15/08/2015

**Match Fileds**

Ingress Port 32  
Metadata 64  
MAC Src\* 48  
MAC Dst\* 48  
Eth Type 16  
VLAN ID\* 12  
VLAN Priority\* 3  
MPLS Label\* 20  
MPLS Traffic Cls\* 3  
IPv4 Src \* 32  
IPv4 Dst\* 32  
IPv4 Protocol 8  
IPv4 ToS\* 6  
TCP Src\* 16  
TCP Dst\* 16  
 ----  
 356

Bits are allocated in the Match Field array in the order given above. Bit no. 357 is reserved for pointing whether that entry is the start of a new mask group or not, as a single mask can have many flow entries (This is memory conserving, but not sure whether this is the actual case). If bit no. 357 is set, then the entry is the starting entry of a new mask group.

Bit 358, 359, 360 are not used for any purpose. That gives 45 bytes.

Byte 46 is for storing the priority of the flow entry, byte 47 for counter and 48 for timeout.

So, a flow entry in the flow table is 48 bytes long. A mask is only 45 bytes long, as priority, counter and timeout fields are not checked when matching a flow.

**Action Set & Flag**

For the time being, only actions of type Drop, Forward and Set Field are considered. Fields indicated with ‘\*’ above are the fields that can be set to a new value. They total up to 236 bits. So 30 bytes are needed for storing the new values of the fields (Last 4 bits of this byte array are not used for anything). Another byte is added to provide the output port.

So, an action set (or action value set) is 31 bytes long. Space is allocated for fields in the order they are placed in the above list (May need to change the order to the order these fields appear on the header of a TCP/IP packet).

In the action flag, 11 bits are needed to point whether a specific field in the packet will be written a new value, as there are 11 fields that can be set. Another bit is used to indicate whether the packet should be dropped or not. So, 12 bits, 2 bytes, are needed for the flag. First bit is for Drop action, next 11 for set action and the remaining 4 (last 4) are not used for anything.

**Matching a Flow**

It is assumed that a flow (preprocessed and made into format) will be 45 bytes long, and in the order specified above. It is possible for a single flow to match with more than one different flow entries, with different masks (network-wide flow entry and specific flow entry, may be for an ACL). So each flow is compared with every single entry in the flow table and out of the matching entries, the action set of the entry with highest priority is the one that will be applied.

*Note: I feel that the entry that has a mask with the lowest number of don’t care bits should be the one with the highest priority (as the set that will match into it is a subset of the set of an entry with a larger number of don’t cares). If we structure the tables like this, it is possible to make the matching faster, as the first entry that is matched will be the one with highest priority. But not sure about this, so I did not implement this in the code.*

Searching flow entries will provide us with the index of the action value set and flag.

**Adding a Flow**

For adding a flow, mask, match field set (with priority, and maybe timeout although that is not implemented in the code), action value set and flag should be given as input.

First the mask and match field set are added to the table, as follows.

Does mask already exist?  
 Yes -> Does the match field set match with any other flow in the mask’s group?  
 Yes -> Entry is duplicate, flow entry not added (This may change, as priorities can be diff)  
 No -> Add the entry to that mask’s group  
 May need to restructure tables  
 No -> Add flow entry and mask to the first available slot in the tables

Then the action value set and flag are added to the correct position (same position as that of the match field set, as we are having one action set for each flow), which will be provided by the function that adds the match field set and mask.

**Modifying a Flow**

*Give the exact same match field set and new action values and a flag (action values calculated by the processor). Locate the flow entry and change the fields of the action value set as needed.*

*This is not yet implemented, so if the approach just described is not suitable, please provide me with a better approach ASAP.*

**Deleting a Flow**

*Not yet implemented.*

16/08/2015

**Modifying a Flow**

For modifying a flow, it is assumed that the flow that needs modification is the matching flow with highest priority. In the code, modification is implemented in that way.

For modifying, apart from the match fields, a flag and the new action values (in a byte array) should be given. The new action values array holds only the new values; it is not of the same size as an action set.

Flag is a byte array of size 2. First bit is for indicating ‘Drop’ action. Next 11 bits are for set fields. If a certain set field is to be given a new value, bit of that field will be set in the provided flag, and we should extract that new value from the action values array. Bit 13 is for changing output port, 14 for priority of the flow entry, 15 for counter (not sure whether we need this) and 16 for time out. It should be noted that if last 3 bits are set, modification should be done to the flow entry, not to the action value set.

**Deleting a Flow**

I came up with 3 deletion cases.

Case 1: The entry to be deleted is the only one in its mask group, in which case mask should also be deleted.

Case 2: Entry is the first (not the only) entry in its mask group. After deleting the entry, should set the mask indicating bit of its successor.

Case 3: Normal Case

After deleting, tables need to be restructured and both deletion (over writing) and restructuring is done in one go.

In deleting as well, flow entry, or rather match fields, should be provided and the same thing as in modifying is assumed.

*Modifications: Some slight modifications were done to the tcam structure and ‘flow\_entry.h’ file. Modifications are mainly with action flags. Earlier it was a 16 bit integer, now it is an array of 2 bytes. Some more methods (to handle deletion and modification) were added to ‘flow\_entry.h’.*