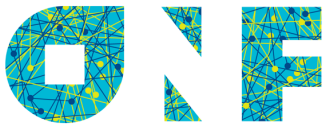

OPEN NETWORKING FOUNDATION

NDM Negotiation OpenFlow Extension

Version 1.0

ONF TR-536

2016-09-08

ONF Document Type: Technical Recommendation
ONF Document Name: NDM Negotiation OpenFlow Extension

Disclaimer

Open Networking Foundation
2275 E. Bayshore Road, Suite 103, Palo Alto, CA 94303
www.opennetworking.org

Table of Contents

1 Introduction

This document describes an OpenFlow extension that enables negotiating and (re)activating a Negotiable Datapath Model (NDM), (re)configuring parameters for the active NDM, as well as obtaining details w.r.t. the currently active NDM as well as associated parameters. Here the NDM may be a Table Type Pattern (TTP), or any other class of NDM.

2 Concepts

2.1 Negotiable Datapath Models and Table Type Patterns

A Negotiable Datapath Model (NDM) is a representation of the constraints of the forwarding model supported by a networking device or required by a controller / application.

The NDM itself is expressed in JavaScript Object Notation (JSON), or another suitable format. The NDM specification allows for alternate representations as long as they conform to the specified information model (structure, keywords, etc.). The specification (in effect the schema) for NDMs can be downloaded from the ONF's web site.

A NDM is typically stored in a file or distributed via a web server, a version control repository, or a similar repository. Each NDM has an associated identifier, and every final / released NDM is expected to have a globally unique identifier.

A Table Type Pattern (TTP) is an instance of a Negotiable Datapath Model.

2.2 NDM Identifiers

The exact format of NDM identifiers is defined in the NDM specification. NDM identifiers are comprised of the following fields: (authority, type, name, version). NDM identifiers specify the authority responsible for creating the NDM, the specific NDM created by that organization / individual, and version identifying details. For the purposes of this document, a NDM identifier is a string consisting of non-whitespace printable characters.

2.3 NDM Parameters

NDMs can define parameters. The parameters are identified using parameter names, which are again defined as strings of non-whitespace printable characters for the purposes of this document. Associated with each parameter is a data type, a default value, and the permissible values (e.g. the permissible value range for numeric parameters). Some of these details (e.g. type and range) w.r.t. each parameter are specified in the NDM, but other details (e.g. default value or a more or less restrictive value range) may be artifacts of the switch implementation. If a parameter value is not supplied and a switch does not supply an appropriate default value, or if values are out of range, an error message may be returned. Parameter values can be set (i.e. written) when activating or re-activating a NDM. The currently active parameter values can also be queried (i.e. read back) from the switch. It is possible to read parameters that have not been set, in which case default or switch selected values will be returned. Some parameters may be read only.

2.4 NDM Negotiation

The purpose of negotiating a NDM is to arrive at a NDM (i.e. identify and activate a NDM) mutually acceptable to the OpenFlow controller (acting in collaboration with a northbound application) and the networking device (OpenFlow switch). During the negotiation process, NDM IDs and parameter names/values are exchanged as detailed in the rest of this document.

3 NDM Experimenter ID

The experimenter ID of this extension is:

```
ONF_ODWG_EXPERIMENTER_ID = 0xFF000006
```

4 NDM Experimenter Messages

4.1 Common Header and Experimenter Type

All messages share the following header at the start of the message:

```
/* Common header for NDM extension messages */
struct onf_ndm_header {
    struct ofp_header header;
    uint32_t experimenter;       /* ONF_ODWG_EXPERIMENTER_ID */
    uint32_t exp_type;           /* One of ONF_ET_*_NDM_* */
};
#if OPENFLOW_VERSION <= 13
OFP_ASSERT(sizeof(struct onf_ndm_header) ==
           sizeof(struct ofp_experimenter_header));
#else
OFP_ASSERT(sizeof(struct onf_ndm_header) ==
           sizeof(struct ofp_experimenter_msg));
#endif
```

The `header` field is a standard OpenFlow header as defined in the specification.

The `experimenter` field is the Experimenter ID (see section 3).

The `exp_type` field is the experimenter message type. It enables this extension to have many types of messages. The currently defined experimenter message types are:

```
/* Message types */
enum onf_ndm_exp_type {
    ONF_ET_GET_SUPPORTED_NDM_REQUEST = 1,
    ONF_ET_GET_SUPPORTED_NDM_REPLY   = 2,
    ONF_ET_GET_ACTIVE_NDM_REQUEST    = 3,
    ONF_ET_GET_ACTIVE_NDM_REPLY      = 4,
    ONF_ET_SET_ACTIVE_NDM_REQUEST    = 5,
    ONF_ET_SET_ACTIVE_NDM_REPLY      = 6,
};
```


4.2 Get Supported NDMs

The ONF_ET_GET_SUPPORTED_NDM_REQUEST message type uses the following message structure:

```
/* Get supported NDMs request. */
struct onf_get_supported_ndm_request {
    struct ofp_header header;
    uint32_t experimenter; /* ONF_ODWG_EXPERIMENTER_ID */
    uint32_t exp_type;     /* ONF_ET_GET_SUPPORTED_NDM_REQUEST */
};
```

It results in the ONF_ET_GET_SUPPORTED_NDM_REPLY message type, which uses the following message structure:

```
/* Get supported NDMs reply. */
struct onf_get_supported_ndm_reply {
    struct ofp_header header;
    uint32_t experimenter; /* ONF_ODWG_EXPERIMENTER_ID */
    uint32_t exp_type;     /* ONF_ET_GET_SUPPORTED_NDM_REPLY */
    uint32_t ndm_list_len; /* Length of following field */
    char     ndm_list[0];  /* JSON array (list) of NDM IDs in double quoted strings */
    uint8_t  pad[0];       /* Pad to 4 byte alignment */
};
```

Should errors occur, an error message is returned instead.

4.3 Get Active NDM

The ONF_ET_GET_ACTIVE_NDM_REQUEST message type uses the following message structure:

```
/* Get active NDM request. */
struct onf_get_active_ndm_request {
    struct ofp_header header;
    uint32_t experimenter; /* ONF_ODWG_EXPERIMENTER_ID */
    uint32_t exp_type;     /* ONF_ET_GET_ACTIVE_NDM_REQUEST */
};
```

It results in the ONF_NDM_ET_GET_ACTIVE_REPLY message type, which uses the following message structure:

```
/* Get active NDM and parameters reply. */
struct onf_get_active_ndm_reply {
    struct ofp_header header;
    uint32_t experimenter;   /* ONF_ODWG_EXPERIMENTER_ID */
    uint32_t exp_type;       /* ONF_ET_GET_ACTIVE_NDM_REPLY */
    uint32_t active_ndm_len; /* Length of following field */
    char     active_ndm[0];  /* NDM ID in double quoted string */
    uint8_t  pad1[0];        /* Pad to 4 byte alignment */
    uint32_t parameters_len; /* Length of following field */
    char     parameters[0];  /* Parameters: key, value... in JSON object (map) */
    uint8_t  pad2[0];        /* Pad to 4 byte alignment */
};
```

4.4 Set Active NDM

The ONF_ET_SET_ACTIVE_NDM_REQUEST message type uses the following message structure:

```
/* Set active NDM and parameters request. */
struct onf_set_active_ndm_request {
    struct ofp_header header;
    uint32_t experimenter;   /* ONF_ODWG_EXPERIMENTER_ID */
    uint32_t exp_type;       /* ONF_ET_GET_ACTIVE_NDM_REQUEST */
    uint32_t active_ndm_len; /* Length of following field */
    char     active_ndm[0];  /* NDM ID in double quoted string */
    uint8_t  pad1[0];        /* Pad to 4 byte alignment */
    uint32_t parameters_len; /* Length of following field */
    char     parameters[0];  /* Parameters: key, value... in JSON object (map) */
    uint8_t  pad2[0];        /* Pad to 4 byte alignment */
};
```

If the supplied NDM ID is "default", the currently active NDM (if any) is deactivated, and the switch reverts to the mode that was active prior to NDM negotiation. De-activating the NDM if no NDM is active is permissible, and no error code is returned in this case.

This specification does not define how existing state (e.g. flow table entries and counter values) is affected by NDM deactivation, NDM re-activation (i.e. re-negotiation), and upgrading of NDM versions, as this aspect is covered by individual NDMs combined with vendor supplied specifications defining the behavior of individual switch implementations.

It results in the ONF_ET_SET_ACTIVE_NDM_REPLY message type, which uses the following message structure:

```
/* Set active NDM and parameters reply. */
struct onf_set_active_ndm_reply {
    struct ofp_header header;
    uint32_t experimenter;   /* ONF_ODWG_EXPERIMENTER_ID */
    uint32_t exp_type;       /* ONF_ET_SET_ACTIVE_NDM_REPLY */
    uint32_t active_ndm_len; /* Length of following field */
    char     active_ndm[0];  /* NDM ID in double quoted string */
    uint8_t  pad1[0];        /* Pad to 4 byte alignment */
    uint32_t parameters_len; /* Length of following field */
    char     parameters[0];  /* Parameters: key, value... in JSON object (map) */
    uint8_t  pad2[0];        /* Pad to 4 byte alignment */
};
```

The reply message returns the actually activated NDM ID as well as the parameters that are currently in use. Should errors occur, an error message is returned instead.

5 Error Codes

The switch returns an error code for the following conditions:

- NDM not supported
- NDM ID invalid
- NDM cannot be changed
- Parameters bad (name not valid, value out of range or unsupported)

Where error codes are not specifically defined below, implementations must use error codes defined in the OpenFlow specification to report error conditions.

Errors specific to this extension have the following structure:

```
/* Message structure for all NDM errors. */

struct onf_ndm_error_msg {
    struct ofp_header header;
    uint16_t type;         /* OFPET_EXPERIMENTER. */
    uint16_t exp_type;     /* One of the ONF_NDM_ET_* error codes. */
    uint32_t experimenter; /* ONF_ODWG_EXPERIMENTER_ID. */
    uint8_t data[0];       /* Up to 64 bytes of failed request data. */
};
OFP_ASSERT(sizeof(struct onf_ndm_error_msg) ==
           sizeof(struct ofp_error_experimenter_msg));
```

The type field must be set to OFPET_EXPERIMENTER.

The experimenter field is the Experimenter ID (see 3).

The data field contains a copy of the failed request message, truncated to 64 bytes.

The exp_type field is the NDM extension error type. The currently defined NDM extension error types are:


```
/* Error codes */
enum onf_ndm_error_exp_type {
   ONF_NDM_ET_TOO_BIG = 1,              /* Message or field in message too big. */
   ONF_NDM_ET_READ_ONLY = 2,            /* Changes not permitted. */
   ONF_NDM_ET_MSG_UNSUPPORTED = 3,      /* Message not supported. */
   ONF_NDM_ET_NDM_UNSUPPORTED = 4,      /* NDM not supported. */
   ONF_NDM_ET_BAD_NDM_ID = 5,           /* Invalid NDM ID. */
   ONF_NDM_ET_BAD_PARAMETER_NAME = 6,   /* Invalid parameter name. */
   ONF_NDM_ET_BAD_PARAMETER_VALUE = 7,  /* Invalid parameter value. */
};
```

When a REQUEST message is sent by the switch to the controller, or a REPLY message is sent by the controller to the switch, the standard BAD_MSG error code is used.

6 Usage Guidelines

The following sequence of messages is typically used.

1. A controller can optionally at any time issue a `GET_ACTIVE_NDM` query to obtain the currently active NDM ID and the currently active parameters. If a switch has a pre-defined pre-activated NDM that does not need to be specifically manually activated, its identifier is returned. If no specific NDM is active, the special identifier “none” is returned.
2. Negotiation
    a. Controller queries and switch returns list of supported NDMs (IDs only, no parameters) using the `GET_SUPPORTED_NDM` message
        - This is optional - if controller already has list, it need not issue this query
    b. Controller activates a NDM (identified by the NDM ID) with specified parameters using the `SET_ACTIVE_NDM` message.
        - Response is active NDM ID and parameters.
        - If switch returns an error code, refusing activation, controller can retry activation, with a different NDM ID and/or different parameters.
3. Activation of a different NDM and/or adjustment of parameters[1]
    a. Controller activates a NDM (identified by the NDM ID) with specified parameters using the `SET_ACTIVE_NDM` message.
        - If the NDM ID has not been changed, parameters are changed.
        - If the NDM ID has been changed, the NDM is upgraded (version changed) or de-activated and re-activated (completely different NDM ID).
        - Note: Whether or not upgrading is supported, and whether or not upgrading / re-activation can be accomplished without losing state (e.g. table entries / counter values etc.), is implementation dependent.

[1] Support for these capabilities may vary between switch vendors and/or NDMs

4. De-activation of NDM
    a. Controller de-activates the active NDM (if any) by issuing the `SET_ACTIVE_NDM` message with a string containing the identifier "`default`" in the NDM ID field.
        - De-activating the NDM if no NDM is active is permissible, and no error code is returned in this case.

6 Acknowledgements

The following individuals contributed to this document: Curt Beckmann, Joel Halpern, Ben Mack-Crane, Lyndon Ong, Johann Tönsing.

7 Revision History

Date	Revision	Description	Editor
2016-09-08	1.0	Initial version for publication	J H Tönsing

