OPEN NETWORKING FOUNDATION

Flow entry notifications Extension

Version 0.1

December 23, 2014

Disclaimer

Contents

1 Introduction

This document describes an ONF extension for OpenFlow version 1.3.X that allows a controller to monitor changes to OpenFlow flow tables. This "flow monitor" extension provides a way for multiple controllers to keep a coherent view of flow tables even in the presence of unanticipated changes.

2 How it works

1. The controller sends an `ONFST_FLOW_MONITOR` request to begin monitoring flows. The `id` in the request must be unique among all monitors that the controller has started and not yet canceled on this OpenFlow connection.

2. The switch responds with an `ONFST_FLOW_MONITOR` reply. If the request's `flags` included `ONFFMF_INITIAL`, the reply includes all the flows that matched the request at the time of the request (with event `ONFFME_ADDED`). If `flags` did not include `ONFFMF_INITIAL`, the reply is empty.

   The reply uses the `xid` of the request (as do all replies to OpenFlow requests).

3. Whenever a change to a flow table entry matches some outstanding monitor request's criteria and flags, the switch sends a notification to the controller as an additional `ONFST_FLOW_MONITOR` reply with `xid` 0.

   When multiple outstanding monitors match a single change, only a single notification is sent. This merged notification includes the information requested in any of the individual monitors. That is, if any of the matching monitors requests actions (`ONFFMF_ACTIONS`), the notification

includes actions, and if any of the monitors request full changes for the controller's own changes (`ONFFMF_OWN`), the controller's own changes will be included in full.

4. The controller may cancel a monitor with `ONFT_FLOW_MONITOR_CANCEL`. No further notifications will be sent on the basis of the canceled monitor afterward.

3 Buffer Management

OpenFlow messages for flow monitor notifications can overflow the buffer space available to the switch, either temporarily (e.g. due to network conditions slowing OpenFlow traffic) or more permanently (e.g. the sustained rate of flow table change exceeds the network bandwidth between switch and controller).

When the notification buffer space current used for flow monitoring reaches a limiting threshold, the switch reacts as follows:

1. The switch sends an `ONFT_FLOW_MONITOR_PAUSED` message to the controller, following all the already queued notifications. After it receives this message, the controller knows that its view of the flow table, as represented by flow monitor notifications, is incomplete.
2. As long as the notification buffer is not empty:
   - `ONFMFE_ADD` and `ONFFME_MODIFIED` notifications will not be sent.
   - `ONFFME_DELETED` notifications will still be sent, but only for flows that existed before the switch sent `ONFT_FLOW_MONITOR_PAUSED`.
   - `ONFFME_ABBREV` notifications will not be sent. They are treated as the expanded version (and therefore only the `ONFFME_DELETED` components, if any, are sent).
3. When the notification buffer empties, the switch sends `ONFFME_ADD` notifications for flows added since the buffer reached its limit and `ONFFME_MODIFIED` notifications for flows that existed before the limit was reached and changed after the limit was reached.
4. The switch sends an `ONFT_FLOW_MONITOR_RESUMED` message to the controller. After it receives this message, the controller knows that its view of the flow table, as represented by flow monitor notifications, is again complete.

This allows the maximum buffer space requirement for notifications to be bounded by the limit plus the maximum number of supported flows.

4 "Flow Removed" Messages

The flow monitor mechanism is independent of `OFPT_FLOW_REMOVED`. Flow monitor updates for deletion are sent if `ONFFMF_DELETE` is set on a monitor, regardless of whether the `OFPFF_SEND_FLOW_REM` flag was set when the flow was added.

5 Experimenter ID

The Experimenter ID of this extension is:

`ONF_EXPERIMENTER_ID` = 0x4F4E4600

6 Experimenter messages

6.1 Headers

The multipart experimenter messages described in this document should be understood to follow a `struct onf_experimenter_multipart_msg` header, conforming to the OpenFlow specification for experimenter multipart messages:

```
/* Header for experimenter multipart messages. */
struct onf_experimenter_multipart_msg {
    struct ofp_header header;
    uint16_t type;              /* OFPMP_EXPERIMENTER.. */
    uint16_t flags;             /* OFPMPF_REQ_* flags. */
    uint8_t pad[4];
    uint32_t experimenter;      /* ONF_EXPERIMENTER_ID. */
    uint32_t mp_type;           /* One of ONFMP_*. */
};
OFP_ASSERT(sizeof(struct onf_experimenter_multipart_msg) == 24);
```

A single multipart message type is defined:

```
enum onf_flow_monitor_mp_type {
    ONFMP_FLOW_MONITOR = 1870
};
```

The other experimenter messages described in this document should be understood to follow a `struct onf_experimenter_header` header, conforming to the OpenFlow specification for experimenter messages:

```
/* Header for experimenter requests and replies. */
struct onf_experimenter_header {
    struct ofp_header header;
    uint32_t vendor;            /* ONF_EXPERIMENTER_ID. */
    uint32_t subtype;           /* One of ONFT_*. */
};
OFP_ASSERT(sizeof(struct onf_experimenter_header) == 16);
```

The following experimenter message `mp_types` are defined:

```
enum onf_flow_monitor_msg_type {
    ONFT_FLOW_MONITOR_CANCEL = 1870,
    ONFT_FLOW_MONITOR_PAUSED = 1871,
    ONFT_FLOW_MONITOR_RESUMED = 1872
};
```


6.2 Flow monitor requests

The ONFST_FLOW_MONITOR request's body consists of an array of zero or more instances of this structure. The request arranges to monitor the flows that match the specified criteria, which are interpreted in the same way as for ONFST_FLOW.

id identifies a particular monitor for the purpose of allowing it to be canceled later with ONFT_FLOW_MONITOR_CANCEL. id must be unique among existing monitors that have not already been canceled.

The reply includes the initial flow matches for monitors that have the ONFFMF_INITIAL flag set. No single flow will be included in the reply more than once, even if more than one requested monitor matches that flow. The reply will be empty if none of the monitors has ONFFMF_INITIAL set or if none of the monitors initially matches any flows.

For ONFFMF_ADD, an event will be reported if out_port matches against the actions of the flow being added or, for a flow that is replacing an existing flow, if out_port matches against the actions of the flow being replaced. For ONFFMF_DELETE, out_port matches against the actions of a flow being deleted. For ONFFMF_MODIFY, an event will be reported if out_port matches either the old or the new actions.

```
struct onf_flow_monitor_request {
    uint32_t id;                /* Controller-assigned ID for this monitor. */
    uint16_t flags;             /* ONFFMF_*. */
    uint16_t match_len;         /* Length of oxm_fields. */
    uint32_t out_port;          /* Required output port, if not OFPP_NONE. */
    uint8_t table_id;           /* One table's ID or 0xff for all tables. */
    uint8_t zeros[3];           /* Align to 64 bits (must be zero). */
    /* Followed by:
     *   - Exactly match_len (possibly 0) bytes containing the oxm_fields, then
     *   - Exactly (match_len + 7)/8*8 - match_len (between 0 and 7) bytes of
     *     all-zero bytes. */
};
OFP_ASSERT(sizeof(struct onf_flow_monitor_request) == 16);
```

The flags member in struct onf_flow_monitor_request is a combination of bits in enum onf_flow_monitor_flags.

```
/* 'flags' bits in struct onf_flow_monitor_request. */
enum onf_flow_monitor_flags {
    /* When to send updates. */
    ONFFMF_INITIAL = 1 << 0,     /* Initially matching flows. */
    ONFFMF_ADD = 1 << 1,         /* New matching flows as they are added. */
    ONFFMF_DELETE = 1 << 2,      /* Old matching flows as they are removed. */
    ONFFMF_MODIFY = 1 << 3,      /* Matching flows as they are changed. */

    /* What to include in updates. */
    ONFFMF_ACTIONS = 1 << 4,     /* If set, actions are included. */
    ONFFMF_OWN = 1 << 5,         /* If set, include own changes in full. */
};
```


6.3 Flow monitor replies

The body of an `ONFST_FLOW_MONITOR` reply is an array of variable-length structures, each of which begins with this header. The `length` member may be used to traverse the array, and the `event` member may be used to determine the particular structure.

Every instance is a multiple of 8 bytes long.

```
/* ONFST_FLOW_MONITOR reply header. */
struct onf_flow_update_header {
    uint16_t length;            /* Length of this entry. */
    uint16_t event;             /* One of ONFFME_*. */
    /* ...other data depending on 'event'... */
};
OFP_ASSERT(sizeof(struct onf_flow_update_header) == 4);
```

The `event` member takes its value from `enum onf_flow_update_event`:

```
/* 'event' values in struct onf_flow_update_header. */
enum onf_flow_update_event {
    /* struct onf_flow_update_full. */
    ONFFME_ADDED = 0,           /* Flow was added. */
    ONFFME_DELETED = 1,         /* Flow was deleted. */
    ONFFME_MODIFIED = 2,        /* Flow (generally its actions) was changed. */

    /* struct onf_flow_update_abbrev. */
    ONFFME_ABBREV = 3,          /* Abbreviated reply. */
};
```

6.3.1 Full update monitor reply

When `event` in `struct onf_flow_update_header` has the value `ONFFEM_ADDED`, `ONFFME_DELETED`, or `ONFFME_MODIFIED`, the full monitor reply takes the form of `struct onf_flow_update_full`:

```
/* ONFST_FLOW_MONITOR reply for ONFFME_ADDED, ONFFME_DELETED, and
 * ONFFME_MODIFIED. */
struct onf_flow_update_full {
    uint16_t length;            /* Length is 24. */
    uint16_t event;             /* One of ONFFME_*. */
    uint16_t reason;            /* OFPRR_* for ONFFME_DELETED, else zero. */
    uint16_t priority;          /* Priority of the entry. */
    uint16_t idle_timeout;      /* Number of seconds idle before expiration. */
    uint16_t hard_timeout;      /* Number of seconds before expiration. */
    uint16_t match_len;         /* Length of oxm_fields. */
    uint8_t table_id;           /* ID of flow's table. */
    uint8_t pad;                /* Reserved, currently zeroed. */
    uint64_t cookie;            /* Opaque controller-issued identifier. */
    /* Followed by:
     *   - Exactly match_len (possibly 0) bytes containing the oxm_fields, then
     *   - Exactly (match_len + 7)/8*8 - match_len (between 0 and 7) bytes of
     *     all-zero bytes, then
     *   - Instructions to fill out the remainder 'length' bytes (always a
```



```
     *     multiple of 8).  If ONFFMF_ACTIONS was not specified, or 'event' is
     *     ONFFME_DELETED, no actions are included.
     */
};
OFP_ASSERT(sizeof(struct onf_flow_update_full) == 24);
```

6.3.2 Abbreviated update monitor reply

When `event` in `struct onf_flow_update_header` has the value `ONFFME_ABBREV`, the full monitor reply takes the form of `struct onf_flow_update_abbrev`:

```
/* ONFST_FLOW_MONITOR reply for ONFFME_ABBREV. */
struct onf_flow_update_abbrev {
    uint16_t length;            /* Length is 8. */
    uint16_t event;             /* ONFFME_ABBREV. */
    uint32_t xid;               /* Controller-specified xid from flow_mod. */
};
OFP_ASSERT(sizeof(struct onf_flow_update_abbrev) == 8);
```

When the controller does not specify `ONFFMF_OWN` in a monitor request, any flow tables changes due to the controller's own requests (on the same OpenFlow channel) will be abbreviated, when possible, to this form, which simply specifies the `xid` of the OpenFlow request (e.g. an `OFPT_FLOW_MOD`) that caused the change.

Some changes cannot be abbreviated and will be sent in full:

- Changes that only partially succeed. This can happen if, for example, an `OFPT_FLOW_MOD` with type `OFPFC_MODIFY` affects multiple flows, but only some of those modifications succeed (e.g. due to hardware limitations).

  This cannot occur with the current implementation of the Open vSwitch software datapath. It could happen with other datapath implementations. We are considering changes to Open vSwitch that would prevent it from occurring even with other datapath implementations.

- Changes that race with conflicting changes made by other controllers or other `OFPT_FLOW_MOD` commands (not separated by barriers) by the same controller.

  This cannot occur with the current Open vSwitch implementation (regardless of datapath) because Open vSwitch internally serializes potentially conflicting changes.

An `OFPT_FLOW_MOD` that does not change the flow table will not trigger any notification, even an abbreviated one. For example, a "modify" or "delete" that does not match any flows will not trigger a notification. Whether an "add" or "modify" that specifies all the same parameters that a flow already has triggers a notification is switch-specific.

The switch must always send the notifications for a given flow table change before the reply to a `OFPT_BARRIER_REQUEST` request that follows the flow table change. Thus, if the controller does not receive an abbreviated (or unabbreviated) notification for an `OFPT_FLOW_MOD` before the next `OFPT_BARRIER_REPLY`, it will never receive one.

7 History

The extension described in this document differs from the extension implemented in Open vSwitch in a few ways. The existing Open vSwitch extension is implemented on top of OpenFlow 1.0 rather than 1.3. This required a new `struct nx13_flow_monitor_request` to be created with a 32-bit `out_port` and OXM fields. Similarly, `nx_flow_update_full` was defined to contain OXM fields and instructions (instead of NXM fields and actions). Other than this, the only intended changes are terminological, e.g. Open vSwitch uses the term "vendor" instead of "experimenter" and "statistics" instead of "multipart."

This extension was further modified to use the ONF Experimenter IDs, standardised id numbers and use the "onf" prefix for all definitions.