What's NeMo – A *Net*work *Mo*deling Language?

NeMo is a transaction based North Bound (NB) API which allows applications to use intent-based policy to create virtual networks comprised of nodes with policy-controlled flows. Intent based policy is prescriptive (ñgo to the storeò) rather than descriptive (ñfollow this route to the storeò), leaving the details to the network. NeMoôs NB API connects the application to a controller and operates using 10 commands which include: 4 basic network commands (Node, Link, Flow, Policy) and 6 basic controller communication commands (connect, disconnect, transact, commit, notification, query). NeMo sends these 10 commands via the REST protocol to exchange the commands with the controller.

Why NeMo?

Software Defined Networking (SDN) and Network Function Virtualization (NFV) are moving the IT world from a network-centric view to an application view. Google considers the datacenter to be comprised of compute devices, storage devices, and networks. Applications running on the Google Cloud must be rewritten to run within this Cloud environment that takes care of placing applications on compute devices that have the appropriate amount of storage and network connections. NeMo provides a simple NB API which gives the application the power to setup and take down virtual networks between virtual nodes.

NeMo Compared with Open Daylight's Group based product

Open Daylight has a group-based policy project that defines an ñapplication-centricò model that tries to abstract application policy from details of network policy. Two groups of endpoint systems communicate across a shared communication path governed by a contract on the exchange on which one system (EGP 1) and a second system (EGP 2) agree. The contract consists of policy rules which govern flows specified as match-action pairs. (Details on this project can be found at: https://wiki.opendaylight.org/view/Group_Policy:Main).

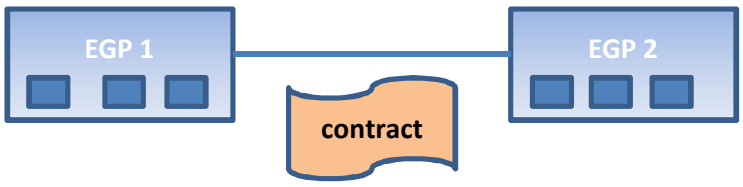
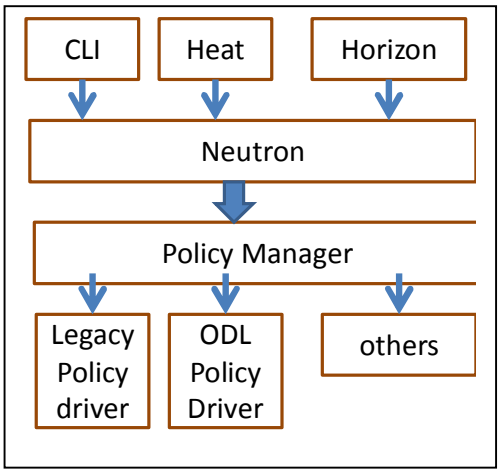


Figure 1 ï Open Daylight Group-based Policy concepts

NeMo compared with Yang and Open Daylight Group Policy

The NeMo project group applauds the Group based Policy project (GBP) for its adoption of the Internet-based framework based on the PCIM work in RFC3060 and RFC3460, and their use of the prescriptive rather than descriptive language. Both NeMo and Open Daylights Group-Based Policy manager are looking to provide Intent driven networking. However, the proof of concept from the GBP policy manager picked different places for the integration. NeMo provides a simple API for the applicationôs interface to middle-layers where the NeMo Language engine communicates with the Virtual Network Engine or multi-vendor SDN controllers. GBP places the command after the middle layer, causing more details to be given by the application to the policy manager at a deeper layer. NeMo can treat the Open Daylight policy manager as a separate engine that it translates, similar to a multi-vendor SDN controller.

Open DayLight Policy Manager



Nemo Policy API and Language Engine

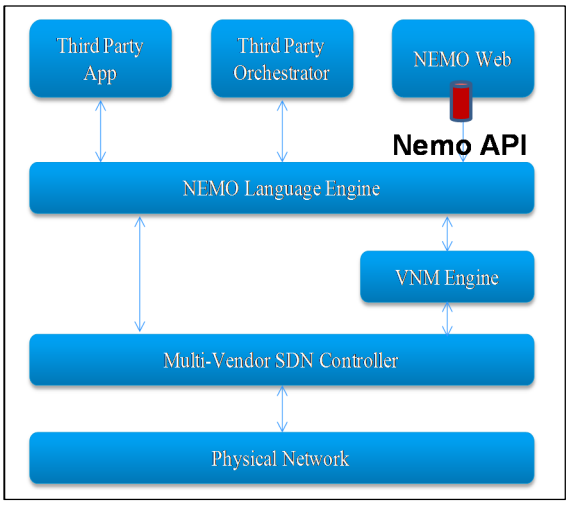


Figure 2 ï Comparison of Placement in stack of Open Daylightôs Policy Manager and Nemoôs Policy API and Nemoôs language engine

Open-Daylightôs Group-based policy (GBP) has the following differences from NeMo:

- **GBP focus on the policy to control flow behavior, but NeMo handles the whole network.**

  Why this is important: Without the capacity to create and control a node and links, it is hard to manage NFV devices.

- **GBP examines a contract between two end systems** without considering the requirements for paths, but **NeMo considers the whole path.**

  Why this is important: While an abstract pathway that GBP provides may be useful for

some Data Centers, it may not describe paths required by legal restrictions (for medical data), or best-cost network routes.

- **GBP operates below the Application interface** which does not allow the application to signal directly its need, and has a mixture of high-level abstraction and lots of details on the commands**. NeMo provides an API for Intent-Driven application** that uses only 15 sentences of three action types (node, link, flow), three behaviors types (query, notification, and policy), 4 transaction messages (connect, disconnect, transaction and commit),

  ***Why this is important*:** Past studies (UT-Dallas and others) have shown that the best utilization of network, compute, and storage is gained when the application and network cooperate to direct paths. NEMO provides a simple interface that applications can implement, and allows the vendor to provide a NEMO engine. NEMO combined with LibVirt may be able to control compute, storage, and network.

- GBP may/may not be extensible to multiple vendorôs SDNs. NEMO was designed with multiple SDN controllers in mind.

  Why this is important: Interoperable SDN controllers remove a single-point of failure in the network.

NEMO	GBP	Description
Node	EndPoint	End point in GBP is more like a port, which can be used to attach VM allocated by NOVA(an OpenStack compute component) **Problems:** GBP cannot describe NFV device
Node->Logical Node	EndPointGroup	NEMO allows the nodes at the application view to be a multiple node cluster with a single set of IP addresses at the IP layer. **Problem:** GBP provides point-to-point policy instead of network wide-policy.
Link	Implicitly expressed by Contract	1. GBP is a very high level abstraction at the mid-layer **Problem:** information like topology is not obvious for application users. 2. Bandwidth and delay which are more intuitive to be a link property. However GBP describes this as a QoS policy. **Problem:** User may be confused by QoS policy.

NeMo compared with Yang and Open Daylight Group Policy

Flow	Classifier in a policy rule	NeMoôs and GBPôs flow match item both support to IP packet fields, plus interfaces, and non-IP protocols.
Policy	Policy rules	GBP has a classifier instead of NEMOôs ñconditionò in its policy rule. This syntactical difference may create differences in conditions. . **Problem:** GBP policy only applies to flow object. NEMO policy is extended to link and node.
Notification	No	
Query	No	

NeMo vs OpFlex

The OpFlex protocol (draft-smith-opflex-00) is another intent-based protocol which declares the policy infrastructure and allows controls to make it happen. The NeMo team is pleased to see policy repository work that considers centralized methods of policy storage. The OpFlex protocol stores policies in a central policy repository (PR) and distributes these policies to policy elements (PE) where the policies are enforced. The OpFlex control protocol allows bidirectional communication between the PR and the PE so that the policy, events, statistics and faults can go both directions. OpFlex re-invents many of the same concepts of centralized policy control point (PCP) and policy enforcement point (PEP) that the COPS protocol (RFC2748, RFC3084, and RFC4261) contained. This work is an exciting placement outgrowth of the intent-based policy for centralized storage, but a consideration of virtualized centralized storage (with physically logical points) may also need to be considered.

Possible Steps for NEMOôs and Open Daylight Group-based Policy

- NEMOôs API is very simple at the application layer so this could be added to the Neutron application layer,
- NEMOôs code integrated into OpenDaylight Stack - Nemoôs API and Language Engine can be used within the OpenDaylight stack above an Open-Flow controller,
- NEMOôs code interfaces to the Policy Manager in the Open Daylight stack, and
- Testing to compare effectiveness of OpenDaylights Group-based Policy engine versus Nemo.

The Nemo Project would like your feedback on these next steps. Please send us a note at nemo@hickoryhill-consulting.com

NeMo's 4 Network Commands in Formal Language

<node_cu> := **Node** <node_id> **Type** <node_type> **LogicNW** <node_id>

[Property {<property_name>: <value>}];

<link_cu> := **Link** <link_id> **EndNodes** <node_type> : <node_id>, <node_type> : <node_id>

[Property{<property_name>: <value>}];

<flow_cu> := **Flow** <flow_id> **Match** {<property_name>: <value>

| **Range** (<value>, <value>) | **List** ({<value>})}

<policy_cu> := Policy <policy_id> ApplyTo <entity_type> : <entity_id>

Priority <integer> [**Condition** {<expression>}]
Action {<action_type> : {<value>}};

A ñNoò in front of each of these commands deletes these commands. An example of this is:

<node_del> := **NoNode** <node_id>;

NeMo's 6 Controller Interaction Commands in Formal Language

!*These commands connect and disconnect the application from the controller.*

<connect_cmd> := **Connect** < conn_id > **Address** <ip_addr> **Port** <integer>

<disconnet_cmd> := **Disconnect** < conn_id >

!*These commands group a set of policy that need to be enacted together.*

<transaction_begin> := **Transaction**

<transaction_end> := **Commit**

! *These commands query or provide notification of policy information*

<query_cmd> := **Query** {< property_name>} **From** <entity_type> | <policy_type>: <UUID>

<notification_cu> := **Notification** <notification_id> **Query** {< property_name>}
From <entity_type> : <entity_id>
Every <integer> **Listener** <callbackfunc>;