OpenState Specification

Version 0.x

xxx

Contents

1 Introduction

This document describes an extension to the OpenFlow specification v1.x () to enable support to stateful packet forwarding inside OpenFlow-enabled switches. Backward compatibility with OpenFlow is always guaranteed an exisitng elements and primitives are not modified in a way that breaks compatibility.

- Motivation (Do we really need it?)
- Recall OpenFlow match/action flow table -¿ Stateless
- State machine abstraction

2 Glossary

3 Stateful Pipeline

As defined by the OpenFlow specification, a packet entering an OpenFlow switch is processed through a pipeline comprised of a set of linked flow tables that provide matching, forwarding, and packet modification. We indicate with the term *stateless stage* the processing operated by a single stateless OpenFlow's flow table. Conversely, we define as *stateful stage* (Figure 3.1) a logical stage comprising a state table and and a flow table, and implementing our abstraction.

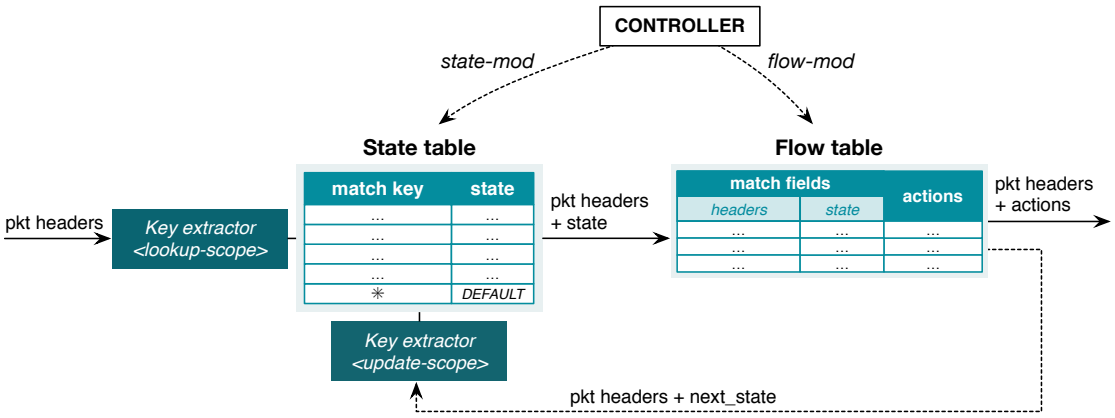

Figure 3.1: Architecture of an OpenState stateful stage

When a packet enters a stateful stage, it is first processed by a *key extractor* which produces a string of bits representing the key to be used to match a row in the state table. The key is derived by concatenating the header fields defined in the *lookup-scope*. The matched state label is appended to the packet headers as an additional header field. By exiting the state table, packet headers along with the returned state label are matched in the flow table.

The flow table is extended by adding support to a new "state" virtual header field to be used to match packets along with other header fields (MPLS, IP, TCP, etc.). We say this header is virtual because it is not really appended to the packet header and it is valid only for current processing trough the flow table of the stateful stage. Because state values are valid only for a specific flow (defined by the flow key), we call them "flow states". Finally, a new "set-state" action is introduced to allow the update of the state value for a given flow in a given stateful stage. The set-state action can be appended to the packet action set as any other OpenFlow action.

By default all the flow tables in the switch are intended as stateless stages, the controller can then enable stateful processing for one or more stages by sending a special control message to the switch and by configuring the key extractors (lookup-scope and update-scope) associated with the state table. Similarly to flow tables, new modification message called "state-mod" has been defined to allow the controller to configure the state entries

and key extractors.

Similarly to flow states, OpenState introduces the concept of “global states”, called also “flags”. As suggested by the name, flags are valid globally for every packet processed by the switch. A controller can specify flags as a match field on the header packet, and it can be seen as a filtering of the flow table. A “set-flags” action is defined to allow the update of flags directly from the pipeline processing.

4 Flow States

4.1 Flow Identification

Flow states are associated with packets and are valid only inside that stateful stage that produced them. Inside a stateful stage, flows can be arbitrary defined by using "flow scopes", which can be seen as the vector of header fields that distinguish one flow from another. For example a Layer 2 flow can be defined by using just the MAC source address and MAC destination address (2 fields), while a flow in the socket sense can be defined by using the whole L2-L4 header (6 fields).

In OpenState states for a given flow can be updated by events occurring on *different* flows. A prominent example is MAC learning: packets are forwarded using the *destination* MAC address, but the forwarding database is updated using the *source* MAC address. Similarly, the handling of bidirectional flows may encounter the same needs; for instance, the detection of a returning TCP SYNACK packet could trigger a state transition on the opposite direction. And in protocols such as FTP, a control exchange on port 21 could be used to set a state on the data transfer session on port 20. For this reason, two types of flow scopes are defined, the "lookup-scope" and the "update-scope", as the ordered sequence of header fields that shall be used to produce the key used to access the state table and perform, respectively, a lookup or an update operation.

The lookup-scope and the update-scope are intrinsic to the state table and are used to configure the key extraction process.

4.2 State Table

Key	State	Timeouts

Table 4.1: Main components of a state entry in the state table

A state table consists of state entries. Each state table entry (see Fig.4.1) contains:

- **Key:** String of bit used to match the packet flow key obtained from the key extractor;
- **State:** value associated with a specific flow key
- **Timeouts:** Maximum amount of time or idle time before the entry is expired by the switch;

The match on the state table is performed using the key extracted using the lookup-scope, and it is performed exactly, in other words wildcards are not allowed. In case of a table-miss (the key is not found) then a *DEFAULT* state is appended to the packet

headers. If the header fields specified by the lookup-scope are not found (e.g. extracting the IP source address when the Ethernet type is not IP), a special state value *NULL* is returned.

4.2.1 Timeouts

If the header fields specified by the update-scope are not found in the packet, the set-state action is not executed.

4.2.2 State Modification Messages

4 different state modification messages are defined by OpenState:

- **Set-lookup-extractor:** allows the controller to set the header fields vector for the lookup-scope of the state table.
- **Set-update-extractor:** allows the controller to set the header fields vector for the update-scope of the state table.
- **Set-flow-state:** allows the controller to add or update a state entry in the state table.
- **Delete-flow-state:** allows the controller to delete a state entry in the state table. This command is equivalent to invoking a set-flow-state command or a set state action 4.3 with `DEFAULT` state.

4.3 Set-state Action

In addition to state modification messages 4.2.2, state transitions can be triggered as a consequence of packet matching in a flow entry. By adding a set-state action to the action set, it is possible to execute state transitions in the same stage or in any other (stateful) stage of the pipeline. Multiple state transitions are allowed by defining more than one set-state action in the action set. OpenFlow allows to execute actions in the group table, so it is possible to perform state transitions from the group table by inserting a set-state action in the action bucket.

When the switch executes a set-state action, the packet header is processed by the update-scope key extractor of the specific state table, the corresponding entry is then updated.

4.3.1 Atomicity

As defined in OpenFlow, actions are usually executed at the end of the pipeline. The same applies for the set-state action, thus making the stateful steps "lookup/update" not atomic by default. Not enforcing atomicity can bring to consistency issues when more than one packet are processed by the pipeline at the same time. The only way to guarantees state consistency between packets is to call the set-state action from the apply-action instruction

(instead of the write-actions instruction) in order to be sure to update the value contained in the state table when exiting a specific stage of the pipeline.

5 Global States

By extending the flow state concept some states could be shared among multiple flows. For this reason global states have been developed. These states (a.k.a. flags) are defined at datapath level and are not related to a single flow of a particular stage. Now each incoming flow's packet can be matched also according to the current value of global states. Global states can be updated c by means of a new action "set-flags" triggered by a match in the flow table. Furthermore, the controller is able to modify and reset global states value of a specific switch exploiting the new flag modification messages.

5.1 Flag Modification Messages

The following types of flag modification messages are defined:

- **Set-flags**: allows the controller to update the value of global states. Values can be totally overwritten or, by using a mask, selectively modified.
- **Reset-flags**: allows the controller to reset global states to the default value.

5.2 Set-flag Action

In addition to flag modification messages 5.1, the global states can be modified as a consequence of packet matching in a flow entry. By adding a set-flag action to the action set, it is possible to modify the global state values in any stage of the pipeline. OpenFlow allows to execute actions in the group table, so it is possible to update global state values from the group table by inserting a set-flag action in the action bucket. Using the set-flag action values can be totally overwritten or, by using a mask, selectively modified.

6 Protocol

6.1 Capability

A new OFPC_OPENSTATE capability has been introduced. If a switch is OpenState-compatible, the OFPC_OPENSTATE capability bit is set, thus enabling the controller to configure a stage of the pipeline as stateful by sending a table feature message [Sec. 6.2].

```
/*
 * OpenState capability if supported by the datapath.
 * To be added to enum ofp_capabilities.
 */
OFPC_OPENSTATE = 1 << 9
```

6.2 Stateful Stage Configuration

If OFPTC_TABLE_STATEFUL bit is set in the table features' config bitmap, right after the packet headers are parsed, the flow state is retrieved and written in the state field, otherwise the packet id directly processed by the flow table.

```
/*
 * Configuration bit for the stateful stage.
 * To be added add to bitmap of OFPTC_* flags)
 */
OFPTC_TABLE_STATEFUL = 1 << 4
```

6.3 State Modification Messages

Modifications to the state table from the controller are done with the OFPT_STATE_MOD message:

```
/*
 * Controller to switch message.
 * To be added to enum ofp_type.
 */
OFPT_STATE_MOD = 30

/*
```

```
 * Structure of the state modification message.
 */
struct ofp_state_mod {
    struct   ofp_header header;
    uint8_t  table_id;
    uint8_t  command;
    uint8_t  payload[];
};

/*
 * Possible command values for ofp_state_mod.
 */
enum ofp_state_mod_command {
    OFPSC_SET_L_EXTRACTOR = 0,
    OFPSC_SET_U_EXTRACTOR,
    OFPSC_SET_FLOW_STATE,
    OFPSC_DEL_FLOW_STATE
 };
```

The `table_id` field specifies the table to be modified, both for setting up the key extractors or to set/delete state entries). The `payload` field structure is defined by `ofp_state_entry` or `ofp_extraction` according to the value of `command` field. The differences between the four commands are explained in section 4.2.2.

An `OFPT_STATE_MOD` message with `command` field set to `OFPSC_SET_L_EXTRACTOR` or `OFPSC_SET_U_EXTRACTOR` must have a `payload` structure as defined by `ofp_extraction`.

```
/*
 * Max number of fields that can be used to compose the key extractor vector
 */
#define OFPSC_MAX_FIELD_COUNT 6

struct ofp_extraction {
    uint32_t field_count;
    uint32_t fields[OFPSC_MAX_FIELD_COUNT];
};
```

The `field_count` field specifies the number of fields provided in `fields[]`, which contains the TLV vector of fields composing the key extractor.

An `OFPT_STATE_MOD` message with `command` field set to `OFPSC_SET_FLOW_STATE` or `OFPSC_DEL_FLOW_STATE` must have a `payload` structure as defined by `ofp_state_entry`.

```
/*
 * Number of bytes composing the state key.
 */
#define OFPSC_MAX_KEY_LEN 48

struct ofp_state_entry {
```

```
    uint32_t key_len;
    uint32_t state;
    uint32_t state_mask;
    uint8_t  key[OFPSC_MAX_KEY_LEN];
};
```

The `key_len` field specifies the key size (number of bytes) of the key provided in `key[]`.

The `state` field contains the state to be inserted (or updated) in the state table. In case `command` field is set to `OFPSC_DEL_FLOW_STATE`, the `state` field can take any value because only `key` field is used to delete the corresponding entry in the state table.

The `state_mask` field specifies which bits of the state should be modified. A `state_mask` with value `0xFFFFFFFFFFFFFFFF` indicates that the `state` field should be entirely overwritten.

The `key` field contains the key used to access the state table, splitted in bytes (e.g: ip 10.0.0.1 is stored as [10,0,0,1]).

Checks and Errors

- When `command` field is set to `OFPSC_SET_L_EXTRACTOR` or `OFPSC_SET_U_EXTRACTOR`, `field_count` must be consistent with the number of fields provided in `fields[]` and should be always greater than 0, otherwise the switch must return an `ofp_error_msg` with `OFPET_BAD_REQUEST` type and `OFPBRC_BAD_LEN` code at message unpack time.
- When `command` field is set to `OFPSC_SET_FLOW_STATE` or `OFPSC_DEL_FLOW_STATE`, `key_count` field must be consistent with the length of the key provided in `key[]` and should be always greater than 0, otherwise the switch must return an `ofp_error_msg` with `OFPET_BAD_REQUEST` type and `OFPBRC_BAD_LEN` code at message unpack time.
- When `command` field is set to `OFPSC_SET_FLOW_STATE` or `OFPSC_DEL_FLOW_STATE`, `key_count` field must be consistent with the number of bytes of the update-scope fields (previously configured with an `OFPSC_SET_U_EXTRACTOR` command). This check is performed at message execution time.
- Lookup-scope and update-scope should provide keys with same length. This check should be performed by the switch when a message with `command` field set to `OFPSC_SET_L_EXTRACTOR` or `OFPSC_SET_U_EXTRACTOR` is received: if the other extractor has already been configured, the new extractor key must have the same length. This check is performed at message execution time.
- When `command` field is set to `OFPSC_SET_FLOW_STATE` or `OFPSC_DEL_FLOW_STATE`, `table_id` field must have a value less or equal than the number of pipeline's tables, otherwise the switch must return an `ofp_error_msg` with `OFPET_BAD_REQUEST` type and `OFPBRC_BAD_TABLE_ID` code. This check is performed at message unpack time (since the number of table is fixed).

6.4 Set-state Action

The OFPAT_SET_STATE action allows to set flow states in a particulare stage of the pipeline. The following structure describes the body of the set-state action:

```
/*
 * New action type.
 * To be added to enum ofp_action_type.
 */
OFPAT_SET_STATE = 28

/*
 * Action structure for OFPAT_SET_STATE.
 */
struct ofp_action_set_state {
    uint16_t type;
    uint16_t len;
    uint32_t state;
    uint32_t state_mask;
    uint8_t table_id;
    uint8_t pad[3];    /* Align to 64-bits. */
};
OFP_ASSERT(sizeof(struct ofp_action_set_state) == 16);
```

The type field should be set to OFPAT_SET_STATE. The len field should be set to 16. The state field is used to set the value to be inserted (or updated) in the state table. The state_mask field specifies which bits of the state field should be modified. A state_mask with value 0xFFFFFFFFFFFFFFFF indicates that the state field must be entirely overwritten. The table_id field specifies the target stage of the state update action.

6.4.1 Checks and Errors

- Set-state action should be called only on stateful stage. This check is performed at action execution time because the flow-mod message with a set-state action could be received by the switch before configuring a stage as stateful. The important thing is that the stage is stateful at action execution time (since the number of table is fixed).
- Set-state action should be performed onto a stage with stage_id less or equal than the number of pipeline's tables, otherwise the switch must return an ofp_error_msg with OFPET_BAD_REQUEST type and OFPBRC_BAD_TABLE_ID code. This check is performed at message unpack time (the number of table is fixed, so installing a flow with a wrong action does not make sense).

6.4.2 Priority

The actions in an action set are applied in the order specified in the OpenFlow specification, regardless of the order that they were added to the set. The new `OFPAT_SET_STATE` action has to be executed with an higher priority with respect to the `OFPAT_SET_FIELD` action. Given an action set containing both a set-field and a set-state action, with this setting we avoid that the set field modifies header fields described in the update-scope before the set-state action execution.

6.5 State Match Field

The `OXM_OF_STATE` field is the field used in the flow table to match on the state value defined in the virtual packet header field returned by a state table in a stateful stage. It is a 32 bit field.

`OXM_OF_STATE` is maskable, so it is possible to match it either exactly or with wildcards. A 0 bit in the mask means i-th state's bit is "do not care", while a 1 bit value means "exact match".

```
/*
 * Flow state field definition (oxm-match.h)
 */
#define OXM_OF_STATE OXM_HEADER     (0x8000, 41, 4)
#define OXM_OF_STATE_W OXM_HEADER_W (0x8000, 41, 4)
```

6.6 Flag modification messages

Modifications to the global states from the controller are done with the `OFPT_FLAG_MOD` message.

```
OFPT_FLAG_MOD = 31,  /* Controller/switch message */

struct ofp_flag_mod {
    struct ofp_header header;
    uint32_t flag;
    uint32_t flag_mask;
    uint8_t command;
    uint8_t pad[7]; /* Pad to 64 bits. */
};
```

The `flag` field specifies the new value of global states. The `flag_mask` field specifies which bits of the global state should be modified. A `flag_mask` with value `0xFFFFFFFFFFFFFFFF` indicates that the global state field should be entirely overwritten.

The `command` field must be one of the following:

```
enum ofp_flag_mod_command {
    OFPSC_MODIFY_FLAGS = 0,
    OFPSC_RESET_FLAGS
};
```

The differences between the two commands are explained in section 5.1.

```
#define OFP_GLOBAL_STATES_DEFAULT 0
```

In case command field is set to `OFPSC_RESET_FLAGS`, both the `flag` field and the `flag_mask` can take any value and global states are reset to the default value defined in `OFP_GLOBAL_STATES_DEFAULT`.

6.7 Set-flag action

The `OFPAT_SET_FLAG` action is used to set flags' value. The following structure describes the body of set-flag action:

```
/*
 * Set a single flag value of the global state.
 * To be added where?
 */
OFPAT_SET_FLAG = 29

/*
 * Action structure for OFPAT_SET_FLAG
 */
struct ofp_action_set_flag {
    uint16_t type;
    uint16_t len;
    uint32_t flag;
    uint32_t flag_mask;
    uint8_t pad[4];   /* Align to 64-bits. */
};
OFP_ASSERT(sizeof(struct ofp_action_set_flag) == 16);
```

The `type` field should be set to `OFPAT_SET_FLAG`. The `len` field should be set to 16. The `flag` field specifies the new value of global states. The `flag_mask` field specifies which bits of the global state should be modified. A `flag_mask` with value `0xFFFFFFFFFFFFFFFF` indicates that the global state field should be entirely overwritten.

6.8 Flags match field

If a switch supports OpenState (capability `OFPC_OPENSTATE`), right after the packet headers are parsed, the global states are retrieved and written in the flags field. `OXM_OF_FLAGS` is

a field with mask, so it is possible to match it either exactly or with wildcards. A 0 bit in the mask means i-th flags value is "do not care", while a 1 bit value means "exact match".

```
/* Global States */
#define OXM_OF_FLAGS OXM_HEADER     (0x8000, 40, 4)
#define OXM_OF_FLAGS_W OXM_HEADER_W (0x8000, 40, 4)
```

Example match:

```
flags=(4,5)
```

This command allows to match over **********1*0 flags configuration (4 in binary is 100 and the mask 5 is 101 that is exact match on LSB 1 (0 value) and LSB 3 (1 value) and "don't care" over all the other flags. In order to perform an exact match on flags value no mask is required.

Example match:

```
flags=4
```

NB: this match is very differend from the previous one. With this command we are matching over 00000000000000000000000000000100 flags configuration, so it is an exact match.

Credits

Spec contributions, in alphabetical order: Giuseppe Bianchi, Marco Bonola, Antonio Capone, Carmelo Cascone, Luca Pollini, Davide Sanvito