

# Optimizing Service Chain ID Generation for Flow Rule Compression

Om Prakash Nirankari

A Thesis Submitted to  
Indian Institute of Technology Hyderabad  
In Partial Fulfillment of the Requirements for  
The Degree of Master of Technology

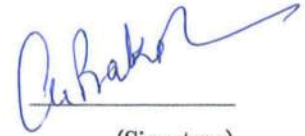


Department of Computer Science and Engineering

June 2016

## Declaration

I declare that this written submission represents my ideas in my own words, and where ideas or words of others have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources that have thus not been properly cited, or from whom proper permission has not been taken when needed.



(Signature)

---


(Om Prakash Nirankari)

cs14mtech11011

(Roll No.)


## Approval Sheet

This Thesis entitled Optimizing Service Chain ID Generation for Flow Rule Compression by Om Prakash Nirankari is approved for the degree of Master of Technology from IIT Hyderabad

  
(.....T. Bheemadurga Reddy.....) Internal Examiner  
Department of Computer Science and Engineering  
IITH

(.....) External Examiner  
.....  
.....

  
(Dr. Kotaro Kataoka) Adviser  
Department of Computer Science and Engineering  
IITH

  
(.....ANTONY.....FRANKLIN.....) Chairman  
Department of Computer Science and Engineering  
IITH

## Acknowledgements

First of all, I would like to express my sincere gratitude to my guide Dr. Kotaro Kataoka for his guidance, valuable feedback and immense knowledge. Without his constant motivation, support and encouragement, I would not have been able to write this thesis.

I would also like to thank all my fellow lab mates for the valuable discussions and fun we had. I am also grateful to Mr. Prakash Pawar who helped in my thesis work. Special thanks to Mr. Uttam Dhabas, Mr. Sandeep R. B. and Mr. Naman Grover.

Finally, I would like to thank friends and family for providing me with unfailing support and continuous encouragement.

## Abstract

Service Chain define the order in which a packet will go through middleboxes. Service Chaining provides opportunities for network and service providers to implement their services and policies with finer granularity of an individual user or an application. Software defined Networking (SDN) provides programmable and flexible way for implementing Service Chaining. SDN switches uses Ternary Content Addressable Memory (TCAM) for storing flow rules that decides the forwarding actions. A significant amount of research has been done on Service Chaining implementation but not satisfactory in terms of TCAM exhaustion, scalability and demand modification in middlebox software.

However, the increasing number of Service Chains and middleboxes will introduce a larger number of flow rules and more consumption of TCAM, whose capacity is limited due to high cost and power consumption. The flow compression techniques have been proposed for reducing the TCAM consumption but they are not applicable to flow rules related to service chaining related flow rules. Service Chain (SC-ID) is used to uniquely identify a service chain in a network. The packet tagging uses this SC-ID to encode the service chain context. The flow rule consumption by the SC-ID tags in the network is proportional to the number of middleboxes present service chains. SC-IDs flow rules can be merged at the switches connected to middleboxes to reduce the TCAM consumption.

This thesis proposes to compress the flow rules for service chaining by optimizing the generation of SC-IDs. The naive approach for generating SC-IDs for  $N$  service chains will take  $N!$  which is computationally not feasible. A Greedy Algorithm for generating SC-IDs is proposed. Our proposed solution 1) makes service chain IDs aggregatable based on Common Forwarding Actions (CFAs) among the service chains, and 2) reduces the number of flow rules at each SDN switch to execute a larger number of forwarding actions for service chaining.

The evaluation results showed that the proposed algorithm can reduce up to 76% of the flow rules using the randomly generated 4000 service chains for 7 middleboxes. The time complexity of the proposed algorithm is polynomial.

The proposed algorithm suggests the effective placement of middleboxes in the given network to reduce the to-and-fro movement of packets. The flow rule compression also helps in performance gain of software switches like Open-vSwitch (OVS) which does not have TCAM by reducing the L1 and L2 caches misses. Our solution is beneficial for both hardware switches as well as software switches. Dynamic addition of new service chains in the network is not supported by our proposed solution. It creates a trade-off between to use the newly added service chains without flow rule compression or re-generate the SC-IDs for all the service chains again. Because the generation of Service Chain ID does not interfere the other flow rule compression techniques, our algorithm can also be used as a plug-in to the other Service Chaining mechanisms to optimize their ID generation.

The proposed solution is using greedy approach that does not guarantee the optimal solution in all cases. As a part of future work, other heuristic solutions can be proposed for SC-IDs generation.

# Contents

Declaration . . . . .	ii
Approval Sheet . . . . .	iii
Acknowledgements . . . . .	iv
Abstract . . . . .	v
<b>Nomenclature</b>	<b>vi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Software Defined Networking (SDN) . . . . .	1
1.2 Service Chaining . . . . .	1
1.3 Ternary Content Addressable Memory (TCAM) . . . . .	1
<b>2 Related Work</b>	<b>2</b>
<b>3 Need for Flow Rules compression in Service Chaining</b>	<b>3</b>
<b>4 Compressing Flow Rules for Service Chaining</b>	<b>4</b>
<b>5 Greedy Algorithm for SC-ID Allocation</b>	<b>7</b>
5.1 Finding CFAs and Forming Node Pairs in the Tree . . . . .	7
5.2 Generating the Service Chain IDs for Flow Rule Compression . . . . .	8
<b>6 Evaluation</b>	<b>12</b>
6.1 Proof of Concept (PoC) Implementation . . . . .	12
6.2 Network Setup for Evaluation . . . . .	13
6.3 Evaluation Results . . . . .	13
6.3.1 Effect of increasing service chains over fixed number of middleboxes . . . . .	13
6.3.2 Effect of increasing the number of middleboxes over fixed number of service chains . . . . .	13
6.3.3 Effect of increasing the number of middleboxes over different service chain sets	13
6.3.4 Time taken to generate SC-IDs . . . . .	14
<b>7 Discussion</b>	<b>17</b>
<b>8 Conclusion and Future Work</b>	<b>19</b>
<b>References</b>	<b>20</b>

# Chapter 1

## Introduction

### 1.1 Software Defined Networking (SDN)

SDN separates the control plane from data plane of a network device. This separation of control plane provides us new opportunities for programming, managing, scaling and monitoring in the way we want. The control plane of SDN is centralized. SDN uses OpenFlow protocol for communication between switches and controller.

### 1.2 Service Chaining

In Network Function Virtualization (NFV), Service Chaining is a key technique to service providers for applying and enforcing policy to the specific traffic. In Service Chaining, a packet has to travel through a set of middleboxes in a pre-defined order as described in its service and network policies. An example for a service chain can be like, data from a host should go through middle boxes in the sequence Traffic Shaper - IPS/IDS - Content Filtering - WAN accelerator- Firewall. In traditional networks, Service Chaining implementation requires manual configuration on switches, routers and middleboxes, which is complex, rigid and error-prone. Software Defined Networking (SDN) provides programmable and flexible way for implementing Service Chaining. SDN uses OpenFlow protocol for communication between switches and controller.

### 1.3 Ternary Content Addressable Memory (TCAM)

TCAM is used in switches/routers for increasing the speed of route lookup, classifying and forwarding the packets. On SDN switches, flow rules describe the forwarding actions being stored in TCAM. The size of TCAM is limited and cannot be increased beyond an extent due to its high power consumption and cost. TCAM supports three states 0, 1 and X (don't care/masked). Once the TCAM is exhausted, the network performance will start to degrade with introducing packet loss. Optimizing the usage of available TCAM is interesting and challenging problem to scale Service Chaining. The optimization of TCAM consumption is done using the proper utilization of "don't care" state.

## Chapter 2

# Related Work

FlowTags [4] introduces simple extensions to middleboxes for operating additional tags, carried in packet headers. Position [8] uses Destination MAC Address for Service Chaining implementation. OpenSCaaS [9] uses Source MAC Address to encode service chain context. Network Service Header (NSH) [5] pushes an additional header to implement Service Chaining. Steering [1] does smart encoding of the forwarding rules using the pipelining feature introduced in OpenFlow v1.1. SPFRI [6] proposed a double tagging approach using VLAN+MPLS label for Service Chaining. Roberto et. al [7] proposed to use 1) VLAN for tunneling the packets and handling the mangling middleboxes, and 2) Flow Identifier to implement Service Chaining. These proposals so far have provided a way for implementing Service Chaining. But our focus is how to generate the aggregatable and compressible tags that carry service minimum context to reduce the TCAM consumption and to maintain Service Chaining still functional.

The flow rule compression technique has been proposed using prefix aggregation (e.g. IP address) and trie data structure [10]. This technique combines nodes of the tree to reduce flow entries. Angelos et. al [11] proposed algorithm for dynamic aggregation of the flow rules considering QoS of the traffic. The non-prefixed based flow rule compression technique is used for Access Control List rules [12].



## Chapter 3

# Need for Flow Rules compression in Service Chaining

A significant amount of research has been done on Service Chaining implementation [1–7], but not satisfactory in terms of TCAM exhaustion, scalability and demand modification in middlebox software.

Some techniques on flow rule compression have been proposed [10–12]. In the Service Chaining implementation, packet tagging techniques are used to identify a service chain and to indicate the corresponding action to be taken (go to the next middlebox). Lang et. al's technique [10] uses prefix property i.e. IP address. One non-prefix based compression technique was proposed for ACL rule compression, namely Bit Weaving algorithm [12]. However, these techniques can compress only the flow rules that are already assigned to handle the service chains.

Service Chaining for a set of given middleboxes can generate a large number of permutations. For example, per-user and per-application service chains with variety of middleboxes can lead to generate large number of flow rules. The current available implementation techniques [1–7] add at least one flow rule per switch for each service chain. So, in order to support variety of service chains with limited TCAM resource, the compression of service chains is necessary.

## Chapter 4

# Compressing Flow Rules for Service Chaining

Packet tagging is a popular approach in Service Chaining implementation to encode service chain context in the packets. Service Chain ID (SC-ID) is used to indicate to which service chain a particular packet (frame) belongs. Single Tagging techniques puts service context along with forwarding actions the next middlebox in the same tag. Double Tagging separates SC-ID and forwarding actions into two different tags. The double tagging approach is scalable as it can support  $2^N$  service chains for given N-bit service tag field. The flow rule compression will be the most effective when a packet carries both of SC-ID and the next middlebox information in the separate maskable fields.

In this thesis, we propose to use double tagging for the Service Chaining implementation as shown in Figure 4.1. The double tagging allows a packet to carry the service context and next middlebox into two different tags say inner tag and outer tag. The service chain ID part will remain fixed in the inner tag while the next middlebox will keep on changing in the outer as a packet traverses. Therefore, at a particular switch, multiple service chains having the same next middlebox can be aggregated at the level of flow rules, and TCAM consumption can be reduced.



**Figure 4.1:** Double Tagging Approach

For implementing the double tagging approach, `Vlan_tci` and Destination MAC Address field of OpenFlow [13] protocol can be used as the inner and the outer tags respectively. Both fields are maskable as per OpenFlow v1.3.

There are three segments as per Figure 4.2 i.e. Ingress Segment, Service Segment and Egress Segment. The ingress switch (SWI) and egress switch (SWE) are the entry and exit points for inbound and outbound traffic. At the ingress switch, depending upon five-tuple rule the inner tag is assigned to a packet. At the egress switch, the inner tag is removed from the packet. In the service segment, middleboxes running on hypervisors apply services and functions to the packets.

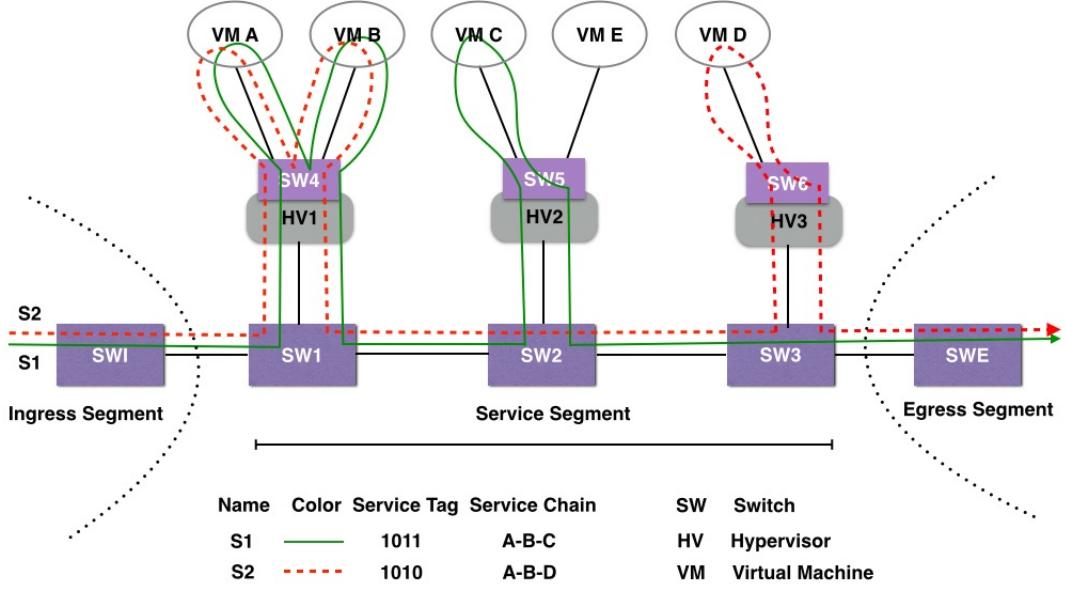


Figure 4.2: Service chain segments

The possible number of service chains with  $N$  different middleboxes is ( $T_{sc}$ )

$$T_{sc} = \sum_{i=1}^N {}^N C_i * (i!) = \sum_{i=1}^N {}^N P_i$$

A service chain with  $K$  middleboxes will have  $K$  different forwarding actions requiring at least  $K$  flow rules at the switches on the service chain path. The Total number of flow rules required to implement  $N!$  service chains with  $K$  middleboxes will be at least of order  $N! * K$ . To implement the service chains of  $N!$  order by keeping TCAM consumption still low the compression of flow rules is necessary. The common sequence of middleboxes among different service chains helps in compression of flow rules.

Considering the example in Figure 4.2, two service chains S1 (A-B-C) and S2 (A-B-D) have one common sequence of middleboxes (A-B). Switch (SW4) executes a common forwarding action to the packets of S1 and S2. SC-ID is maskable, the multiple SC-IDs can be merged into a single flow rule in the flow table denoted as Figure 4.3.

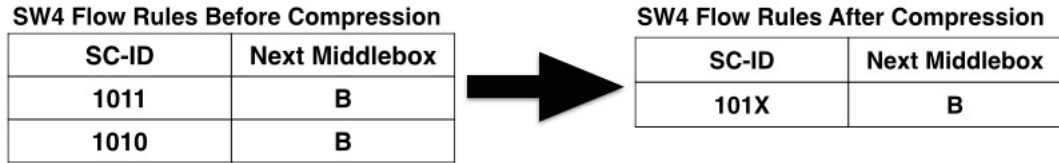


Figure 4.3: Compression of SC-IDs

The compression applied on the inner tag is for the switches that are connected to middleboxes. On rest of the switches in service segment the compression on the outer tag i.e. used for forwarding can be applied using existing TCAM compression techniques.

The number of common forwarding actions between the given set of service chains keep on

In a network, the total number of service chains is known in advance, preprocessing those will

yield the common forwarding actions among various service chains. This common forwarding action can be used as a heuristic to assign SC-IDs, to the service chains, using that Service Chaining can achieve the maximum flow rule compression on the switches.

## Chapter 5

# Greedy Algorithm for SC-ID Allocation

The naive approach to determine the optimal SC-IDs for a given set of service chains is to try out all possible patterns and to figure out the best assignment. The maximum number of service chains with  $n$ -bits field is  $2^n$ . The naive approach for  $2^n$  service chains requires to examine  $(2^n)!$  combinations to get the optimal assignment of SC-IDs. Even with a 12-bits field, trying out all 4096! assignments is not practical to implement. Greedy Algorithm is proposed for the quick approximation to the optimum.

Greedy Algorithm proposed for SC-ID generation is based on the heuristic of **Common Forwarding Action (CFA)** among various service chains. To illustrate the algorithm, the example dataset is shown in TABLE 5.1.

**Table 5.1:** Service Chain Dataset Example

Service Chain	#VM1	#VM2	#VM3	#VM4	#VM5	#VM6
S1	A	B	C	D	E	
S2	E	F	G	H		
S3	E	F	G			
S4	A	B	C	D		
S5	E	F	G	H	I	
S6	F	G	H	I		
S7	A	B	C			
S8	A	B	C	D	E	F

The proposed algorithm uses binary tree data structure. The height of tree is determined by the total number of service chains. The binary tree is initialized with leaves pointing to the respective service chains in the given order as shown in Figure 5.1.

### 5.1 Finding CFAs and Forming Node Pairs in the Tree

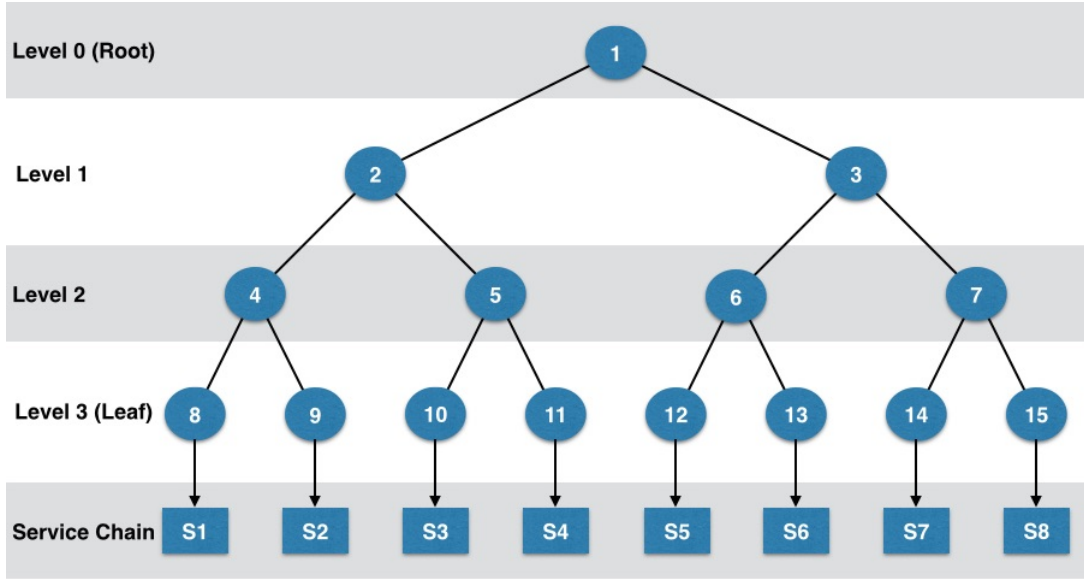
Greedy algorithm performs three steps at each level of the tree except the root, starting from the leaf level. The first step is to determine the CFAs between nodes at a level of the tree. In the second

---

**Algorithm 1** Greedy Algorithm for SC-ID Generation

---

- 1: **Input:** Set of Service Chains.
  - 2: **Output:** Service Chain ID's for the given Service Chains.
  - 3: Height of tree ( $H$ )= $\log_2$  (Number of Service Chains).
  - 4: Create a binary tree with height  $H$ .
  - 5: Attach Service Chains to the leaves of the tree in the given order.
  - 6: **for**  $i = H$  to 1 **do**
  - 7:     Find CFAs among all the nodes at level  $i$ .
  - 8:     Form Node Pairs on basis of most CFAs between them.
  - 9:     Move CFAs from Node Pair to parent node.
  - 10: **end for**
  - 11: Assign bit 0 and 1 to left edges and right edges of tree respectively.
  - 12: Traverse tree from to root to leaves taking values on tree edges in order to generate SC-IDs.
- 



**Figure 5.1:** Tree after Initialization

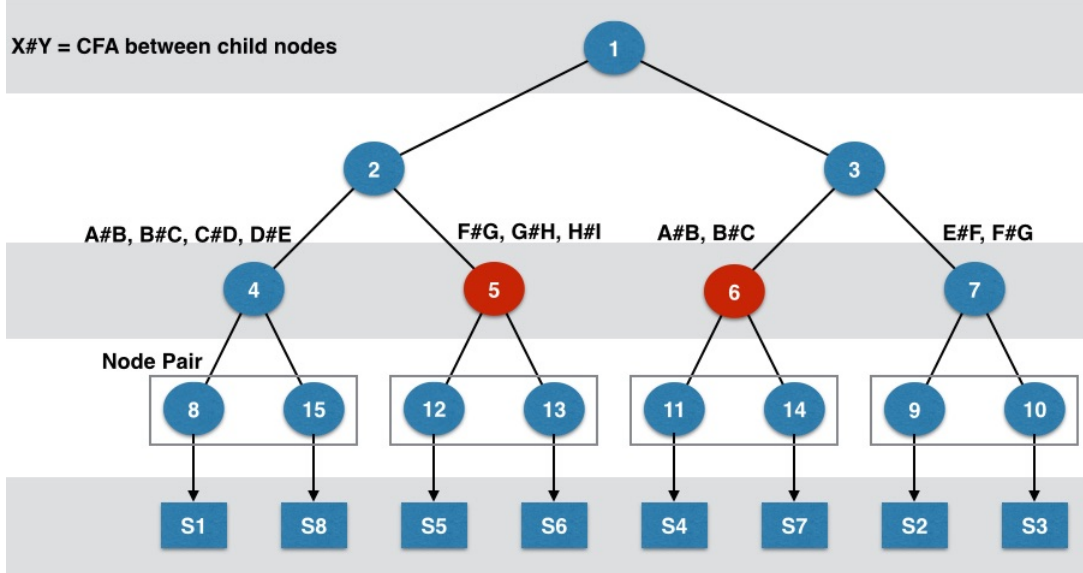
step, the two nodes having the most CFAs form a node pair by swapping operation in the binary tree. The swapping of nodes happens along with the subtrees of the participating nodes. The node pairs formed at leaf level along with their CFAs is shown in TABLE 5.2. The third step involves the movement of CFAs to their parent node from each node pair formed in the step two. Figure 5.2 shows the update tree after the first iteration of all the three steps at Level 3 (Leaf Level).

## 5.2 Generating the Service Chain IDs for Flow Rule Compression

The resultant tree after completion of the algorithm is shown in Figure 5.3. Swapping of nodes 5 and 6 at tree Level 2 happened along with their subtrees during the iteration of line number 7, 8 and 9 in Algorithm 1. The edges of the tree are marked bit values 0 or 1 depending upon its connection to the left or the right child. SC-IDs are generated by performing the traversal of the tree from the

**Table 5.2:** Node Pairs and CFAs Formed at Level 3 (Leaf Level)

Node Pair	Service Chain Pair	Common Forwarding Actions (CFAs)
(8, 15)	(S1, S8)	A#B, B#C, C#D, D#E
(12, 13)	(S5, S6)	F#G, G#H, H#I
(11, 14)	(S4, S7)	A#B, B#C
(9, 10)	(S2, S3)	E#F, F#G



**Figure 5.2:** Tree after 1st Iteration of Loop in Greedy Algorithm (Line number 7, 8 and 9 in Algorithm 1)

root to the leaf. Bit string formed by the path from the root to a leaf determines the SC-ID for the corresponding service chain pointed by the leaf. Finally TABLE 5.3 shows the list of SC-IDs to the given service chain dataset.

**Table 5.3:** Generated Service Chain IDs for Example Dataset

Service Chain	Service Chain ID
S1	000
S2	110
S3	111
S4	010
S5	100
S6	101
S7	011
S8	001

5.4 shows the generated SC-IDs before and after compression at the switch connected to the middlebox A.

The number of flow rules saved by masking SC-IDs,  $N_{saved}$ , can be calculated from the following formula.

$$N_{saved} = \sum_{i=0}^{H-1} [(X = (C_i - \sum_{j=i-1}^0 C_j)) * 2^{H-i} - X]$$

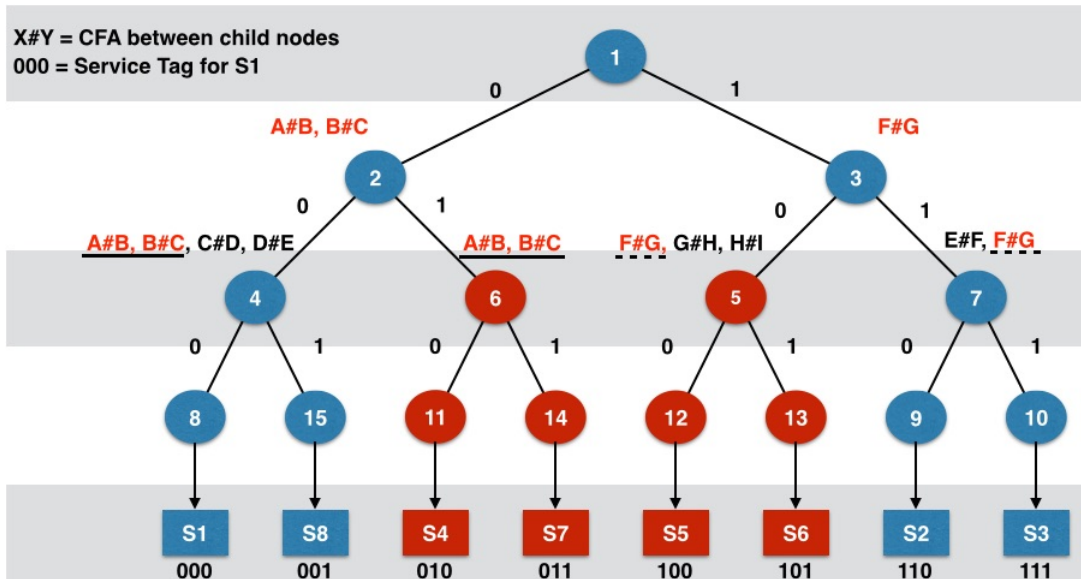


Figure 5.3: Resultant Tree at the End of Greedy Algorithm

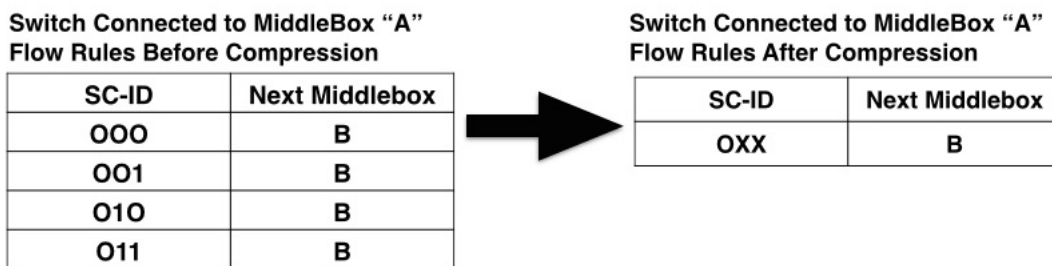


Figure 5.4: Masking of Generated SC-IDs

In the above formula,  $C_i$  denotes the total number of CFAs at the  $i^{\text{th}}$  level of resultant tree. At a particular level of the tree, the subtraction of ancestor node's CFAs is necessary to avoid duplication of CFAs into the result. The position of a CFA in the tree at a particular level determines the number of entries that can be masked into a single entry.

The time complexity of Greedy Algorithm is  $O(M^2 * L * \log_2 M)$  where  $M$  is equal to the number of service chains and  $L$  is the maximum length of a service chain. The space complexity of Greedy algorithm is  $O(M * L)$ .

Finding CFAs among service chains step dominates in time complexity of Greedy Algorithm. To support the very high number of service chains Map-Reduce version of algorithm is given.



---

**Algorithm 2** Determines CFAs among given service chains using Map-Reduce

---

```
1: function MAP1(KEY, SERVICE CHAINS WITH SEQNO)
2:   for each Service chain do
3:     Get the Service_SeqNo
4:     Create a Service_tuple(#VM1,#VM2)
5:     Pass to REDUCE1(Service_tuple, Service_SeqNo)
6:   end for
7: end function

8: function REDUCE1(SERVICE_TUPLE, SERVICE_SEQNO)
9:   Create Service_SeqNo_list=null;
10:  for each Service_tuple do
11:    Get the Service_SeqNo
12:    Service_SeqNo_list.add(Service_Id
13:    Write to temp_output1 (Service_tuple, Service_SeqNo_list)
14:  end for
15: end function

16: function MAP2(SERVICE_TUPLE , SERVICE_SEQ_LIST )
17:   n=Service_SeqNo_list.size()
18:   for  $i = 1$  to  $n - 1$  do
19:      $x$ =Service_SeqNo_list.get( $i$ )
20:     for  $j = 1$  to  $n$  do
21:        $y$ =Service_SeqNo_list.get( $j$ )
22:       Service_SeqNo_tupe=( $x,y$ )
23:       Pass to REDUCE2(Service_SeqNo_tupe , Service_tuple)
24:     end for
25:   end for
26: end function

27: function REDUCE2(SERVICE_SEQNO_TUPE , SERVICE_TUPLE)
28:    $sum = 0$ ;
29:   for each Service_SeqNo_tuple do
30:      $sum = sum + 1$ 
31:     Write to output (Service_SeqNo_tuple, Sum)
32:   end for
33: end function
```

---

# Chapter 6

## Evaluation

### 6.1 Proof of Concept (PoC) Implementation

In OpenFlow v1.3, the two fields **Vlan\_tci** and **Destination MAC Address** are maskable and are used as the inner and outer tags respectively for Service Chaining implementation. The size of **Vlan\_tci** field is 12 bits supporting  $2^{12}$  different service chains. The **Vlan\_tci** keeps the SC-ID intact when the packet passes through middleboxes.

Followings are the three cases considered in the PoC implementation for evaluating the proposed algorithm:

1. No Compression: The SC-ID field is non maskable and the flow rule compression does not take place. The number of flow rules ( $N$ ) required by SC-IDs in this case is equal to the summation of middleboxes in each service chain.
2. Sequential ID Generation with Compression: SC-IDs are assigned sequentially to the service chains. The existing solutions for service chaining mentioned in Related work do not focus on the ID generation, thereby uses sequential allocation. To calculate masked entries in the sequential ID generation, a binary tree is formed similar to one used in Greedy algorithm. However, in the binary tree, only CFAs between adjacent nodes are moved to the parent node without any swapping of the nodes. The number of flow rules ( $SE$ ) saved by masking is calculated using the formula  $N_{saved}$  mentioned in Chapter 5. The total number of flow entries consumed for this service chaining implementation is  $N-SE$ .

$$\% \text{ Reduction in FlowRules} = \frac{SE}{N} * 100$$

3. Greedy ID Generation with Compression: Greedy algorithm proposed in Chapter 5 is used to generate SC-IDs. The number of saved flow entries ( $GE$ ) is calculated using tree formed by Greedy algorithm. The total number of flow entries required by service chains is  $N-GE$ .

$$\% \text{ Reduction in FlowRules} = \frac{GE}{N} * 100$$

## 6.2 Network Setup for Evaluation

The proof of concept is verified with linear topology (15 switches) created with Mininet 2.2.0 [14], enabling Floodlight 1.1 [15] as SDN controller, and Open vSwitch(OVS) version 2.3.1 [16] as SDN switch on mininet VM. Middleboxes are implemented with libpcap for imposing various types of middlebox behavior on the network traffic. With the 12-bits `Vlan_tci` as the inner tag, 4096 service chains are available. The half of service chains in the dataset is generated randomly and the other half are obtained for their reverse direction of the chains. 7 middleboxes can generate approximate  $7! = 5040$  service chains, the least possible value for middleboxes that exceeds the capacity of the inner tag using `Vlan_tci`. The results shown are averaged over five iterations of experiments. The proposed compression of SC-IDs takes CFAs among service chains as heuristic, that does not reflect the actual path between the middleboxes and does make the Greedy Algorithm topology independent.

## 6.3 Evaluation Results

### 6.3.1 Effect of increasing service chains over fixed number of middleboxes

Figure 6.1 and 6.2 show the comparison results of the three implementations of the flow rule compression in the number and percentage. Figure 6.1 shows that even if the flow rule compression is attempted, the sequential ID generation does not reduce the number of flow rules significantly. In contrast, the compression with the proposed SC-ID generation exhibits the maximum of 76% reduction of the flow rules as shown in Figure 6.2.

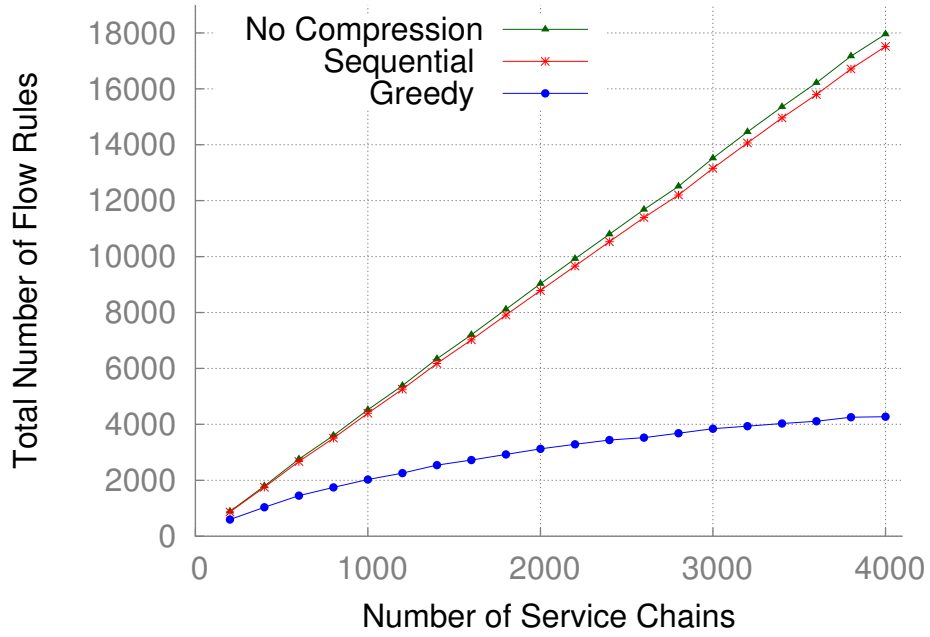
7 middleboxes can generate  $7! = 5040$  service chains, as per results choosing less number of service chains from 5040 available combination, will have less number of CFAs and hence less compression. For 200 service chains i.e. the % Reduction is reduced to 28% in comparison to 76% for 4000 service chains. This shows that the actual compression ratio of flow rules depends upon the number of CFAs upon service chains.

### 6.3.2 Effect of increasing the number of middleboxes over fixed number of service chains

Given the fixed number of service chains, the effectiveness of compression is compared between the sequential and proposed algorithm. The increasing number of middleboxes introduces the degradation of compression efficiency of proposed algorithm as shown in Figure 6.3. This is because the increase in the number of middleboxes decreases the number of CFAs. However, the comparison shows that the proposed approach always shows the better performance even if the number of middleboxes gets reasonably large in a service chain.

### 6.3.3 Effect of increasing the number of middleboxes over different service chain sets

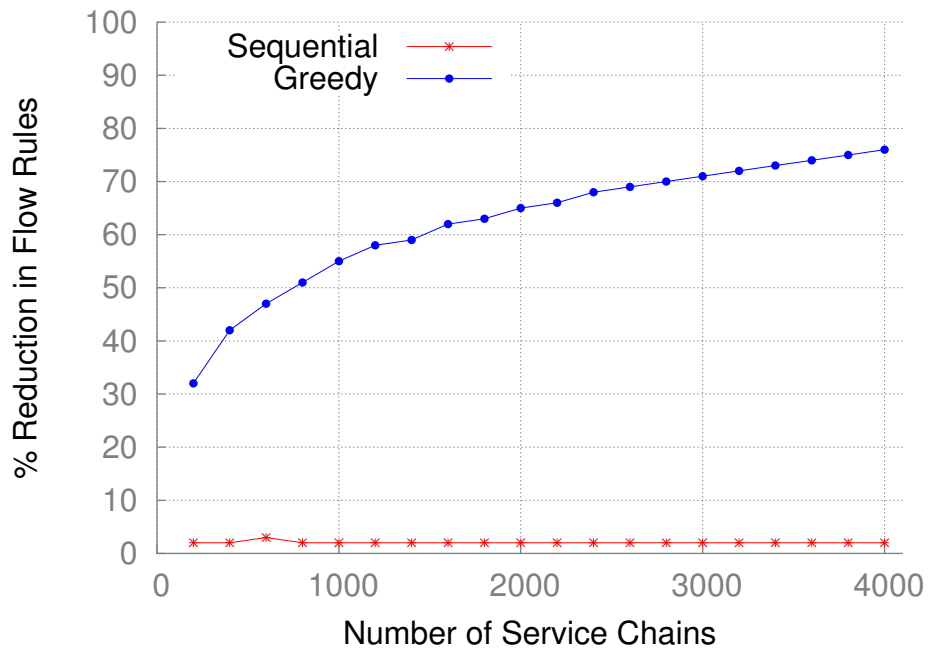
Figure 6.4 shows that having a bigger ID space is reasonable to accommodate the more number of service chains with higher efficiency of flow rule compression.



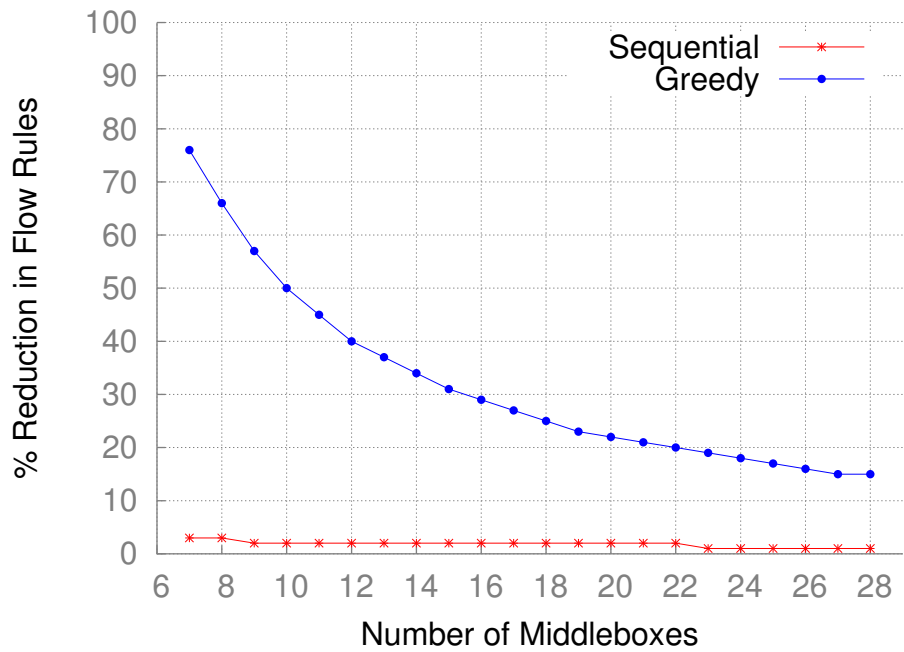
**Figure 6.1:** The number of Flow Rules for 7 Middleboxes using different ID generation and compression schemes

### 6.3.4 Time taken to generate SC-IDs

The step of finding CFAs dominates in time complexity of the proposed algorithm. The increase in service chains for the fixed number of middleboxes produces more CFAs resulting in the longer time consumption as shown in Figure 6.5.



**Figure 6.2:** Ratio of Flow Rule Reduction against the increasing number of Service Chains for 7 Middleboxes



**Figure 6.3:** Ratio of Flow Rule Reduction against the increasing number of Middleboxes for 4096 Service Chains

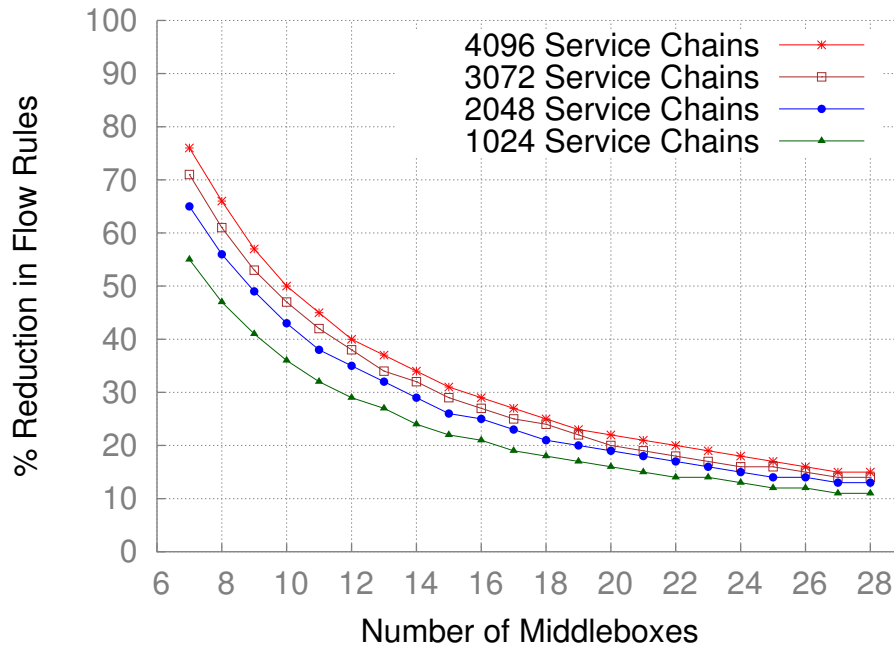


Figure 6.4: Ratio of Flow Rule Reduction for the different number of Service Chain Sets

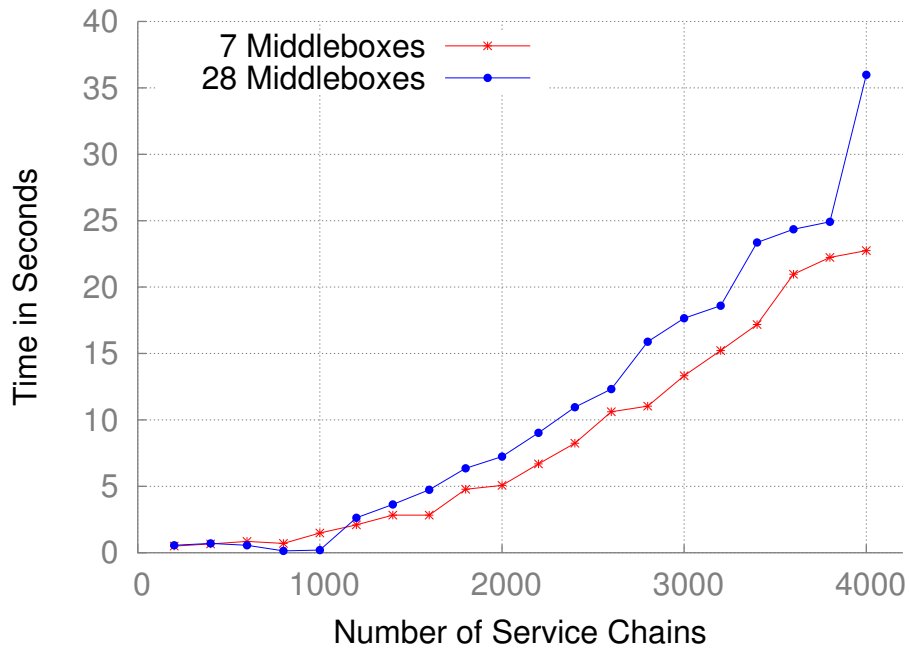


Figure 6.5: Comparison of Time Taken for Generating Service Chain IDs

# Chapter 7

## Discussion

**Placement of middleboxes** The placement of middleboxes in the network impacts the to-and-fro movement of the packets. Our proposed algorithm suggests the effective placement of middleboxes for implementing service chains. The CFAs, which are nearer to the root of the tree, appear in the more number of service chains. The middleboxes present in those CFAs can be placed on the same hypervisor or switch to further reduce the to-and-fro movement of packets.

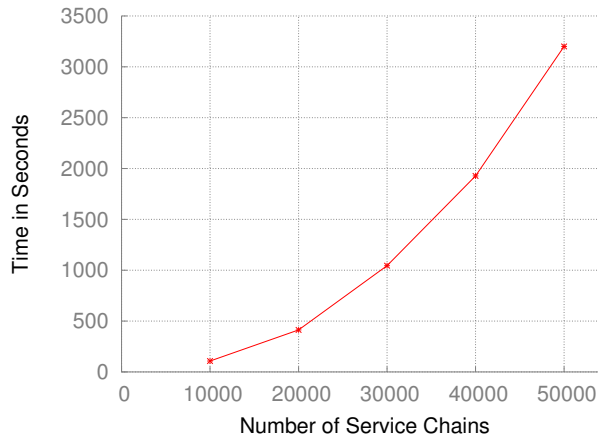
**How does software switches take the advantage of flow rule aggregation?** The proposed flow rule compression works greatly when the switch is attached to a middlebox. However, in the cloud computing environment, hypervisors of VMs may have a software switch like OVS that does not have TCAM. The performance evaluation of OVS conducted by Emmerich et. al [17] showed that the increasing number of flow rules introduces the degradation of forwarding throughput due to the increase of L1 and L2 cache misses on a CPU core. Although the testing result was about the case of forwarding traffic from Physical NIC to Physical NIC through OVS, we assume this result will also apply to the scenarios involving Virtual NICs. In this case, the proposed flow rule compression still benefits the performance on the packet forwarding on software switches as well as the resource management on hypervisors.

**Applicability of the compressed SC-ID generation** If a hardware switch is not attached to middleboxes, do we have a chance to get benefit of the proposed algorithm? We have observed the existing and emerging demands on the variety of service functions and their possible shapes, including QoS, Network Slicing, Docker-based middleboxes and etc. especially in 5G deployment scenarios. Such demands introduce great chances where an intermediate hardware switches in a segment turns to be attached to various actual or pseudo middleboxes at anytime. By well planning the SC-IDs and its ID space, the current SDN deployment, using both software and hardware switches, can mitigate the impact of dynamic change of service function trends in a scalable manner in terms of flow rule and TCAM resource management.

**Finding the effective length and number of service chains for a given network** TCAM consumption is affected by parameters like the length of service chains, the number of service chains and the CFAs pattern among those service chains. The proposed algorithm helps in finding effective size of service chains i.e. whether a very long service chain would remain as it is or be broken into

shorter ones to achieve the higher flow rule compression ratio and the less number of flow rules as total.

The current implementation is using 12-bits `vlan_tci` field. The proposed algorithm can be extended to support longer SC-IDs. However, is having much more number of service chains realistic? In the case of longer tags, the time consuming step of finding CFAs can be done through parallel algorithms or Map-Reduce framework [18] instead of simple loops. We have also implemented the algorithm for finding CFAs using Hadoop. Figure 7.1 shows the time taken for finding CFAs at the leaf level for 28 middleboxes using Hadoop v2.6.0 over single node cluster. Even if finding CFAs can be done within the polynomial time, the benefit of having a very large number of service chains is highly doubtful expecting the demand for very short response time.



**Figure 7.1:** Time Taken to Find CFAs at Leaf Level for 28 middleboxes using Hadoop Framework

**Dynamic addition of new service chains** The proposed Greedy algorithm doesn't consider the dynamic change in the number of service chains or middleboxes. To set up the new service chains dynamically, the unused ID space can be used meanwhile. Once the benefit of compressing the new service chains becomes more than the cost of rerunning the algorithm for all the service chains, then we can remove the existing service chains and assign the new SC-IDs all over again. This introduces the trade-off between to use the existing bit field space without compression or to regenerate new IDs for all the service chains.

Well planning the use of ID space, i.e. managing the number of bits in the inner tag for immediate use and for the future, will also mitigate this trade-off. The IDs for only new service chains can be generated without involving the existing ones. Even if the effectiveness of compression may not be optimal, the cost and impact of reinstalling the flow rules at the switch can be much less.

**Avoidance of bandwidth saturation** Our approach considers only the shortest path for forwarding to a middlebox. Therefore, the aggregation of traffic multiple flows over a single path may lead to bandwidth saturation. In the presence of multiple paths, the bandwidth saturation can be avoided by considering the same middlebox as multiple pseudo instances in the algorithm via the available paths. As a trade-off, aggressive use in the combination scenario of multi-path and sparse middlebox deployment may result in less flow rule compression.



## Chapter 8

# Conclusion and Future Work

This thesis proposed an approach for optimizing the generation of service chain IDs for compressing flow rules for scalable Service Chaining. The key ideas are 1) finding Common Forwarding Actions (CFAs) among multiple service chains and 2) generating the optimum and maskable service chain IDs in order to effectively compress the flow rules for packet forwarding on service chaining. The evaluation results showed that the proposed approach can reduce the flow rules up to 76%, and we also studied how the efficiency of compression may become even higher or low according to the available size of ID space and other properties of service chains. As an approach of flow rule compression, the optimization of service chain ID has not attracted much attention compared with the regular packet forwarding in SDN. Our approach will be promising to achieve a significant reduction of the number of flow rules on the existing and potential packet tagging approaches on Service Chaining.

As a part of future work, we can explore other heuristics for the service chain ID generation. Several points raised in Discussion chapter will be interesting to explore with implementing Service Chaining in the live networks. Making our algorithm as an open source API will also be contributing so that the other Service Chaining mechanisms can offload the task of ID generation to our approach.

# References

- [1] Y. Zhang, N. Beheshti, L. Beliveau, G. Lefebvre, R. Manghirmalani, R. Mishra, R. Patneyt, M. Shirazipour, R. Subrahmaniam, C. Truchan *et al.*, “Steering: A software-defined networking for inline service chaining,” in *Network Protocols (ICNP), 2013 21st IEEE International Conference on*. IEEE, 2013, pp. 1–10.
- [2] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, “Simple-fying middlebox policy enforcement using SDN,” in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 27–38.
- [3] Y. Ben-Itzhak, K. Barabash, R. Cohen, A. Levin, and E. Raichstein, “Enfordsn: Network policies enforcement with sdn,” in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015, pp. 80–88.
- [4] S. K. Fayazbakhsh, V. Sekar, M. Yu, and J. C. Mogul, “Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions,” in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN ’13. New York, NY, USA: ACM, 2013, pp. 19–24.
- [5] P. Quinn, R. Fernando, J. Guichard, S. Kumar, P. Agarwal, R. Manur, A. Chauhan, M. Smith, N. Yadav, B. McConnell, and C. Wright., “Network service header,” *Internet-Draft draft-quinn-nsh-03, IETF Secre-tariat*, July 2014.
- [6] P. Pawar and K. Kataoka, “Segmented proactive flow rule injection for service chaining using sdn,” in *2nd IEEE Conference on Network Softwarization 2016*, pp. 38 – 42.
- [7] A. S. Roberto Bifulco, Anton Matsiuk, “Ready-to-deploy service function chaining for mobile networks,” in *2nd IEEE Conference on Network Softwarization 2016*, pp. 175 – 183.
- [8] J. J. Blendin, J. Ruckert, N. Leymann, G. Schyguda, and D. Hausheer, “Position: Software-defined network service chaining,” in *European Workshop on Software Defined Network (EWSDN)*. IEEE, 2014, pp. 139 – 140.
- [9] W. Ding, W. Qi, J. Wang, and B. Chen, “Openscaas: an open service chain as a service platform toward the integration of sdn and nfv,” *IEEE Network*, vol. 29, pp. 30–35, May 2015.
- [10] B. Leng, L. Huang, X. Wang, H. Xu, and Y. Zhang, “A mechanism for reducing flow tables in software defined network,” in *2015 IEEE International Conference on Communications (ICC)*, June 2015, pp. 5302–5307.

- [11] J. S. Angelos Mimidis, Cosmin Caba, “Dynamic aggregation of traffic flows in sdn,” in *2nd IEEE Conference on Network Softwarization 2016*, pp. 136 – 140.
- [12] C. R. Meiners, A. X. Liu, and E. Torng, “Bit weaving: A non-prefix approach to compressing packet classifiers in tcams,” *IEEE/ACM Transactions on Networking*, vol. 20, pp. 488–500, April 2012.
- [13] “Openflow protocol,” <http://www.openflow.org>.
- [14] B. Lantz, B. Heller, and N. McKeown, “A network in a laptop: Rapid prototyping for software-defined networks,” in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX. New York, NY, USA: ACM, 2010, pp. 19:1–19:6.
- [15] “Floodlight openflow controller,” <http://www.projectfloodlight.org/floodlight/>.
- [16] “Open virtual switch,” <http://openvswitch.org/>.
- [17] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle, “Performance characteristics of virtual switching,” in *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, Oct 2014, pp. 120–125.
- [18] “T. white. hadoop: The definitive guide. yahoo press, 2010 pdf.”