Peer Mount

Eric Voit
Alex Clemm

13-Nov-2014

Four Drafts

Requirements for Peer Mounting of YANG subtrees from Remote Datastores
draft-voit-netmod-peer-mount-requirements-01
(Requirements & Use Cases showing how/why Peer Mount.)

Mounting YANG-Defined Information from Remote Datastores
draft-clemm-netmod-mount-02
(Technology draft. Multiple implementations including OpenDaylight MD-SAL)

Cloud SLA YANG Model incorporating Peer Mount Semantics
draft-tripathy-cloud-sla-yang-model-00
(Example YANG model using Peer Mount syntax. Will show running Demo)

Tomorrow

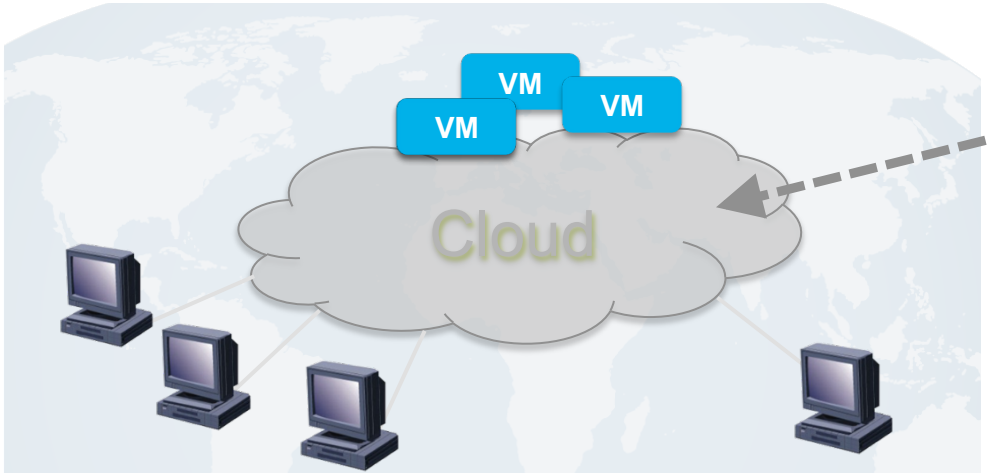
Subscribing to datastore push updates
draft-netmod-clemm-datastore-push-00
(Pub/Sub mechanism for YANG objects, applicable well beyond Peer Mount)

Requirements Draft

Requirements for Peer Mounting of YANG subtrees from Remote Datastores
draft-voit-netmod-peer-mount-requirements-01
E. Voit, A. Clemm, S. Mertens

- Example Use Case
- “Peer Mount” Basics
- Contents of Requirements Draft

Network Services delivered via a federated set of devices in a Cloud

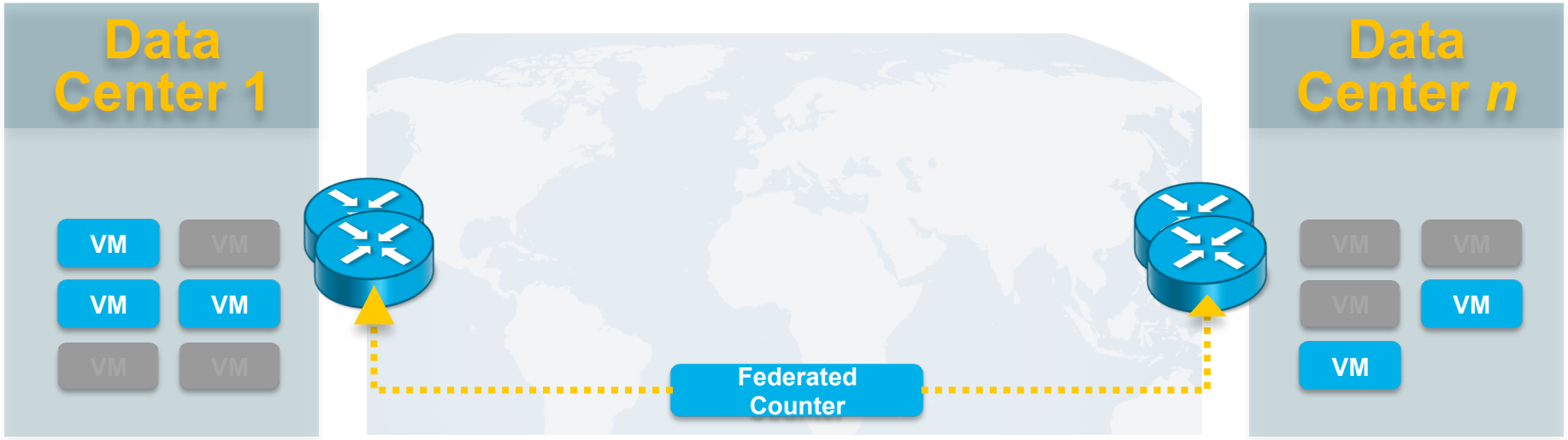


Network acts as a unit for targeted abstractions

Location agnostic many-to-many policing

SLA protections between tenants

Cloud SLA spanning a set of Network Elements

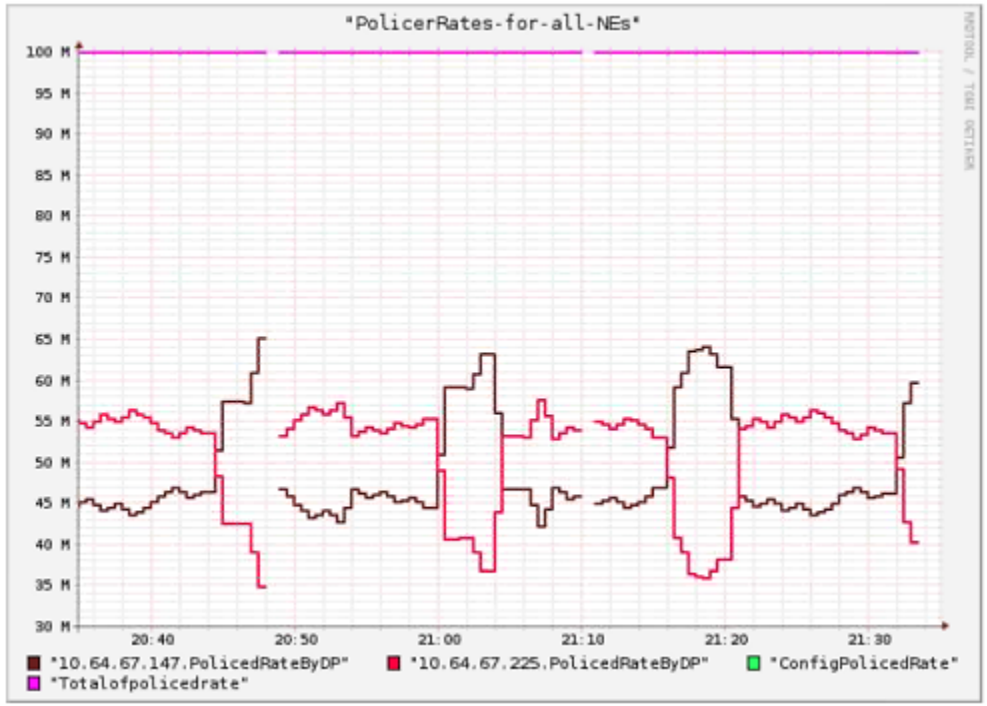


- Dynamically adjust policers as traffic moves about the cloud
- Forward to DDoS Appliance if Bandwidth Threshold hit

Cloud Policer + DDoS Thresholding

Currently running on OpenDaylight

From 2 rate 3 color policer by Device, to “n” rate “m” actions for a Cloud



Device policers dynamically updated against Cloud SLA of 100Mb/s

DDoS Scrubber inserted when Traffic Spikes over 600 Mb/s Threshold

What is needed from a Solution

Laser Focus on Application Developer Simplicity

- App developers want distributed database and convergence complexities hidden. And they certainly don't want to learn and invoke protocols to get remote objects.

Single Authoritative owner for an Object

- Applications don't want to care which Network Element across a set of devices is the authoritative owner of a particular replicated object.
- Multiple controllers and overlapping domains is a reality. Apps still don't care.

Transparent Subscriptions and Caching

- Applications need timely access to remote information, which can necessitate the introduction cache infrastructure.

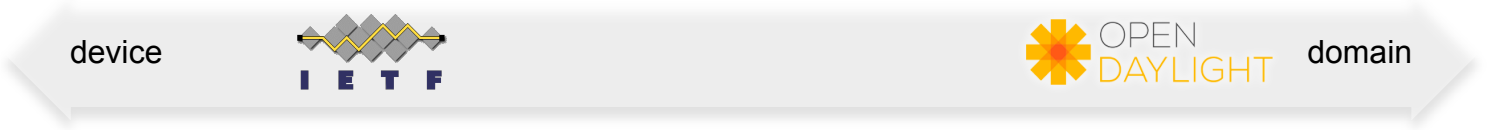
Network Abstractions already span Devices

IETF has specified many Controllers. Industry is adding more.

There are classes of applications which expose multi-device abstractions across overlapping sets of devices.

Example controllers

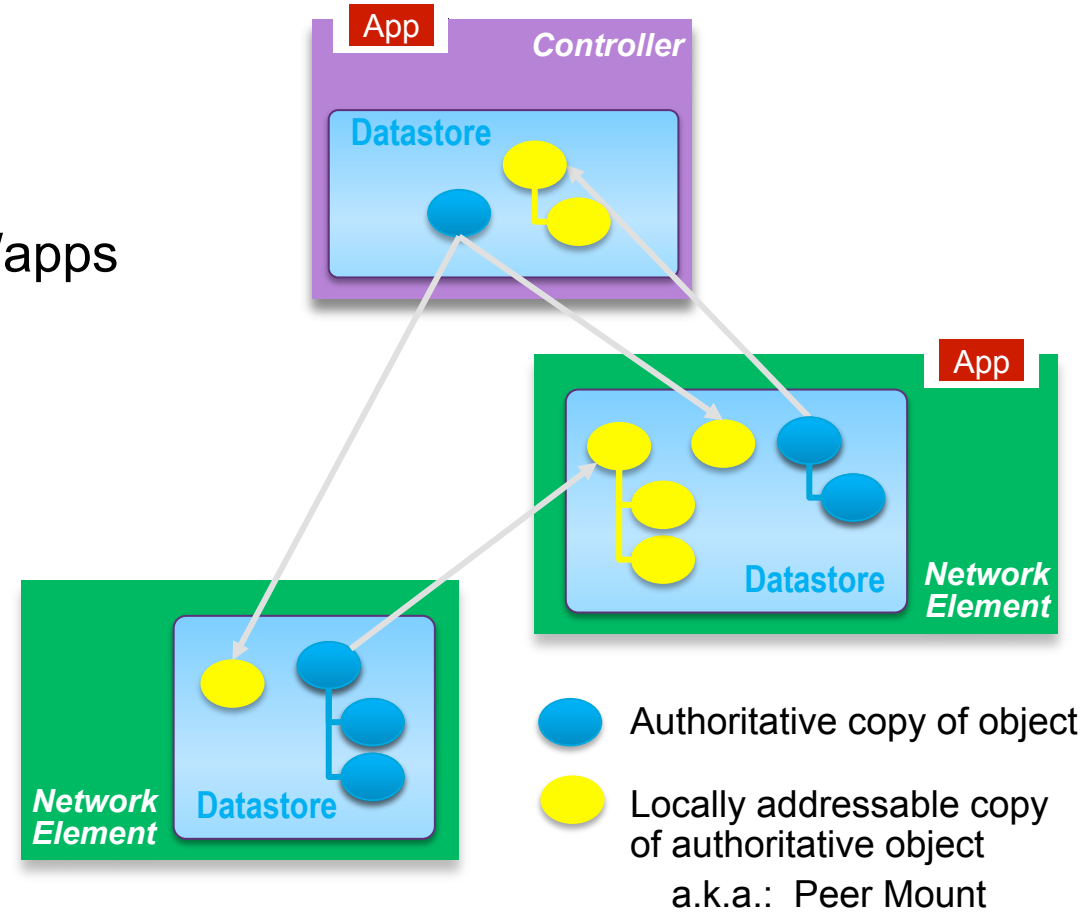
High Availability	Topology Maintenance	Cloud Functionality
• VRRP	• OSPF Designated Router	• Cloud Policer
• mLACP/ICCP	• BGP Route Reflection	• Traffic Redirection
• Anycast-RP	• BGP LS	



Layered abstractions build on each other, they must interoperate and converge

Peer Mounting the Authoritative Copy
Network Wide Data, Locally Addressable

- Excerpt of authoritative network-wide datastore assembled on each device
- Local & remote objects treated identically by users/apps
- Device type agnostic: peering of Controller & Network Element Intelligence

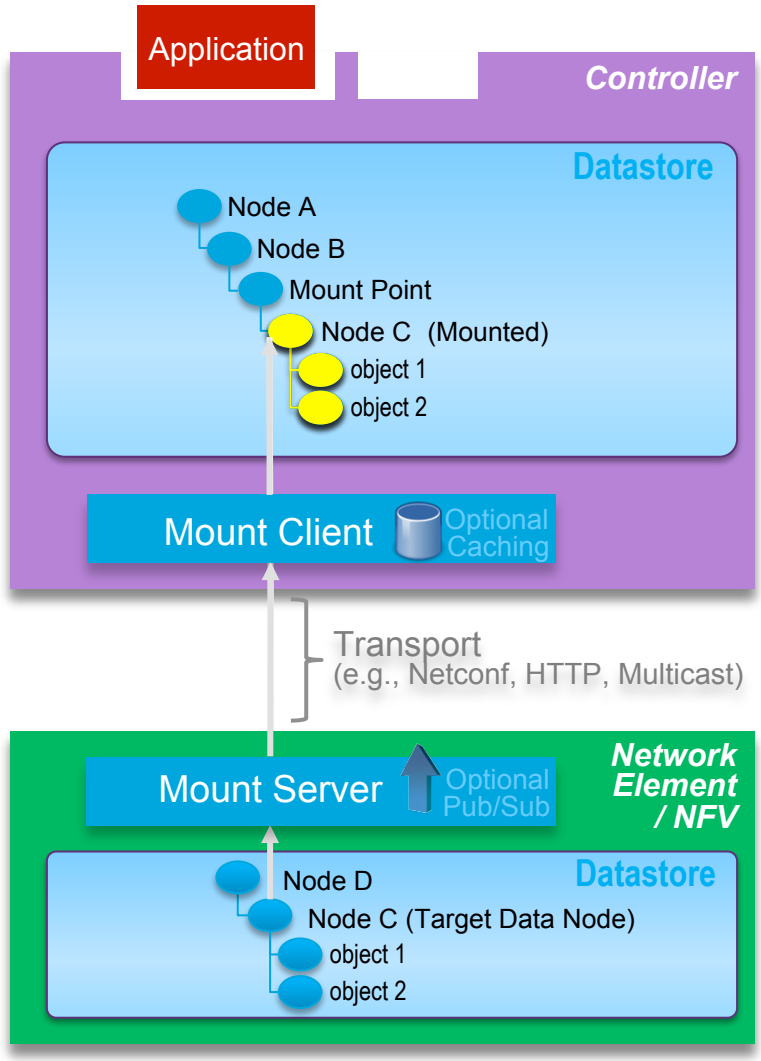


Peer Mounting the Authoritative Copy

"Mount" is a Remote Object Binding

- Application requests object from local Datastore
- Target Data Node Mount request is sent to object owner
  - *On Demand*: Subtree containing requested object is returned; object passed back to application
- Optional: Transparent caching mechanisms + Pub/Sub speeds performance as needed
  - *Periodic*: Object update passed every 'X' seconds. Timeframes can be synchronized across devices
  - *On Change*: Push Object when there us change on Mount Server

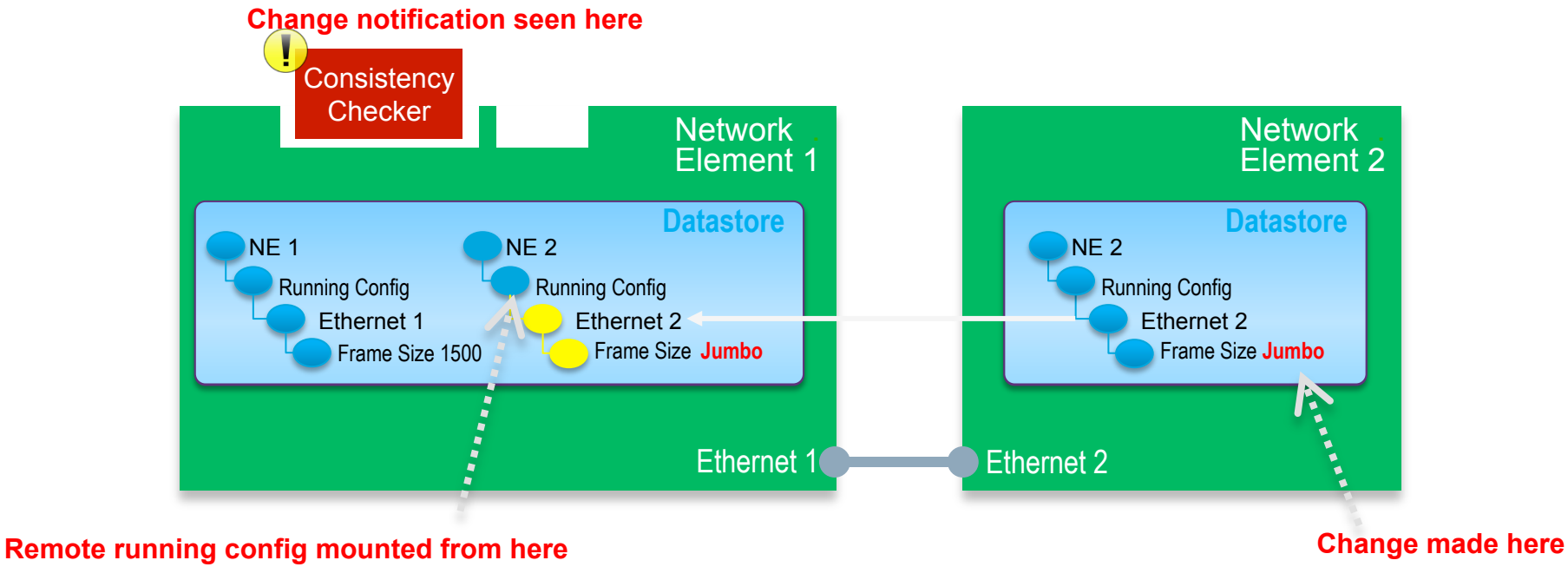
Underlying technologies have broad applicability and can be lightly coupled with Peer Mount



Peer Mounting the Authoritative Copy

Transparent remote visibility also viable for device based Apps

- Auto-discovery of link, group, or area misconfigurations. Controller optional.
- Applications have visibility into one or more exposed local YANG abstractions



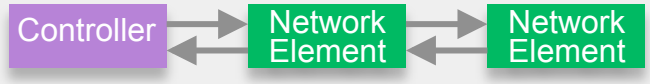
Requirements Draft

Blue – decent shape
Yellow – Work in progress

1. Business Problem
2. Terminology
3. Solution Context
   - 3.1. Peer Mount
   - 3.2. Eventual Consistency and YANG 1.1
4. Example Use Cases
   - 4.1. Cloud Policer
   - 4.2. DDoS Thresholding
   - 4.3. Service Chain Classification, Load Balancing and Capacity Management
5. Requirements
   - 5.1. Application Simplification
   - 5.2. Caching Considerations
     - 5.2.1. Caching Overview
     - 5.2.2. Pub/Sub of Object Updates
   - 5.3. Lifecycle of the Mount Topology
     - 5.3.1. Discovery and Creation of Mount Topology
     - 5.3.2. Restrictions on the Mount Topology
   - 5.4. Mount Filter
   - 5.5. Auto-Negotiation of Peer Mount Client QoS
   - 5.6. Datastore Qualification
   - 5.7. Local Mounting
   - 5.8. Mount Cascades
   - 5.9. Transport
   - 5.10. Security Considerations
   - 5.11. High Availability
     - 5.11.1. Reliability
     - 5.11.2. Alignment to late joining peers
     - 5.11.3. Liveliness
     - 5.11.4. Merging of datasets
     - 5.11.5. Distributed Mount Servers
   - 5.12. Configuration
   - 5.13. Assurance and Monitoring

Mirrors existing implementations of Data Distribution Services (DDS)

Peer Mount
Benefits Seen in current prototypes

Application Robustness	• Distributed YANG object trees available to applications • Intersecting domain abstractions 
Application Simplification	• Major code size reduction → data access only to local Datastore • Transparent access to data independent of location • Caching and polling becomes hidden infrastructure
Performance	• Unsolicited & periodic object Pub/Sub with transparent caching • Ability to cascade updates across multiple tiers of Mount
Synchronization	• Avoid need for redundant models for local and remote information • Receipt of unsolicited push of Authoritative object change • Multiple Controller support
Misconfiguration Types Eliminated	• Authoritative copy is explicitly known • Mount & monitor the actual state of the network (not just one-time “ACK”).

Peer Mount Technology Draft

Mounting YANG-Defined Information from Remote Datastores
draft-clemm-netmod-mount-02
A Clemm, J Medved, E Voit

(Multiple implementations including OpenDaylight MD-SAL)

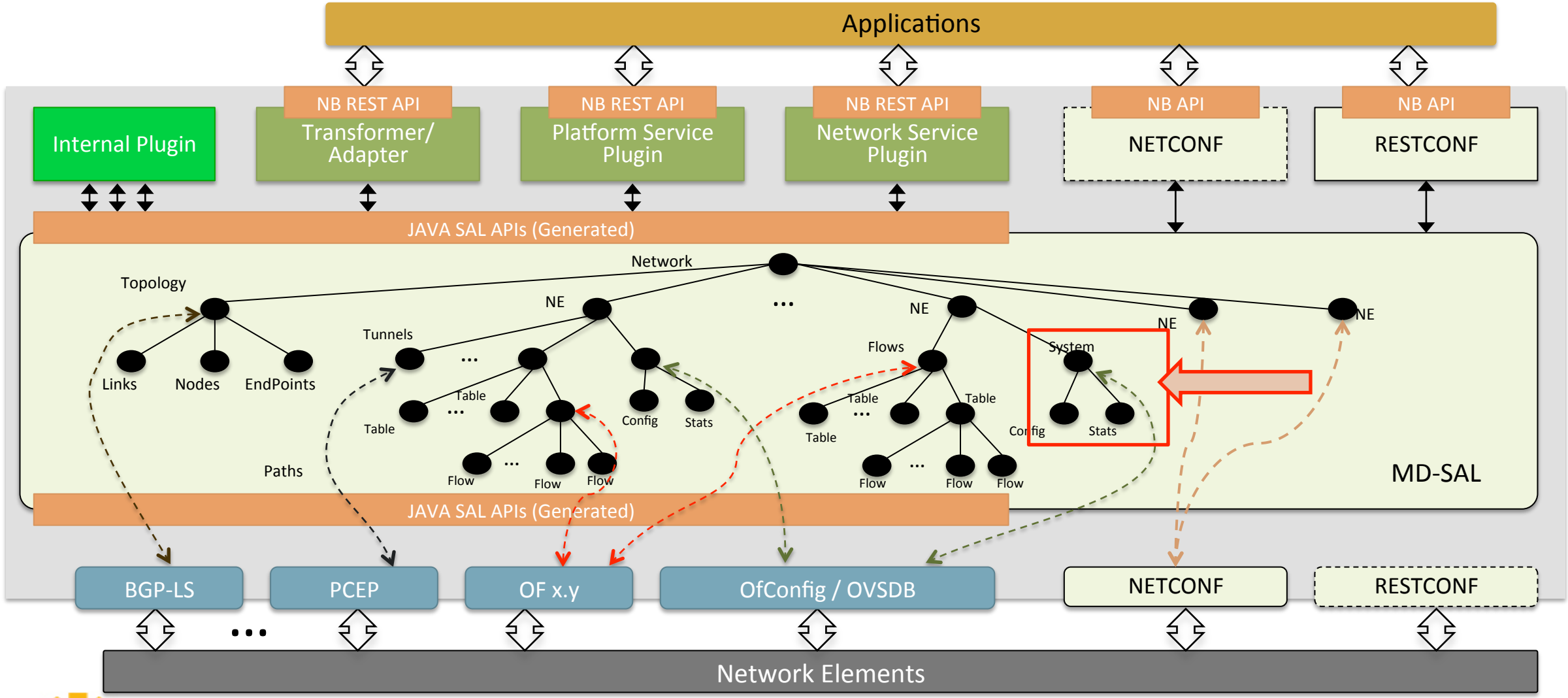
Purpose

- Allow YANG Datastores to reference information in remote datastores
- YANG Server (Netconf, RESTconf) allows its clients/applications to access data that is conceptually federated across a network
  - Incorporate information from remote systems into consolidated view
  - Visibility and validation of parameters with cross-device dependencies
  - Location transparency*
    - *Caveats apply with regards to cross-network transactions. Focus on data retrieval.
- Why - draft-voit-netmod-peer-mount-requirements
  - Ease for app developers/users
    - Federated information from a single source, without need for additional brokering/middleware
  - Avoid need for redundant models
    - NMS Model Layering: “device”-level concepts must be reiterated at “network” layer where needed
    - Heavy systems, slow TTM, less standards adoption as a result
  - “It can be used also for controllers”
    - Controller/device boundaries are blurry, less relevant

Datastore mount concept

- Mount client:
  - Contains mount points at which to attach remote subtrees into data tree
  - Requests whose scope contains remote data are proxied/forwarded to remote system
  - Acts as application/client to the remote system
- Mount server
  - Authoritative owner of the data
  - May not be aware that mounting occurs (mount client is “just another application”)
- Notes
  - Caching optimizations possible → draft-netmod-datastore-push-00
  - Circular mounting prohibited
  - Focus on retrieval of remote data
    - NETCONF-style network transactions per ACID results in issues
    - We can keep configuration out of scope for now
  - Notifications and RPCs currently outside scope

Open Daylight - Model-Driven SAL



OPENDAYLIGHT

www.opendaylight.org

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 {
          ...
     }
     mnt:mountpoint "interfaces" {
         mnt:target "./element-address";
         mnt:subtree "/if:interfaces";
     }
  }
...
```

Mountpoint declaration

- YANG modules 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

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  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
RPCs for manual mount, unmount
```

Mount policies grouping

- Mountpoints can be system-administered
- Applications&users are not exposed to this
- System administration can add bindings

  Update on-demand, periodic, on-change
- Not shown:

  Mount bindings - data update subscriptions

Summary of changes since -01

- Caching considerations and mount bindings – tie-in with datastore push
- Usage guidelines - modeling best practices
- Editorial improvements throughout
- TBD issues:
  - System administration – keep target as part of mountpoint declaration?
  - Remove possibility for configuration? (currently usage discouraged/limited but still permitted)

Example Usage of Mount

Cloud SLA YANG Model incorporating Peer Mount Semantics
draft-tripathy-cloud-sla-yang-model-00
A Tripathy, E Voit, A Clemm

- “Peer Mount” Semantics used in a Model
- Worthy of Note
- Demo (Model in Action)

YANG Model including Mount Semantics

```
+--rw ietf-cloud-sla
     +--rw policies
       +--rw policy [policy-name]
         +--rw policy-name                  string
         +--rw policy-max-bw?               uint64      Set Multi-device abstraction value
         +--ro network-aggregate-bw?        uint64      Derived  (∑ of mounted device counters)
         +--rw nes
           +--rw ne [ne-id]
             +--rw ne-id                    string
             +--rw policing-policies
             |   +--rw policed-bandwidth*   [ifref]
             |      +--rw ifref             mounted-interface-ref   Get interface info from elsewhere
             |      +--rw bandwidth?        uint64      Add a policy object, continually update
             +--ro interfaces-state
                +--M interface-statistics               Mount of Device info (extract RFC 7223)
```

Worthy of Note

- Mounting RFC 7223 interface stats = more objects than desired
  - Value of Mount Filter highlighted
- Network Element counters exposed under policy
  - Read-only copy is effectively an alias (making application writing easier)
- Interface Ref enables adding centrally managed information with Mounted objects
  - New objects can in turn be Mounted back to distributed devices
- Subscription to counters superior to continual requests
  - Current standards don’t cover. One reason for Push draft.

Cloud Policer Demo

Cloud Service on Federated Network Devices

- OpenDaylight MD-SAL Controller
- Cisco Asr9K Network Elements
- Peer Mounting of Network Element YANG counters by controller
- Peer Mounting of controller application derived Policer Rate by Network Elements
- Controller application does nothing but read and write from local YANG abstraction

Summary of First Three Drafts

Some Mailing List Q&A

Question/Assertion	Answer
How can this scale and converge?	The Object Management Group has been standardizing a Data Distribution Service (DDS). It is very similar. Current revenue is ~ $100M+ and growing.
YANG is for devices only	Implementation in OpenDaylight and elsewhere have multi-device abstractions. IETF has multi-device abstractions. If IETF doesn't serve these needs, the technology will fork. Unnecessarily. It is true that WG in charge of Netconf+Netmod architecture is unclear. RFC6244 asserts a device boundary which has been surpassed by reality. We should get clarity for evolving charters.
YANG should support ACID only	Eventual consistency is industry proven for classes of problems. This includes convergence between some abstract and granular policies.
IETF doesn't care about controllers	Declaring multi-device abstractions out-of-scope means that certain applications won't be portable between controllers and Network Devices. In any case, the IETF has defined controllers for years.
"Peer Mount" should be read only.	Agree. Write is possible, but very hard. So let's ignore it for now.

Recommended Actions

1. Netconf will be analyzing “Peer Mount” drafts in interim meetings. Work with Netconf and Ops AD to apportion WG ownership and charters appropriately.

2. Based on output of (1), explore netmod Charter change to include multi-device abstractions. (e.g., something like Peer-Mount)

End Day 1

Subscribing to datastore push updates draft-netmod-clemm-datastore-push-00.txt

Alexander Clemm, Alberto Gonzalez Prieto, Eric Voit

Motivation

- Many applications require continuous updates to datastore contents
  - Mount clients (peer mount): caching of remote data
  - Service assurance: continuous monitoring
  - Big Data: analyze network state
- Periodic polling has limitations (known from SNMP)
  - Additional load on network and devices
  - Lack of robustness, dealing with missed polling cycles
  - Difficult calibration, synchronization of polling cycles across network (makes polled data difficult to compare)
- Current interactions with datastore are request/response based
  - RFC 6470 defines configuration change notifications
  - YANG datastores contain increasingly operational data

Solution Requirements (1/2)

- Provide push mechanism as alternative to polling
- Configuration and management of subscriptions
  - Create/Delete
  - Subscription scope
    - Operational data
    - Subtrees and filters
  - Subscription policy
    - Periodic
    - On change (with dampening)
  - Optional: subscription monitoring
  - Optional: suspend/resume

Solution Requirements (2/2)

- Negotiation capability
  - Resource limitations: not every subscription may be supported
  - Implementation limitations (on change may be difficult)
  - Negotiate update frequency, size, policy (on change vs periodic
- Tie-in with security
  - RFC 6536/NACM – receive updates only for authorized data
- Work in conjunction with Netconf/RESTconf/YANG framework
  - Leverage RFC 5277 notification capability
  - Possibility to decouple transport and subscriptions
    Allow for pub/sub, multicast transports at a later point, outside scope

Subscription Model

```
module: ietf-datastore-push
     +--rw datastore-push-subscription
        +--rw stream                    string
        +--rw subscription-id           subscription-identifier
        +--rw (filter)?
        |  +--:(subtree)
        |  |  +--rw subtree-filter
        |  +--:(xpath)
        |     +--rw xpath-filter        yang:xpath1.0
        +--rw (notification-trigger)
        |  +--:(periodic)
        |  |  +--rw period              yang:timeticks
        |  +--:(on-change)
        |     +--rw (change-policy)
        |        +--:(update-dampening)
        |        |  +--rw period        yang:timeticks
        |        +--:(delta-policy)
        |           +--rw delta         uint32
        +--rw start-time?               yang:date-and-time
        +--rw stop-time?                yang:date-and-time
```

(next revision)

Selected discussion items

- RW vs RO
  and create method
  (edit vs.
  <create-subscription>
- On-change subpolicies options or choices
- “on-change” feature
- Delta policy

Subscription Negotiation

- Leverage RFC 5277 <create-subscription>
- Server may reject a subscription request
  - Implementation limitations (e.g. on-change)
  - Resource limitations
    (e.g. update size, frequency)
- Response to include “acceptable” parameter settings (no guarantee)
- Additional notifications to indicate if server cannot keep “subscription promise)
- Optional: client throttling of subscription via suspend/ resume

Selected discussion items
- <create-subscription> vs edit-config
- Subscription throttling via suspend/resume

Push Data Stream and Transport

- Push-update notifications
  - Subscription correlator
    - Ties update to a specific subscription
  - Data node with datastore update
    - Per subscription
    - Filtered per NACM rules
- Leverage <notification> element (per RFC 5277)
- Alternative transport mappings conceivable but outside scope

Conclusion

- There is a need for a mechanism for datastore push updates, and subscribing to such updates
- Drivers
  - Peer Mount
  - Service Assurance
  - Operational data increasingly part of YANG data models
  - Move beyond SNMP-style polling-based management
- Properties of the solution
  - Data model at its core → Netmod
  - Fits in with YANG/Netconf/RESTconf framework
  - Addresses subscription, negotiation, transport
  - Addresses requirements, PoC exists

Ask: Adopt as WG Document

Backup

Technology Gaps

Why Netconf Event Notifications (RFC 5277) is not a full solution

- Not Pub/Sub
- Pre-dates YANG
- Tied to Netconf, not to the YANG model.
  - Want data to be distributed should be agnostic to transport protocol
  - Different boxes could use different protocols under the covers
- No mechanisms for time synchronized delivery across domain
- Application developer simplicity
  - Transparent caching: requires additional mechanisms to ensure single view across domain

Business Imperative

Virtualization = explosion in Objects

50%+ of outages from mis-config

Speed to activation too slow

Mechanize logic in human brains

Evolving choices in abstraction


CLI


API


GUI

Easy Button

Peering of Controller & Network Element Intelligence