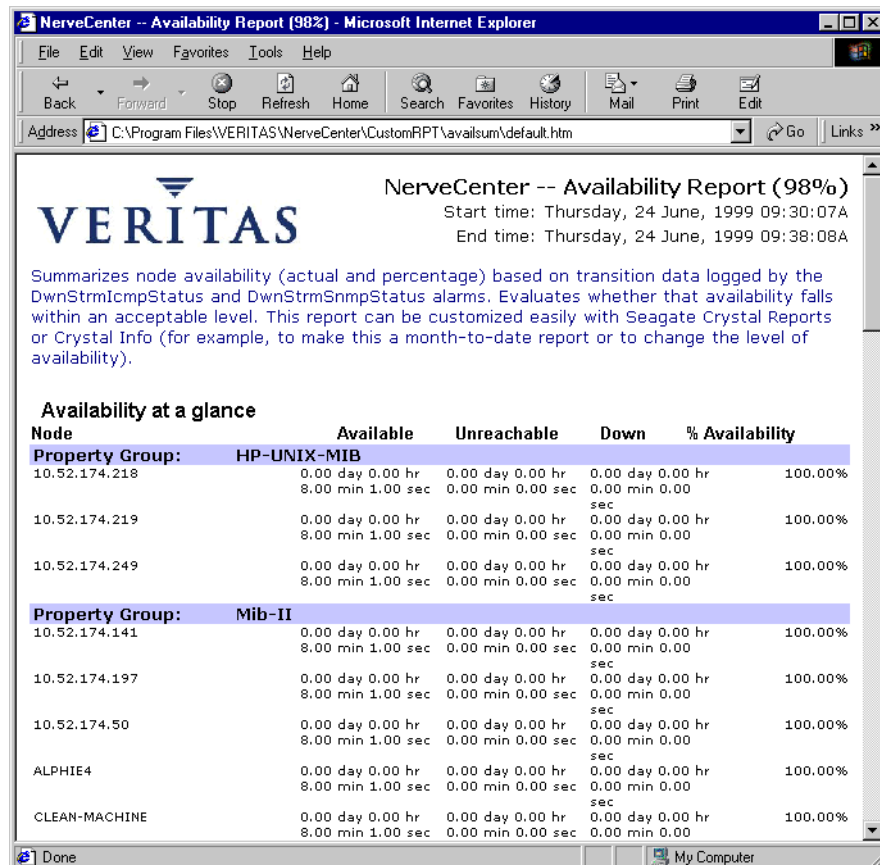
Figure 4. Updating a Node's Status Depends on When the Parent's Status Was Last Updated



As long as the parent's status remains down and is more current than the node's status every time the Perl subroutine checks it, the node's status is refreshed.

If you are running the NerveCenter Server on Windows NT, you can run reports on the availability of managed nodes. Three reports included with this version of NerveCenter include a summary of availability (availsum.rpt), the status of each node by property group (availstat.rpt), and a list of all transitions for each node (availtrans.rpt). The following is an example of the summary of availability report.

Figure 5. Summary of Node Availability



For more details about the new downstream alarm suppression model, see ["Understanding the Technical Details"](#) on page 13.



Testing the Model

The alarm suppression model is based on this concept: by monitoring whether nodes are dependent on other nodes (parent-child relationships) and by keeping each node's status updated proactively, the model can make accurate assessments as to what the statuses of dependent, or child, nodes are.

The following sections describe how to test the models:

1. [“Importing the New Model”](#) on page 7
2. [“Identifying Parent-Child Relationships”](#) on page 8
3. [“Making the Relationship Information Available to NerveCenter”](#) on page 9
4. [“Testing the Alarm Suppression Model”](#) on page 10
5. [“Running Node Availability Reports”](#) on page 11



Importing the New Model

The new alarm suppression model is included in the default database that is installed with NerveCenter v3.6. However, if you upgraded an existing database, you must import the model before you can use it. The model includes all of the objects you need, including the alarms, polls, masks, Perl subroutines, and so on.

Caution Older versions of the imported objects will be overwritten (for example, IcmpPoll).

▼ **To import the new model:**

1. Start the Client and connect to the NerveCenter Server.
2. From the Server menu, select Import Objects and Nodes.
3. Select Browse.
4. Double-click the node_status directory.
5. Select nodestatus_dwnstrm.mod and then Open.
6. Select OK.

A message is displayed when the file has been imported. See *Designing and Managing Behavior Models* for complete details on importing models.



Identifying Parent-Child Relationships

You can let OVPA extract relationship information from HP OpenView or TME 10 NetView and either store it in the NerveCenter database or in a text file. You can also create the text file manually.

Note By default, OVPA does not get information about a node's parents from your network management platform. You must configure OVPA to collect that information by doing the following steps.

▼ To identify parent-child relationships using OVPA:

1. Make sure HP OpenView or TME 10 NetView is running. Also make sure the NerveCenter Server is running.
2. Make sure that your network management platform is set up as your node source in the NerveCenter Administrator.
3. If OVPA is running, stop it by typing `ovstop ovpa` at the command line.
4. Start OVPA in parenting mode from the command line by typing one of the following commands:
 - ◆ `ovpa -pc`
OVPA runs and computes parenting information.
 - ◆ `ovpa -pc_noupdate`
OVPA starts and computes parenting information only on starting and when NerveCenter does a full resync.
 - ◆ `ovpa -pc hostname`
hostname is the name of the machine on which the NerveCenter Server runs. OVPA computes the parenting information, writes it to a file named *hostname_PC.dat*, and then stops.

See *Integrating NerveCenter with a Network Management Platform* for complete details about starting and stopping OVPA and the NerveCenter Server, as well as instructions for setting up a node source.

▼ To identify parent-child relationships manually:

1. Open a new text file.
2. Include a line for each node that has parents. Use the following syntax:

```
child parent
```

where *child* is the name of the node and *parent* is the name of each node on which the child is dependent. If you have more than one parent, separate parents by typing a space between each one.

Note If NerveCenter uses a full domain name for the node, use the full name in this file to refer to that node.

For example, nodeA is dependent on nodeB.domain.com and nodeC and nodeB.domain.com is dependent on nodeD. The contents of the text file would look like this:

```
nodeA nodeB.domain.com nodeC
nodeB.domain.com nodeD
```

3. Save and close the file.



Making the Relationship Information Available to NerveCenter

If you created a text file with the relationship information—either manually or by using OVPA—you must load that information into NerveCenter.

▼ **To load relationship information into NerveCenter:**

1. In the NerveCenter Client, create an alarm that you can transition on demand. On the transition, call a Perl subroutine that includes the following function:

```
NC::LoadParentsFromFile(FileName);
```

where *FileName* is the name of the file you created.

2. Transition the alarm. After the alarm transitions, you can turn the alarm off.

Caution If you modify the file, you must repeat this procedure.

To make sure the contents of the file were read correctly, you can create another alarm with a Perl subroutine that includes the following function:

```
NC::DumpParentsToFile(FileName);
```

The information will be written to the file on the local machine.

To remove relationship information, you can create an alarm with a Perl subroutine that includes the following function:

```
NC::RemoveAllParents();
```



Testing the Alarm Suppression Model

You can test the model by turning the DwnStrmSnmpStatus alarm (see “[DwnStrmSnmpStatus Alarm](#)” on page 15) and DwnStrmIcmpStatus alarm (see “[DwnStrmIcmpStatus Alarm](#)” on page 19) on, and then simulating a node being unreachable.

▼ **To test the alarm suppression model:**

1. Make sure the Client is connected to the NerveCenter Server.
2. From the Admin menu, select Alarm Definition List.
The Alarm Definition List dialog is displayed.
3. In the listbox, right-click on DwnStrmSnmpStatus and select On.
4. In the listbox, right-click on DwnStrmIcmpStatus and select On.

The DwnStrmSnmpStatus and DwnStrmIcmpStatus alarms monitor the status of the managed nodes.

Note To use the model to log data against which you can run availability reports, turn on the LogToDB versions of the alarms.

To simulate a node being unreachable, you can change the IP address to an invalid address for your network. (For example, you might use 10.10.10.10.) If you have a test network available, you can also make nodes unreachable by unplugging devices—for example, a router.

NerveCenter detects errors since the node no longer responds to polls. As a result, NerveCenter reevaluates and updates the status of the node. If the alarm is in an AgentDown, DeviceDown, or Unreachable state, NerveCenter suppresses suppressible alarms for that node until it is available again.

To make sure the statuses of the nodes are correct, you can create an alarm with a Perl subroutine that includes the following function:

```
NC::DumpNodeStatusToFile(FileName) ;
```

The information will be written to the file on the local machine.

Running Node Availability Reports

If you turned on the alarms that log transition data, you can run node availability reports against that data.

Note Currently, these reports are available for NerveCenter Servers running on Windows NT only. Also, you must have imported the model and turned the correct alarms on.

▼ To run the node availability reports:

1. From the Admin menu in the NerveCenter Client, select Report List.



The Report List dialog box is displayed.

2. Select New.

The Add Report dialog box is displayed.

Figure 6. Add Report Dialog Box

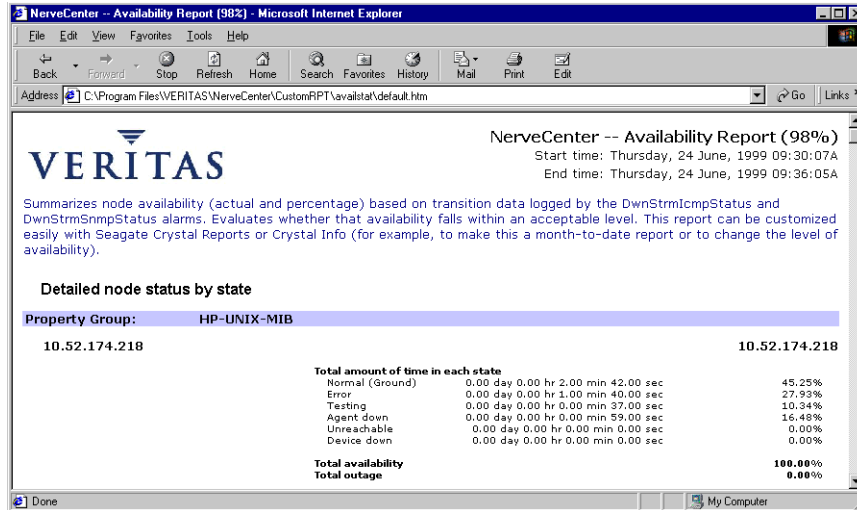
3. In the Report Select List, select one of the following reports:
 - ◆ availsum.rpt—Lists each node and percent availability by property group. This report offers availability information at a glance.
 - ◆ availstat.rpt—Lists each node and the amount of time it spent in each state, as well as the overall outage and availability. This report includes both actual times and percentages of time.
 - ◆ availtrans.rpt—Lists each node and its state transitions. If you have a large number of nodes, this report can be quite long.
4. In the Report Name field, type a name for your report (optional).
5. In the Report Author field, type your name (optional).
6. In the Description field, type any information that will help you or others understand the report or why it was generated (optional).
7. Select the Override Server Name in Report checkbox.
8. Select OK.

The report is added to the report list.
9. Select the report, and then select Run.

The report is generated and displayed. The following report is an example of the availstat.rpt report.



Figure 7. availstat.rpt Report



See *Monitoring Your Network* for more details about running and viewing reports.



Understanding the Technical Details

The two particular areas of interest in the model are the alarms used to monitor device status and the Perl subroutines used to store and evaluate relationship and status information. See the following sections for details about those types of objects:

- ◆ “Alarms” on page 15
- ◆ “Perl Subroutines” on page 24

The following objects are imported when you import the MOD file.

Caution Older versions of the imported objects will be overwritten (for example, `IcmpPoll`).

- ◆ Alarms
 - ◆ `DwnStrmSnmpStatus` (off)
 - ◆ `DwnStrmSnmpStatus_LogToDB` (off)
 - ◆ `DwnStrmIcmpStatus` (off)
 - ◆ `DwnStrmIcmpStatus_LogToDB` (off)
- ◆ Properties
 - ◆ `icmpStatus`
 - ◆ `nl-ping`
 - ◆ `system`
- ◆ Polls
 - ◆ `SnmpFastPoll` (on)
 - ◆ `SnmpPoll` (on)
 - ◆ `IcmpPoll` (on)
 - ◆ `IcmpFastPoll` (on)
- ◆ Masks
 - ◆ `ColdStart` (on)
 - ◆ `WarmStart` (on)
- ◆ Triggers
 - ◆ `nodeUpFast`
 - ◆ `nodeUp`
 - ◆ `agentUp`
 - ◆ `agentUpFast`
 - ◆ `warmStart`
 - ◆ `coldStart`
 - ◆ `Down`
 - ◆ `UnReachable`
 - ◆ `Testing`
 - ◆ `ERROR`
 - ◆ `NODE_UNREACHABLE`



- ◆ NET_UNREACHABLE
- ◆ ICMP_TIMEOUT
- ◆ PORT_UNREACHABLE
- ◆ SNMP_TIMEOUT
- ◆ Severities
 - ◆ Minor
 - ◆ Inform
 - ◆ Critical
 - ◆ Normal
- ◆ Perl subroutines
 - ◆ SetNodeStatusUp
 - ◆ TestParentStatus
 - ◆ SetNodeStatusUnReachable
 - ◆ SetNodeStatusTesting
 - ◆ SetNodeStatusDown
 - ◆ TestParentSetNode



Alarms

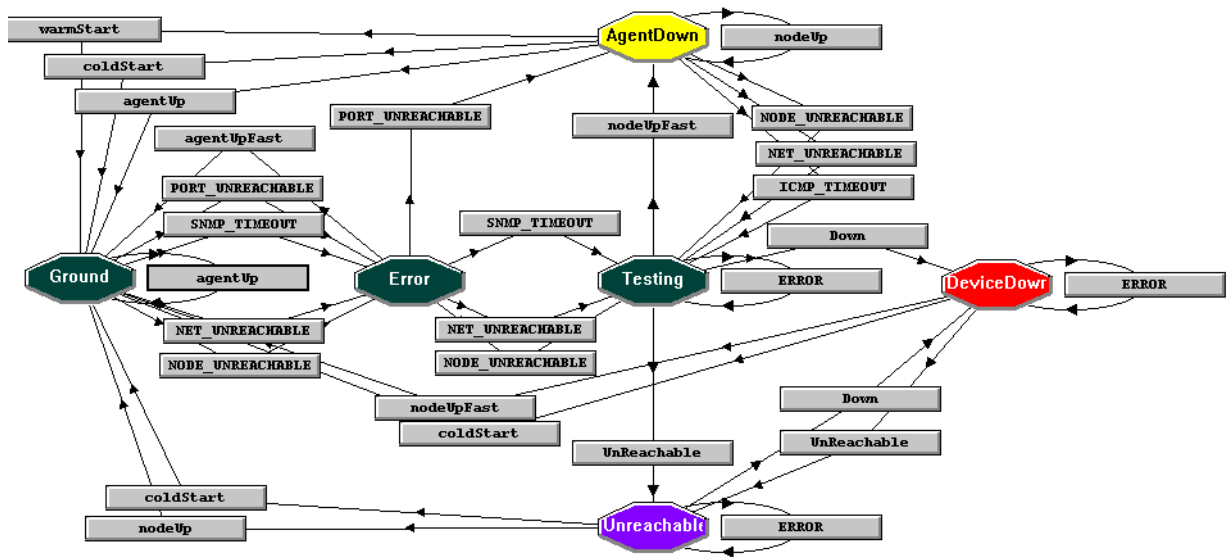
The new downstream alarm suppression behavior model monitors node status using both SNMP and ICMP. This section includes descriptions of the following alarms:

- ◆ “DwnStrmSnmpStatus Alarm” on page 15
- ◆ “DwnStrmIcmpStatus Alarm” on page 19

DwnStrmSnmpStatus Alarm

This alarm accurately monitors the status of nodes and their SNMP agents by taking into consideration the status of the nodes’ parents.

Figure 8. DwnStrmSnmpStatus Alarm State Diagram



The following table lists the severity of each state:

Table 2. Severities of Each State in DwnStrmSnmpStatus

State	Severity	Color
Ground	Normal	Green
Error	Normal	Green
Testing	Normal	Green
AgentDown	Minor	Yellow
DeviceDown	Critical	Red
Unreachable	Inform	Purple

When this alarm is turned on, the following polls and masks cause state transitions:

- ◆ IcmpFastPoll (ICMP echo request, or ping)
- ◆ IcmpPoll (ICMP echo request, or ping)
- ◆ SnmpFastPoll (SNMP get request)
- ◆ SnmpPoll (SNMP get request)
- ◆ ColdStart (trap mask)



- ◆ WarmStart (trap mask)

This alarm uses the following Perl subroutines:

- ◆ “[SetNodeStatus Perl Subroutines](#)” on page 24
- ◆ “[TestParentStatus Perl Subroutine](#)” on page 24
- ◆ “[TestParentSetNode Perl Subroutine](#)” on page 27

Note Before turning this alarm on, NerveCenter must have loaded the relationship data. See “[Identifying Parent-Child Relationships](#)” on page 8 and “[Making the Relationship Information Available to NerveCenter](#)” on page 9.

The following sections describe the states in the DwnStrmSnmppStatus alarm and the transitions and actions that can happen from those states:

- ◆ “[Ground State](#)” on page 16
- ◆ “[Error State](#)” on page 16
- ◆ “[Testing State](#)” on page 17
- ◆ “[AgentDown State](#)” on page 18
- ◆ “[Unreachable State](#)” on page 18
- ◆ “[DeviceDown State](#)” on page 19

Ground State

In Ground state, the node is reachable and the SNMP agent is up.

As long as the node and agent respond to the SnmpPoll and SnmpFastPoll requests, the agentUp circular transition is triggered. The agentUp transition calls the SetNodeStatusUp Perl subroutine (see “[SetNodeStatus Perl Subroutines](#)” on page 24) to refresh the update time.

If the node does not respond to the polls, the following triggers can transition the alarm from Ground to Error:

- ◆ NET_UNREACHABLE
- ◆ NODE_UNREACHABLE
- ◆ PORT_UNREACHABLE
- ◆ SNMP_TIMEOUT

Transitions to the Error state call the SetNodeStatusTesting Perl subroutine (see “[SetNodeStatus Perl Subroutines](#)” on page 24) to update the status to Testing.

Error State

The alarm suppression behavior model uses the Error state to confirm that there is actually a problem (as opposed to a dropped packet, for example). From the Error state, a node can transition back to Ground, to Testing, or to AgentDown.

If the node and agent respond to the SnmpFastPoll request, the agentUpFast transition is triggered. The agentUpFast transition does the following:

- ◆ Returns the alarm to Ground state
- ◆ Calls the SetNodeStatusUp Perl subroutine (see “[SetNodeStatus Perl Subroutines](#)” on page 24) to update the status so that NerveCenter can evaluate the states of the children of this node, if there are any, accurately

If the node still does not respond to the poll, several transitions can happen.

- ◆ The following triggers transition the alarm from Error to Testing
 - ◆ NET_UNREACHABLE
 - ◆ NODE_UNREACHABLE
 - ◆ SNMP_TIMEOUT
- ◆ The PORT_UNREACHABLE trigger does the following:
 - ◆ Transitions the alarm to an AgentDown state
 - ◆ Uses the Set Attribute action to suppress the node so the node won't be polled by suppressible polls while the agent is down
 - ◆ Calls the SetNodeStatusUp Perl subroutine (see "[SetNodeStatus Perl Subroutines](#)" on page 24) to update the status so that NerveCenter can evaluate the states of the children of this node, if there are any, accurately

Testing State

While an alarm is in the Testing state, NerveCenter identifies whether the node is:

- ◆ Down
- ◆ Unreachable
- ◆ Up, but its agent is down

If nodeUpFast is triggered in response to an IcmpFastPoll poll while the node is in the Testing state, the trigger:

- ◆ Transitions the alarm to an AgentDown state
- ◆ Uses the Set Attribute action to suppress the node so the node won't be polled by suppressible polls while the agent is down
- ◆ Calls the SetNodeStatusUp Perl subroutine (see "[SetNodeStatus Perl Subroutines](#)" on page 24) to update the status so that NerveCenter can evaluate the states of the children of this node, if there are any, accurately

If nodeUpFast results in a circular ERROR transition, the TestParentStatus Perl subroutine (see "[TestParentStatus Perl Subroutine](#)" on page 24) looks up the status of the parents. If TestParentStatus can determine the node's state based on the parents' status, TestParentStatus fires the appropriate trigger: UnReachable or Down.

- ◆ The UnReachable trigger:
 - ◆ Transitions the alarm to an Unreachable state
 - ◆ Uses the Set Attribute action to suppress the node so the node won't be polled by suppressible polls while the node is unreachable
 - ◆ Calls the SetNodeStatusUnreachable Perl subroutine (see "[SetNodeStatus Perl Subroutines](#)" on page 24) to update the status so that NerveCenter can evaluate the states of the children of this node, if there are any, accurately
- ◆ The Down trigger:
 - ◆ Transitions the alarm to a DeviceDown state
 - ◆ Uses the Set Attribute action to suppress the node so the node won't be polled by suppressible polls while the node is unreachable



- ◆ Calls the `SetNodeStatusDown` Perl subroutine (see “[SetNodeStatus Perl Subroutines](#)” on page 24) to update the status so that NerveCenter can evaluate the states of the children of this node, if there are any, accurately
- ◆ Sends an Inform action to notify a network management platform or another NerveCenter of the status of this node

AgentDown State

While an alarm is in the AgentDown state, NerveCenter continues to monitor the node for any changes.

As long as the node responds to the `IcmpPoll` and `IcmpFastPoll` requests, the `nodeUp` transition is triggered. The `nodeUp` transition calls the `SetNodeStatusUp` Perl subroutine (see “[SetNodeStatus Perl Subroutines](#)” on page 24) to refresh the update time.

If the node does not respond to the polls, the following triggers transition the node from AgentDown to Testing:

- ◆ `NET_UNREACHABLE`
- ◆ `NODE_UNREACHABLE`
- ◆ `ICMP_TIMEOUT`

Each transition calls the `SetNodeStatusTesting` Perl subroutine (see “[SetNodeStatus Perl Subroutines](#)” on page 24) to update the status to Testing.

If NerveCenter receives a `warmStart` trap or a `coldStart` trap, or `agentUp` is triggered in response to an `SnmpPoll` response, the trigger:

- ◆ Transitions the alarm to a Ground state
- ◆ Uses the Set Attribute action to turn poll suppression off so NerveCenter can resume all normal polling
- ◆ Calls the `SetNodeStatusUp` Perl subroutine (see “[SetNodeStatus Perl Subroutines](#)” on page 24) to update the time of the last status change so that NerveCenter can evaluate the states of the children of this node, if there are any, accurately

Unreachable State

While an alarm is in the Unreachable state, NerveCenter continues to monitor the node for any changes.

If NerveCenter receives a `coldStart` trap or `nodeUp` is triggered by a response to `IcmpPoll`, the trigger:

- ◆ Transitions the alarm to a Ground state
- ◆ Uses the Set Attribute action to turn poll suppression off so NerveCenter can resume all normal polling
- ◆ Calls the `SetNodeStatusUp` Perl subroutine (see “[SetNodeStatus Perl Subroutines](#)” on page 24) to update the status so that NerveCenter can evaluate the states of the children of this node, if there are any, accurately

If the poll does not get a response and an `ERROR` transition is triggered, NerveCenter calls the `TestParentSetNode` Perl subroutine (see “[TestParentSetNode Perl Subroutine](#)” on page 27), which looks up the status of the parents. If `TestParentSetNode` can determine the node’s state based on the parents’ status, `TestParentSetNode` fires the Down trigger or refreshes the node’s update time.

The Down trigger:

- ◆ Transitions the alarm to a DeviceDown state
- ◆ Calls the SetNodeStatusDown Perl subroutine (see “[SetNodeStatus Perl Subroutines](#)” on page 24) to update the status so that NerveCenter can evaluate the states of the children of this node, if there are any, accurately
- ◆ Sends an Inform action to notify a network management platform or another NerveCenter of the status of this node

DeviceDown State

While an alarm is in the DeviceDown state, NerveCenter continues to monitor the node for any changes.

If NerveCenter receives a coldStart trap or the nodeUpFast transition is triggered by an IcmpFastPoll, the trigger:

- ◆ Transitions the alarm to a Ground state
- ◆ Uses the Set Attribute action to turn poll suppression off so NerveCenter can resume all normal polling
- ◆ Calls the SetNodeStatusUp Perl subroutine (see “[SetNodeStatus Perl Subroutines](#)” on page 24) to update the time of the last status change so that NerveCenter can evaluate the states of the children of this node, if there are any, accurately

If the poll does not get a response and a circular ERROR transition is triggered, the TestParentSetNode Perl subroutine (see “[TestParentSetNode Perl Subroutine](#)” on page 27), which looks up the status of the parents. If TestParentSetNode can determine the node’s state based on the parents’ status, TestParentStatus fires the Unreachable trigger or refreshes the node’s update time.

The Unreachable trigger:

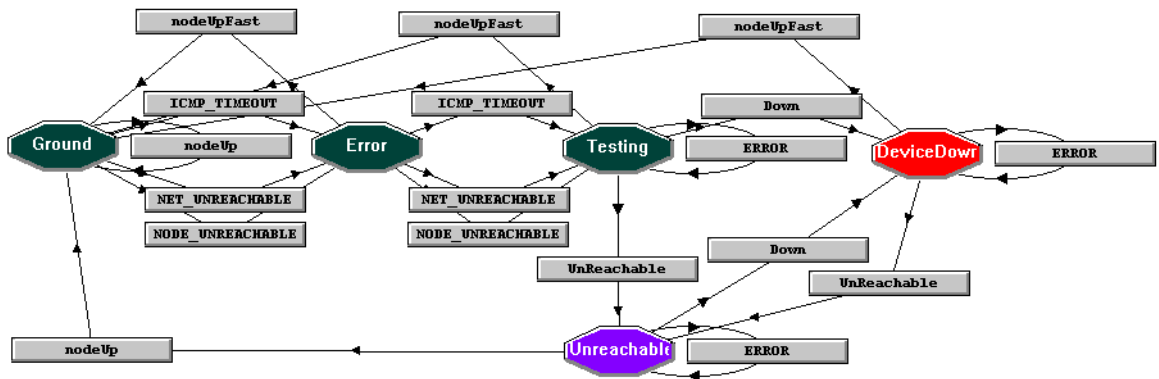
- ◆ Transitions the alarm to an Unreachable state
- ◆ Calls the SetNodeStatusUnReachable Perl subroutine (see “[SetNodeStatus Perl Subroutines](#)” on page 24) to update the status so that NerveCenter can evaluate the states of the children of this node, if there are any, accurately

DwnStrmlcmpStatus Alarm

This alarm accurately monitors the status of nodes by taking into consideration the status of the nodes’ parents.



Figure 9. DwnStrmIcmpStatus Alarm State Diagram



The following table lists the severity of each state:

Table 3. Severities of each state in DwnStrmSnmpStatus

State	Severity	Color
Ground	Normal	Green
Error	Normal	Green
Testing	Normal	Green
DeviceDown	Critical	Red
Unreachable	Inform	Purple

When this alarm is turned on, the following polls cause state transitions:

- ◆ IcmpFastPoll (ICMP echo request, or ping)
- ◆ IcmpPoll (ICMP echo request, or ping)

This alarm uses the following Perl subroutines:

- ◆ [“SetNodeStatus Perl Subroutines”](#) on page 24
- ◆ [“TestParentStatus Perl Subroutine”](#) on page 24
- ◆ [“TestParentSetNode Perl Subroutine”](#) on page 27

Note Before turning this alarm on, NerveCenter must have loaded the relationship data. See [“Identifying Parent-Child Relationships”](#) on page 8 and [“Making the Relationship Information Available to NerveCenter”](#) on page 9.

The following sections describe the states in the DwnStrmIcmpStatus alarm and the transitions and actions that can happen from those states:

- ◆ [“Ground State”](#) on page 21
- ◆ [“Error State”](#) on page 21
- ◆ [“Testing State”](#) on page 21
- ◆ [“Unreachable State”](#) on page 22
- ◆ [“DeviceDown State”](#) on page 22

Ground State

In Ground state, the node is reachable.

As long as the node responds to the `IcmpPoll` and `IcmpFastPoll` requests, the `nodeUp` circular transition is triggered. The `nodeUp` transition calls the `SetNodeStatusUp` Perl subroutine (see “[SetNodeStatus Perl Subroutines](#)” on page 24) to refresh the update time.

If the node does not respond to the polls, the following triggers can transition the alarm from Ground to Error:

- ◆ `NET_UNREACHABLE`
- ◆ `NODE_UNREACHABLE`
- ◆ `ICMP_TIMEOUT`

Transitions to the Error state call the `SetNodeStatusTesting` Perl subroutine (see “[SetNodeStatus Perl Subroutines](#)” on page 24) to update the status to Testing.

Error State

The alarm suppression behavior model uses the Error state to confirm that there is actually a problem (as opposed to a dropped packet, for example). From the Error state, an alarm can transition back to Ground or to Testing.

If the node responds to the `IcmpFastPoll` request, the `nodeUpFast` transition is triggered. The trigger:

- ◆ Returns the alarm to Ground state
- ◆ Calls the `SetNodeStatusUp` Perl subroutine (see “[SetNodeStatus Perl Subroutines](#)” on page 24) to update the status so that `NerveCenter` can evaluate the states of the children of this node, if there are any, accurately

If the node still does not respond to the poll, the following triggers transition the alarm from Error to Testing:

- ◆ `NET_UNREACHABLE`
- ◆ `NODE_UNREACHABLE`
- ◆ `ICMP_TIMEOUT`

Testing State

While an alarm is in the Testing state, `NerveCenter` identifies whether the node is down or unreachable.

If `nodeUpFast` is triggered in response to an `IcmpFastPoll` poll while the node is in the Testing state, the trigger:

- ◆ Transitions the alarm to Ground
- ◆ Calls the `SetNodeStatusUp` Perl subroutine (see “[SetNodeStatus Perl Subroutines](#)” on page 24) to update the status so that `NerveCenter` can evaluate the states of the children of this node, if there are any, accurately

If `nodeUpFast` results in a circular `ERROR` transition, `NerveCenter` calls the `TestParentStatus` Perl subroutine (see “[TestParentStatus Perl Subroutine](#)” on page 24) looks up the status of the parents. If `TestParentStatus` can determine the node’s state based on the parents’ status, `TestParentStatus` fires the appropriate trigger: `UnReachable` or `Down`.

- ◆ The `UnReachable` trigger:



- ◆ Transitions the alarm to an Unreachable state
- ◆ Uses the Set Attribute action to suppress the node so the node won't be polled by suppressible polls while the node is unreachable
- ◆ Calls the SetNodeStatusUnreachable Perl subroutine (see “[SetNodeStatus Perl Subroutines](#)” on page 24) to update the status so that NerveCenter can evaluate the states of the children of this node, if there are any, accurately
- ◆ The Down trigger:
 - ◆ Transitions the alarm to a DeviceDown state
 - ◆ Uses the Set Attribute action to suppress the node so the node won't be polled by suppressible polls while the node is unreachable
 - ◆ Calls the SetNodeStatusDown Perl subroutine (see “[SetNodeStatus Perl Subroutines](#)” on page 24) to update the status so that NerveCenter can evaluate the states of the children of this node, if there are any, accurately
 - ◆ Sends an Inform action to notify a network management platform or another NerveCenter of the status of this node

Unreachable State

While an alarm is in the Unreachable state, NerveCenter continues to monitor the node for any changes.

If nodeUp is triggered by a response to IcmpPoll, the trigger:

- ◆ Transitions the alarm to a Ground state
- ◆ Uses the Set Attribute action to turn poll suppression off so NerveCenter can resume all normal polling
- ◆ Calls the SetNodeStatusUp Perl subroutine (see “[SetNodeStatus Perl Subroutines](#)” on page 24) to update the status so that NerveCenter can evaluate the states of the children of this node, if there are any, accurately

If the poll does not get a response and a circular ERROR transition is triggered, NerveCenter calls the TestParentSetNode Perl subroutine (see “[TestParentSetNode Perl Subroutine](#)” on page 27), which looks up the status of the parents. If TestParentSetNode can determine the node's state based on the parents' status, TestParentSetNode either fires the Down trigger or refreshes the node's update time.

The Down trigger:

- ◆ Transitions the alarm to a DeviceDown state
- ◆ Calls the SetNodeStatusDown Perl subroutine (see “[SetNodeStatus Perl Subroutines](#)” on page 24) to update the status so that NerveCenter can evaluate the states of the children of this node, if there are any, accurately
- ◆ Sends an Inform action to notify a network management platform or another NerveCenter of the status of this node

DeviceDown State

While an alarm is in the DeviceDown state, NerveCenter continues to monitor the node for any changes.

If the nodeUpFast transition is triggered by an IcmpFastPoll, the trigger:

- ◆ Transitions the alarm to a Ground state

- ◆ Uses the Set Attribute action to turn poll suppression off so NerveCenter can resume all normal polling
- ◆ Calls the SetNodeStatusUp Perl subroutine (see “[SetNodeStatus Perl Subroutines](#)” on page 24) to update the time of the last status change so that NerveCenter can evaluate the states of the children of this node, if there are any, accurately

If the poll does not get a response and a circular ERROR transition is triggered, the TestParentSetNode Perl subroutine (see “[TestParentSetNode Perl Subroutine](#)” on page 27), which looks up the status of the parents. If TestParentSetNode can determine the node’s state based on the parents’ status, TestParentSetNode fires the Unreachable trigger or refreshes the node’s update time.

The Unreachable trigger:

- ◆ Transitions the alarm to an Unreachable state
- ◆ Calls the SetNodeStatusUnReachable Perl subroutine (see “[TestParentStatus Perl Subroutine](#)” on page 24) to update the status so that NerveCenter can evaluate the states of the children of this node, if there are any, accurately



Perl Subroutines

The new downstream alarm suppression behavior model uses several Perl subroutines to store parent-child relationships and maintain node statuses. This section includes descriptions of the following Perl subroutines:

- ◆ “[SetNodeStatus Perl Subroutines](#)” on page 24
- ◆ “[TestParentStatus Perl Subroutine](#)” on page 24
- ◆ “[TestParentSetNode Perl Subroutine](#)” on page 27

SetNodeStatus Perl Subroutines

For the DwnStrmSnmpStatus and DwnStrmIcmpStatus alarms, all state transitions—except transitions from Error to Testing—call one of the following Perl subroutines:

- ◆ SetNodeStatusTesting
- ◆ SetNodeStatusDown
- ◆ SetNodeStatusUnreachable
- ◆ SetNodeStatusUp

These Perl subroutines update the node status so the node’s children can accurately update their statuses based on the node’s status.

SetNodeStatusTesting

```
my $Return;
$return = NC::SetNodeStatus($NodeName, "Testing");
#If $Return = 0, operation failed
```

SetNodeStatusDown

```
my $Return;
$return = NC::SetNodeStatus($NodeName, "Down");
#If $Return = 0, operation failed
```

SetNodeStatusUnreachable

```
my $Return;
$return = NC::SetNodeStatus($NodeName, "Unreachable");
#If $Return = 0, operation failed
```

SetNodeStatusUp

```
my $Return;
$return = NC::SetNodeStatus($NodeName, "Up");
#If $Return = 0, operation failed
```

TestParentStatus Perl Subroutine

For the DwnStrmSnmpStatus and DwnStrmIcmpStatus alarms, if a node is in a Testing state, the ERROR trigger is fired every time the node is polled and doesn’t respond. Each resulting ERROR transition calls the TestParentStatus Perl subroutine.

The TestParentStatus Perl subroutine tests the parent node status and determines the status of the node by doing the following:



- ◆ If the node has parents, TestParentStatus evaluates each parent's last update time. Based on the following rules, TestParentStatus sets a flag (TriggerFlag) that determines what trigger, if any, should be fired.
 - ◆ If no parents have an update time more recent than the node's update time, then TriggerFlag is set to Testing.
 - ◆ If at least one parent has a more recent update time but is not up, the flag is set to Testing.
 - ◆ If at least one parent has a more recent update time and is up, the flag is set to Down, regardless of the status or time of last update of any other parent.
 - ◆ If all parents have more recent update times and no parent is up or in testing, the flag is set to Unreachable.
- ◆ If the node has no parents, TriggerFlag is set to Down.

If TriggerFlag is set to Testing, TestParentStatus does nothing because TestParentStatus must have more information to make an accurate decision. If the alarm should be in another state, TestParentStatus fires the appropriate trigger to transition the node into that state.

The code for this subroutine is:

```
# The purpose of this subroutine is to test the parent
# node status and fire the appropriate trigger to take the
# alarm to either down or unreachable. You must make sure
# that all parents are being monitored with the status
# alarms.
use NC;
my $NodeUpdateTime; # Last time node status was updated
my $LastNodeStatus; # Last node status
my @Parents = (); # Array of parents
my $Parent; # Parent Node
my $ParentUpdateTime; # Last time parent node status was updated
my $ParentStatus; # Last parent status
my $TriggerFlag = "NotSet";
my $ParentNotUpdated = 0; # Remember if we have any parents not updated

#Define all triggers that can be fired
DefineTrigger('UnReachable');
DefineTrigger('Down');
DefineTrigger('Testing');

# Get the last node status and update time for this node
($LastNodeStatus,$NodeUpdateTime) = NC::GetNodeStatus($NodeName);

# Get the array of parents for this node
@Parents = NC::GetParents($NodeName);
if( defined( $Parents[0] ) )
```



```
{
# Test each parent, if ANY are ok, we assume the node
# is reachable. Parents update time must be past the
# last time the node was updated or we can't assume the
# status is accurate.
foreach $Parent (@Parents)
{
($ParentStatus,$ParentUpdateTime) = NC::GetNodeStatus($Parent);
if( $ParentUpdateTime >= $NodeUpdateTime )
{
# Using TriggerFlag to store name of trigger to be fired. If any
# parent is found to be up, then the flag will be set to down. If
# all parents are down or unreachable, then the flag will be set
# to unreachable. If no parents are down and at least one parent
# is testing, set flag to testing. Otherwise, it will remain not
# set and we will update the node's current status and time. Testing
# handles the case where one parent is testing and another is
# unreachable. We need to make sure we do not mark the node as
# unreachable until the parent node in testing goes to some final
# state because that state could be agent down which is treated
# as up.
if( ($ParentStatus eq "Down" || $ParentStatus eq "UnReachable") && $TriggerFlag eq
"NotSet")
{
$TriggerFlag = "UnReachable";
}
elseif( $ParentStatus eq "Up" )
{
$TriggerFlag = "Down";
}
elseif( $ParentStatus eq "Testing" && $TriggerFlag ne "Down" )
{
$TriggerFlag = "Testing";
}
}
else
{
# Remember that we have at least one parent that hasn't been updated.
$ParentNotUpdated = 1;
}
}
}
```

```

}
else
{
  # If no parents, assume node is down.
  $TriggerFlag = "Down";
}

# If I have at least one parent not updated and I do not have
# any Up parents, Set TriggerFlag to testing.
if( $ParentNotUpdated && $TriggerFlag ne "Down" )
{
  $TriggerFlag = "Testing";
}

if( $TriggerFlag ne "Testing" )
{
  # Fire trigger if node's status should change.
  if( $TriggerFlag ne $LastNodeStatus )
  {
    # Fire trigger
    FireTrigger( $TriggerFlag );
  }
}

```

TestParentSetNode Perl Subroutine

For the DwnStrmSnmpStatus and DwnStrmIcmpStatus alarms, if an alarm is in a DeviceDown or Unreachable state, the ERROR trigger is fired every time the node is polled and doesn't respond. Each resulting ERROR transition calls the TestParentSetNode Perl subroutine.

The TestParentSetNode Perl subroutine tests the parent node status and determines the status of the node by doing the following:

- ◆ If the node has parents, TestParentSetNode evaluates each parent's last update time. Based on the following rules, TestParentSetNode sets a flag (TriggerFlag) that determines what trigger, if any, should be fired.
 - ◆ If no parents have an update time more recent than the node's update time, then TriggerFlag is set to Testing.
 - ◆ If at least one parent has a more recent update time but is not up, the flag is set to Testing.
 - ◆ If at least one parent has a more recent update time and is up, the flag is set to Down, regardless of the status or time of last update of any other parent.
 - ◆ If all parents have more recent update times and no parent is up or in testing, the flag is set to Unreachable.
- ◆ If the node has no parents, TriggerFlag is set to Down.



If `TriggerFlag` is set to `Testing`, `TestParentSetNode` does nothing because `TestParentSetNode` must have more information to make an accurate decision. If the alarm should be in another state, `TestParentSetNode` fires the appropriate trigger to transition the alarm into that state. If the alarm is already in the correct state, `TestParentSetNode` just refreshes the node update time so the node's children can accurately update their statuses based on the node's status.

The code for this subroutine is:

```
# The purpose of this subroutine is to test the parent
# node status and, if the node is not in a terminal state
# but should be, fire a trigger to make it so. If the node
# is already in the correct state, just refresh the node
# update time. You must make sure that all parents are
# being monitored with the status alarms.
use NC;
my $NodeUpdateTime; # Last time node status was updated
my $LastNodeStatus; # Last node status
my @Parents = (); # Array of parents
my $Parent; # Parent Node
my $ParentUpdateTime; # Last time parent node status was updated
my $ParentStatus; # Last parent status
my $TriggerFlag = "NotSet";
my $ParentNotUpdated = 0; # Remember if we have any parents not updated

#Define all triggers that can be fired
DefineTrigger('UnReachable');
DefineTrigger('Down');
DefineTrigger('Testing');

# Get the last node status and update time for this node
($LastNodeStatus,$NodeUpdateTime) = NC::GetNodeStatus($NodeName);

# Get the array of parents for this node
@Parents = NC::GetParents($NodeName);
if( defined( $Parents[0] ) )
{
    # Test each parent, if any are ok, we assume the node
    # is reachable. Parents update time must be past the
    # last time the node was updated or we can't assume the
    # status is accurate.
    foreach $Parent (@Parents)
    {
        ($ParentStatus,$ParentUpdateTime) = NC::GetNodeStatus($Parent);
```

```

if( $ParentUpdateTime >= $NodeUpdateTime )
{
    # Using TriggerFlag to store name of trigger to be fired. If any
    # parent is found to be up, then the flag will be set to down. If
    # all parents are down or unreachable, then the flag will be set
    # to unreachable. If no parents are down and at least one parent
    # is testing, set flag to testing. Otherwise, it will remain not
    # set and we will update the node's current status and time. Testing
    # handles the case where one parent is testing and another is
    # unreachable. We need to make sure we do not mark the node as
    # unreachable until the parent node in testing goes to some final
    # state because that state could be agent down which is treated
    # as up.
    if( ($ParentStatus eq "Down" || $ParentStatus eq "UnReachable") && $TriggerFlag eq
"NotSet" )
    {
        $TriggerFlag = "UnReachable";
    }
    elseif( $ParentStatus eq "Up" )
    {
        $TriggerFlag = "Down";
    }
    elseif( $ParentStatus eq "Testing" && $TriggerFlag ne "Down" )
    {
        $TriggerFlag = "Testing";
    }
}
else
{
    # Remember that we have at least one parent that hasn't been updated.
    $ParentNotUpdated = 1;
}
}
}
else
{
    # Node does not have parents so assume down
    $TriggerFlag = "Down";
}

# If I have at least one parent not updated and I do not have

```



```
# any up parents, Set TriggerFlag to testing.
if( $ParentNotUpdated && $TriggerFlag ne "Down" )
{
    $TriggerFlag = "Testing";
}

if( $TriggerFlag ne "Testing" )
{
    # Fire trigger if node's status should change. Otherwise
    # refresh the time for the node's current state.
    if( $TriggerFlag ne $LastNodeStatus )
    {
        # Fire trigger
        FireTrigger( $TriggerFlag );
    }
    else
    {
        # Refresh node status
        NC::SetNodeStatus($NodeName,$LastNodeStatus);
    }
}
}
```

Glossary

The following terms are used in this white paper.

alarm

A VERITAS NerveCenter construct that correlates one or more detected network or system events.

behavior model

The group of alarm, trigger, and property group definitions required to detect and handle a particular network or system behavior.

child

A node whose network path to the VERITAS NerveCenter server is dependent on another device (a router, for example). A child may have one or more parents.

circular transition

A transition that does not change an alarm's state.

managed node

A device on a network that is actively monitored and managed by VERITAS NerveCenter.

mask

A construct that determines which SNMP traps generate NerveCenter triggers.

node

A device on a network.

parent

A node between the VERITAS NerveCenter server and other managed nodes. A parent may have one or more children.

Perl subroutine

A Perl script that is run when the transition to which it is assigned is triggered.

poll

A construct that solicits MIB data from managed nodes running SNMP agents, evaluates the returned data, and, depending on the results, can trigger an alarm transition.

Smart Polling

VERITAS NerveCenter technology that lets NerveCenter suppress polling to nodes that are unreachable so unnecessary network traffic is minimized.

state

Part of the finite state machine that represents an alarm. Alarms transition from state to state.



status

The real-time status of a node—up, down, testing, or unreachable.

transition

A change from one state in alarm state diagram to another. During a transition, VERITAS NerveCenter carries out any actions assigned to the transition.

trigger

A flag sent to an alarm to indicate that an event has been detected or a condition has been satisfied.

