# Opennms Linkd

* [1 Availability](http://www.opennms.org/wiki/Linkd#Availability)
* [2 Quick Start](http://www.opennms.org/wiki/Linkd#Quick_Start)
* [3 Concept](http://www.opennms.org/wiki/Linkd#Concept)
  + [3.1 Discovery Using Bridge Forwarding Table, Spanning Tree Information and Ip Net To Media Table](http://www.opennms.org/wiki/Linkd#Discovery_Using_Bridge_Forwarding_Table.2C_Spanning_Tree_Information_and_Ip_Net_To_Media_Table)
  + [3.2 Discovery Using Ip Routes](http://www.opennms.org/wiki/Linkd#Discovery_Using_Ip_Routes)
  + [3.3 Discovery Using Cisco Discovery Protocol](http://www.opennms.org/wiki/Linkd#Discovery_Using_Cisco_Discovery_Protocol)
  + [3.4 Discovery Using Link Layer Discovery Protocol](http://www.opennms.org/wiki/Linkd#Discovery_Using_Link_Layer_Discovery_Protocol)
  + [3.5 Discovery Using OSPF](http://www.opennms.org/wiki/Linkd#Discovery_Using_OSPF)
  + [3.6 Discovery Using IS-IS](http://www.opennms.org/wiki/Linkd#Discovery_Using_IS-IS)
* [4 Database](http://www.opennms.org/wiki/Linkd#Database)
* [5 Tips](http://www.opennms.org/wiki/Linkd#Tips)
  + [5.1 Sample Config](http://www.opennms.org/wiki/Linkd#Sample_Config)
* [6 Vlans](http://www.opennms.org/wiki/Linkd#Vlans)
* [7 AutoDiscovery](http://www.opennms.org/wiki/Linkd#AutoDiscovery)
* [8 WEB User interface](http://www.opennms.org/wiki/Linkd#WEB_User_interface)
* [9 Troubleshooting](http://www.opennms.org/wiki/Linkd#Troubleshooting)
  + [9.1 Link is not discovered](http://www.opennms.org/wiki/Linkd#Link_is_not_discovered)
* [10 Version History/Availability](http://www.opennms.org/wiki/Linkd#Version_History.2FAvailability)
* **Availability**

This feature first appeared in the 1.3.2 unstable release.

Linkd experienced significant changes between the following series of releases:

**1.6.x » 1.8.x**

Many changes. Some users reported significant regressions across this delta, while others seemed mostly unaffected. The types of devices in the network seem to play a major role in the experience.

**1.8.x » 1.10.x**

Many bug fixes aimed at getting the 1.10 Linkd to functional parity with the 1.6 Linkd in environments where regressions were reported. A big part of this work involved refactoring Linkd's code to make it easier to target in unit tests.

**1.10.x » 1.12.x**

Tons of work including both bug fixes and new functionality. Support for link discovery using OSPF and LLDP MIBs is new in this release series.

**1.13.x**

More bug fixes including support for duplicated ip addresses, support for IS-IS protocol links discovery

The following notes apply to 1.13.0-SNAPSHOT release.

* **Quick Start**

OpenNMS by default has linkd disabled, so you should enable it.

To enable linkd follow these simple steps:

* Go to service-configuration.xml and uncomment the following lines:

<service>

<name>OpenNMS:Name=Linkd</name>

<class-name>org.opennms.netmgt.linkd.jmx.Linkd</class-name>

<invoke at="start" pass="0" method="init"/>

<invoke at="start" pass="1" method="start"/>

<invoke at="status" pass="0" method="status"/>

<invoke at="stop" pass="0" method="stop"/>

</service>

* Restart openNMS, you can follow the progress of Linkd in the ${opennms.home}/logs/daemon/link.log

The linkd daemon collects a lot of data from nodes to perform the network topology discovery, this should affect performance, so if you are running opennms with default memory allocation remeber to increase JAVA\_HEAP\_SIZE by 64Mbyte for each 5 threads used. By default linkd uses 5 thread to do the work.

Links and data collected by Linkd can be visualized in **maps** and in **WEB UI** element section.

* **Concept**

Linkd is layer 2/3 iso/osi model network topology discovery daemon.

Linkd Daemon is controlled by a '"Configuration File*linkd-configuration.xml (located in the ${opennms.home}/etc) and do it work in two steps:*

A) Performs SNMP data collection over the nodes (Java Class SnmpCollection)

B) Discover network topology using previously collected data (Java Class DiscoveryLink)

The number of threads is configurable using the "threads" property in linkd-configuration.xml (defatults to 5). The configuration property "initial\_sleep\_time" defines a timeout for the linkd daemon prior starting the Snmp Data Collection. The snmp\_poll\_interval is the default interval for rescheduling data collection and topology discovery. The discovery\_link\_interval is a delay in scheduling (B). In practice B must be run when all the collection over nodes has been completed: this interval must be configured according to the size of the network and to the time it is used to perform all the collections. Rescheduling for Discovery Link is performed using the snmp\_poll\_interval.

Linkd has packages like all the old style daemon service in opennms, so it is possible to group the nodes together and to configure snmp data collection (linkd specific) and discovery strategy for each package.

Linkd has 6 discovery strategy controlled by correspondant properties in the configuration file. Note that all these property can be overwritten in packages.

* **Discovery Using Bridge Forwarding Table, Spanning Tree Information and Ip Net To Media Table**

Linkd defaults to discovery to use use-bridge-discovery property. Set the use-bridge-discovery="true|false" in linkd-configuration.xml to enable/disable discovery using the bridge discovery strategy.

This strategy uses the bridge forwarding table on bridges to assign bridge port to mac addresses. A mac addresses with a forwarding rule on a bridge port means that the mac address is learned on that bridge port. If the bridge port it is not a backbone bridge port (that is a link between two bridges) then you got a "mac link" on the port, or you can guess that the mac address (and the device that holds that mac address) is directly linked to the bridge port. This strategy uses the ip net to media address table to find what is the ip address associated with the discovered "mac link" (this is the same as running show arp or arp -a). If the ip address belong to an opennms node then the link can be saved. Clearly previously linkd founds the "backbone links" i.e. the links between the device that are bridge devices. The way in which backbone links are found is either walking the spanning tree informations on the bridges (the spanning tree has the information about the topology of the network) or running a "diskra algoritm" to discover itself the topology among bridges.

For each node when using use-bridge-discovery strategy linkd walks the following snmp tables:

* ipNetToMediaTable (oid .1.3.6.1.2.1.4.22.1) (ipNetToMediaIfIndex, ipNetToMediaPhysAddress, ipNetToMediaNetAddress, ipNetToMediatype) that holds information about mac/ip association.
* dot1dBaseGroup (root oid .1.3.6.1.2.1.17.1) (dot1dBaseBridgeAddress,dot1dBaseNumPorts,dot1dBaseType) which holds the basic fundamental information of the bridge.
* dot1dBaseTable (oid .1.3.6.1.2.1.17.1.4.1) (dot1dBasePort,dot1dBasePortIfIndex,dot1dBasePortCircuit,dot1dBasePortDelayExceededDiscards,dot1dBasePortMtuExceededDiscards) which holds the association between the bridge port and the ifindex
* dot1dStpGroup (oid .1.3.6.1.2.1.17.2) (dot1dStpProtocolSpecification, dot1dStpDesignatedRoot, dot1dStpRootPort) which holds all the basic information regarding the spanning tree protocol
* dot1dStpTable (oid .1.3.6.1.2.1.17.2.15.1) (dot1dStpPort,dot1dStpPortDesignatedRoot,dot1dStpPortDesignatedCost,dot1dStpPortDesignatedBridge,dot1dStpPortDesignatedPort) the designated port and designated bridge identify the next bridge, this is the main way to find backbone links.
* dot1dTpFdbTable (oid .1.3.6.1.2.1.17.4.3.1) (dot1dTpFdbAddress,dot1dTpFdbPort,dot1dTpFdbStatus) that holds the information related to the bridge port.

The bridge information walk is performed on some devices (like cisco) over each vlan defined, in fact these devices run a bridge instance with its own information and agent for each vlan. See the vlan entry for details about vlans.

The ipNeTToMediaTable is saved in **atinterface** database table. The dot1dBaseGroup and dot1dStpGroup are saved in **stpnode** database table The dot1dBaseTable and dot1dStpTable are saved in **stpinterface**database table

It is possible to control the persistence of the stpnode and stpinterface using the following configuration properties in linkd-configuration.xml: save-stp-node-table="true|false" save-stp-interface-table="true|false"

The default is to persist data (true).

if you do not want to use bridge discovery and persist data in stpnode and stpinterface you must set the following properties:

use-bridge-discovery="false" save-stp-node-table="false" sva-stp-interface-table="false".

See ${opennms.home}/share/xsds/linkd-configuration.xsd for detailed configuration option.

* **Discovery Using Ip Routes**

Linkd defaults to true to use this ip route next hop strategy. Set use-ip-route-discovery="true|false" to enable/disable ip route discovery strategy. This strategy look at the next hop in nodes routing table to identify a possible node. If next hop is a valid ip address then a link is found between the walked node and the node holding the next hop ip. This is a layer 3 network topology discovery strategy. This strategy was originally written to discover links over Serial, Frame Relay and ATM interfaces.

For each node that is set to use-ip-route-discovery to true linkd walks the ip route table.

Linkd use by default IpCidrRouteTable (oid .1.3.6.1.2.1.4.24.4.1). It is possible to use an old supported snmp ip route table IpRouteTable (oid .1.3.6.1.2.1.4.21.1) in linkd-configuration.xml using the <iproutes> element. Here follows the configuration in default lin kd-configuration.xml to forse to use the IpRouteTable to get the routing table information.

<iproutes>

<vendor vendor\_name="netscreen" sysoidRootMask=".1.3.6.1.4.1.3224"

class-name="org.opennms.netmgt.linkd.snmp.IpRouteTable">

<specific>1.51</specific>

</vendor>

<vendor vendor\_name="cisco" sysoidRootMask=".1.3.6.1.4.1.9"

class-name="org.opennms.netmgt.linkd.snmp.IpRouteTable">

<specific>1.569</specific>

<specific>5.42</specific>

</vendor>

<vendor vendor\_name="darwin" sysoidRootMask=".1.3.6.1.4.1.8072.3"

class-name="org.opennms.netmgt.linkd.snmp.IpRouteTable">

<specific>2.255</specific>

</vendor>

</iproutes>

The ip route table is persisted by linkd defaults in **iprouteinterface** database table. use save-ip-route-interface-table="true|false" to enable/disable persisting.

To disable ip route discovery and ip route persisting use the following configuration: use-ip-route-discovery="false" save-ip-route-interface-table="false".

The ip route discovery strategy disabled by default over ethernet interface (as it was in the old times ethernet used only on LAN and you cab use bridge discovery there to get the links). It is possible to discovery ip route next hop links over an ethernet interface setting force-ip-route-discovery-on-ethernet="true" (while it defaults to false).

For devices and networks with big routing table (using OSPF or BGP in a big network) it is recommended to disable this discovery strategy.

* **Discovery Using Cisco Discovery Protocol**

Linkd defaults to true to use cisco discovery protocol strategy. To enable/disable cdp discovery set use-cdp-discovery="true|false" in linkd-configuration.xml.

Linkd Cisco Discover Protocol Strategy use the CDP Cache Table to get links among Cisco Nodes.

For each node that has use-cdp-discovery="true" the following table is walked:

* cdpCacheTable (oid .1.3.6.1.4.1.9.9.23.1.2.1.1) (cdpCacheIfIndex,cdpCacheDeviceIndex,cdpCacheAddressType,cdpCacheAddress, cdpCacheVersion, cdpCacheDeviceId,cdpCacheDevicePort)

this table contains the information for the Cisco CDP protocol neighbor devices

No persistance is done for the collected data

* **Discovery Using Link Layer Discovery Protocol**

Linkd by default use the LLDP discovery strategy. Configure use-lldp-discovery="true|false" to enable/disable discovery links using the lldp protocol.

The lldp protocol holds the data for the discovered links on a specified device. For each node with use-lldp-discovery="true" the following data is collected:

* lldpLocalGroup (oid .1.0.8802.1.1.2.1.3) (lldpLocChassisIdSubtype,lldpLocChassisId,lldpLocSysName) used to identify the lldp nodes
* lldpLocTable (oid .1.0.8802.1.1.2.1.3.7.1) (lldpLocPortNum,lldpLocPortIdSubtype,lldpLocPortId) used to identify the ports
* lldpRemTable (oid .1.0.8802.1.1.2.1.4.1.1) (lldpRemLocalPortNum,lldpRemChassisIdSubtype,lldpRemChassisId,lldpRemPortIdSubtype,lldpRemPortId,lldpRemSysName) used to identify linked lldp nodes.

No persisting for collected data.

* **Discovery Using OSPF**

The Open Short Path discovery Strategy use the OSPF routing info for finding the links. It is enabled by default. Set use-ospf-discovery="true|false" in linkd-configuration.xml to enable or disable ospf discovery strategy.

For each node supporting use-ospf-discovery="true" the following snmp data is collected:

* ospfGeneralGroup (oid .1.3.6.1.2.1.14.1) (ospfRouterId) the router identifier is found. This is used by OSPF protocol to uniquely identify the device.
* ospfNbrTable (oid .1.3.6.1.2.1.14.10.1) (ospfNbrIpAddr,ospfNbrAddressLessIndex,ospfNbrRtrId,ospfNbrState) which holds the information about the ospf nei.

In practice the link is found using the intersection of information between the ospf nei table of the ospf routers.

This is a strictly layer 3 link discovery strategy.

No information is persisted.

* **Discovery Using IS-IS**

The Intermediate System to Intermediate System discovery Strategy use the IS-IS protocol routing information for finding the links. It is enabled by default. Set use-isis-discovery="true|false" in linkd-configuration.xml to enable or disable isis discovery strategy.

For each node supporting use-isis-discovery="true" the following snmp data is collected:

* IsisSystemID (oid .1.3.6.1.2.1.138.1.1.1.3.0) the IS-IS router system identifier is found. This is used by IS-IS protocol to uniquely identify the device.
* IsisAdminState (oid .1.3.6.1.2.1.138.1.1.1.8.0) the IS-IS router admin state is found. This is used by IS-IS protocol to check IS-IS status of the device.
* isisCircTable (oid .1.3.6.1.2.1.138.1.3.2) which holds the information about the IS-IS circuit status.
* isisISAdjTable (oid .1.3.6.1.2.1.138.1.6.1) (isisISAdjState,isisISAdjNeighSNPAAddress,isisISAdjNeighSysType,isisISAdjNeighSysID,isisISAdjNbrExtendedCircID) which holds the information about the is is adjenciences.

In practice a link is found using the intersection of information between the isis adj table of the isis routers.

This is a strictly layer 3 link discovery strategy.

No information is persisted.

* **Database**

DiscoveryLink process save discovered links (the topology) into database table datalinkinterface.

Each db table (atInterface, stpNode, stpInterface,vlan, ipRouteInterface, DataLinkInterface) row has a status tag that signify the status of the row.

"A" means Active (that means that linkd has this information updated)

"N" means Not Polled(that means that linkd has this information but was not lastly updated)

"D" means Deleted (that means that linkd has not more this information)

The row are deleted if they are setted to 'D' and after the last poll time is older the three times snmp\_poll\_interval.

* **Tips**

linkd-configuration.xml default configuration is suitable for small or medium sized network (200/300 nodes). Try to limit your initial linkd discovery to a /24 subnet to experiment. Linkd against a large network range can take a long time and use a lot of server resources.

Network topology discovery on ethernet LANs is done using the bridge and ipnettomedia tables. So if a complete discovery is needed it's necessary to have all these data. The bridge table is achieved just managing and enabling SNMP protocols on each layer 2 device. The information on ipnettomedia is shared by nodes on the network. Usually an almost complete table is found on the LAN's default gateway.

'To full discovery network topology it's recommended that every network device in the area network works using SNMP.

In an multivlan environnement to resolve ip to mac address it is required that layer 3 switches should be accessed via SNMP.

It is also suggested to enable SNMP on OpenNMS server to get ip to mac addresses ipnettomedia table.

Remember that increasing threads requires an increment of JAVA\_HEAP\_SIZE as previously discussed.

* **Sample Config**

Here are a couple sample configs tuned to different environments. Maintaining control of linkd discovery protocols makes the process more efficient and faster.

<package name="CampusLAN" use-ospf-discovery="false"

enable-vlan-discovery="true"

use-ip-route-discovery="false"

save-route-table="false"

use-cdp-discovery="true"

use-bridge-discovery="false"

save-stp-node-table="false"

save-stp-interface-table="false"

use-lldp-discovery="false">

<filter>IPADDR != '0.0.0.0'</filter>

<include-range begin="10.255.122.129" end="10.255.122.190"/>

</package>

<!-- Maybe comment out different configs until you need to test

<package name="CampusLAN2" use-ospf-discovery="false"

enable-vlan-discovery="true"

use-ip-route-discovery="false"

save-route-table="false"

use-cdp-discovery="true"

use-bridge-discovery="false"

save-stp-node-table="false"

save-stp-interface-table="false"

use-lldp-discovery="false">

<filter>IPADDR != '0.0.0.0'</filter>

<include-range begin="10.255.122.65" end="10.255.122.126"/>

</package>

-->

<package name="T1-WAN-Network"

enable-vlan-discovery="true"

use-ip-route-discovery="false"

save-route-table="false"

use-cdp-discovery="true"

use-ospf-discovery="true"

use-bridge-discovery="false"

save-stp-node-table="false"

save-stp-interface-table="false"

use-lldp-discovery="false">

<filter>IPADDR LIKE '10.255.120.%'</filter>

<include-range begin="10.255.120.1" end="10.255.120.254"/>

</package>

* **Vlans**

Linkd try to collect vlan information. Set enable-vlan-discovery="true|false" to enable/disable vlan discovery.

To discovery vlan over a node a "Vlan" class must be specified. The classes are defined inside the <vlans> element in linkd-configuration.xml. If there is no matching class for the node no collection could be performed to get vlan.

The association between Vlan Class and nodes is made using the node sysoid.

The vlans element maps Vendor Devices Sysoid with the specific class to use to get snmp Vlan table.

Each vendor has its own specific snmp vlan table.

Once the general settings are in place the only thing left to tell the linkd is for which sysoids try to collects vlan This is controlled by the following elements:

* vlans: specify a list of sysoids for which download SNMP vlan table.
* vendor: Allow you to specify a set of sysoids whose root is set by a vendor sysoidrootmask attribute and using the following tags.
* include-range: <begin>start-sub-oid</begin><end>end-sub-oi</end>
* specific: sub-oid
* exclude-range: <begin>start-sub-oid</begin><end>end-ip-address</end>

The vendor tags in the default configuration file are tested. If you have switches with different sysoid you have to find an appropriate vlan class to get Vlans and add a vendor tag for this.

For example consider that you have a Cisco catalyst295024 whose sysoid is: 1.3.6.1.4.1.9.1.324

And you want to get Vlan info on that device. You should look under Cisco vendor tag if this sysoid is in place:

<vendor vendor\_name="cisco" sysoidRootMask=".1.3.6.1.4.1.9"

class-name="org.opennms.netmgt.linkd.snmp.CiscoVlanTable">

<specific>1.217</specific>

<specific>1.218</specific>

<specific>1.221</specific>

<specific>1.247</specific>

<specific>1.248</specific>

<specific>1.283</specific>

<specific>1.300</specific>

<specific>1.324</specific>

<specific>1.366</specific>

<specific>1.367</specific>

<specific>5.46</specific>

</vendor>

and you see is in place.

if you want Vlan on cat6506 whose sysoid is: 1.3.6.1.4.1.9.1.282 you should add to cisco vendor tag:

<specific>1.282</specific>

* **AutoDiscovery**

AutoDiscovery Let linkd send a newSuspectEvent when an unknown ip address is found. Set auto-discovery="true|false" in linkd-configuration.xml to enable disable autodiscovery.

By default autodiscovery is disabled.

* **WEB User interface**

There are some JSP (like box in includes dir and pages in element dir) to take a look at the infos saved in db tables. Here you see the complete list of what is going to do:

A. mac address search in element/index.jsp

B. Include mac address info for element/interface.jsp

C. Include link to element/linkednode.jsp in element/node.jsp

D. Include link to element/routeipnode.jsp in element/node.jsp

E. Include link to element/bridgenode.jsp in element/node.jsp

F. Include box includes/nodebridge-box.jsp in element/node.jsp

G. Include box includes/noderouteInfo-box.jsp in element/node.jsp

H. Include box includes/nodeStpint-box.jsp in element/node.jsp

I. Include box includes/interfaceLink-box.jsp in element/interface.jsp

L. Include box includes/interfaceStp-box.jsp in element/interface.jsp

Links should also be visualized on [maps](http://www.opennms.org/wiki/Maps)

* **Troubleshooting**
* **Link is not discovered**
* is the sysoid of your device included in the configuration described above? Some devices using non-standard mib tables won't work. If the sysoid is not included but a similar device from the same vendor is referenced in the config try to add the sysoid of your device. If it works open an enhancement bug to get it included in future releases.
* is the ip address of your device included in a range in the configuration described above?
* is the snmp community configured in opennms? Go to the node page of your device in opennms and check if "SNMP Attributes" is visible on the node page
* can you snmp walk your device from your opennms server?
* If you can do an snmp walk grep in the output from the snmp walk for the MAC address of the device on the other side of the missing link. If the MAC is in there, and opennms has the snmp values of the device, linkd should discover the link.
* What is your intervall for linkd to run? Maybe you just have to wait for the next run of linkd. Check the log files when it is running.
* Is the device on the other end of the missing link always busy talking? If it is "quiet" while linkd collects the snmp tables it depends on the timeouts of the device where linkd tries to get the MAC tables - if the timeout for the MAC tables is too short the MAC will be thrown away and linkd can't find anything.
* try enable-discovery-download="true" if you have many failing linkd data collection. This flag enables DiscoveryLink class to try to download itself some missing bridge data that could have been discarded.
* **Version History/Availability**
* This feature was added in [version 1.3.2](http://www.opennms.org/wiki/Releases#OpenNMS_1.3.2)
* This feature was enhanced or modified in [version 1.10.0](http://www.opennms.org/wiki/Releases#OpenNMS_1.10.0)
* This feature was enhanced or modified in [version 1.11.3](http://www.opennms.org/wiki/Releases#OpenNMS_1.11.3)

# Link monitor

* [1 Feature Description](http://www.opennms.org/wiki/Link_Monitoring#Feature_Description)
  + [1.1 Prerequisites](http://www.opennms.org/wiki/Link_Monitoring#Prerequisites)
  + [1.2 Configuration](http://www.opennms.org/wiki/Link_Monitoring#Configuration)
    - [1.2.1 Configure EndPoint Types](http://www.opennms.org/wiki/Link_Monitoring#Configure_EndPoint_Types)
      * [1.2.1.1 Ping EndPoint](http://www.opennms.org/wiki/Link_Monitoring#Ping_EndPoint)
      * [1.2.1.2 SNMP Match EndPoint](http://www.opennms.org/wiki/Link_Monitoring#SNMP_Match_EndPoint)
    - [1.2.2 Configure SNMP](http://www.opennms.org/wiki/Link_Monitoring#Configure_SNMP)
    - [1.2.3 Configure the Poller](http://www.opennms.org/wiki/Link_Monitoring#Configure_the_Poller)
    - [1.2.4 Configure Link Adapter](http://www.opennms.org/wiki/Link_Monitoring#Configure_Link_Adapter)
    - [1.2.5 Configure Foreign Source](http://www.opennms.org/wiki/Link_Monitoring#Configure_Foreign_Source)
* [2 Version History/Availability](http://www.opennms.org/wiki/Link_Monitoring#Version_History.2FAvailability)
* **Feature Description**

This feature is a link service monitor that monitors the status of a link end point. A link end point is a node that has the end point service. For a node to have the end point service it must match at least one of the conditions in the configuration. When a endpoint status changes it will update the link status on the map. This feature has a number of nifty features and configuring it might be a little tricky hopefully this page clearly demonstrates the configuration and usage of this feature.

A wireless link service monitor will be developed that can be used to represent the status of a link. This service monitor will most likely use SNMP but could use HTTP to determine the link status. Links on the map will be updated to display the link state (up/down).

See [Link Monitoring Design](http://www.opennms.org/wiki/Link_Monitoring_Design) for an overview of the architecture.

* **Prerequisites**

This feature depends on the [Automatic\_Map\_Creation](http://www.opennms.org/wiki/Automatic_Map_Creation" \o "Automatic Map Creation) feature. Prior to configuring this feature you will need to install and configure the opennms-plugin-provisioning-map and opennms-plugin-provisioning-link packages, as well as configure them. You can find all that [here](http://www.opennms.org/wiki/Automatic_Map_Creation)

* **Configuration**

To configure link monitoring you will need to configure the following, depending on your network configuration and devices. This feature is highly customizable and should be able to cover many different device types.

* **Configure EndPoint Types**

This configuration covers what type of devices and how to detect and poll the end point service. There are two types of end points a pingable end point and a Snmp end point. You can configure as many as you would like. The default configuration file is $OPENNMS\_HOME/etc/endpoint-configuration.xml. Most likely you may have a combination of different devices that serve as end points to a link and you can have as many devices configured.

* **Ping EndPoint**

This configuration defines one pingable device. Because we use Snmp sysOid to determine the device you will need Snmp to be enabled, even though the device's status is determined by pinging it. This is for the initial end point detection.

<endpoint-types xmlns="<http://xmlns.opennms.org/xsd/config/endpoint-types>" endpoint-service-name="EndPoint">

<endpoint-type name="pingable">

<sysoid-mask>.1.3.6.1.4.1.XXXX</sysoid-mask>

<pingable />

</endpoint-type>

</endpoint-types>

* **SNMP Match EndPoint**

If the device has multiple Snmp oids that need to be checked to determine the status of the end point you can add some match criteria that will be checked everytime the node is polled. The configuration should be fairly easy to understand, anything in the <and> tag will be the criteria for calculating the status of the end point. You can have unlimited nested conditions with either <and/> or< or/> tags.

<endpoint-types xmlns="<http://xmlns.opennms.org/xsd/config/endpoint-types>" endpoint-service-name="EndPoint">

<endpoint-type name="snmpMatch">

<sysoid-mask>.1.3.6.1.4.1.YYYY</sysoid-mask>

<and>

<match-oid oid=".1.3.6.1.4.1.YYYY.1">^1$</match-oid>

<match-oid oid=".1.3.6.1.4.1.YYYY.2">^1$</match-oid>

<match-oid oid=".1.3.6.1.4.1.YYYY.3">^1$</match-oid>

<or>

<match-oid oid=".1.3.6.1.4.1.YYYY.4">^1$</match-oid>

<match-oid oid=".1.3.6.1.4.1.YYYY.5">^1$</match-oid>

</or>

</and>

</endpoint-type>

</endpoint-types>

* **Configure SNMP**

If your endpoint devices do not use the default "public" SNMP community name, you will have to configuresnmp-config.xml to define IP address ranges and community strings for your devices.

* **Configure the Poller**

Add the EndPoint service monitor to poller-configuration.xml.

<poller-configuration threads="30" serviceUnresponsiveEnabled="false">

...

<package name="example1">

...

<service name="EndPoint" interval="300000" user-defined="false" status="on">

<parameter key="retry" value="2" />

<parameter key="timeout" value="3000" />

</service>

</package>

...

<monitor service="EndPoint" class-name="org.opennms.netmgt.provision.adapters.link.EndPointMonitor" />

...

</poller-configuration>

The default configuration file location is $OPENNMS\_HOME/etc/endpoint-configuration.xml.

* **Configure Link Adapter**

The link adapter will need to know the name of the endpoint service you created.

<link-adapter-configuration xmlns="<http://xmlns.opennms.org/xsd/config/map-link-adapter>"

endpoint-service-name="EndPoint">

<for match="^link-(.\*?)-to-(.\*?)$">

<link>link-$2-to-$1</link>

</for>

</link-adapter-configuration>

* **Configure Foreign Source**

Next, you should configure the foreign source so that it has an EndPoint Detector plugin available.

Create a detector service, the name should match the EndPoint service name used above.

This will make sure that nodes which are endpoints in a link are detected when they are provisioned.

* The detector plugin can optionally take a endpoint-type-url parameter which defines the location of the[endpoint configuration](http://www.opennms.org/wiki/Link_Monitoring#Configuration) file. The default is $OPENNMS\_HOME/etc/endpoint-configuration.xml.
* The detector plugin can optionally take a endpoint-name-pattern parameter which defines a regular expression to apply to the node name. Only nodes with node labels that match this pattern can be an EndPoint.
* **Version History/Availability**
* This feature was added in [version 1.7.8](http://www.opennms.org/wiki/Releases#OpenNMS_1.7.8)