IETF118, Prague, Czech Republic
November 2024

Mounting YANG-Defined Information from Remote Datastores

draft-clemm-netmod-peermount-02

Alexander Clemm (Futurewei)

Eric Voit (Cisco)

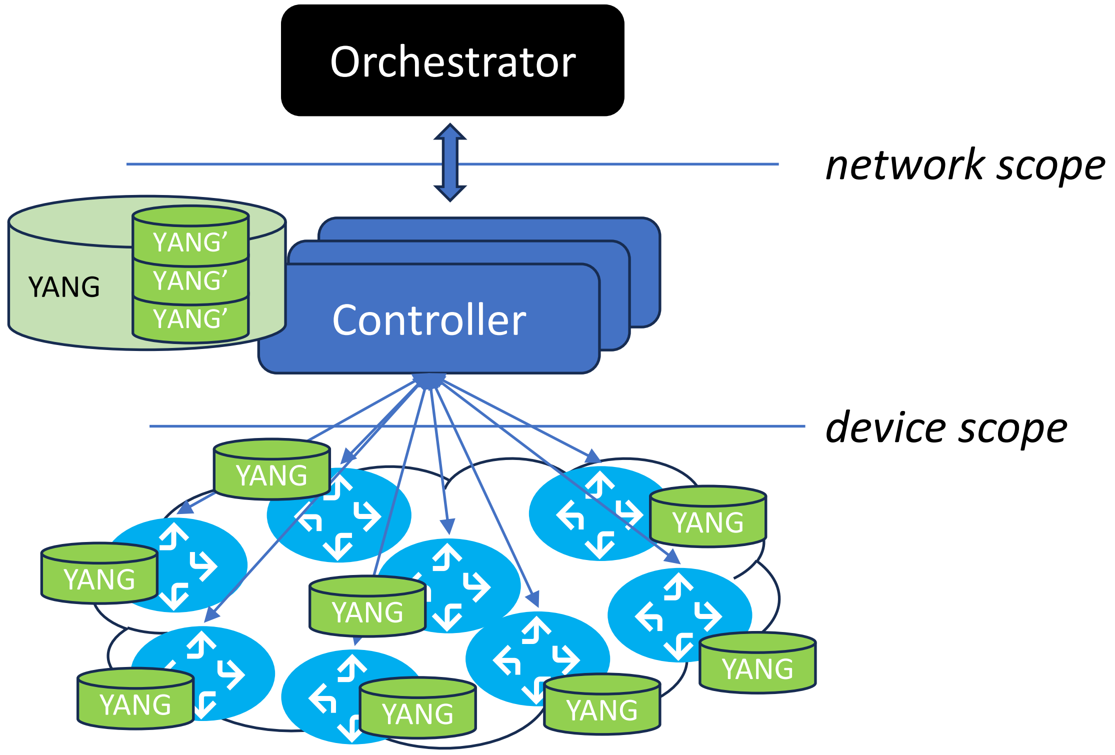
Aihua Guo (Futurewei)

Ignacio Dominguez (Telefonica)

draft-clemm-netmod-peermount@ietf.org

Motivation

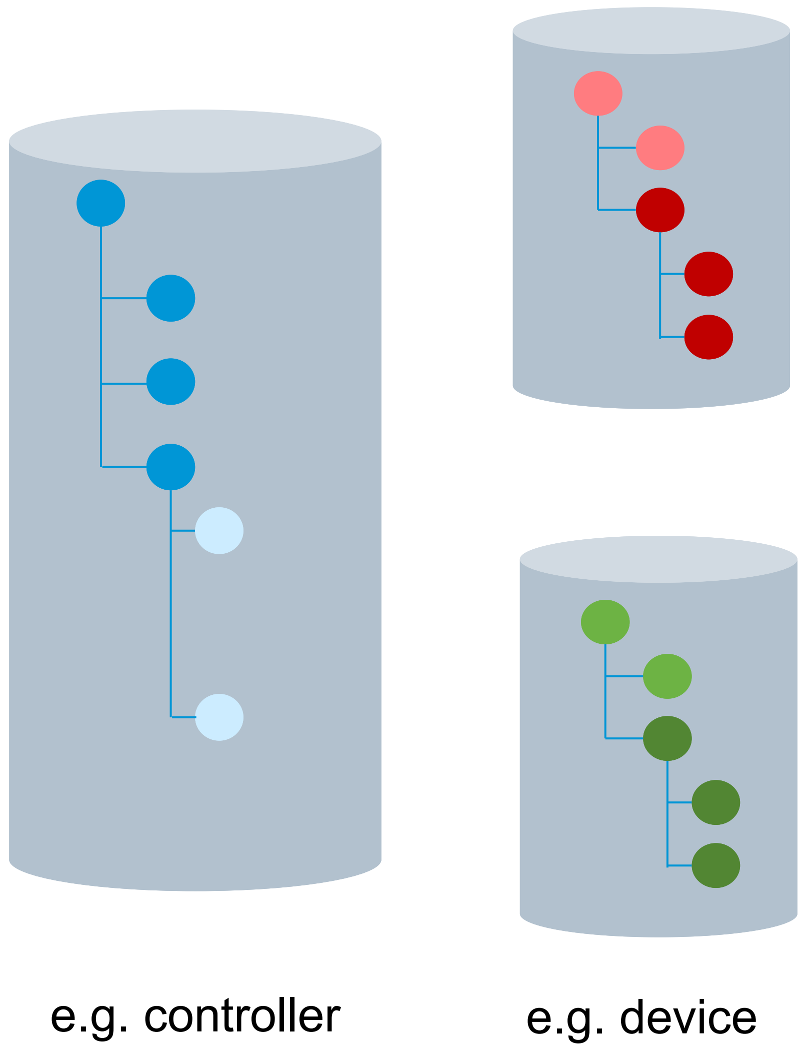
- YANG datastores provide local management data with a device-level scope
- Increasingly, use cases appear that require more holistic, network-wide views
    - Topology, Digital Map, Network Inventory, Network Digital Twin
    - Required data may become increasingly redundant (e.g. status, aspects of configuration)
    - Provided as part of a management hierarchy (e.g. device – controller – orchestrator)



Motivation (contd.)

- Issues
  - Need redundant model definitions for device and for network context
    - Risk of model misalignments at controller vs at network element
      (e.g. deviations, different speeds at which models become available, …)
    - Need for redundant augmentations
  - Separate implementation and instrumentation at device and controller level
  - Synchronization of redundant data
  - Operational inefficiencies (e.g. redundant manual population)
  - 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: a federated datastore that provides a holistic view of a network
  - Federated data resides across nodes that authoritatively "own" their data
  - Accessed by clients as one conceptual datastore
  - Use to provide additional configuration & status information of nodes in network context
    eg. for Topology, Inventory, Digital Twin

Mount Concept – Peer Mount



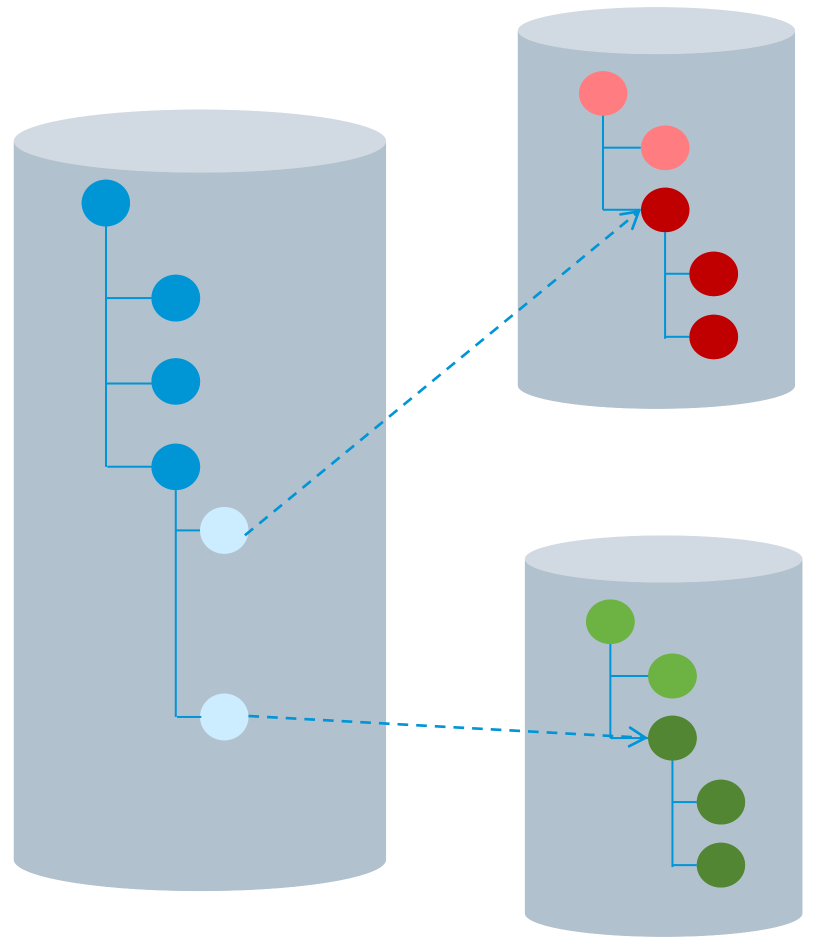
Concept:

- 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
- Analogies with mountpoints in a distributed file system (YANG data nodes vs files/directories)

Why:

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

Mount Concept – Peer Mount



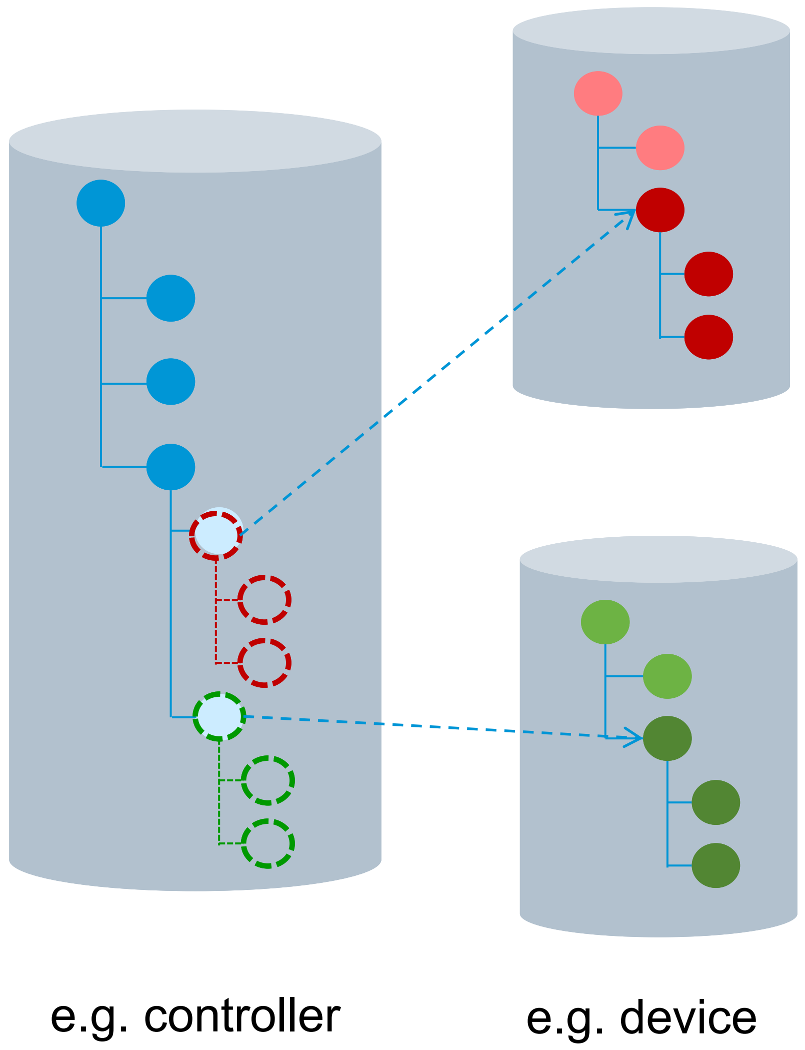
Concept:

- 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
- Analogies with mountpoints in a distributed file system (YANG data nodes vs files/directories)

Why:

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

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
- Analogies with mountpoints in a distributed file system (YANG data nodes vs files/directories)

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 peer 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-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> ...
  </network-element>
  <network-element>
    <element-id>NE2</element-id>  ...
    <interfaces>
      <if:interface> ...
```

Instance information

Example uses

- Provide network-wide view of device configuration aspects (in an inventory, in a Digital Twin, ...)

  For example, system management settings, data on hardware/firmware, location information, ...

- Provide network-wide status information (in a topology, in a digital map, ...)

  For example, power statistics, link status, interface statistics

- Design pattern:

  Define Mount Point for additional information in network element list elements

  Mount subtrees with the desired information

```
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

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

Dealing with heterogeneity & legacy

- Not every remote device may support / provide the information that is to be mounted
- In those situations, a controller may still need to populate the information manually (or mount alternative data)
- This can be addressed through design patterns that accommodate different options / choices

```
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)
                +--:(data to be mounted supported by remote system)
                |  +--rw name
                |  +--M node-hardware -->/hardware/component[name]
                +--:(controller-populated)
                   +--ro component* [uuid]
                   +--ro uuid  yang:uuid
                   +--ro location
                   ...
```

Datastore mountpoint YANG module

- YANG Extensions:
  - mountpoint
  - target
  - subtree

- 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)

Datastore mountpoint YANG module

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

- Possibly include mount status as metadata on data retrieval

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

Additional considerations

- Mount cascades supported (but circular mounting is prohibited)
- Supported operations: data retrieval only (at this point), other operations 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)
- Authorization
    - Target system is the authoritative owner, NACM applies – mount client “just another application”
- 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

Final remarks

- A historical remark
  - 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
- Next steps
  - Time may be ripe now as network-wide models in IETF scope (e.g. network inventory, Digital Twin)
  - This draft revives the earlier proposal with modification and simplifications in view of new context

Is there interest in taking up this this work?

Questions, comments, suggestions? Please reach out to us

Thank you!

Backup

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

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)