Network Working Group
Internet-Draft
Intended status: Experimental
Expires: March 26, 2014

A. Clemm
J. Medved
E. Voit
Cisco Systems
September 22, 2013

Mounting YANG-Defined Information from Remote Datastores
draft-clemm-netmod-mount-01.txt

Abstract

This document introduces a new capability that allows YANG datastores to reference and incorporate information from remote datastores. This is accomplished using a new YANG data model that allows to define and manage datastore mount points that reference data nodes in remote datastores. The data model includes a set of YANG extensions for the purposes of declaring such mount points.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 26, 2014.

Table of Contents

1. Introduction

This document introduces a new capability that allows YANG datastores [RFC6020] to incorporate and reference information from remote datastores. This is provided by introducing a mountpoint concept. This concept allows to declare a YANG data node as a "mount point", under which a remote datastore subtree can be mounted. To the user

of the primary datastore, the remote information appears as an integral part of the datastore. It allows remote data nodes and datastore subtrees to be inserted into the local data hierarchy, arranged below local data nodes. The concept is reminiscent of concepts in a Network File System that allows to mount remote folders and make them appear as if they were contained in the local file system of the user's machine.

The ability to mount information from remote datastores is new and not covered by existing YANG mechanisms. Until now, management information provided in a datastore has been intrinsically tied to the same server. In contrast, the capability introduced here allows the server to represent information from remote systems as if it were its own and contained in its own local data hierarchy.

YANG does provide means by which modules that have been separately defined can reference and augment one another. YANG also does provide means to specify data nodes that reference other data nodes. However, all the data is assumed to be instantiated as part of the same datastore, for example a datastore provided through a NETCONF server [RFC6241]. Existing YANG mechanisms do not account for the possibility that some information that needs to be referred not only resides in a different subtree of the same datastore, or was defined in a separate module that is also instantiated in the same datastore, but that is genuinely part of a different datastore that is provided by a different server.

The ability to mount data from remote datastores is useful to address various problems that several categories of applications are faced with:

One category of applications that can leverage this capability concerns network controller applications that need to present a consolidated view of management information in datastores across a network. Controller applications are faced with the problem that in order to expose information, that information needs to be part of their own datastore. Today, this requires support of a corresponding YANG data module. In order to expose information that concerns other network elements, that information has to be replicated into the controller's own datastore in the form of data nodes that may mirror but are clearly distinct from corresponding data nodes in the network element's datastore. In addition, in many cases, a controller needs to impose its own hierarchy on the data that is different from the one that was defined as part of the original module. An example for this concerns interface configuration data, which would be contained in a top-level container in a network element datastore, but may need to be contained in a list in a controller datastore in order to be able to distinguish instances from different network elements under

the controller's scope. This in turn would require introduction of redundant YANG modules that effectively replicate the same information save for differences in hierarchy.

By directly mounting information from network element datastores, the controller does not need to replicate the same information from multiple datastores, nor does it need to re-define any network element and system-level abstractions to be able to put them in the context of network abstractions. Instead, the subtree of the remote system is attached to the local mount point. Operations that need to access data below the mount point are in effect transparently redirected to remote system, which is the authorative owner of the data. The mounting system does not even necessarily need to be aware of the specific data in the remote subtree.

A second category of applications concerns decentralized networking applications that require globally consistent configuration of parameters. When each network element maintains its own datastore with the same configurable settings, a single global change requires modifying the same information in many network elements across a network. In case of inconsistent configurations, network failures can result that are difficult to troubleshoot. In many cases, what is more desirable is the ability to configure such settings in a single place, then make them available to every network element. Today, this requires in general the introduction of specialized servers and configuration options outside the scope of NETCONF, such as RADIUS [RFC2866] or DHCP [RFC2131]. In order to address this within the scope of NETCONF and YANG, the same information would have to be redundantly modeled and maintained, representing operational data (mirroring some remote server) on some network elements and configuration data on a designated master. Either way, additional complexity ensues.

Instead of replicating the same global parameters across different datastores, the solution presented in this document allows a single copy to be maintained in a subtree of single datastore that is then mounted by every network element that requires access to these parameters. The global parameters can be hosted in a controller or a designated network element. This considerably simplifies the management of such parameters that need to be known across elements in a network and require global consistency.

The capability of allowing to mount information from remote datastores into another datastore is accomplished by a set of YANG extensions that allow to define such mount points. For this purpose, a new YANG module is introduced. The module defines the YANG extensions, as well as a data model that can be used to manage the mountpoints and mounting process itself. Only the mounting module

and server needs to be aware of the concepts introduced here. Mounting is transparent to the models being mounted; any YANG model can be mounted.

2. Definitions and Acronyms

Data node: An instance of management information in a YANG datastore.

DHCP: Dynamic Host Configuration Protocol.

Datastore: A conceptual store of instantiated management information, with individual data items represented by data nodes which are arranged in hierarchical manner.

Data subtree: An instantiated data node and the data nodes that are hierarchically contained within it.

Mount client: The system at which the mount point resides, into which the remote subtree is mounted.

Mount point: A data node that receives the root node of the remote datastore being mounted.

Mount server: The server with which the mount client communicates and which provides the mount client with access to the mounted information. Can be used synonymously with mount target.

Mount target: A remote server whose datastore is being mounted.

NACM: NETCONF Access Control Model

NETCONF: Network Configuration Protocol

RADIUS: Remote Authentication Dial In User Service.

RPC: Remote Procedure Call

Remote datastore: A datastore residing at a remote node.

URI: Uniform Resource Identifier

YANG: A data definition language for NETCONF

3. Example scenarios

The following example scenarios outline some of the ways in which the ability to mount YANG datastores can be applied. Other mount topologies can be conceived in addition to the ones presented here.

3.1. Network controller view

Network controllers can use the mounting capability to present a consolidated view of management information across the network. This allows network controllers to not only expose network abstractions, such as topologies or paths, but also network element abstractions, such as information about a network element's interfaces, from one consolidated place.

While an application on top of a controller could in theory also bypass the controller to access network elements directly for network-element abstractions, this would come at the expense of added inconvenience for the client application. In addition, it would compromise the ability to provide layered architectures in which access to the network by controller applications is truly channeled through the controller.

Without a mounting capability, a network controller would need to at least conceptually replicate data from network elements to provide such a view, incorporating network element information into its own controller model that is separate from the network element's, indicating that the information in the controller model is to be populated from network elements. This can introduce issues such as data consistency and staleness. Even more importantly, it would in general lead to the redundant definition of data models: one model that is implemented by the network element itself, and another model to be implemented by the network controller. This leads to poor maintainability, as analogous information has to be redundantly defined and implemented across different data models. In general, controllers cannot simply support the same modules as their network elements for the same information because that information needs to be put into a different context. This leads to "node"-information that needs to be instantiated and indexed differently, because there are multiple instances across different data stores.

For example, "system"-level information of a network element would most naturally placed into a top-level container at that network element's datastore. At the same time, the same information in the context of the overall network, such as maintained by a controller, might better be provided in a list. For example, the controller might maintain a list with a list element for each network element, underneath which the network element's system-level information is

contained. However, the containment structure of data nodes in a module, once defined, cannot be changed. This means that in the context of a network controller, a second module that repeats the same system-level information would need to be defined, implemented, and maintained. Any augmentations that add additional system-level information to the original module will likewise need to be redundantly defined, once for the "system" module, a second time for the "controller" module.

By allowing a network controller to directly mount information from network element datastores, the controller does not need to replicate the same information from multiple datastores. Perhaps even more importantly, the need to re-define any network element and system-level abstractions to be able to put them in the context of network abstractions is avoided. In this solution, a network controller's datastore mounts information from many network element datastores. For example, the network controller datastore could implement a list in which each list element contains a mountpoint. Each mountpoint mounts a subtree from a different network element's datastore.

This scenario is depicted in Figure 1. In the figure, M1 is the mountpoint for the datastore in Network Element 1 and M2 is the mountpoint for the datastore in Network Element 2. MDN1 is the mounted data node in Network Element 1, and MDN2 is the mounted data node in Network Element 2.

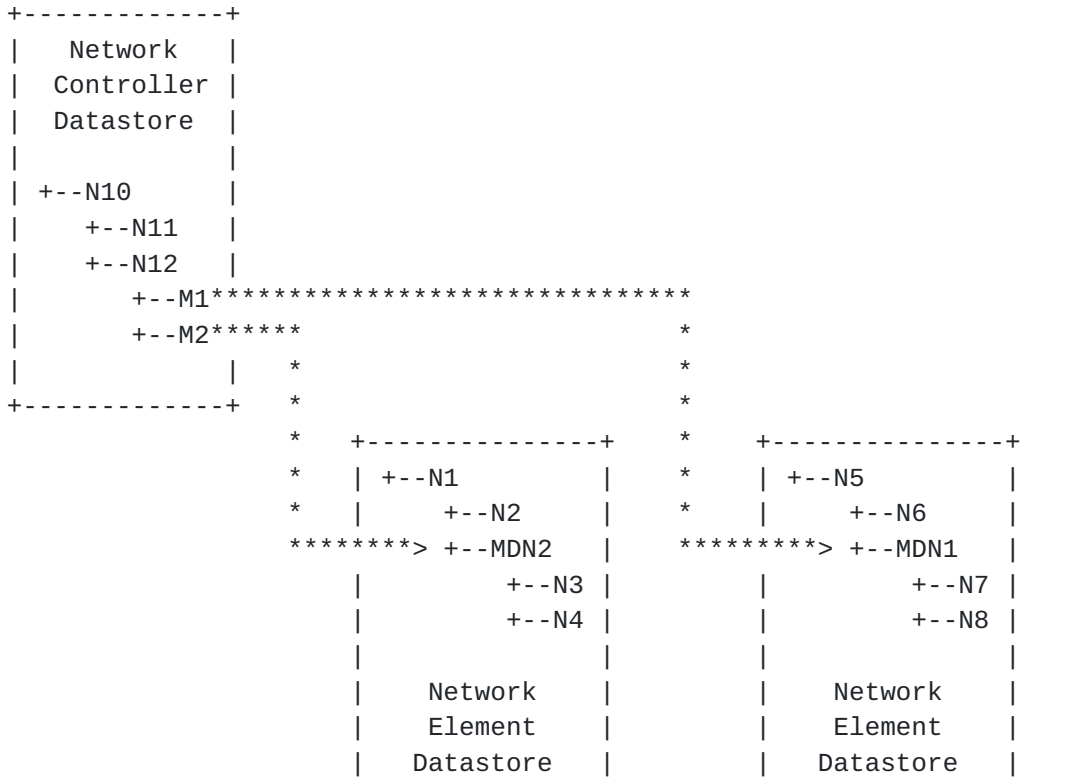


Figure 1: Network controller mount topology

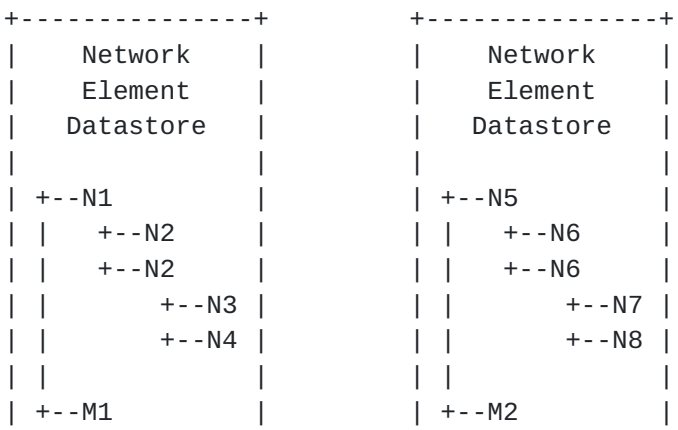
3.2. Distributed network configuration

A second category of applications concerns decentralized networking applications that require globally consistent configuration of parameters that need to be known across elements in a network. Today, the configuration of such parameters is generally performed on a per network element basis, which is not only redundant but, more importantly, error-prone. Inconsistent configurations lead to erroneous network behavior that can be challenging to troubleshoot.

Using the ability to mount information from remote datastores opens up a new possibility for managing such settings. Instead of replicating the same global parameters across different datastores, a single copy is maintained in a subtree of single datastore. This datastore can hosted in a controller or a designated network element. The subtree is subsequently mounted by every network element that requires access to these parameters.

In many ways, this category of applications is an inverse of the previous category: Whereas in the network controller case data from many different datastores would be mounted into the same datastore with multiple mountpoints, in this case many elements, each with their own datastore, mount the same remote datastore, which is then mounted by many different systems.

The scenario is depicted in Figure 2. In the figure, M1 is the mountpoint for the Network Controller datastore in Network Element 1 and M2 is the mountpoint for the Network Controller datastore in Network Element 2. MDN is the mounted data node in the Network Controller datastore that contains the data nodes that represent the shared configuration settings.



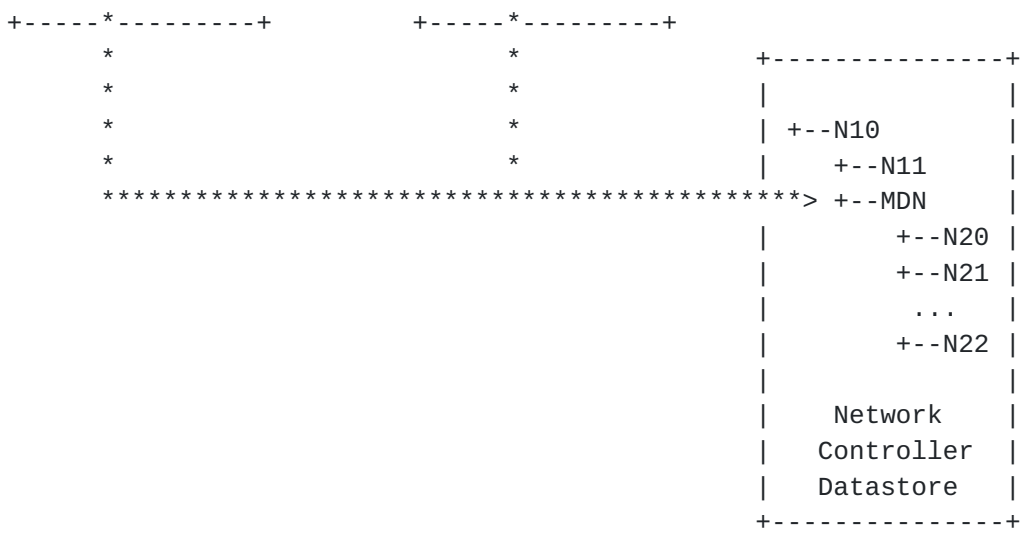


Figure 2: Distributed config settings topology

4. Operating on mounted data

This section provides a rough illustration of the operations flow involving mounted datastores.

The first thing that should be noted about these operations flows concerns that a mount client essentially constitutes a special management application that interacts with a remote system. To the remote system, the mount client constitutes in effect just another application. The remote system is the authorative owner of the data. While it is conceivable that the remote system (or an application that proxies for the remote system) provides certain functionality to facilitate the specific needs of the mount client, the fact that another system decides to expose a certain "view" of that data is fundamentally not its concern.

When a client makes a request to a server that involves data that is mounted from a remote system, the server will effectively act as a proxy to the remote system on the client's behalf. It will extract from the request the portion that involves the mounted subtree from the remote system. It will strip that portion of the local context, i.e. remove any local data paths and insert the data path of the mounted remote subtree, as appropriate. The server will then forward the transposed request to the remote system that is the authorative owner of the mounted data. Upon receiving the reply, the server will transpose the results into the local context as needed, for example map the data paths into the local data tree structure, and combine those results with the results of the remainder portion of the original request.

In the simplest and at the same time perhaps the most common case, the request will involve simple data retrieval. In that case, a "get" or "get-configuration" operation might be applied on a subtree whose scope includes a mount point. When resolving the mount point, the server issues its own "get" or "get-configuration" request against the remote system's subtree that is attached to the mount point. The returned information is then inserted into the data structure that is in turn returned to the client that originally invoked the request.

Requests that involve editing of information and "writing through" to remote systems are more complicated, particularly where they involve the need for transactions and locking. While not our primary concern at this time, implications are briefly discussed in section Section 5.4.5.

Since mounted information involves in general communication with a remote system, there is a possibility that the remote system does not respond within a certain amount of time, that connectivity is lost, or that other errors occur. Accordingly, the ability to mount datastores also involves mountpoint management, which includes the ability to configure timeouts, retries, and management of mountpoint state (including dynamic addition removal of mountpoints).

As a final note, it is conceivable that caching schemes are introduced. Caching can increase performance and efficiency in certain scenarios (for example, in the case of data that is frequently read but that rarely changes), but increases implementation complexity. Whether to perform caching is purely a local implementation decision. This specification has not requirement that caching be introduced and makes no corresponding assumptions; there is no dependency on any caching scheme.

5. Data model structure

5.1. YANG mountpoint extensions

At the center of the module is a set of YANG extensions that allow to define a mountpoint.

- The first extension, "mountpoint", is used to declare a mountpoint. The extension takes the name of the mountpoint as an argument.

- The second extension, "target", serves as a substatement underneath a mountpoint statement. It takes an argument that identifies the target system. The argument is a reference to a data node that contains the information that is needed to identify

and address a remote server, such as an IP address, a host name, or a URI [RFC3986].

- The third extension, "subtree", also serves as substatement underneath a mountpoint statement. It takes an argument that defines the root node of the datastore subtree that is to be mounted, specified as string that contains a path expression.

A mountpoint MUST be contained underneath a container. Future revisions might allow for mountpoints to be contained underneath other data nodes, such as lists, leaf-lists, and cases. However, to keep things simple, at this point mounting is only allowed directly underneath a container.

Only a single data node can be mounted at one time. While the mount target could refer to any data node, it is recommended that as a best practice, the mount target SHOULD refer to a container. It is possibly to maintain e.g. a list of mount points, with each mount point each of which has a mount target an element of a remote list. However, to avoid unnecessary proliferation of the number of mount points and associated management overhead, in order to mount lists or leaf-lists, a container containing the list respectively leaf-list SHOULD be mounted.

It is possible for a mounted datastore to contain another mountpoint, thus leading to several levels of mount indirections. However, mountpoints MUST NOT introduce circular dependencies. In particular, a mounted datastore MUST NOT contain a mountpoint which specifies the mounting datastore as a target and a subtree which contains as root node a data node that in turn contains the original mountpoint. Whenever a mount operation is performed, this condition MUST be validated by the mount client.

5.2. Mountpoint management

The YANG module contains facilities to manage the mountpoints themselves.

For this purpose, a list of the mountpoints is introduced. Each list element represents a single mountpoint. It includes an identification of the mount target, i.e. the remote system hosting the remote datastore and a definition of the subtree of the remote data node being mounted. It also includes monitoring information about current status (indicating whether the mount has been successful and is operational, or whether an error condition applies such as the target being unreachable or referring to an invalid subtree).

In addition to the list of mountpoints, a set of global mount policy settings allows to set parameters such as mount retries and timeouts.

Each mountpoint list element also contains a set of the same configuration knobs, allowing administrators to override global mount policies and configure mount policies on a per-mountpoint basis if needed.

There are two ways how mounting occurs: automatic (dynamically performed as part of system operation) or manually (administered by a user or client application). A separate mountpoint-origin object is used to distinguish between manually configured and automatically populated mountpoints.

When configured automatically, mountpoint information is automatically populated by the datastore that implements the mountpoint. The precise mechanisms for discovering mount targets and bootstrapping mount points are provided by the mount client infrastructure and outside the scope of this specification. Likewise, when a mountpoint should be deleted and when it should merely have its mount-status indicate that the target is unreachable is a system-specific implementation decision.

Manual mounting consists of two steps. In a first step, a mountpoint is manually configured by a user or client application through administrative action. Once a mountpoint has been configured, actual mounting occurs through an RPCs that is defined specifically for that purpose. To unmount, a separate RPC is invoked; mountpoint configuration information needs to be explicitly deleted.

The structure of the mountpoint management data model is depicted in the following figure, where brackets enclose list keys, "rw" means configuration, "ro" operational state data, and "?" designates optional nodes. Parantheses enclose choice and case nodes. The figure does not depict all definitions; it is intended to illustrate the overall structure.

```
rw mount-server-mgmt
+-- rw mountpoints
|   +-- rw mountpoint [mountpoint-id]
|       +-- rw mountpoint-id  string
|       +-- rw mount-target
|       |   +--: (IP)
|       |   |    +-- rw target-ip  yang:ip-address
|       |   +--: (URI)
|       |   |    +-- rw uri  yang:uri
|       |   +--: (host-name)
|       |   |    +-- rw hostname  yang:host
```

```
|       |   +-- (node-ID)
|       |   |    +-- rw node-info-ref  mnt:subtree-ref
|       |   +-- (other)
|       |        +-- rw opaque-target-id  string
|       +-- rw subtree-ref  mnt:subtree-ref
|       +-- ro mountpoint-origin enumeration
|       +-- ro mount-status  mnt:mount-status
|       +-- rw manual-mount? empty
|       +-- rw retry-timer? uint16
|       +-- rw number-of-retries? uint8
+-- rw global-mount-policies
    +-- rw manual-mount? empty
    +-- rw retry-time? uint16
    +-- rw number-of-retries? uint8
```

5.3. YANG structure diagrams

YANG data model structure overviews have proven very useful to convey the "Big Picture". It would be useful to indicate in YANG data model structure overviews the fact that a given data node serves as a mountpoint. We propose for this purpose also a corresponding extension to the structure representation convention. Specifically, we propose to prefix the name of the mounting data node with upper-case 'M'.

```
rw network
+-- rw nodes
    +-- rw node [node-ID]
        +-- rw node-ID
        +-- M node-system-info
```

5.4. Other considerations

5.4.1. Authorization

Whether a mount client is allowed to modify information in a mounted datastore or only retrieve it and whether there are certain data nodes or subtrees within the mounted information for which access is restricted is subject to authorization rules. To the mounted system, a mounting client will in general appear like any other client. Authorization privileges for remote mounting clients need to be specified through NACM (NETCONF Access Control Model) [RFC6536].

Users and implementers need to be aware of certain issues when mounted information is modified, not just retrieved. Specifically, in certain corner cases validation of changes made to mounted data

may involve constraints that involve information that is not visible to the mounting datastore. This means that in such cases the reason for validation failures may not always be fully understood by the mounting system.

Likewise, if the concepts of transactions and locking are applied at the mounting system, these concepts will need to be applied across multiple systems, not just across multiple data nodes within the same system. This capability may not be supported by every implementation. For example, locking a datastore that contains a mountpoint requires that the mount client obtains corresponding locks on the mounted datastore as needed. Any request to acquire a lock on a configuration subtree that includes a mountpoint MUST NOT be granted if the mount client fails to obtain a corresponding lock on the mounted system. Likewise, in case transactions are supported by the mounting system, but not the target system, requests to acquire a lock on a configuration subtree that includes a mountpoint MUST NOT be granted.

5.4.2. Datastore qualification

It is conceivable to differentiate between different datastores on the remote server, that is, to designate the name of the actual datastore to mount, e.g. "running" or "startup". However, for the purposes of this spec, we assume that the datastore to be mounted is generally implied. Mounted information is treated as analogous to operational data; in general, this means the running or "effective" datastore is the target. That said, the information which targets to mount does constitute configuration and can hence be part of a startup or candidate datastore.

5.4.3. Local mounting

It is conceivable that the mount target does not reside in a remote datastore, but that data nodes in the same datastore as the mountpoint are targeted for mounting. This amounts to introducing an "aliasing" capability in a datastore. While this is not the scenario that is primarily targeted, it is supported and there may be valid use cases for it.

5.4.4. Mount cascades

It is possible for the mounted subtree to in turn contain a mountpoint. However, circular mount relationships MUST NOT be introduced. For this reason, a mounted subtree MUST NOT contain a mountpoint that refers back to the mounting system with a mount target that directly or indirectly contains the originating mountpoint. As part of a mount operation, the mount points of the mounted system need to be checked accordingly.

5.4.5. Implementation considerations

Implementation specifics are outside the scope of this specification. That said, the following considerations apply:

Systems that wish to mount information from remote datastores need to implement a mount client. The mount client communicates with a remote system to access the remote datastore. To do so, there are several options:

- The mount client acts as a NETCONF client to a remote system. Alternatively, another interface to the remote system can be used, such as a REST API using JSON encodings, as specified in [I-D.bierman-netconf-restconf]. Either way, to the remote system, the mount client constitutes essentially a client application like any other. The mount client in effect IS a special kind of client application.

- The mount client communicates with a remote mount server through a separate protocol. The mount server is deployed on the same system as the remote NETCONF datastore and interacts with it through a set of local APIs.

- The mount client communicates with a remote mount server that acts as a NETCONF client proxy to a remote system, on the client's behalf. The communication between mount client and remote mount server might involve a separate protocol, which is translated into NETCONF operations by the remote mount server.

It is the responsibility of the mount client to manage the association with the target system, e.g. validate it is still reachable by maintaining a permanent association, perform reachability checks in case of a connectionless transport, etc.

It is the responsibility of the mount client to manage the mountpoints. This means that the mount client needs to populate the mountpoint monitoring information (e.g. keep mount-status up to data and determine in the case of automatic mounting when to add and remove mountpoint configuration). In the case of automatic mounting, the mount client also interacts with the mountpoint discovery and bootstrap process.

The mount client needs to also participate in servicing datastore operations involving mounted information. An operation requested involving a mountpoint is relayed by the mounting system's infrastructure to the mount client. For example, a request to retrieve information from a datastore leads to an invocation of an internal mount client API when a mount point is reached. The mount client then relays a corresponding operation to the remote datastore. It subsequently relays the result along with any responses back to the invoking infrastructure, which then merges the result (e.g. a retrieved subtree with the rest of the information that was retrieved) as needed. Relaying the result may involve the need to transpose error response codes in certain corner cases, e.g. when mounted information could not be reached due to loss of connectivity with the remote server, or when a configuration request failed due to validation error.

6. Datastore mountpoint YANG module

```
<CODE BEGINS>
file "mount@2013-09-22.yang"
module mount {
    namespace "urn:cisco:params:xml:ns:yang:mount";
    // replace with IANA namespace when assigned

    prefix mnt;

    import ietf-yang-types {
        prefix yang;
    }

    organization
        "IETF NETMOD (NETCONF Data Modeling Language) Working Group";

    contact
        "WG Web: http://tools.ietf.org/wg/netmod/
        WG List: netmod@ietf.org

        WG Chair: David Kessens
        david.kessens@nsn.com
```

```
            WG Chair: Juergen Schoenwaelder
            j.schoenwaelder@jacobs-university.de

            Editor: Alexander Clemm
            alex@cisco.com";

        description
            "This module provides a set of YANG extensions and definitions
             that can be used to mount information from remote datastores.";

        revision 2013-09-22 {
            description "Initial revision.";
        }

        feature mount-server-mgmt {
            description
                "Provide additional capabilities to manage remote mount
                 points";
        }

        extension mountpoint {
            description
                "This YANG extension is used to mount data from a remote
                system in place of the node under which this YANG extension
                statement is used.

                This extension takes one argument which specifies the name
                of the mountpoint.

                This extension can occur as a substatement underneath a
                container statement, a list statement, or a case statement.
                As a best practice, it SHOULD occur as statement only
                underneath a container statement, but it MAY also occur
                underneath a list or a case statement.

                The extension takes two parameters, target and subtree, each
                defined as their own YANG extensions.
                A mountpoint statement MUST contain a target and a subtree
                substatement for the mountpoint definition to be valid.

                The target system MAY be specified in terms of a data node
                that uses the grouping 'mnt:mount-target'.  However, it
                can be specified also in terms of any other data node that
                contains sufficient information to address the mount target,
                such as an IP address, a host name, or a URI.

                The subtree SHOULD be specified in terms of a data node of
                type 'mnt:subtree-ref'. The targeted data node MUST
```

```
            represent a container.

            It is possible for the mounted subtree to in turn contain a
            mountpoint.  However, circular mount relationships MUST NOT
            be introduced. For this reason, a mounted subtree MUST NOT
            contain a mountpoint that refers back to the mounting system
            with a mount target that directly or indirectly contains the
            originating mountpoint.";

        argument "name";
    }

    extension target {
        description
            "This YANG extension is used to specify a remote target
             system from which to mount a datastore subtree.  This YANG
             extension takes one argument which specifies the remote
             system. In general, this argument will contain the name of
             a data node that contains the remote system information. It
             is recommended that the reference data node uses the
             mount-target grouping that is defined further below in this
             module.

             This YANG extension can occur only as a substatement below
             a mountpoint statement. It MUST NOT occur as a substatement
             below any other YANG statement.";

        argument "target-name";
    }

    extension subtree {
        description
            "This YANG extension is used to specify a subtree in a
            datastore that is to be mounted.  This YANG extension takes
            one argument which specifies the path to the root of the
            subtree. The root of the subtree SHOULD represent an
            instance of a YANG container.  However, it MAY represent
            also another data node.

            This YANG extension can occur only as a substatement below
            a mountpoint statement. It MUST NOT occur as a substatement
            below any other YANG statement.";

        argument "subtree-path";
    }

    typedef mount-status {
        description
```

```
            "This type is used to represent the status of a
            mountpoint.";
        type enumeration {
            enum ok; {
                description
                    "Mounted";
            }
            enum no-target {
                description
                    "The argument of the mountpoint does not define a
                     target system";
            }
            enum no-subtree {
                description
                    "The argument of the mountpoint does not define a
                      root of a subtree";
            }
            enum target-unreachable {
                description
                    "The specified target system is currently
                     unreachable";
            }
            enum mount-failure {
                description
                    "Any other mount failure";
            }
            enum unmounted {
                description
                    "The specified mountpoint has been unmounted as the
                    result of a management operation";
            }
        }
    }
    typedef subtree-ref {
        type string;  // string pattern to be defined
        description
            "This string specifies a path to a datanode. It corresponds
            to the path substatement of a leafref type statement.  Its
            syntax needs to conform to the corresponding subset of the
            XPath abbreviated syntax. Contrary to a leafref type,
            subtree-ref allows to refer to a node in a remote datastore.
            Also, a subtree-ref refers only to a single node, not a list
            of nodes.";
    }
    rpc mount {
        description
            "This RPC allows an application or administrative user to
            perform a mount operation.  If successful, it will result in
```

```
            the creation of a new mountpoint.";
        input {
            leaf mountpoint-id {
                type string {
                    length "1..32";
                }
            }
        }
        output {
            leaf mount-status {
                type mount-status;
            }
        }
    }
    rpc unmount {
        "This RPC allows an application or administrative user to
        unmount information from a remote datastore.  If successful,
        the corresponding mountpoint will be removed from the
        datastore.";
        input {
            leaf mountpoint-id {
                type string {
                    length "1..32";
                }
            }
        }
        output {
            leaf mount-status {
                type mount-status;
            }
        }
    }
    grouping mount-monitor {
        leaf mount-status {
            description
                "Indicates whether a mountpoint has been successfully
                mounted or whether some kind of fault condition is
                present.";
            type mount-status;
            config false;
        }
    }
    grouping mount-target {
        description
            "This grouping contains data nodes that can be used to
            identify a remote system from which to mount a datastore
            subtree.";
        container mount-target {
```

```
            choice target-address-type {
                mandatory;
                case IP {
                    leaf target-ip {
                        type yang:ip-address;
                    }
                case URI {
                    leaf uri {
                        type yang:uri;
                    }
                }
                case host-name {
                    leaf hostname {
                        type yang:host;
                    }
                }
                case node-ID {
                    leaf node-info-ref {
                        type subtree-ref;
                    }
                }
                case other {
                    leaf opaque-target-ID {
                        type string;
                        description
                            "Catch-all; could be used also for mounting
                            of data nodes that are local.";
                    }
                }
            }
        }
    }
    grouping mount-policies {
        description
            "This grouping contains data nodes that allow to configure
            policies associated with mountpoints.";
        leaf manual-mount {
            type empty;
            description
                "When present, a specified mountpoint is not
                automatically mounted when the mount data node is
                created, but needs to mounted via specific RPC
                invocation.";
        }
        leaf retry-timer {
            type uint16;
            units "seconds";
            description
```

```
                "When specified, provides the period after which
                mounting will be automatically reattempted in case of a
                mount status of an unreachable target";
            }
            leaf number-of-retries {
                type uint8;
                description
                    "When specified, provides a limit for the number of
                    times for which retries will be automatically
                    attempted";
            }
        }

        container mount-server-mgmt {
            if-feature mount-server-mgmt;
            container mountpoints {
                list mountpoint {
                    key "mountpoint-id";

                    leaf mountpoint-id {
                        type string {
                            length "1..32";
                        }
                    }
                    leaf mountpoint-origin {
                        type enumeration {
                            enum client {
                                description
                                    "Mountpoint has been supplied and is
                                    manually administered by a client";
                            }
                            enum auto {
                                description
                                    "Mountpoint is automatically
                                     administered by the server";
                            }
                        config false;
                        }
                    }
                    uses mount-target;
                    leaf subtree-ref {
                        type subtree-ref;
                        mandatory;
                    }
                    uses mount-monitor;
                    uses mount-policies;
                }
            }
```

```
            container global-mount-policies {
                uses mount-policies;
                description
                    "Provides mount policies applicable for all mountpoints,
                    unless overridden for a specific mountpoint.";
            }
        }
    }
    <CODE ENDS>
```

7. Security Considerations

TBD

8. Acknowledgements

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from Tony Tkacik, Robert Varga, Lukas Sedlak, and Benoit Claise.

9. References

9.1. Normative References

[RFC2131] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, March 1997.

[RFC2866] Rigney, C., "RADIUS Accounting", RFC 2866, June 2000.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

[RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, March 2012.

9.2. Informative References

[I-D.bierman-netconf-restconf]
Bierman, A., Bjorklund, M., Watsen, K., and R. Fernando, "RESTCONF Protocol", draft-bierman-netconf-restconf-01 (work in progress), September 2013.

Appendix A. Example

In the following example, we are assuming the use case of a network controller that wants to provide a controller network view to its client applications. This view needs to include network abstractions that are maintained by the controller itself, as well as certain information about network devices where the network abstractions tie in with element-specific information. For this purpose, the network controller leverages the mount capability specified in this document and presents a fictitious Controller Network YANG Module that is depicted in the outlined structure below. The example illustrates how mounted information is leveraged by the mounting datastore to provide an additional level of information that ties together network and device abstractions, which could not be provided otherwise without introducing a (redundant) model to replicate those device abstractions

```
rw controller-network
+-- rw topologies
|   +-- rw topology [topo-id]
|       +-- rw topo-id                node-id
|       +-- rw nodes
|       |   +-- rw node [node-id]
|       |       +-- rw node-id        node-id
|       |       +-- rw supporting-ne  network-element-ref
|       |       +-- rw termination-points
|       |           +-- rw term-point [tp-id]
|       |               +-- tp-id     tp-id
|       |               +-- ifref     mountedIfRef
|       +-- rw links
|           +-- rw link [link-id]
|               +-- rw link-id        link-id
|               +-- rw source         tp-ref
|               +-- rw dest           tp-ref
+-- rw network-elements
    +-- rw network-element [element-id]
        +-- rw element-id             element-id
        +-- rw element-address
        |   +-- ...
        +-- M interfaces
```

The controller network model consists of the following key components:

- A container with a list of topologies. A topology is a graph representation of a network at a particular layer, for example, an IS-IS topology, an overlay topology, or an Openflow topology. Specific topology types can be defined in their own separate YANG modules that augment the controller network model. Those augmentations are outside the scope of this example

- An inventory of network elements, along with certain information that is mounted from each element. The information that is mounted in this case concerns interface configuration information. For this purpose, each list element that represents a network element contains a corresponding mountpoint. The mountpoint uses as its target the network element address information provided in the same list element

- Each topology in turn contains a container with a list of nodes. A node is a network abstraction of a network device in the topology. A node is hosted on a network element, as indicated by a network-element leafref. This way, the "logical" and "physical" aspects of a node in the network are cleanly separated.

- A node also contains a list of termination points that terminate links. A termination point is implemented on an interface. Therefore, it contains a leafref that references the corresponding interface configuration which is part of the mounted information of a network element. Again, the distinction between termination points and interfaces provides a clean separation between logical concepts at the network topology level and device-specific concepts that are instantiated at the level of a network element. Because the interface information is mounted from a different datastore and therefore occurs at a different level of the containment hierarchy than it would if it were not mounted, it is not possible to use the interface-ref type that is defined in YANG data model for interface management [] to allow the termination point refer to its supporting interface. For this reason, a new type definition "mountedIfRef" is introduced that allows to refer to interface information that is mounted and hence has a different path.

- Finally, a topology also contains a container with a list of links. A link is a network abstraction that connects nodes via node termination points. In the example, directional point-to-point links are depicted in which one node termination point serves as source, another as destination.

The following is a YANG snippet of the module definition which makes use of the mountpoint definition.

```
<CODE BEGINS>
module controller-network {
    namespace "urn:cisco:params:xml:ns:yang:controller-network";
    // example only, replace with IANA namespace when assigned
    prefix cn;
    import mount {
        prefix mnt;
    }
    import interfaces {
        prefix if;
    }
    ...
    typedef mountedIfRef {
        type leafref {
            path "/cn:controller-network/cn:network-elements/"
            +"cn:network-element/cn:interfaces/if:interface/if:name";
            //  cn:interfaces corresponds to the mountpoint
        }
    }
    ...
    list termination-point {
        key "tp-id";
        ...
        leaf ifref {
            type mountedIfRef;
        }
        ...
        list network-element {
            key "element-id";
            leaf element-id {
                type element-ID;
            }
            container element-address {
                ... // choice definition that allows to specify
                // host name,
                // IP addresses, URIs, etc
            }
            mnt:mountpoint "interfaces" {
                mnt:target "./element-address";
                mnt:subtree "/if:interfaces";
            }
        ...
    }
...
<CODE ENDS>
```

Finally, the following contains an XML snippet of instantiated YANG information. We assume three datastores: NE1 and NE2 each have a datastore (the mount targets) that contains interface configuration data, which is mounted into NC's datastore (the mount client).

Interface information from NE1 datastore:

```
<interfaces>
  <interface>
    <name>fastethernet-1/0</name>
    <name>ethernetCsmacd</type>
    <location>1/0</location>
  </interface>
  <interface>
    <name>fastethernet-1/1</name>
    <name>ethernetCsmacd</type>
    <location>1/1</location>
  </interface>
<interfaces>
```

Interface information from NE2 datastore:

```
<interfaces>
  <interface>
    <name>fastethernet-1/0</name>
    <name>ethernetCsmacd</type>
    <location>1/0</location>
  </interface>
  <interface>
    <name>fastethernet-1/2</name>
    <name>ethernetCsmacd</type>
    <location>1/2</location>
  </interface>
<interfaces>
```

NC datastore with mounted interface information from NE1 and NE2:

```
<controller-network>
  ...
  <network-elements>
    <network-element>
      <element-id>NE1</element-id>
      <element-address> .... </element-address>
      <interfaces>
        <if:interface>
          <if:name>fastethernet-1/0</if:name>
          <if:type>ethernetCsmacd</if:type>
          <if:location>1/0</if:location>
```

```
          </if:interface>
          <if:interface>
            <if:name>fastethernet-1/1</if:name>
            <if:type>ethernetCsmacd</if:type>
            <if:location>1/1</if:location>
          </if:interface>
        <interfaces>
      </network-element>
      <network-element>
        <element-id>NE2</element-id>
        <element-address> .... </element-address>
        <interfaces>
          <if:interface>
            <if:name>fastethernet-1/0</if:name>
            <if:type>ethernetCsmacd</if:type>
            <if:location>1/0</if:location>
          </if:interface>
          <if:interface>
            <if:name>fastethernet-1/2</if:name>
            <if:type>ethernetCsmacd</if:type>
            <if:location>1/2</if:location>
          </if:interface>
        <interfaces>
      </network-element>
    </network-elements>
    ...
  </controller-network>
```

Authors' Addresses

Alexander Clemm
Cisco Systems

EMail: alex@cisco.com

Jan Medved
Cisco Systems

EMail: jmedved@cisco.com

Eric Voit
Cisco Systems

EMail: evoit@cisco.com