Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: October 4, 2014

M. Smith
M. Dvorkin
Cisco Systems, Inc.
Y. Laribi
Citrix
V. Pandey
IBM
P. Garg
Microsoft Corporation
N. Weidenbacher
Sungard Availability Services

April 2, 2014

OpFlex Control Protocol
draft-smith-opflex-00

Abstract

The OpFlex architecture provides a distributed control system based on a declarative policy information model. The policies are defined at a logically centralized policy repository (PR) and enforced within a set of distributed policy elements (PE). The PR communicates with the subordinate PEs using the OpFlex Control protocol. This protocol allows for bidirectional communication of policy, events, statistics, and faults. This document defines the OpFlex Control Protocol.

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 October 4, 2014.

Copyright Notice

Table of Contents

1. Introduction

As software development processes merge with IT operations, there is an increasing demand for automation and agility within the IT infrastructure. Application deployment has been impeded due to the existing IT infrastructure operational models. Management at scale is a very difficult problem and existing imperative management models typically falter when challenged with the heterogeneity of various platforms, applications, and releases. In such environments, declarative management models have shown to cope quite well. In these systems, agents have autonomy of control and provide a declaration of intent regarding behavior. Declarative policy is rendered locally to provide desired system behavior. The OpFlex architecture is founded in these concepts.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

1.2. Terminology

AD: Administrative Domain. A logical instantiation of the OpFlex system components controlled by a single administrative policy.

EP: Endpoint. A device connected to the system.

EPR: Endpoint Registry. A logically centralized entity containing the endpoint registrations within associated administrative domain.

OB: Observer. A logically centralized entity that serves as a repository for statistics, faults, and events.

PE: Policy Element. A function associated with entities comprising the policy administrative domain that is responsible for local rendering of policy.

PR: Policy Repository. A logically centralized entity containing the definition of all policies governing the behavior of the associated administrative domain.

OpFlex Device: Entity under the management of a Policy Element.

JSON: Javascript Object Notation [RFC4627]

XML: Extensible Markup Language [XML]

2. Scope

This document defines the OpFlex Control Protocol used between OpFlex system components. It does not define the policy object model or the policy object model schemas. A System Overview section is provided for reference.

3. System Overview

OpFlex is a policy driven system used to control a large set of physical and virtual devices. The OpFlex system architecture consists of a number of logical components. These are the Policy Repository (PR), Endpoint Registry (EPR), Observer, and the Policy Elements (PE). These components and their interactions are described in the following subsections.

3.1. Policy Repository

Within each administrative domain of the OpFlex system, there is a single logical entity referred to as the Policy Repository (PR) that serves as the single source of all policies. The PR handles policy resolution requests from the Policy Elements within the same administrative domain. An example scope of an administrative domain

would be a datacenter fabric. These policies are configured directly by the user via a policy administration interface (API/UI/CLI/etc.) or indirectly (implicitly through the application of higher order policy constructs). These policies represent a declarative statement of desired state. Policies are typically abstracted from the underlying implementation.

3.1.1. Management Information Model

All of the physical and logical components that comprise the administrative domain are represented in a hierarchical management information model (MIM), also referred to as the management information tree (MIT). The hierarchical structure starts at a root node and all policies within the system can be reached via parent and child containment relationships. Each node has a unique Uniform Resource Identifier (URI) [RFC3986] that indicates its place in the tree.

3.1.1.1. Managed Object

Each node in the tree represents a managed object (MO) or group of objects and contains its administrative state and operational state. An MO can represent a concrete object, such as a switch or adapter, or a logical object, such as a policy or fault. An MO consists of the following items:

Properties: A property is a named instance of policy data and is interpreted by the Policy Element in local rendering of the policy.

Child Relations: A containment relationship between MOs where the children MOs are contained within the parent MO.

Parent Relation: The inverse of the children relationship. This relation is implicit and is implied through the hierarchical name of the MO name.

MO Relations: Relationships with other MOs in the system that are not containment relationships. These relationships can be unidirectional or bidirectional. The

relationships can also be 1:1, 1:n, or m:n.

Statistics: These are child MOs that track statistics relevant to the parent MOs. These MOs are reported to the Observer.

Faults: These are child MOs that track faults relevant to the parent MOs. These MOs are reported to the Observer.

Health: These are child MOs that track the overall health relevant to the parent MOs. This is often represented in the form of a health score. These MOs are reported to the Observer.

MOs that contain statistic, fault, or health MOs are said to be observable.

3.2. Endpoint Registry

The Endpoint Registry (EPR) is the component that stores the current operational state of the endpoints (EP) within the system. PEs register the EPs with the EPR upon EP attachment to the local device where the PE is resident. Upon EP detachment, the registration will be withdrawn. The EP registration information contains the scope of the EP such as the Tenant or logical network as well as location

information such as the hypervisor where the EP resides. The EPR can be used by PEs to query the current EPR registrations as well as receive updates when the information changes.

3.3. Observer

The Observer serves as the monitoring subsystem that provides a detailed view of the system operational state and performance. It serves as a data repository for information related to trending, forensics, and long-term visibility data such as statistics, events, and faults. Statistical data is reported to the Observer at expiration of reporting intervals and statistics will be rolled up for longer-term trend analysis.

3.4. Policy Element

Policy elements (PEs) are logical functional abstractions of member elements within the administrative domain. Policy elements reside on physical or virtual devices that are subjected to policy control under a given administrative domain. PEs receives policy triggers through local triggers or triggers invoked by other PEs. Local triggers involve local MO state transitions such as new control node additions, removals, or other operational events. Policy triggers invoked by other PEs are transmitted using the OpFlex Control Protocol. Both types of policy triggers result in policy resolution. Policies are resolved with the PR using the OpFlex protocol. This protocol allows bidirectional communication, and allows the exchange of policy information. Policies are represented as managed object "sub-trees". Upon policy resolution, the PE renders the policy to the configuration of the underlying subsystem, and continuously performs health monitoring of the subsystem. PEs perform local corrective actions as needed for the enforcement of policies in its scope. Operational transitions can also cause new or additional/ incremental policy resolutions such as the attachment of new EPs to the corresponding device.

4. OpFlex Control Protocol

The OpFlex Control Protocol is used by OpFlex system components to communicate policy and operational data. The protocol uses JSON, XML, or OpFlex-Binary-RPC as the wire encoding. This document describes the JSON format and uses JSON-RPC version 1.0 [JSON-RPC]. The JSON-RPC transport SHOULD be over TCP. The description of the encoding and transport of XML and OpFlex-Binary-RPC are left to later revisions of this document.

4.1. JSON Usage

The descriptions below use the following shorthand notations for JSON values. Terminology follows [RFC4627].

<string>:
: A JSON string. Any Unicode string is allowed. Implementations SHOULD disallow null bytes.

<integer>:
: A JSON number with an integer value, within the range -(2**63)...+(2**63)-1.

<json-value>:
: Any JSON value.

<nonnull-json-value>:
: Any JSON value except null.

<URI>:
: A JSON string in the form of a Uniform Resource Identifier[RFC3986].

<status>:
: An enumeration specifying one of the following set of strings: "created", "modified", or "deleted".

<role>:
: An enumeration specifying one of the following set of strings: "policy_element", "observer", "policy_repository", or "endpoint_registry".

<mo>:
: A JSON object with the following members:

```
"name": <URI>
"properties": [{"name":<string>, "data": <string>}*]
"children": [<mo>*]
"statistics": [<mo>*]
"from_relations": [<mo>*]
"to_relations": [<mo>*]
"faults": [<mo>*]
"health": [<mo>*]
```

All of the members of the JSON object are REQUIRED. However, the corresponding value MAY consist of the empty set for all members except for "name". It is REQUIRED that the "name" be specified.

The "name" uniquely identifies the managed object within the

scope of the administrative domain and indicates its location within the MIT.

The "properties" holds a set of named policy data.

The "children" identifies a set of MOs where each MO is considered a child of this particular MO.

The "statistics" identifies a set of MOs containing statistic data maintained by the policy rendered from this particular MO.

The "from_relationships" identifies a set of relationship MOs. Each relationship MO has a reference to the MOs that have relationship to this particular MO.

The "to_relationships" identifies a set of relationship MOs. Each relationship MO has a reference to the MOs that have relationship from this particular MO.

The "faults" identifies a set of MOs containing fault information maintained by the policy rendered from this particular MO.

The "health" identifies a set of MOs containing health metrics maintained by the policy rendered from this particular MO.

In the case of MOs used as policies, there will be no statistics, faults, or health.

4.2. RPC Methods

The following subsections describe the RPC methods that are supported. As described in the JSON-RPC 1.0 specification, each request comprises a string containing the name of the method, a (possibly null) array of parameters to pass to the method, and a request ID, which can be used to match the response to the request. Each response comprises a result object (non-null in the event of a successful invocation), an error object (non-null in the event of an error), and the ID of the matching request. More details on each method, its parameters, and its results are described below.

A Policy Element is configured with the connectivity information of at least one peer OpFlex Control Protocol participant. The connectivity information consists of the information necessary to establish the initial connection such as the IP address and wire

encapsulation. A Policy Element MAY be configured with the connectivity information for one or more of the OpFlex logical components. A Policy Element MUST connect to each of the configured OpFlex logical components.

4.2.1. Identity

This method identifies the participant to its peer in the protocol exchange and MUST be sent as the first OpFlex protocol method. The method indicates the transmitter's role and the administrative domain to which it belongs. Upon receiving an Identity message, the response will contain the configured connectivity information that the participant is using to communicate with each of the OpFlex components. If the response receiver is a Policy Element and is not configured with connectivity information for certain OpFlex logical components, it SHOULD use the peer's connectivity information to establish communication with the OpFlex logical components that have not been locally configured.

The Identity request contains the following members:

- "method": "send_identity"
- "params": [

```
"name": <string>
"domain": <string>
["my_role": <role>]+
]
```

- "id": <nonnull-json-value>

The "name" is an identifier of the OpFlex Control Protocol participant that is unique within the administrative domain.

The "domain" is a globally unique identifier indicating the administrative domain that this participant exists.

The "my_role" states the particular OpFlex component contained within this participant. Since a participant may be capable of acting as more than 1 type of component, there may be multiple "my_role" parameters passed.

The response object contains the following members:

- "result": [

```
        "name": <string>
        [ "my_role": <role> ]+
        "domain": <string>
        [ {"role": <role>
        "connectivity_info": <string>}* ]
        ]
```

- "error": null
- "id": same "id" as request

The "name" is the identifier of the OpFlex Control Protocol participant sending the response.

The "my_role" states the OpFlex component roles contained within the participant sending the response.

The "domain" is a globally unique identifier indicating the administrative domain that the participant sending the response exists.

The "role" and associated "connectivity_info" give the reachability information (i.e. IP address or DNS name) and the role of the entity that the participant is communicating using the OpFlex Control Protocol. This information MAY be gleaned by a receiving participant to resolve reachability for various OpFlex components.

In the event that the administrative domains do not match, an error response of the following form:

- "result": null
- "error": "Domain mismatch"
- "id": same "id" as request

4.2.2. Policy Resolution

This method retrieves the policy associated with the given policy

name. The policy is returned as a set of managed objects. This method is typically sent by the PE to the PR.

The request object contains the following members:

- "method": "resolve_policy"

- "params": [


```
"subject": <string>
"context": <string>
"policy_name": <string>
"on_behalf_of": <URI>
"data": <string>
]
```

- "id": <nonnull-json-value>

The "subject" provides the class of entity for which the policy is being resolved. The applicable object classes are dependent on the particular MIT.

The "context" is used to scope the policy resolution request. Common examples would be scoping within a particular tenant name.

The "policy_name" is the name of the policy needs to be resolved.

The "on_behalf_of" indicates the MO that triggered this policy resolution.

The "data" provides additional opaque data that may be used to assist in the policy resolution.

Upon successful policy resolution, the response object contains the following members:

- "result": [

```
"policy": <mo>+,
"prr": <integer>]
```

- "error": null
- "id": same "id" as request

The "policy" parameter contains the managed objects that represent the resolved policy. These objects are used by the Policy Element to render and apply the local policy. The application of the local policy may cause the local PE to deliver policy triggers to other PEs in the system.

The "prr" or Policy Refresh Rate provides the amount of time that a PE should use the policy as provided in the request. The <integer> indicates the time in seconds that the policy should be kept by the PE. A PE SHOULD issue another policy resolution request before the expiration of the prr timer if the PE still requires the policy. If the PE is unable to subsequently resolve the policy after the prr

timer expires, the PE MAY continue to use the resolved policy. The PE SHOULD raise an alarm if the policy cannot be resolved after multiple attempts.

In the event that the policy named in the resolution request does not exist, an error response of the following form:

- "result": null
- "error": "unknown policy name"
- "id": same "id" as request

4.2.3. Policy Update

This method is sent to Policy Elements when there has been a change of policy definition for policies which the Policy Element has requested resolution. Policy Updates will only be sent to Policy Element for which the policy refresh rate timer has not expired.

The Policy Update contains the following members:

- "method": "update_policy"
- "params": [

```
      "context": <named_tlv>
      ["subtree": <mo>+]+
      "prr": <integer>
      ]
```

- "id": <nonnull-json-value>

The "context" is used to indicate the scope of the policy. This is typically the same as the context in the original policy resolution request but it may be different.

The "subtree" contains one or more subtrees of the MIT. Each subtree is a collection of MOs that represent the changed policy.

The "prr" or Policy Refresh Rate provides the amount of time that a PE should use the policy as provided in the request. The <integer> indicates the time in seconds that the policy should be kept by the PE. A PE SHOULD issue another policy resolution request before the expiration of the prr timer if the PE still requires the policy. If the PE is unable to subsequently resolve the policy after the prr timer expires, the PE MAY continue to use the resolved policy. The PE SHOULD raise an alarm if the policy cannot be resolved after

multiple attempts.

The response object contains the following members:

- "result": {}
- "error": null
- "id": same "id" as request

4.2.4. Echo

The "echo" method can be used by OpFlex Control Protocol peers to verify the liveness of a connection. It MUST be implemented by all participants. The members of the request are:

- "method": "echo"

- "params": JSON array with any contents
- "id": <nonnull-json-value>

The response object has the following members:

- "result": same as "params"
- "error": null
- "id": same "id" as request

4.2.5. Policy Trigger

A policy trigger is issued from one Policy Element to a peer Policy Element in order to trigger a policy resolution on the peer. It is typically done to indicate a attachment state change or a change in the consumption of the peer resources. For example, a Policy Element in a switch may cause a policy trigger in the upstream switch to enable a particular VLAN on the peer's interface. It may also be issued upon receiving a Policy Update or Policy Resolution response.

The Policy Trigger contains the following members:

- "method": "trigger_policy"
- "params": [


```
        "policy_type": <string>
        "context": <string>
        "policy_name": <string>
        "prr": <integer>
        ]
```

- "id": <nonnull-json-value>

The response object contains the following members:

- "result": {}

- "error": null
- "id": same "id" as request

4.2.6. Endpoint Declaration

This method is used to indicate the attachment and detachment of an endpoint. It is sent from the Policy Element to the Endpoint Registry.

The Endpoint Declaration contains the following members:

- "method": "endpoint_declaration"
- "params": [

```
"subject": <string>
"context": <string>
"policy_name": <string>
"location": <URI>
["identifier": <string>]+
["data": <string>]*
"status": <status>
"prr": <integer>
]
```

- "id": <nonnull-json-value>

The "subject" provides the class of entity for which the declaration applies. This will typically be the class representing the endpoint. The applicable object classes are dependent on the particular MIT.

The "context" is used to scope the endpoint declaration.

The "policy_name" is used to identify the policy that must be resolved and applied when this endpoint attaches, detaches, or is

otherwise modified.

The "location" is used to identify the managed object indicating the point where the endpoint connects to the system. An example would be

a managed object representing a certain physical port on a ethernet switch.

The "identifier" is a label that is used in identifying the particular instance of the endpoint. Some examples of an identifier would be a MAC address, VLAN, and IP address.

The "data" are used along with the context, endpoint class, endpoint MO, and the policy_name to select the policy that will be applied to the particular endpoint. These are typically labels used in identifying particular endpoint or endpoint location characteristics. Some examples would include trusted, untrusted, production, test, etc.

The "status" indicates whether this declaration is an endpoint attachment, detachment, or modification.

The "prr" or Policy Refresh Rate provides provides the amount of time that the endpoint declaration will remain valid. The <integer> indicates the time in seconds that the endpoint declaration should be kept by the EPR. A PE SHOULD issue another endpoint declaration before the expiration of the prr timer if the endpoint is to continue existing within the system.

The response object contains the following members:

- "result": {}
- "error": null
- "id": same "id" as request

4.2.7. Endpoint Request

This method queries the EPR for the registration of a particular EP. The request is made using the identifiers of the endpoint. Since multiple identifiers may be used to uniquely identify a particular endpoint, there may be more than 1 endpoint returned in the reply if the identifiers presented do not uniquely specify the endpoint.

The Endpoint Request contains the following members:

- "method": "endpoint_request"

- "params": [

```
"subject": <string>
"context": <string>
["identifier": <string>]+
]
```

- "id": <nonnull-json-value>

The "subject" provides the class of entity for which the request applies. This will typically be the class representing the endpoint. The applicable object classes are dependent on the particular MIT.

The "context" is used to scope the endpoint resolution.

The "identifier" is a label that is used in identifying the particular instance of the endpoint. Some examples of an identifier would be a MAC address, VLAN, and IP address.

The "prr" or Policy Refresh Rate provides provides the amount of time that the endpoint information will remain valid. The <integer> indicates the time in seconds that the endpoint information should be kept by the PE. A PE SHOULD issue another endpoint request before the expiration of the prr timer if the communication is still required with the endpoint.

The response object contains the registrations of one or more endpoints. Each endpoint contains the same information that was present in the original registration. The following members are present in the response:

- "result": {

```
[ endpoint :
{"subject": <string>
"context": <string>
"policy_name": <string>
"location": <URI>
["identifier": <string>]+
["data": <string>]*
"status": <status>
"prr": <integer>
} ]+
}
```

- "error": null
- "id": same "id" as request

The following error response object is returned if no endpoints match the identifiers presented in the request:

- "result": {}
- "error": "No endpoints found."
- "id": same "id" as request

4.2.8. Endpoint Policy Update

This method is sent to Policy Elements by the EPR when there has been a change relating to the EP Declaration for an Endpoint that the Policy Element has requested. Policy Updates will only be sent to Policy Elements for which the Policy Refresh Rate timer timer for the Endpoint Request has not expired.

The Endpoint Policy Update contains the following members:

- "method": "endpoint_update_policy"
- "params": [

```
"subject": <string>
"context": <string>
"policy_name": <string>
"location": <URI>
["identifier": <string>]+
["data": <string>]*
"status": <status>
"ttl": <integer>
]
```

- "id": <nonnull-json-value>

All of the "params" contain identical information to the descriptions

given as part of the Endpoint Declaration.

The response object contains the following members:

- "result": {}
- "error": null

- "id": same "id" as request

4.2.9. State Report

This method is sent by the Policy Element to the Observer. It provides fault, event, statistics, and health information in the form of managed objects.

The State Report contains the following members:

- "method": "report_state"
- "params": [

```
"subject": <URI>
"context": <string>
"object": <mo>
["fault": <mo>]*
["event": <mo>]*
["statistics": <mo>]*
["health": <mo>]*
]
```

- "id": <nonnull-json-value>

The "subject" provides the class of entity for which the State Report applies. The applicable object classes are dependent on the particular MIT.

The "context" is used to scope the subject.

The "object" is the specific managed object that the faults, events, statistics, and health reports in this method apply.

The "fault" is an optional field that contains one or more managed objects representing faults.

The "events" is an optional field that contains one or more managed objects representing events.

The "statistics" is an optional field that contains one or more managed objects representing statistics.

The "health" is an optional field that contains one or more managed objects representing health statistics applicable.

The response object contains the following members:

- "result": {}
- "error": null
- "id": same "id" as request

5. IANA Considerations

A TCP port will be requested from IANA for the OpFlex Control Protocol.

6. Security Considerations

The OpFlex Control Protocol itself does not address authentication, integrity, and privacy of the communication between the various OpFlex components. In order to protect the communication, the OpFlex Control Protocol SHOULD be secured using Transport Layer Security (TLS) [RFC5246]. The distribution of credentials will vary depending on the deployment. In some deployments, existing secure channels can be used to distribute the credentials.

7. Acknowledgements

The authors would like to thank Vijay Chander, Mike Cohen, and Brad McConnell for their comments and contributions.

8. Normative References

[JSON-RPC] "JSON-RPC Specification, Version 1.0", <http://json-rpc.org/wiki/specification>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

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

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[XML] Bray, T., Jean Paoli, Sperberg-McQueen, C., and E. Maler, Ed., "Extensible Markup Language (XML) 1.0 (Second Edition)", October 2000, <http://www.w3.org/TR/REC-xml>.

Authors' Addresses

Michael Smith
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, California 95134
USA

Email: michsmit@cisco.com

Mike Dvorkin
Cisco Systems, Inc.
170 West Tasman Drive

San Jose, California 95134
USA

Email: midvorki@cisco.com

Youcef Laribi
Citrix
4988 Great America Parkway
Santa Clara, California 95054
USA

Email: Youcef.Laribi@citrix.com

Vijoy Pandey
IBM
4400 N First Street
San Jose, California 95134
USA

Email: vijoy.pandey@us.ibm.com

Pankaj Garg
Microsoft Corporation
1 Microsoft Way
Redmond, Washington 98052
USA

Email: pankajg@microsoft.com

Nik Weidenbacher
Sungard Availability Services
Philadelphia, Pennsylvania

USA

Email: nik.weidenbacher@sungard.com