Equivalent Forwarding Set Evaluation in Software Defined Networking

Liang Yang, Bryan Ng, Winston K.G. Seah, Lindsay Groves
School of Engineering and Computer Science, Victoria University of Wellington, New Zealand
{ liang.yang, bryan.ng, winston.seah, lindsay }@ecs.vuw.ac.nz

***Abstract*—Network devices rely on forwarding rules to forward packets. With the latest OpenFlow standard (the de-facto software defined network standard), the rules that make up a forwarding set may be expressed in the form of a single table or multiple linked tables. This raises the question of how to reconcile the forwarding behaviour of different flow tables? In this paper we propose two approaches to convert various forwarding sets into a canonical form called *equivalent forwarding set* and formalise the process of evaluating equivalence between two forwarding sets in terms of networking function.**

***Index Terms*—Software Defined Networking (SDN), Equivalent Forwarding Set (EFS), Multiple Flow Tables (MFT)**

I. Introduction

In software defined networking (SDN), forwarding sets are presented in the form of a single table or chained multiple tables to enable more efficient packets processing. The OpenFlow standard specifies the structure of a multiple-flow-table (MFT) forwarding pipeline which is illustrated in Fig. 1 [1]. MFT is one of the most significant enhancements between OpenFlow 1.1 and OpenFlow 1.0, which adds power and flexibility to an OpenFlow switch [2]. MFT allows a packet being proceed by more than one table and it has been studied in [3] and [4]; both works were concerned with the conversion between different types of forwarding tables, which motivates the underpinnings of this paper, i.e., guaranteeing the correctness in terms of networking functionality upon conversions between a single table and MFT.

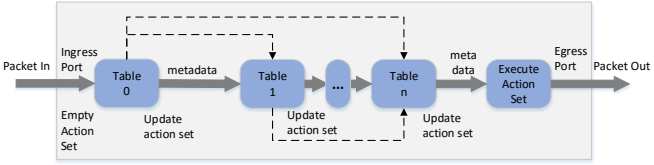

Fig. 1. Multi-table Forwarding Set Specified by OpenFlow

We have found the OTN (the conversion of an One-stage flow table into N-stage flow tables) proposed in [4] is NOT always correct. Its core idea is to populate the corresponding match field value of a one-stage flow entry to multi-stage entries based on their contained match field types, as illustrated in Fig. 2. In this conversion, the original single table (ST0) has been converted into a multi-table structure (MFT0 and MFT1). Take the first entry in ST0 as an example, the original match fields {A, B} are decomposed and distributed into {A} in MFT0 and {B} in MFT1, respectively. However, closer analysis reveals that packets matching either {A, B} or {C, D} will behave the same, while packets with {A, D} or {C, B} will not. Packets with attributes {A, D} do NOT match any entries in the single table and will be dropped by default; however, in the converted multiple flow tables, the same packets can match the first entry in MFT0 and then the second entry in MFT1 sequentially, which means the matching packets will be forwarded to egress port 2. Thus for the same packets, the forwarding behaviour will not always be identical, and thus the single table is not equivalent to the multiple flow table. This flaw can be easily detected and avoided by the approaches we propose in this paper.

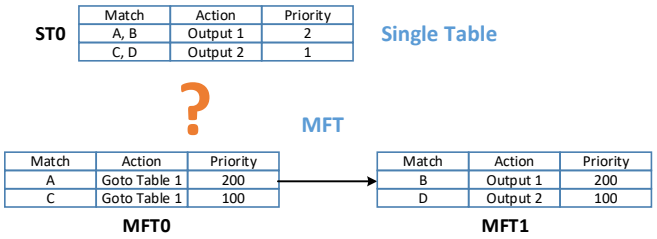

Fig. 2. Match Fields distribution of OTN conversion proposed by [4]

The motivation of this paper lies in the fact that forwarding sets have different representations across various networking elements. The table structure and size in hardware switches vary significantly due to different hardware's specification. In an SDN controller, a logical single table representation is used to abstract the intricate implementation/pipeline details of a hardware switch. For any transformation of a forwarding set among these various networking elements, their functionality equivalence must be guaranteed. This paper proposes two approaches for evaluating the equivalence of any two forwarding sets. The generic principle is to convert different types of forwarding sets into a canonical form and then compare them with the help of boolean reasoning.

II. Equivalent Forwarding Set Conversion

Two forwarding sets F_g and F_h are considered as equivalent forwarding sets if they perform the exact same operations for all incoming packets P. Here the operations include any modification on the packets and their output interface(s).

In this section, we will discuss the conversions from a forwarding set into an equivalent canonical form with the

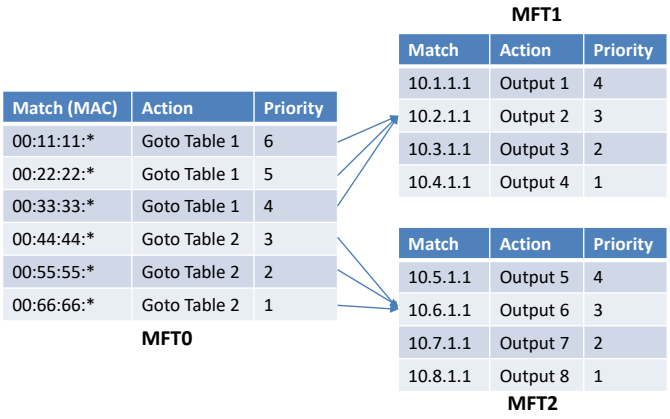

Fig. 3. A Multi-table Forwarding Example. Forwarding rules have Match, Action and Priority fields and the contents of these fields are called attributes. Flow entries match packets in priority order, with the first matching entry in each table being used.

same multi-table forwarding example in Fig. 3. The first row "00:11:11:*, Goto Table 1, 6" in MFT0 is a flow table entry which is associated with MFT1 via the action attribute "Goto Table 1". In Fig. 3, this association relationship is represented by the link between the individual entry and its associated table. Two approaches are proposed to achieve the equivalent canonical form: *match-field oriented approach (MFA)*, which builds indexing on match-fields; and *action oriented approach (ACA)*, which constructs its entries based on actions.

1) MFA Conversion: In match fields oriented approach, a forwarding entry in one table and each entry in its associated tables will generate a new composite entry. This new entry preserves the functionality of the original entries. Thus, the link between two tables can be removed once we replace the original entry and the link with an equivalent composite entry. Algorithm 1 describes the conversion process which removes one table per step until a composite single table is achieved.

Take the MFT set in Fig. 3 as an example, MFT0 (with its entries) has two associated tables: MFT1 and MFT2. Each entry in MFT0 generates a new entry for every entry in its associated table. Thus, the equivalent single table will have 24 entries which is far higher than the sum of the entries in the original three tables.

Once a MFT set has been converted into a single-table set, the equivalence evaluation of any two sets turns to a boolean comparison between two individual tables. Algorithm 2 presents a comparison of two tables based on a strategy in which all items of one table will be successively eliminated. For two given tables T_1 and T_2, every item $T_1[i]$ in T_1 will be compared with all the items in T_2 in the decreasing order of priority. If two items share same match fields while their actions differ, we can conclude these two tables are not equivalent (Lines 8-11). For those two items with the same action, the common match fields will be removed from both items (Lines 19-28). This process is repeated until the match fields in the item $T_1[i]$ yields an empty set, which means $T_1[i]$ is subsumed by T_2. Otherwise, these two sets are not

Algorithm 1 Match Fields Oriented Table Join

1: $T \leftarrow$ input: An array of all the tables in a switch
2: $S \leftarrow$ output: Equivalent Single Table
3: *MAX_PRI* $\leftarrow$ constant: Maximum value of priority
4: *.cnt* $\leftarrow$ Properties: Count of an array
5: *.m, .act* $\leftarrow$ Properties: Match fields and actions of an entry
6: **procedure** $Multi\_table\_join(T)$
7: $n \leftarrow T.cnt$
8: **for** $i = 0$ *to* $n-1$ **do**
9: **for** $ek \in T_i$ **do**
10: **if** $ek.goto\_table\_id = n-1$ **then**
11: **for** $el \in T_{n-1}$ **do**
12: $e.m \leftarrow ek.m \wedge el.m$
13: $e.act \leftarrow ek.act \cup el.act$
14: remove $ek.goto\_table\_id$ in $e.act$
15: $e.pri \leftarrow ek.pri + el.pri/MAX\_PRI$
16: insert e into T_i
17: **end for**
18: remove ek from T_i
19: Convert all priorities to integer in T_i
20: **end if**
21: **end for**
22: **end for**
23: remove T_{n-1} from T
24: **if** $T.cnt = 1$ **then**
25: $S \leftarrow T$
26: **else**
27: $Multi\_table\_join(T)$
28: **end if**
29: **end procedure**

Note: The $\wedge$ in Line 12 is an "AND"-alike operation. The result of $\wedge$ operation on two match files yield a new match field which means the incoming packet must match both of them.

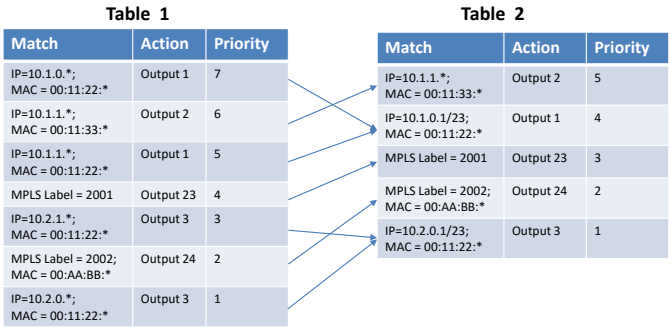

Fig. 4. Successive Elimination Comparison

equivalent (Lines 30-31). Figure. 4 illustrates a successive elimination comparison example of Algorithm 2.

2) ACA Conversion: Action oriented forwarding set is organized as a "dictionary" to store the "action(s)" as a key and "match fields" as its associated value, respectively. If two forwarding sets can be transformed into two dictionaries in which all keys and associated values are identical, they are equivalent.

Algorithm 2 Two-tables-comparison

$T_1, T_2 \leftarrow$ input: Arrays of two signle tables
True, False $\leftarrow$ output: Equivalence of T_1, T_2
.cnt $\leftarrow$ Properties: Count of an array
.m, .act $\leftarrow$ Match fields and actions of an entry
procedure $Table\_comparison(T_1, T_2)$
  for $i = 0$ *to* $T_1.cnt - 1$ **do**
    for $j = 0$ *to* $T_2.cnt - 1$ **do**
      if $T_1[i].act \neq T_2[j].act$ **then**
        if $T_1[i].m \wedge T_2[j].m \neq \emptyset$ **then**
          return False
        end if
      else
        if $T_1[i].m = T_2[j].m$ **then**
          if $(i = T_1.cnt - 1) \& (T_2.cnt = 1)$ **then**
            return True
          end if
          $T_2.RemoveAt(j)$
          break
        else if $T_1[i].m \subset T_2[j].m$ **then**
          $T_2[j].m \leftarrow T_2[j].m \wedge (\neg T_1[i].m)$
          break
        else if $T_2[j].m \subset T_1[i].m$ **then**
          $T_1[i].m \leftarrow T_1[i].m \wedge (\neg T_2[j].m)$
          $T_2.RemoveAt(j)$
        else if $T_1[i].m \wedge T_2[j].m \neq \emptyset$ **then**
          $T_1[i].m \leftarrow T_1[i].m \wedge (\neg T_2[j].m)$
          $T_2[j].m \leftarrow T_2[j].m \wedge (\neg T_1[i].m)$
        end if
      end if
      if $j = T_2.cnt - 1$ **then**
        return False
      end if
    end for
  end for
end procedure

If multiple items share the same action, their respective match fields must be merged together as the value for this action, in this case, an *and(∧)* operation will be performed on all match fields for the same action. However, the match fields in one table are not independent due to the "priority" attribute. If we reorganize a forwarding set into a dictionary-based structure, the "priority" must be removed. Fig. 5 illustrates the deprioritisation process (removing the priority field while preserving the same forwarding functionality) with a five-entries example. In Table 1 (the original table) of Fig. 5 , each entry is associated with a priority which determines the matching sequence of an incoming packet. After adding an "unmatch" property, the dependences among these items are removed, which means their positions in the table are exchangeable and forwarding functionality is unchanged.

Algorithm 3 depicts the process of removing priority and merging match fields while preserving forwarding functionality. In this algorithm, the process of deprioritisation has been

Table 1

Match	Action	Priority
M1_1, M1_2	A1	100
M2_1	A2	90
M3_1, M3_2, M3_3	A1	80
M4_1, M4_2	A4	70
*	A2	0

Table 2

Match	Unmatch	Action	Priority
M1_1, M1_2	NULL	A1	100
M2_1	M1_1, M1_2	A2	90
M3_1, M3_2, M3_3	M1_1, M1_2, M2_1	A1	80
M4_1, M4_2	M1_1, M1_2, M2_1, M3_1, M3_2, M_3_3	A4	70
*	M1_1, M1_2, M2_1, M3_1, M3_2, M_3_3, M4_1, M4_2	A2	0

Fig. 5. Single Table Preprocessing: Deprioritisation

transformed to the set *intersection* operation between each entry's match fields and the *complement* operation of all the match fields with higher priorities.

Algorithm 3 Single Table Preprocessing: Deprioritisation and Merging

$t \leftarrow$ input: Sorted Table based on Priority
$e \leftarrow$ input: Forwarding table entries in Table T
$d \leftarrow$ output: Equivalent Forwarding Dictionary Set
.cnt $\leftarrow$ Properties: Count of an array
.m, .um, .act $\leftarrow$ Match Fields, Unmatch Fields, Actions
procedure $Single\_table\_Preprocessing(T)$
  $e_0.um \leftarrow \emptyset$
  $d.key[e_0.act] \leftarrow e_0.m$
  for $i = 1$ *to* $T.count - 1$ **do**
    $e_i.um \leftarrow e_{i-1}.m \vee e_{i-1}.um$
    $d.key[e_i.act] \leftarrow e_i.m \wedge \overline{e_i.um}$
  end for
end procedure

The core idea of converting a MFT into a dictionary is achieved by a mapping from the linked table structure to a trie in which values are associated with leaf nodes while inner nodes are corresponding to keys of interest. In a MFT converted trie, an edge represents a match field and a leaf node represents an action which can only be reached by the packets matching all the fields along the path from root to that leaf node. In the example illustrated by Fig. 6, there are four different actions: $\{A_{1\_1}, A_{1\_2}, A_{2\_1}, A_{2\_2}\}$, they are the "keys" in the converted forwarding sets while their associated match fields are determined by the fields in each respective path, which are the composite attributes: $\{M_{0\_1}, M_{1\_1}\}$, $\{M_{0\_1}, M_{1\_2}\}$, $\{M_{0\_2}, M_{2\_1}\}$ and $\{M_{0\_2}, M_{2\_2}\}$.

The process is detailed in Algorithm 4 which uses a typical breadth-first search strategy. All entries in a table are iterated and the "goto-table-id" in these entries will be removed by the generation of new entries. Each new entry is a combination of the original entry and all entries in its associated table. Finally all the "goto-table-id" will be removed, which means the remaining action(s) are independent and can be considered as the "key" in the final dictionary and the match fields in these new generated entries will become their corresponding "value".

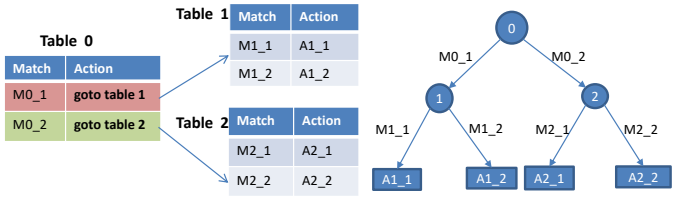

Fig. 6. Multi-table Trie Conversion

Algorithm 4 Multi-table Trie Traversal

```
1: T ← input: An array of all tables in a switch
2: D ← output: EFS Dictionary [key, value] - [action, match]
3: .cnt ← Properties: Count of an array
4: .m, .act ← Match fields and actions of an entry
5: procedure Multi_table_traversal(T, D)
6:     for i to T.cnt − 1 do
7:         for ek ∈ T[i] do
8:             if goto_table_id ∉ ek.act then
9:                 D[ek.act] ← ek.m ∪ D[ek.act]
10:            else
11:                j ← ek.goto_table_id
12:                for el ∈ T[j] do
13:                    e.m ← el.m ∧ ek.m
14:                    e.act ← el.act ∪ ek.act
15:                    remove goto in ek from el.act
16:                    insert e into T[j]
17:                    remove el from T[j]
18:                end for
19:            end if
20:        end for
21:    end for
22: end procedure
```

3) MFA vs ACA: The concept of EFS means that a same packet traversing along two EFSs will have the same forwarding behaviour, which can be explained from two perspectives: i) identical match fields must trigger a unique action; ii) a unique action must be associated with same match fields.

Both MFA and ACA are built on the idea of comparing match fields. The difference lies in that all the match fields of the same action in ACA are combined together while MFA preserves the original match fields. Take the multi-table forwarding set in Fig. 3 as an example, there are total 14 entries in all three tables; However, as shown in Fig. 7, the combinations of all the potential match fields based on their association relationship among these three tables will have 24 entries. Contrary to this, as shown in Fig. 8, the number of entries in ACA is reduced to only eight due to the limited actions (the standard allows for only four predefined action types).

Usually the number of entries in the converted single table by MFA is far larger than ACA; however, the average width of the match fields in ACA is greater than MFA because ACA combines all match fields into a new one for the same action(s) while MFA still maintains original match fields.

Match	Action	Priority
00:11:11:*, 10.1.1.1	Output 1	
00:22:22:*, 10.1.1.1	Output 2	
00:33:33:*, 10.1.1.1	Output 3	
00:11:11:*,10.2.1.1	Output 4	
00:22:11:*,10.2.1.1	Output 1	
00:33:33:*,10.2.1.1	Output 2	
...		
00.55:55:*,10.5.1.1.	Output 5	
00:55:55:*,10.6.1.1	Output 6	
00:55:55:*,10.7.1.1	Output 7	
00:55:55:*,10.8.1.1	Output 8	
...		

Total number of flow table entries: 3*4+3*4=24

Fig. 7. Match-fields Oriented Conversion Approach (MFA). This yields a "tall" table.

Match	Action	Priority
(00:11:11:*, 10.1.1.1) V (00:22:22:*, 10.1.1.1) V (00:33:33:*, 10.1.1.1)	Output 1	
(00:11:11:*, 10.2.1.1) V (00:22:22:*, 10.2.1.1) V (00:33:33:*, 10.2.1.1)	Output 2	
(00:11:11:*, 10.3.1.1) V (00:22:22:*, 10.3.1.1) V (00:33:33:*, 10.3.1.1)	Output 3	
(00:11:11:*, 10.4.1.1) V (00:22:22:*, 10.4.1.1) V (00:33:33:*, 10.4.1.1)	Output 4	
(00:44:44:*, 10.5.1.1) V (00:55:55:*, 10.5.1.1) V (00:66:66:*, 10.5.1.1)	Output 5	
(00:44:44:*, 10.6.1.1) V (00:55:55:*, 10.6.1.1) V (00:66:66:*, 10.6.1.1)	Output 6	
(00:44:44:*, 10.7.1.1) V (00:55:55:*, 10.7.1.1) V (00:66:66:*, 10.7.1.1)	Output 7	
(00:44:44:*, 10.8.1.1) V (00:55:55:*, 10.8.1.1) V (00:66:66:*, 10.8.1.1)	Output 8	

Total number of flow table entries: 8

Fig. 8. Action Oriented Conversion Approach (ACA). This yields a "fat" table.

III. Conclusion

With the introduction of SDN, a forwarding set can be presented as a single table or multiple linked tables. Two algorithms namely ACA and MFA are proposed in this paper to convert a multi-table forwarding set into a canonical form called EFS. Our future work includes the implementation and performance evaluation of the comparison between two EFSs with the help of boolean functions.

References

[1] Open Networking Foundation, "Openflow switch specification, version 1.5. 1," December 19, 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf

[2] Open Networking Foundation, "The benefits of multiple flow tables and TTPs, version number 1.0," February 02, 2015. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_Multiple_Flow_Tables_and_TTPs.pdf

[3] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 99–110.

[4] H. Pan, H. Guan, J. Liu, W. Ding, C. Lin, and G. Xie, "The flowadapter: Enable flexible multi-table processing on legacy hardware," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 85–90.