Description:

Every flow will be assigned an OpenFlow meter by the SDN controller during flow installation phase. This assignment is done by our protocol implemented in Open Daylight when triggered by Packet-In message. In the SDN switch a “meter_id” uniquely identifies a meter within switch local scope. We associate one of these “meter_id” with every flow-entry.

OpenFlow meters use meter bands to define behavior of meters on packets. This behavior is controlled by meter band type. Presently three band types are supported.

1) OFPMBT_DROP
2) OFPMBT_DSCP_REMARK
3) OFPMBT_EXPERIMENTER

We use the third type (i.e. OFPMBT_EXPERIMENTER) to define a band-rate-evaluator. The main components of OpenFlow meter bands are type, rate, burst and counters. We extend the meter band to add new components: superior constraint meter list, inferior constraint meter list and guaranteed rate.

The superior constraint meter list is a list of “meter_id” that has higher priority than this meter.
The inferior constraint meter list contains all meters except in superior constraint meter list and this meter.

The guaranteed rate specifies the bandwidth guarantee for this meter.

The above values can be specified in KBPS or packets per second as set in meter flags.

In the proposed architecture, a meter associated with a flow-entry is applied to every incoming packet of the flow at the ingress port. This is shown in Figure 1. The meter then evaluates the band rate at which it should drop the packet. If the evaluated band rate is greater than present rate, packet is passed, otherwise it is dropped.

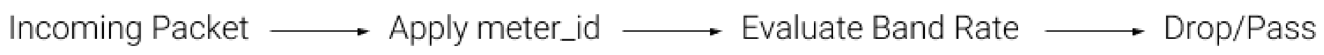

Figure 1: Applying meter band-rate-evaluator to every packet

We illustrate the architecture of the proposed protocol by an example. As shown in Figure 2, the flow-entries are associated with meters. Flow 1 is associated with Meter 1 having meter id “meter_id1” and Flow 2 is associated with Meter 2 having meter id “meter_id2”.

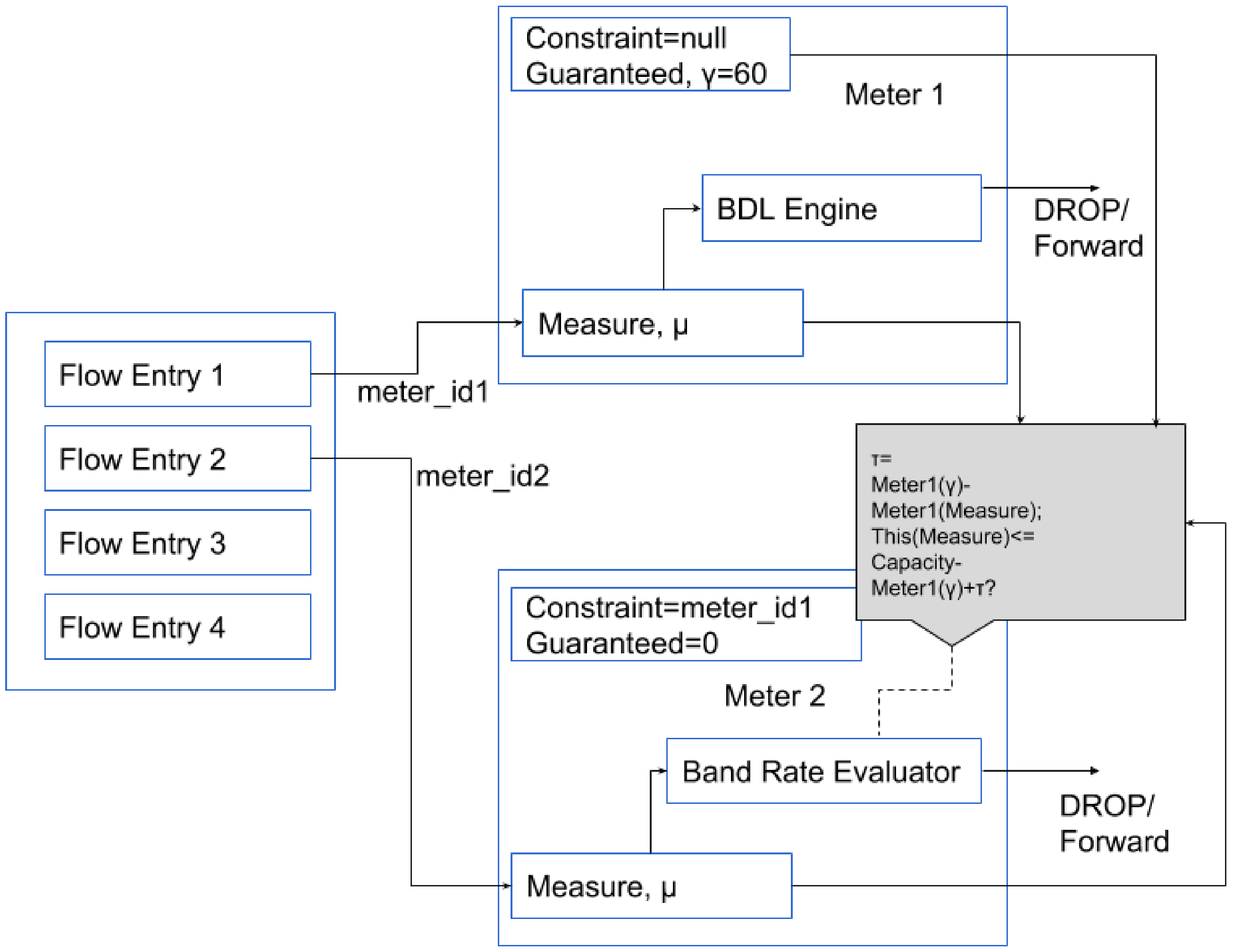

Figure 2: Illustration of the proposed architecture

The evaluation process is done using the following process shown in Figure 3.

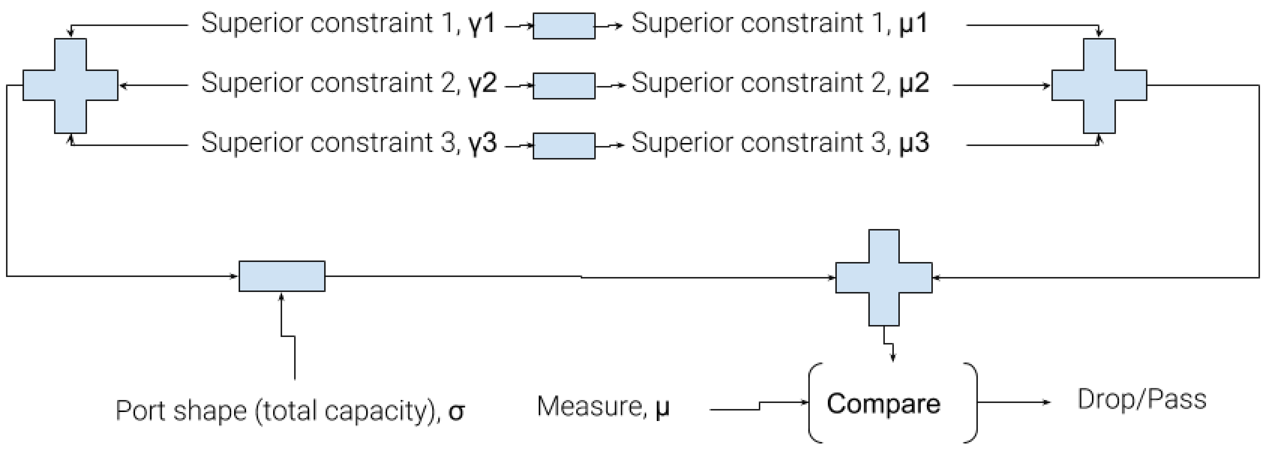
Superior constraint 1, γ1
Superior constraint 1, µ1
Superior constraint 2, γ2
Superior constraint 2, µ2
Superior constraint 3, γ3
Superior constraint 3, µ3
Port shape (total capacity), σ
Measure, µ
Compare
Drop/Pass

Figure 3: Proposed Band Rate Evaluator

Implementation:

Meter Action has the following structure.

```
/* Action structure for OFPAT_METER */
struct ofp_action_meter {
        uint16_t type;      /* OFPAT_METER */
        uint16_t len;
        uint32_t meter_id;
};
```

By meter_id, designated meter is specified for the packet.

Experimenter Structure has the following structure.

```
/* Typical Experimenter structure. */
struct ofp_experimenter_structure {
        uint32_t experimenter;
        uint32_t exp_type;
        uint8_t experimenter_data[0]; * Here we define the additional meter band
components.*
};
```

Meter Modification Message has the following structure.

```
/* Meter configuration. OFPT_METER_MOD. */
struct ofp_meter_mod {
        struct ofp_header header;
        uint16_t command;           /* One of OFPMC_*. */
        uint16_t flags;             /* Bitmap of OFPMF_* flags. */
        uint32_t meter_id;          /* Meter instance. */
        struct ofp_meter_band_header bands[0];    /*    The band list length is
                                                        inferred from the length field
                                                        in the header. */
};
```

meter_id (1 to max_supported meters) uniquely identifies a meter within a switch. Additional virtual meters are also supported. We will use per-flow meters instead of virtual meters in our protocol.

Meter Numbering can be as follows.

```
enum ofp_meter {
        /* Last usable meter. */
        OFPM_MAX = 0xffff0000,
        /* Virtual meters. */
        OFPM_SLOWPATH = 0xfffffffd,          /* Meter for slow datapath. */
        OFPM_CONTROLLER = 0xfffffffe,        /* Meter for controller connection. */
        OFPM_ALL = 0xffffffff,               /* Represents all meters for stat requests
                                                    commands. */
};
```

Virtual Meters are for existing implementations. New implementations are encouraged to use regular per-flow meters instead of virtual meters.

OFPM_CONTROLLER:	For controlling packets sent to controller via Packet-In message.
OFPM_SLOWPATH:	For controlling packets processed by slot datapath.

The command specified in meter modification must be one of the following.

```
/* Meter commands */
enum ofp_meter_mod_command {
        OFPMC_ADD= 0,               /* New meter. */
        OFPMC_MODIFY = 1,           /* Modify specified meter. */
        OFPMC_DELETE = 2,           /* Delete specified meter. */
};
```

Flags in the meter modification message can be from the following.

```
/* Meter configuration flags
enum ofp_meter_flags {
        OFPMF_KBPS = 1 <<0,         /*Rate value in kb/s (kilo-bit per second). */
        OFPMF_PKTPS = 1 <<1,        /*Rate value in packet/sec. */
        OFPMF_BURST = 1 <<2 ,       /*Do burst size. */
        OFPMF_STATS= 1 <<3          /*Collect statistics. */
};
```

Meter bands are defined using the following header.

```
/* Common header for all meter bands */
struct ofp_meter_band_header {
        uint16_t type;          /* One of OFPMBT_*. */
        uint16_t len;           /* Length in bytes of this band. */
        uint32_t rate;          /* Rate for this band. */
        uint32_t burst_size;    /* Size of bursts. */
};
```

The type field is one of the following.

```
/* Meter band types */
enum ofp_meter_band_type {
        OFPMBT_DROP = 1,
        OFPMBT_DSCP_REMARK = 2,
        OFPMBT_EXPERIMENTER = 0xFFFF * Used by the proposed protocol *
};
```