Peer Mount

Alexander Clemm (Futurewei, ludwig@clemm.org)
Eric Voit (Cisco, evoit@cisco.com)

13 September 2023

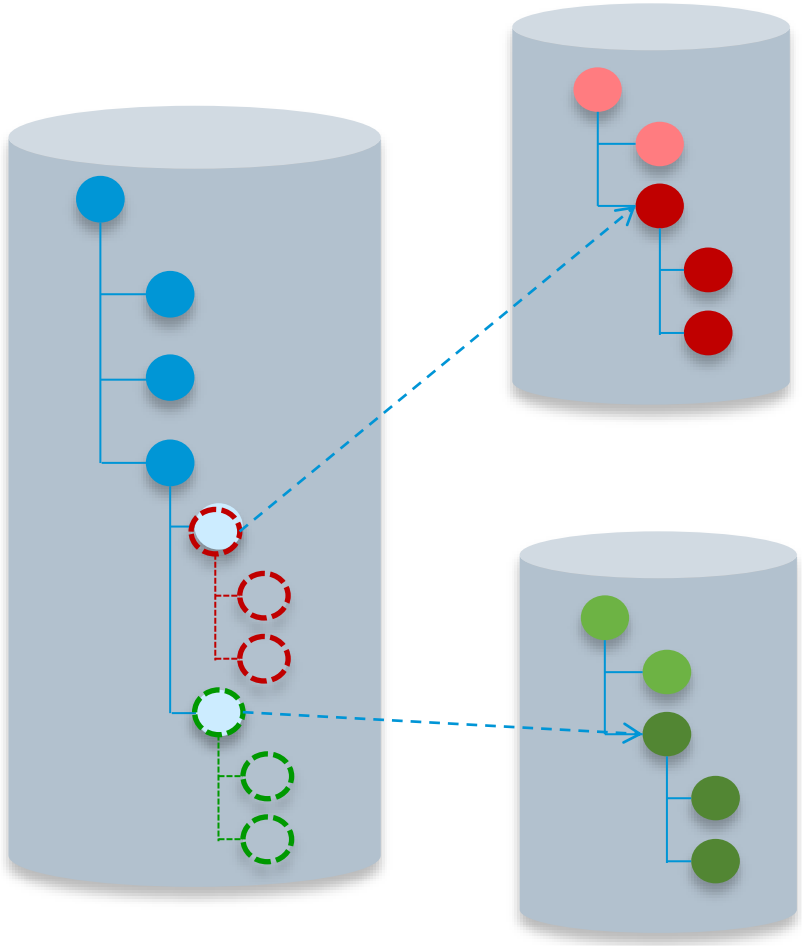
Motivation

- YANG Datastores today provide a view of management data that is maintained and implemented locally – device-level scope
- Increasingly, use cases appear that require more holistic, network-wide views
  - Examples: Topology, Digital Map, Network Inventory, Network Digital Twin
  - Required data may become increasingly redundant (e.g. status, aspects of configuration)
  - Provided as part of a management hierarchy (e.g. device – controller – orchestrator)
- Issues
  - Need for redundant model definitions for device and for network context (and redundant augmentations etc)
  - Separate implementation and instrumentation at device and controller level
  - Risk of model misalignments (e.g. deviations, different speeds at which models become available, …)
  - Synchronization of redundant data
  - In case of data that is not redundantly captured: need for multiple management associations (& potential layer violations) in management hierarchies, mgmt. communication scaling issues
- Needed: an ability to provide a datastore with a holistic network view that avoids these issues

Peer-Mount Concept

- Allow YANG Datastores to reference information in remote datastores
  - Insert (remote) subtrees under a mount point in a (local) datastore
  - Mount client: a YANG server that maintains the mounted “view”
  - Mount server: the original “authoritative” owner of the data
  - For on-demand object access, mount server does not need to be aware of mount client
- Use to provide federated datastore that provides a holistic view of a network
  - Network inventory can provide additional system and configuration information
  - Network topology can provide “live” view of nodes, termination points, links: status, statistics, etc
  - No need for redundant data models to model aspects both at system and at topology/inventory level
  - Avoidance of data synchronization and reconciliation issues
- Analogies with mountpoints in a distributed file system (YANG data nodes vs files/directories)

Mount Concept – Peer Mount



Concept:

- Refer to data nodes / subtrees in remote datastores
- Remote data nodes visible as part of local data store
- Avoid need for data replication and orchestration (caching considerations apply)
- Authority remains with original owner

Why:

- Federated datastore - treat network as a system
- “Borderless Agents”, “Network-as-a-System”
- “Live” network topology, network inventory, digital map

Note: do not confuse with schema mount (RFC 8528)

- *Mount instances of datastore subtrees in remote servers vs. extensions of model to be instantiated locally*

Usage example

```
rw controller-network
    +-- rw network-elements
        +-- rw network-element [element-id]
            +-- rw element-id
            +-- rw element-address
            |   +-- ...
            +-- M interfaces
```

Module structure

```
...
 list network-element {
     key "element-id";
     leaf element-id {
          type element-ID;
     }
     container element-address {
          ...
     }
     pmt:mountpoint "interfaces" {
         pmt:target "./element-address";
         pmt:subtree "/if:interfaces";
     }
  }
...
```

Mountpoint declaration

- YANG module defines YANG mount extensions + data model for mountpoint management
- YANG extensions:

  Mountpoint: Defined under a containing data node (e.g. container, list)

  Target: References data node that identifies remote server

  Subtree: Defines root of remote subtree to be attached

```
<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>
        ...
```

Instance information

In the context of network inventory

```
module: my-new-network-inventory
     +--rw nw:networks
        +--rw nw:network* [nw:network-id]
           ...
           +--rw nw:node* [node-id]
              +--rw nw:node-id              node-id
              +--rw name
              +--M node-hardware -->/hardware/component[name]
              ...
```

from ietf-network-topology per RFC 8345

Note: need to associate target system name with address (may need to add data node)

augmentation
(here: mounted hw component data from ietf-hardware per RFC 8348)

In the context of network inventory

```
module: my-new-network-inventory
    +--rw nw:networks
       +--rw nw:network* [nw:network-id]
          ...
          +--rw nw:node* [node-id]
             +--rw nw:node-id              node-id
             +--(hw-data-origin)
                +--:(rfc8348)
                |  +--rw name
                |  +--M node-hardware -->/hardware/component[name]
                +--:(controller-populated)
                   +--ro component* [uuid]
                   +--ro uuid  yang:uuid
                   +--ro location
                   ...
```

Example only; for an actual network inventory can be integrated with other models
e.g. draft-ietf-ccamp-network-inventory-yang or draft-wzwb-opsawg-network-inventory-management

draft-ietf-ccamp-network-inventory-yang + RFC 8348

```
module: ietf-network-hardware-inventory
     +--ro network-hardware-inventory
        +--ro equipment-rooms
        |  ...................................
        +--ro network-elements
           +--ro network-element* [uuid]
              +--ro uuid                yang:uuid
              +--ro name?               string
              +--ro description?        string
              ...................................
              +--M components --> [uuid:]/hardware/
                 +--ro last-change?   yang:date-and-time
                 +--rw component* [name]
                    +--rw name                string
                    +--rw class               identityref
                    +--ro hardware-rev?       string
                    +--ro firmware-rev?       string
                    +--ro software-rev?       string
                    +--ro serial-num?         string
                    +--ro mfg-name?           string
                    +--ro model-name?         string
                    +--rw alias?              string
                    +--rw asset-id?           string
                    ...
```

Note: need to associate target system UUID with address (may need to add data node)

mounted subtree from RFC8348-compliant NE

draft-wzwb-opsawg-network-inventory-management + RFC 8348

```
module: ietf-network-inventory
  augment /nw:networks/nw:network/nw:node:
    +--rw name?                          string
    +--ro node-type?                     identityref
    +--ro is-virtual?                    boolean
    +--ro is-gateway?                    boolean
    +--ro gateway-ref?                   -> ../name
    +--rw management-ipv6-address?       inet:host
    ...........
    +--ro hardware-rev?                  string
    +--ro firmware-rev?                  string
    +--ro software-rev?                  string
    +--ro serial-num?                    string
    +--ro mfg-name?                      string
    +--ro asset-id?                      string
    +--ro mfg-date?                      yang:date-and-time
    .......
    +--M components --> [name|management-ipv4/v6-address:]/hardware/
       +--rw component* [name]
          +--rw name                     string
          +--rw class                    identityref
          +--ro hardware-rev?            string
          +--ro firmware-rev?            string
          ...
```

Notes:

1) *top level may be obtained from RFC8348 (may need to add data node)*
2) *resolve mgmt. address alternatives (e.g. choice/union)*

mounted subtree from RFC8348-compliant NE

Integration with network inventory effort

- Can be used in conjunction with network inventory models currently being defined
  - draft-ietf-ccamp-network-inventory-yang; draft-wzwb-opsawg-network-inventory-management; etc
- Inventory models can be defined to allow for support of RFC8348-enabled and legacy devices
  - Use “choice” to distinguish cases where required data can be mounted from remote (RFC 8348 supported) or requires manual population
- Decoupling network inventory effort from peer-mount definition effort is possible
  - Allow for the possibility of future support of mounting in the inventory model
  - Use if-feature to create placeholders where mountpoints can be injected once supported

Datastore mountpoint YANG module

- Extensions:
  - mountpoint
  - target
  - subtree
- RPCs:
  - mount
  - unmount
- Mountpoint management:
  - mount status
  - caching policies
  - communication / retry policies

- Declares a mountpoint under a containing data node (container, list, case)
- Two parameters: target and subtree (separate extension)
- Circular mounts prohibited – check on instantiation

- Identifies the target system that is authoritative owner of the data (e.g. IP address, host name, URI)
- Generally, maintained as part of the same datastore (“inventory”)

- Identifies the subtree in the target system that is being mounted
- Generally, a container (but could be another data node)

- Only needed for explicit / on-demand instantiation of mountpoints (vs by system operation)
- Might remove

Mountpoint management

```
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  pmt:subtree-ref
  |       |   +-- (other)
  |       |        +-- rw opaque-target-id  string
  |       +-- rw subtree-ref  pmt:subtree-ref
  |       +-- ro mountpoint-origin enumeration
  |       +-- ro mount-status  pmt: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

+ RPCs for manual mount, unmount
```

- Mountpoints can be system-administered
  - Applications & users will not be exposed to this
  - Manage caching policies, maintain mount status
- Instantiation of mountpoints
  - Via system operation (automatic instantiation)
  - Via mount / unmount RPC (explicit instantiation)
- Either case, where mountpoints can be instantiated must be declared as part of the model
  - Cannot mount in arbitrary locations
  - Retain ability to validate instance documents

Other considerations

- Authorization
  - Target system is the authoritative owner, NACM applies – mount client “just another application”
- Mount cascades supported (but circular mounting is prohibited)
- Focus on read operations and data retrieval, out of scope:
  - Configuration support (would incur transactional ramifications)
  - Notifications (cascading subscriptions conceivable but may lead to event replication)
  - YANG-Push (support for cascading subscriptions is conceivable when need arises)
- Caching
  - Conceivable as an implementation optimization – cache datanodes when #reads>>#updates
  - Implementations could leverage YANG-Push – subscribe to updates from mounted subtree in mount server (distinguish from YANG-Push subscription to the YANG client)
- Mount & connection granularity
  - Can mount multiple (small) subtrees from the same target system
  - Implementations should be smart enough to maintain only a single management association
- Datastore qualification and NMDA TBD

Comparison Peer-Mount – Schema Mount

Peer-Mount	Schema Mount
Provide visibility - create access path to existing instances hosted in a remote server	Reuse existing definitions to create new models that are then locally instantiated and locally hosted
Analogy: soft link* (*with some caveats)	Analogy: grouping/uses (or augments) “after the fact”
Reference mount target has authoritative copy	Mount Point has authoritative copy
No validation of data at or by mountpoint; validation of data is responsibility of authoritative data owner	Validation of data at mount point
Mount point provides visibility to data already instantiated elsewhere (no redundant data)	Mountpoint instantiates new data
The same target mounted in different mountpoints does not result in additional data instances	Same target schema mounted in different mountpoints results in separate unrelated data instances

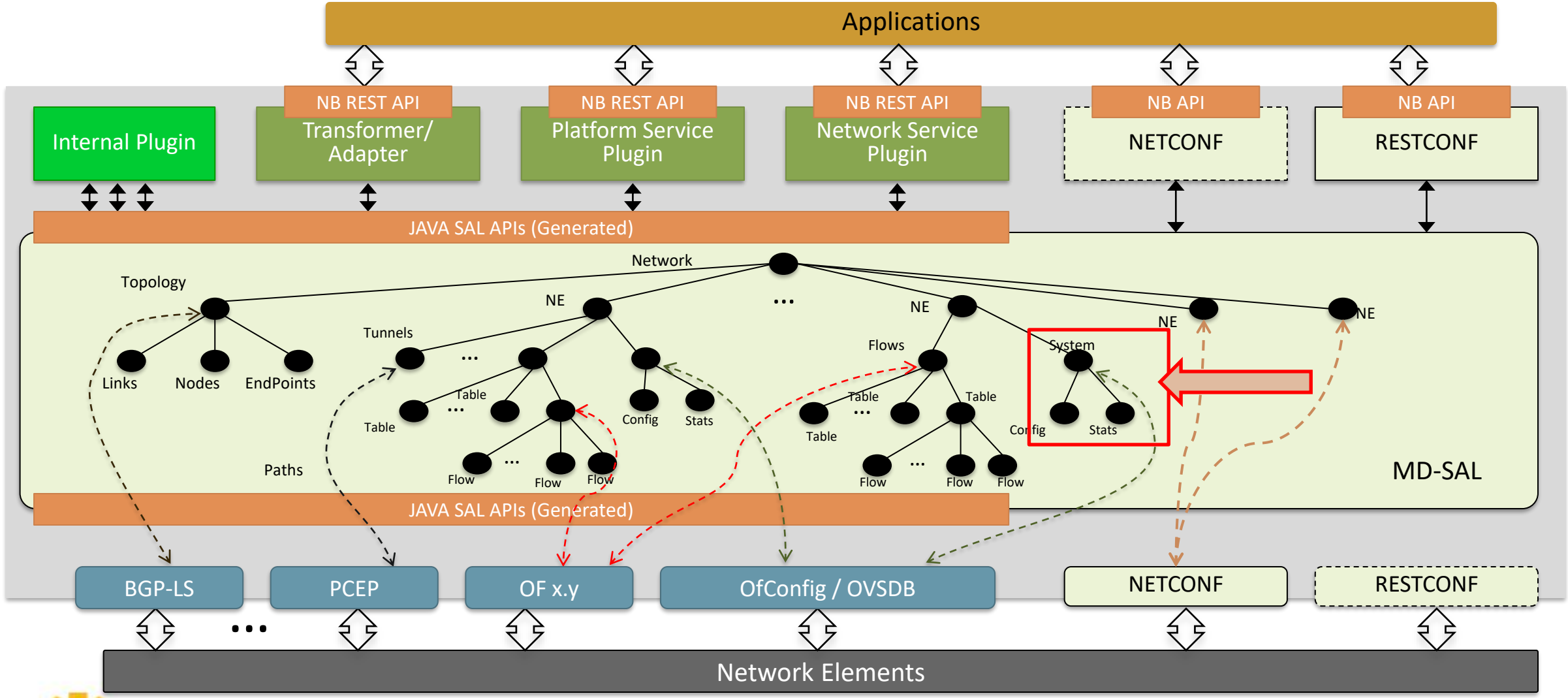
Commonality between Peer-Mount and Schema-Mount: YANG mountpoint extension

- YANG extension introduced to define mountpoints
- Differences in terms of additional parameters (to identify target node and target system)

Miscellaneous

- Past: History
  - An earlier proposal for Peer-Mount was made in 2013 but arguably ahead of its time
  - Included 2 mount variants: alias mount for alternative data tree in addition to peer mount
  - Implementation as part of Open Daylight's MD-SAL (SDN Controller)
  - No IETF interest in data models above device level at the time, so did not gain traction
- Future: Next steps
  - Revive earlier proposal with view of new context, requirements, use cases, refined scope (eg, no alias)
  - -00 version is planned, coauthored with Eric Voit (Cisco), outreach to operators & other people interested in contributing
  - Outline how network inventory/topology/digital map drafts could leverage this

Open Daylight - Model-Driven SAL



OPENDAYLIGHT

www.opendaylight.org