On Scalable Service Function Chaining with $\mathcal{O}(1)$ Flowtable Entries

Yi Ren, Tzu-Ming Huang, Kate Ching-Ju Lin, and Yu-Chee Tseng
Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan
{renyi, tzuming, katelin, yctseng}@cs.nctu.edu.tw,

***Abstract*—The emergence of Network Function Virtualization (NFV) enables flexible and agile service function chaining in a Software Defined Network (SDN). While this virtualization technology efficiently offers customization capability, it however comes with a cost of consuming precious TCAM resources. Due to this, the number of service chains that an SDN can support is limited by the flowtable size of a switch. To break this limitation, this paper presents *CRT-Chain*, a service chain forwarding protocol that requires only *constant flowtable entries*, regardless of the number of service chain requests. The core of *CRT-Chain* is an encoding mechanism that leverages Chinese Remainder Theorem (CRT) to compress the forwarding information into small labels. A switch does not need to insert forwarding rules for every service chain request, but only needs to conduct very simple modular arithmetic to extract the forwarding rules directly from *CRT-Chain*'s labels attached in the header. We further incorporate *prime reuse* and *path segmentation* in *CRT-Chain* to reduce the header size and, hence, save bandwidth consumption. Our evaluation results show that, when a chain consists of no more than 5 functions, *CRT-Chain* actually generates a header smaller than the legacy 32-bit header defined in IETF. By enabling prime reuse and segmentation, *CRT-Chain* further reduces the total signaling overhead to a level lower than the conventional scheme, showing that *CRT-Chain* not only enables scalable flowtable-free chaining but also improves network efficiency.**

I. Introduction

The emergence of Network Function Virtualization (NFV) enables network operators to virtualize their network services as Virtualized Network Functions (VNFs). Different from traditional network components (i.e., standalone physical devices, each with only one network function), VNFs can be efficiently scaled up/down and, hence, provide agility and flexibility to adaptation of network components according to dynamic user demands. With NFV, a Software Defined Network (SDN) is able to deliver elastic services via *Service Function Chaining* (SFC), which allows traffic to go through a customized sequence of services, i.e., VNFs. Such SFC capability is beneficial for supporting a large variety of applications defined by operators or users.

The great flexibility of such VNF service chaining, however, comes with a cost. A flow passing through a specific service chain should be installed as forwarding rules (i.e., flowtable entries) in Ternary Content-Addressable Memory (TCAM) of SDN switches. That is, the required TCAM resources grow linearly with the number of distinct service chain requests. However, while TCAM reduces the forwarding delay significantly, it usually has a fairly limited capability due to its extremely high cost and power consumption. Therefore, the maximum number of service chains that can be served in an SDN is typically limited by the scarce TCAM space of SDN switches. The scalability is, hence, a fundamental challenge of enabling such configurable service chaining.

To provide scalable, customizable service chaining, we present *CRT-Chain*, a service chain forwarding protocol that requires only *constant flowtable entries*, regardless of how many SFC requests being served in the system. The key idea of *CRT-Chain* is to leverage the Chinese Remainder Theorem (CRT) to encode the forwarding rules of a service chain into small labels, which can be embedded in the header of a packet. Each switch then conducts very simple modular arithmetic to extract forwarding rules directly from the CRT labels. By doing this, *CRT-Chain* requires zero flowtable entry to insert every new SFC request. The beauty of *CRT-Chain* is that it replaces the cost of flowtable entries with a small header overhead. Therefore, the proposed *CRT-Chain* can be an efficient backup that compensates for the deficiency of conventional flowtable-based chaining when the number of SFC requests exceeds the TCAM capability or when an operator prefers to reserve the precious TCAM resources for other purposes.

To improve the efficiency of the proposed CRT-based service chaining, we need to overcome some technical challenges. First, in an SDN, a switch is usually connected to multiple VNFs, and an SFC may request to be served by different functions associated with the same switch *in a specific order*. However, a naïve CRT encoding scheme is oblivious of the sequence of forwarding and, thereby, could lead to ambiguity. Thus, we design a novel encoding algorithm that always guarantees the unique chaining path specified by the controller (see Section IV-B). Second, to reduce the label size, we propose a chain segmentation scheme that allows the overall label size of all the segments to be smaller than the label size of the end-to-end path. In addition, the last-hop switch of each segment can discard the sub-header to further save overhead bandwidth consumption (see Section IV-C). Finally, a CRT label size is determined by the primes assigned to the functions along a chain. Fortunately, different network functions usually have heterogeneous importance and popularity. We hence exploit a prime assignment strategy based on the distribution of function popularity to reduce the expected label size (see Section IV-D).

We examine the protocol efficiency of *CRT-Chain* in terms of the header size and the overall overhead bandwidth consumption, with and without path segmentation. The evalua-

tion results demonstrate that, without path segmentation, the average header size of *CRT-Chain* ranges from 5 bits to 55 bits when the length of a service function chain varies from 1 to 10. More specifically, the header of *CRT-Chain* is smaller than the 32-bit legacy header defined in IETF when the chain length is no longer than 5. By enabling path segmentation, *CRT-Chain* can further reduce the total header size and even allow switches to discard sub-headers in the middle of routing. This allows *CRT-Chain* to consume much less bandwidth for signaling as compared to the conventional chaining protocol. We hence conclude that *CRT-Chain* not only saves flowtable entries but also reduces the overall signaling overhead.

The rest of the paper is organized as follows. Section II reviews the related works and Section III introduces the background of service function chaining. Section IV gives an overview of *CRT-Chain* and details the designs of *CRT-Chain*. The implementation via simulations is presented in Section V, followed by performance evaluation in Section VI. We finally conclude in Section VII.

II. Related Work

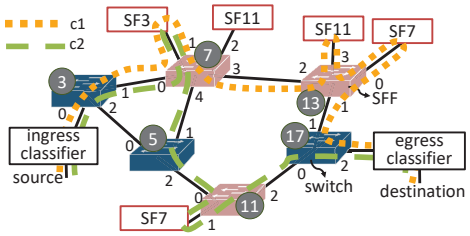
Related works on TCAM management and service chaining fall in three categories.

A. Flowtable Management

Recent works [1]–[4] have investigated how to overcome the limitation of scarce TCAM resources by reducing the number of forwarding rules inserted to each switch. The work [1] proposes to aggregate flows with identical forwarding information so as to reduce the number of flowtable entries. A later work [2] then extends [1] by formulating the aggregation problem as an optimization model and achieving an order of reduction. However, it builds on an assumption that a routing table has a tree structure. On the other hand, a distributed scheme [3] is proposed to decompose a large flowtable into small ones that can be distributed across the network. By optimizing the size of tables across the network, the total number of flow entries can be reduced. It, however, does not perform well when the path length is small due to the fact that the performance is determined by the length of the shortest path. The work [4] enhances [3] by caching the most popular rules in a small TCAM. The rest small amount of cache misses is handled by software. Cache misses, however, introduce a much longer fetching delay between switches and the controller. Our work differs from the above approaches in that we manipulate a label encoding algorithm to replace a large number of chaining rules with a constant number of label forwarding rules.

B. Entry Size Reduction

Another branch of works [5]–[9] attempt to reduce the size of each flowtable entry. Some [5] [6] propose to use a short label to replace the long network address. The work [5] appends a label to each packet for encoding the routing path, while the other [6] attaches a label that encodes the routing information of a flow, which in turn reduces the size of



(a) SFPs of the two SFCs

id	index	next hop
1	3	SF3
1	2	S13
2	2	SF3
2	1	S5

(b) legacy flowtable for SFF 7

S prime	SF prime	next hop
7	-	-
-	3	SF3
-	11	SF11

(c) flowtable for SFF 7 in *CRT-Chain*

Fig. 1: Example of two SFC requests.

Forwarding Information Base (FIB) and complexity of inter-domain routing. They, hence, avoid complex routing table lookup. More recently, [7] proposes a novel forwarding table architecture to compress flow entries. Others [8] [9] propose to use MAC addresses as virtual addresses to minimize the TCAM usage. The work [9] uses the destination MAC address as a universal label to reduce the number of flow entries and the size of each flow entry, without increasing the packet header length. However, it requires explicit path-label assignment, which is proven an NP-complete problem. Instead of reducing the entry size, our design directly eliminates the need of flowtable entries for each service chain, and is hence scalable for a system supporting a large number of service chains.

C. Flowtable-Free Routing

The works most related to ours are [10]–[12]. In [10], the forwarding information of a flow traversing through a path is encoded into a label, which is appended to each packet. Based on [10], the work [11], [12] further shows how to reuse the MAC address as the label, as a result introducing no additional header information. However, the label could become very large when the forwarding path is long or a network has a large number of nodes. Our work leverages the properties of sequential service requests in an SFC to design a novel label encoding algorithm, which introduces a small overhead limited by the number of function types (typically much smaller than the number of nodes).

III. Background of Service Function Chaining

The SFC protocol defined in IETF RFC7665 [13] allows a packet to be forwarded to and served by a sequence of one or more L4-L7 services, which can be realized via NFV on top of an SDN. For example, one can request to send a packet through several service functions sequentially in a specific order, such as firewall, Network Address Translation (NAT), and Deep Packet Inspection (DPI). To improve reliability and load-balancing, for each Service Function (SF), we can create multiple SF instances, each installed in a Virtual Machine (VM) associated with a different Service Function Forwarder (SFF). An SFF can be a physical switch or a virtual switch that

helps forward packets of an SFC to a specified SF instance. The functionality of an SFF is similar to that of a legacy switch except that an SFF can insert/process the forwarding rules generated for an SFC and can further know how to parse the Network Service Header (NSH) of an SFC packet. Hence, in this work, we use a universal term, *forwarder*, to denote either a switch or an SFF.

As there exist many instances for an SF type, for each SFC request, the controller has a flexibility of identifying any available Service Function Path (SFP) that consists of a sequence of the scheduled SF instances based on some performance metrics (e.g., path length or traffic load). Consider an example of two SFC requests, $c1$ and $c2$, as illustrated in Fig. 1. SFC $c1$ requests to access SF3, SF11, and SF7 sequentially, while SFC $c2$ requests to access SF3 and SF7 sequentially. The orange path denotes the SFP of $c1$, which goes through the instances SF3 (associated with SFF7), SF11 (associated with SFF13) and SF7 (associated with SFF13) sequentially. Similarly, the green path denotes the SFP of $c2$.

The forwarding rules of each SF instance along the scheduled SFP are inserted to the flowtable of its associated SFF. When an SFC packet arrives to an SFC-enabled domain, it first enters an ingress classifier, which retrieves the SFP of that SFC from the controller and appends an NSH indicating this SFP to the packet. The packet is then forwarded to the SFFs associated with the SF instances specified in the SFP sequentially. Each SFF receiving this packet will look up its flowtable and forward the packet to the associated SF instance. The ingress classifier also uses the Service Index (SI) (initialized to the SFP length) in NSH to indicate the length of an SFP. Each SF instance along the SFP decreases the value of SI by 1 such that all the SFFs can know whether an SFC has been executed completely and should be forwarded to the egress classifier.

While this forwarding mechanism is simple and easy to implement, it is however not scalable. Since the forwarding rules of each SFP should be inserted to the flowtable of SFFs along the path, the number of the required flowtable entries grows linearly with the number of SFC requests. Consider the same example shown in Fig. 1, where both chains $c1$ and $c2$ are assigned to the SF3 instance associated with SFF7. The conventional protocol will need to insert two sets of rules in the flowtable of SFF7, each for a distinct SFC. This example shows that, even when a large number of service chains share the same forwarding rules, those rules need to be duplicated and, thus, occupy a large number of entries in a flowtable. The number of chains supported in the system is hence limited by the flowtable size.

IV. *CRT-Chain* Design

In this section, we first give an overview of *CRT-Chain*, and then describe its key components. We begin by first introducing the basic label encoding and forwarding procedure of *CRT-Chain*. The extensions for enabling path segmentation and popularity-aware prime assignment are then presented in the end of the section.

A. Overview of CRT-Chain

To resolve the scalability issue, we propose *CRT-Chain*, a flowtable-free service chaining protocol. The high-level idea of *CRT-Chain* is to replace per-chain forwarding rules with *per-function* forwarding rules. More specifically, for all the chains requesting to be served by an SF instance, its associated SFF inserts *only one forwarding rule* for this SF instance, regardless of how many chains assigned to it. To this end, *CRT-Chain* assigns each SF type and each forwarder a prime and leverages the Chinese Remainder Theorem (CRT) to encapsulate the service chain into a path label $\mathcal{X}$ and an SFP label $\mathcal{Y}$, respectively. The labels can replace the 32-bit service path field of the legacy NSH defined in IETF. An SFF then uses very simple modular arithmetic to extract the forwarding information directly from the labels, without knowing which chain it belongs to. Therefore, *CRT-Chain* only requires a constant number of flowtable entries in each SFF and can support a large number of SFC requests without limited by the TCAM capability.

Consider the same example in Fig. 1. We assign SF3 a prime, 3, and only insert one forwarding rule for SF3 based on its prime to the flowtable of SFF7, even when there exist two service chains assigned to its associated SF3 instance. We will explain in detail how to encode the labels and how to forward packets in Section IV-B. As *CRT-Chain* transfers the cost of flowtable entries to the label overhead, we incorporate several designs, including path segmentation (see Section IV-C) and prime assignment (see Section IV-D), to minimize the label size and reduce overhead bandwidth consumption.

B. CRT-based Chaining

CRT-Chain leverages CRT to enable flowtable-free service chaining. We define $\mathcal{F}$ as a set of forwarders, which can be either switches or SFFs that are connected to SF instances. The Ingress Classifier (IC) and Egress Classifier (EC) are also included in $\mathcal{F}$. Let $\mathcal{S}$ denote the set of available SF types. Note that each SF type $s \in \mathcal{S}$ can be installed in multiple VMs associated with different SFFs. Let $\mathcal{C}$ denote the set of SFC requests, where each $c \in \mathcal{C}$ is a sequence of requested SFs. For each SFC request $c \in \mathcal{C}$, we assume that the routing path, denoted by $P(c) = f_1(c) \rightarrow f_2(c) \cdots \rightarrow f_{|P(c)|}(c)$, and the SFP, denoted by $SP(c) = s_1(c) \rightarrow s_2(c) \cdots \rightarrow s_{|SP(c)|}(c)$, are both given (i.e., determined by the controller). Here, $|P(c)|$ and $|SP(c)|$ represent the path length and SFP length, respectively. For example, in Fig. 1(a), the path of service chain $c1$ is $P(c1) =$ IC $\rightarrow$ S3 $\rightarrow$ SFF7 $\rightarrow$ SFF13 $\rightarrow$ S17 $\rightarrow$ EC, and the SFP of $c1$ is $SP(c1) =$ SF3(SFF7) $\rightarrow$ SF11(SFF13) $\rightarrow$ SF7(SFF13). Note that the path and SFP scheduling problems for service chaining have been extensively investigated recently in [14]–[18], and are not the scope of this work.

In our design, we encode the routing path $P(c)$ and service function path $SP(c)$ of each $c \in \mathcal{C}$ into two variable length labels, $\mathcal{X}_c$ and $\mathcal{Y}_c$, respectively. We let $|\mathcal{X}_c|$ and $|\mathcal{Y}_c|$ denote the path label length and the SFP label length, respectively. Each forwarder decodes $\mathcal{X}_c$ and $\mathcal{Y}_c$ to extract the forwarding rules for routing and SFP, respectively, as summarized in

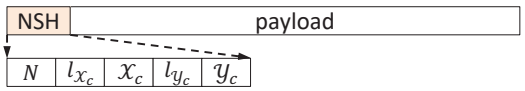


Fig. 2: *CRT-Chain*'s header format.

Algorithm 1. The encoding for $P(c)$ and $SP(c)$ are performed independently, and $\mathcal{X}_c$, $\mathcal{Y}_c$ and a constant length step counter N (initialized to 1) are concatenated together as *CRT-Chain*'s NSH $(N, l_{\mathcal{X}_c}, \mathcal{X}_c, l_{\mathcal{Y}_c}, \mathcal{Y}_c)$, as shown in Fig. 2, in which $l_{\mathcal{X}_c}$ and $l_{\mathcal{Y}_c}$ are the constant length fields indicating $|\mathcal{X}_c|$ and $|\mathcal{Y}_c|$, respectively, and their lengths can be set to $\log_2 |\mathcal{X}_{\max}|$ and $\log_2 |\mathcal{Y}_{\max}|$, where $\mathcal{X}_{\max}$ and $\mathcal{Y}_{\max}$ are the maximum possible path label and SFP label, respectively. We will explain how to derive $\mathcal{X}_{\max}$ and $\mathcal{Y}_{\max}$, respectively, later. This NSH replaces the legacy NSH in each SFC packet.

We assign each forwarder in $\mathcal{F}$ a unique prime, and, similarly, assign each SF type in $\mathcal{S}$ a unique prime. For simplicity, we directly use the prime assigned to a forwarder (SF) as the ID of the forwarder (SF). Note that, since the path and SFP are encoded independently, a forwarder and an SF can share the same prime. That is, $f \neq f'$, for all $f, f' \in \mathcal{F}$, and $s \neq s'$, for all $s, s' \in \mathcal{S}$, but $f = s$ for any $f \in \mathcal{F}, s \in \mathcal{S}$ is allowed. Another thing worth noting is that multiple instances of the same SF type $s \in \mathcal{S}$ use the same prime s. Namely, the number of required SF primes is determined by the number of SF types $|\mathcal{S}|$, instead of the number of SF instances.

Encoding and decoding $\mathcal{X}_c$: The encoding process of $P(c)$ for each $c \in \mathcal{C}$ is similar to the proposal in [10] [11]. However, since *CRT-Chain*'s SFP encoding is designed based on path encoding, we first briefly describe how CRT-based path encoding works. Given a path $P(c)$, the path label $\mathcal{X}_c$ should satisfy the following constraints:

$$\mathcal{X}_c \equiv e_i \,(\text{mod } f_i(c)), \forall 1 \leq i \leq |P(c)|, \tag{1}$$

where $f_i(c)$ is the prime assigned to the i-th forwarder along the path $P(c)$ and e_i is the egress port of flow $P(c)$ in the i-th forwarder $f_i(c)$. The solution of $\mathcal{X}_c$ can be found based on CRT [19] as follows:

$$\mathcal{X}_c = (\sum_{i=1}^{n} w_i \cdot e_i) \,(\text{mod } X), \tag{2}$$

where $X = \prod_{i=1}^{|P(c)|} f_i$, $w_i = Q_i U_i$, $Q_i = \frac{X}{f_i}$, and $U_i = Q_i^{-1} \,(\text{mod } f_i)$. Here, U_i is the multiplicative inverse of Q_i under modulo f_i. Consider service chain $c1$ in the example shown in Fig. 1(a), which traverses through forwarders with primes 3, 7, 13 and 17 using the output ports 1, 3, 1 and 2, respectively. Hence, the path label $\mathcal{X}_{c1}$ should meet the following constraints:

$$\begin{cases} \mathcal{X}_{c1} \equiv 1 \,(\text{mod } 3) \\ \mathcal{X}_{c1} \equiv 3 \,(\text{mod } 7) \\ \mathcal{X}_{c1} \equiv 1 \,(\text{mod } 13) \\ \mathcal{X}_{c1} \equiv 2 \,(\text{mod } 17), \end{cases} \tag{3}$$

and we get $\mathcal{X}_{c1} = 4,252$, which is appended to each packet of $c1$. Based on the above encoding scheme, the maximum possible path label $\mathcal{X}_{\max}$ can be found by encoding the label for the longest possible path traversing through the forwarders assigned the largest primes. The length of the field, $l_{\mathcal{X}_c}$, shown in Fig. 2 can be set accordingly and announced to each forwarder. Note that the length of $l_{\mathcal{Y}_c}$ can be derived similarly.

Algorithm 1: *CRT-Chain*-FORWARD

Input: f: current forwarder; $\mathcal{X}$: path label encoded based on Eq. (2); $\mathcal{Y}$: SFP label encoded based on Eq. (5)

```
1  // extract SFP forwarding rules from Y
2  while true do
3      match ← false
4      for each SF with prime s associated with f do
5          let step be (Y mod s)
6          if step matches N in NSH then
7              match ← true
8              N ← N + 1 and forward the packet to SF s
9              break;
10     if match = false then break
11 // extract routing forwarding rules from X
12 N ← N + 1 // increase step counter
13 e ← X mod f, and forward the packet through port e
14 return
```

To forward an SFC packet, each forwarder decodes $\mathcal{X}_c$ and extracts the forwarding port by taking the same modular arithmetic in Eq. (3) using its assigned prime (line 13 in Algorithm 1). More specifically, forwarder $f_i(c)$ can obtain the output port $e_i = \mathcal{X}_c \text{ mod } f_i(c)$. For example, SFF7 can find its output port by $\mathcal{X}_{c1} \text{ mod } 7 = 3$.

Encoding and decoding $\mathcal{Y}_c$: The encoding of the SFP label $\mathcal{Y}_c$ is different from that of $\mathcal{X}_c$ since an SFF might associate with multiple SF instances and, more importantly, the order of forwarding to different SF instances matters. If we again use the output port toward an SF instance as the remainder, its associated SFF will not know the right forwarding sequence of the requested SFs. For example, for $c1$ in Fig. 1(a), SFF13 connects to two SF instances and $c1$ should be sent to SF11 before SF7. To overcome this problem, we introduce a step counter N for each SFC packet and, alternatively, use the step count of each SF in SFP as the remainder for encoding $\mathcal{Y}_c$.

We use the example illustrated in Fig. 1(a) to explain SFP encoding. In this case, $c1$ goes through SF3 , SF11 and SF7, sequentially, and its SFP label $\mathcal{Y}_{c1}$ should meet the following constraints:

$$\begin{cases} \mathcal{Y}_{c1} \equiv 1 \,(\text{mod } 3) \\ \mathcal{Y}_{c1} \equiv 2 \,(\text{mod } 11) \\ \mathcal{Y}_{c1} \equiv 3 \,(\text{mod } 7), \end{cases} \tag{4}$$

which results in $\mathcal{Y}_{c1} = 178$. Once a packet arrives to an SFF, it first extracts the current step counter N from the header and uses the primes assigned to the SF types of its associated instances to check how the packet should be forwarded. Specifically, when the packet is received by SFF7, it has not

been served yet and, hence, has the step counter initialized to 1. SFF7 uses $\mathcal{Y}_{c1}$ and the primes of SF3 and SF11 to get the remainders $\mathcal{Y}_{c1} \bmod 3 = 1$ and $\mathcal{Y}_{c1} \bmod 11 = 2$ (lines 4–5). Since the remainder of SF3 matches the current step counter, SFF7 knows that the packet should be forwarded to the SF3 instance and then adds the step counter N by 1 (lines 6–9). Since a chain might go through multiple SF instances associated with the same SFF, an SFF should perform such forwarding repetitively until all the remainders obtained by the primes of the associated SFs do not match the current step counter (lines 3 and 10). SFF13 can similarly use the primes of the associated SF instances, 7 and 11, to find that the remainders equal to 3 and 2, respectively, showing that the forwarding order is SF11 followed by SF7.

Ensuring unique SFP forwarding: This simple CRT encoding, however, cannot always guarantee correct SFP scheduled by the controller. Since *CRT-Chain* assigns different instances of an SF type the same prime, an SFC packet might not be forwarded to the specified SF instance correctly if any of SF instances associated with an SFF happens to be the same with the first scheduled SF instance of the next SFF along the SFP. Consider again $c1$ in Fig. 1(a). Although its request for SF11 can be served by either the instance associated with SFF7 or the one associated with SFF13, the controller can exactly specify which instance should be used to improve network performance, e.g., latency reduction or load balancing. In our example, $c1$ is assigned to SF11 associated with SFF13. However, one might observe that SFF7 also connects to an instance of SF11 and will get a remainder matching the step counter (i.e., 2) when the packet returns to SFF7 from SF3. In this case, SFF7 will make a mistake and incorrectly forward $c1$ to its SF11 instance. Though this minor mistake does not affect the serving sequence of functions, it, however, would disturb traffic management expected by the controller.

To avoid this ambiguity, we propose a new way to calculate the step count. More specifically, we combine all the forwarders along $P(c)$ and all the SF instances along $SP(c)$ into a *merged path* $MP(c)$ in their traversing order. For example, for $c1$ in Fig. 1(a), $P(c1)$ and $SP(c1)$ can be merged to a list $MP(c1)$ = (1) S3 $\rightarrow$ (2) SFF7 $\rightarrow$ (3) SF3 $\rightarrow$ (4) SFF13 $\rightarrow$ (5) SF11 $\rightarrow$ (6) SF7 $\rightarrow$ (7) S17, where the number within a brace indicates the step count of the corresponding forwarder or SF instance. Given the merged list $MP(c)$, *CRT-Chain* now encodes the SFP label $\mathcal{Y}_c$ using the step counts of SF instances in $MP(c)$, rather than the original step counts in $SP(c)$, as the remainders. For the above example, the step counts for SF3, SF11, and SF7 become 3, 5, and 6, respectively. This small trick addresses the ambiguity problem since different instances of an SF type associated with various SFFs must correspond to different step counts when a packet traverses from one SFF to another. For example, if we alternatively assign $c1$ in Fig. 1 to SF11 connected to SFF7, the step count of SF11 will be 4 as this SF11 instance becomes located in between SF3 and SFF13 in the resulting merged path.

With this update, $\mathcal{Y}_c$ should now satisfy the following constraints:

$$\mathcal{Y}_c \equiv ix_n \,(\text{mod } s_n(c)), \forall 1 \leq n \leq |SP(c)|, \tag{5}$$

where ix_n denotes the step count (index) of $s_n(c)$ in the merged path $MP(c)$. To ensure unique routing, all the forwarders then modify its decoding process as follows: each forwarder (a switch or SFF) now also needs to add the step counter N in the header by one before a packet departs and heads to the next forwarder (line 12). By doing this, when a packet is sent to a forwarder or an SF instance, its step count N matches the order of the forwarder/SF in $MP(c)$.

C. Chain Segmentation

In *CRT-Chain*, the header overhead, i.e., $|\mathcal{X}_c|$ and $|\mathcal{Y}_c|$, is determined by the primes used in the congruence system, i.e., Eqs. (1,5). Typically, larger primes lead to larger labels, $\mathcal{X}_c$ and $\mathcal{Y}_c$. As *CRT-Chain* assigns each forwarder (SF) a unique prime, the header size scales up with the number of forwarders (SF types) in a network. An intuitive solution to reducing the header size is to allow different forwarders (SFs) to use the same prime, as a result minimizing the number of primes we need. For example, given $\mathcal{F}$ ($\mathcal{S}$), instead of using $|\mathcal{F}|$ ($|\mathcal{S}|$) unique primes, we can use only $\alpha|\mathcal{F}|$ ($\alpha|\mathcal{S}|$) smallest primes, where α is referred to as the *prime reuse rate* and $0 < \alpha \leq 1$. Specifically, each forwarder (SF) is assigned one prime randomly selected from those $\alpha|\mathcal{F}|$ ($\alpha|\mathcal{S}|$) smallest primes. Hence, on average, each prime is reused $1/\alpha$ times by $1/\alpha$ forwarders (SFs). However, to ensure the CRT-based algorithm to work properly, each forwarder (SF) along the same routing path (SFP) should be assigned a unique prime. When some forwarders (SFs) share the same prime, there will be problem if they happen to belong to the same path (SFP). To avoid this problem while embracing the efficiency of prime reuse, we propose a segmentation technique that partitions a path (SFP) into several subpaths, in each of which any two forwarders (SFs) do not share the same prime.

Partitioning a path: To perform conflict-free segmentation, for each path $P(c)$ (SFP $SP(c)$) of a chain c, we trace the path and check whether any $f \in P(c)$ ($s \in SP(c)$) has a prime duplicated with any one locating prior to it. For any duplicated prime found, the path should be cut here, making those forwarders (SFs) prior to it as a conflict-free subpath[1]. Consider an example path $P(c) = f_1 \rightarrow f_2 \rightarrow f_3 \rightarrow f_4 \rightarrow f_5$ assigned the primes $5 \rightarrow 13 \rightarrow 5 \rightarrow 2 \rightarrow 7$. It should be partitioned into two conflict-free subpaths: $P_1(c) = f_1 \rightarrow f_2 = 5 \rightarrow 13$ and $P_2(c) = f_3 \rightarrow f_4 \rightarrow f_5 = 5 \rightarrow 2 \rightarrow 7$. Note that an SFP can be partitioned in a similar way. After partitioning, the labels $\mathcal{X}_{c,i}$ and $\mathcal{Y}_{c,i}$ of each subpath i can be encoded in the similar way as mentioned in Sec. IV-B. Those sub-labels are then concatenated together as the header in the format of $(N, l_{\mathcal{X}_{c,1}}, \mathcal{X}_{c,1}, l_{\mathcal{Y}_{c,1}}, \mathcal{Y}_{c,1}, \cdots, l_{\mathcal{X}_{c,i}}, \mathcal{X}_{c,i}, l_{\mathcal{Y}_{c,i}}, \mathcal{Y}_{c,i}, \cdots)$, where N is again the step counter initialized to 1, and $l_{\mathcal{X}_{c,i}}$ and $l_{\mathcal{Y}_{c,i}}$ indicate $|\mathcal{X}_{c,i}|$ and $|\mathcal{Y}_{c,i}|$, respectively.

[1] A forwarder and an SF in the same $MP(c)$ can have the same prime.

Algorithm 2: FORWARD-SEGMENT

Input: $(N, \mathcal{X}_{c,\text{now}}, \mathcal{Y}_{c,\text{now}}, \mathcal{X}_{c,\text{now}+1}, \mathcal{Y}_{c,\text{now}+1}, \cdots)$: remaining NSH; f: current forwarder;

perform SF forwarding as mentioned in Algorithm 1

// check whether it is the last-hop of the subpath P_{now}

if $(\mathcal{X}_{c,now} \bmod f)$ *is not a valid port* **then**

  discard $\mathcal{X}_{c,\text{now}}$ and $\mathcal{Y}_{c,\text{now}}$

  $\mathcal{X}_{c,\text{now}} \leftarrow \mathcal{X}_{c,\text{now}+1}$ and $\mathcal{Y}_{c,\text{now}} \leftarrow \mathcal{Y}_{c,\text{now}+1}$

$N \leftarrow N + 1$ // increase step counter

$e \leftarrow \mathcal{X}_{c,\text{now}} \bmod f$, and forward the packet through port e

return

Forwarding subpaths: The remaining problem is how can a forwarder know which sub-label $(\mathcal{X}_{c,i}, \mathcal{Y}_{c,i})$ it should decode and when should a sub-label be discarded. To this end, we propose a variation of label encoding such that each forwarder can leverage the similar modular arithmetic and a clever detection rule to determine whether to terminate a subpath. Our design is motivated by an observation that each forwarder has a limited number of output ports. Hence, we can use this parameter to encode the termination rule of a subpath. More specifically, let o_f denote the maximum output port of forwarder f. Recall that in *CRT-Chain*'s routing, each forwarder with prime f finds its output port by $(\mathcal{X}_{c,i} \bmod f)$. If we want to force the last-hop forwarder f to terminate the current subpath $P_i(c)$ and truncate the sub-label $(\mathcal{X}_{c,i}, \mathcal{Y}_{c,i})$, we can modify the modulo constraint for $f \in P_i(c)$ to

$$\mathcal{X}_{c,i} \equiv e_{\text{null}} \,(\text{mod } f), \tag{6}$$

where the remainder e_{null} can be any integer number larger than the maximum output port o_f. By doing this, the last-hop forwarder f of a subpath will get an invalid port e_{null} and easily detect that it should end the current subpath. Then, the forwarder drops the current sub-label $(\mathcal{X}_{c,i}, \mathcal{Y}_{c,i})$, extracts the next one $(\mathcal{X}_{c,i+1}, \mathcal{Y}_{c,i+1})$, and initiates the forwarding process for subpath $P_{i+1}(c)$, as summarized in Algorithm 2.

D. Prime Assignment

The label sizes $|\mathcal{X}_c|$ and $|\mathcal{Y}_c|$ are also related to how primes are assigned to forwarders and SF types. While *CRT-Chain* already enables path segmentation to minimize the maximum prime used in the system, we can further reduce the header overhead by decreasing the probability of using those large primes. More specifically, so far we assume that primes are randomly assigned to forwarders and SFs. However, if, unfortunately, a large prime is used by a forwarder or SF type that is traversed frequently, many paths going through those popular SFs with large primes can output large labels.

To reduce the header size for all the chains in $\mathcal{C}$ in a probabilistic way, we propose to assign primes to forwarders and SFs according to their *popularity* (or loading). Intuitively, a frequently-used forwarder (SF type) is more likely to be included in a path $P(c)$ (SFP $SP(c)$). Hence, to minimize the expected header size of service chains, we should assign small

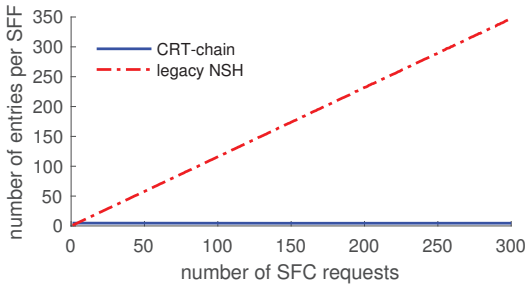


Fig. 3: Impact of the number of SFC requests on flowtable usage.

primes to heavy loaded forwarders and popular SF types, while letting less used ones have large primes. To this end, we count the number of chains that traverse through a forwarder $f \in \mathcal{F}$ (SF $s \in \mathcal{S}$), denoted by the *popularity score* w_f (w_s), and sort $f \in \mathcal{F}$ ($s \in \mathcal{S}$) in descending order of their popularity w_s (w_f). Each forwarder f (SF type s) is then assigned a unique prime, from small to large, in order.

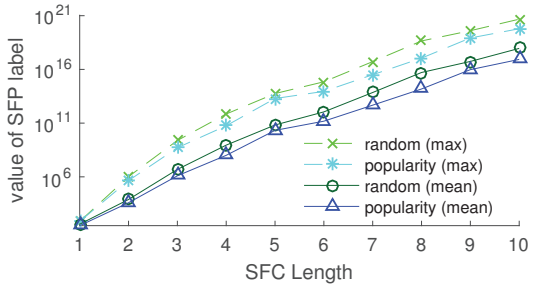
If *CRT-Chain* allows prime reuse, we just need to replicate a prime for $\lceil 1/\alpha \rceil$ times, $0 < \alpha \leq 1$ and, again, sort all the replica in ascending order, which are then assigned to the forwarders (SFs) in descending order of their popularity. By doing this, we can enable popularity-aware prime assignment even when forwarders (SFs) share $\alpha|\mathcal{F}|$ ($\alpha|\mathcal{S}|$) primes.

V. Implementation

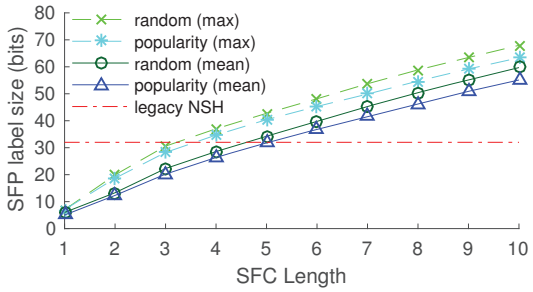
We implement a simulation framework of *CRT-Chain*. The physical switches, each with four ports, are connected as a 4-pod fat-tree topology with the block factor (i.e., the ratio of the number of downlink ports to that of uplink ports) set to 1. Namely, a network consists of 20 switches. Each of the eight edge switches is connected to two physical servers. Each of the 16 servers has a virtual switch acting as an SFF that connects to four VMs, each supporting a service, i.e., an instance of any $s \in \mathcal{S}$. Each forwarder (switch or SFF) is assigned a prime based on Section IV-D.

Note that, in our implementation, a network accommodates 64 VMs and, hence, is capable of supporting 64 SF instances in total. We deploy SF instances in VMs based on two strategies: 1) *random* and 2) *popularity-aware*. For random deployment, all the SFs have an equal number of instances, i.e., $64/|\mathcal{S}|$, and each VM installs an instance of an SF randomly selected from $\mathcal{S}$. For popularity-aware deployment, we randomly assign each SF type a popularity score w_s (i.e., request probability). The popularity scores of all SF types $s \in \mathcal{S}$ are normalized to 1, i.e., $w_s = w_s/(\sum_{s' \in \mathcal{S}} w_{s'})$. We then make the number of instances of SF s proportional to its popularity w_s, but ensure every SF s has at least one instance. Roughly speaking, $\lceil 64 * w_s \rceil$ instances are created for each SF type s, and each is deployed in a randomly selected VM.

To generate an SFC of length l, we iteratively pick an SF type from $\mathcal{S}$ and combine the l randomly selected SF types as a chain. We use the popularity w_s to sample a requested SF. *Random* sampling is equivalent to assigning all the SF types an equal popularity. Since our work assumes that the physical routing path and the service function path of an SFC is given,

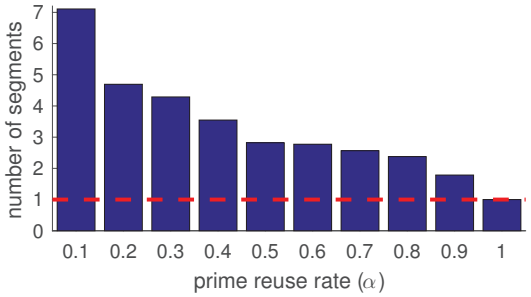


(a) value of SFP label $\mathcal{Y}$

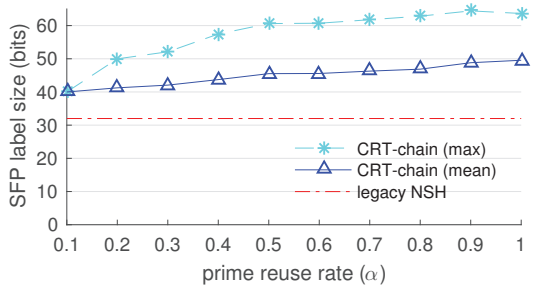


(b) size of SFP label $\mathcal{Y}$

Fig. 4: Impact of the length of SFCs.



(a) average number of segments



(b) total overhead of all SFP sub-labels $\mathcal{Y}_i$

Fig. 5: Impact of prime reuse rate α.

we need to further apply some algorithms to determine the path and SFP for each SFC. Here, we implement a simple and practical algorithm as follows: the SFP $SP(c)$ is initialized as an empty set, and, for each SF request s in the chain c in order, we pick the lightest loaded instance of s and insert it into $SP(c)$. Any pair of consecutive SF instances in $SP(c)$ are then connected by a shortest subpath found by the Floyd-Warshall Algorithm [19], and all the subpaths are combined into the final routing path $P(c)$.

VI. Performance Evaluation

We conduct extensive numerical studies via simulations to evaluate the performance of *CRT-Chain*. Unless otherwise stated, the number of SF types and the SFC length are set to 16 and 10, respectively, by default. Since our CRT-based SFP label $\mathcal{Y}$ can be combined with any routing protocol, we mainly focus on comparing the overhead of $\mathcal{Y}$ to that of the 32-bit service path field in the legacy NSH. We will finally check the overall CRT-based header size as the SFP label $\mathcal{Y}$ is combined with the CRT-based routing label $\mathcal{X}$. In each configuration, we generate 1,000 SFC requests of the same length and report the average result of 100 random runs.

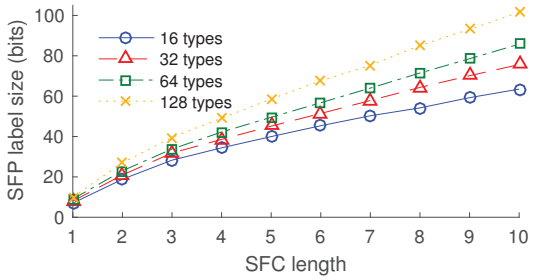
A. Impact of the Number of SFC Requests

We first verify whether *CRT-Chain* can be scalable as supporting an increasing number of SFC requests. Fig. 3 illustrates the average number of flowtable entries inserted into an SFF by conventional service chaining and *CRT-Chain*, respectively, for various numbers of SFCs. The results verify that the number of flowtable entries required by legacy chaining grows linearly as the number of requests increases. However, *CRT-Chain* only needs a few rules representing the primes assigned to each SFF and its associated SFs. Hence, *CRT-Chain* only requires a constant number of flowtable entries in each SFF, regardless of how many service chains going through it. This supports that *CRT-Chain* is especially efficient when a system needs to serve a large number of service chains but is short of TCAM resources in some switches.

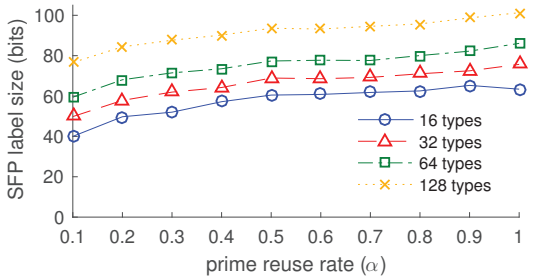
B. Impact of the Length of SFCs

We next examine the impact of the SFC length on *CRT-Chain*'s SFP label size. Recall that, as encoding an SFP label $\mathcal{Y}_c$, the step counts of the SF instances along the SFP are used as the remainders. In addition, the prime assigned to each SF type should be no smaller than the maximum step size, i.e., the SFC length. As a result, a longer SFC could lead to a larger label size $|\mathcal{Y}_c|$. The objective of this simulation is to check how the SFP label size $|\mathcal{Y}_c|$ grows as an SFC gets longer.

Figs. 4(a) and 4(b) show the value of label $\mathcal{Y}_c$ and the label size in bits, respectively, for various SFC lengths. We observe that the label size increases nearly lineally as the SFC length grows. The popularity-aware prime assignment explicitly considers the access probability of each SF type, and, hence, results in a smaller average label size since the popular SFs can use small primes. More specifically, for an SFC including 5 SFs, the label $\mathcal{Y}$ consumes only 34 bits and 32 bits, on average, in random assignment and popularity-aware assignment, respectively. We also report the maximum possible overhead of label $\mathcal{Y}$, which is found by calculating the label for a service chain with l SFs assigned the l largest primes. The results show that, for a chain with 5 SFs, the maximum label sizes in random assignment and popularity-aware assignment are 43 bits and 40 bits, respectively, which are only slightly longer than the legacy 32-bit NSH. This confirms that *CRT-*



(a) various SFC lengths



(b) various prime reuse rates α

Fig. 6: Impact of the number of SF types.

Chain can effectively enable flowtable-free chaining at a fairly small expense of additional overhead.

C. Impact of Prime Reuse and Path Segmentation

We now examine how path segmentation reduces the label overhead when the prime reuse rate α (defined in Section IV-C) varies from 0.1 to 1. In particular, each prime is used by $\lceil 1/\alpha \rceil$ SF types. To avoid forwarding errors, we then leverage *CRT-Chain*'s path segmentation to encode sub-labels such that all the SFs in each sub-SFP have unique primes. Fig. 5(a) plots the average number of SFP segments for various settings of α. As expected, when the prime reuse rate decreases, more SF types use the same prime. Hence, we might need to partition a path into an increasing number of segments. The figure however shows that we only need around 3 segments, on average, even when α is as low as 0.4.

To understand how the number of segments affects the overall label overhead, we further plot in Fig. 5(b) the total label size of all the segments along an SFP for various prime reuse rates. Note that the case of $\alpha = 1$ indicates the *CRT-Chain* scheme without segmentation. The results show that the overall overhead actually decreases continuously, rather than increases, as the prime reuse rate is decreased, resulting in more segments. The average overhead reduction can be up to 20%. The main reason is that, though we need more sub-labels for the partitioned segments, the size of each sub-label becomes much smaller as the SFs can share the smaller primes.

D. Impact of the Number of SF Types

We further examine the impact of the number of SF types on the SFP label size, with and without path segmentation. Fig. 6(a) plots the impact of the SFC length on the SFP label size $|\mathcal{Y}|$ when the number of SF types is set to 16, 32, 64, and 128, respectively. In this simulation, we apply our popularity-aware prime assignment, but disable prime reuse and path segmentation. The results show that, similar to the trend shown in Fig. 4(b), the label size, in general, grows as the SFC length increases. When more SF types are supported in the system, the label size is further increased as some SFs are now assigned large primes. However, the average label size is only increases slightly when the number of SF types increases from 16 to 128.

We also plot in Fig. 6(b) the impact of the prime reuse rate α on the SFP label size $|\mathcal{Y}|$ for various numbers of SF types. In this simulation, the SFC length is set to 10. The figure shows that the amount of overhead reduction from reusing primes is to some extent independent of the number of SF types. More specifically, no matter how many SF types a system supports, reusing primes reduces the label size by around 25 bits at most. This implies that the percentage of overhead saving is higher when there are fewer SF types.

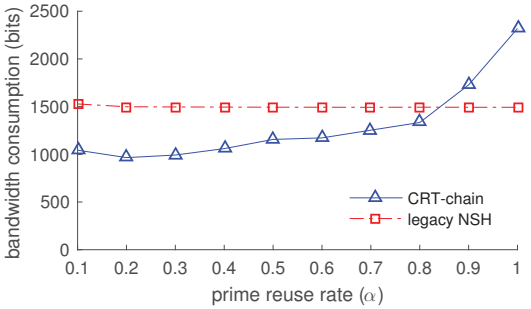


Fig. 7: Impact of prime reuse rate α on bandwidth consumption.

E. Impact of Segmentation on Bandwidth Consumption

Another benefit of path segmentation is that it also helps reduce the overall overhead bandwidth assumption when the last hop of each segment can drop the sub-label of the current subpath before forwarding a packet to the next subpath, making the header become shorter continuously as the packet traverses through its end-to-end path. We hence examine how discarding sub-labels saves the overall bandwidth consumption, which is defined as the total number of header bits sent over any network link. Specifically, a header bit sent through an l-hop path will consume bandwidth resources of l bits.

Fig. 7 illustrates that, as compared to no segmentation (i.e., $\alpha = 1$), segmentation reduces the overall bandwidth consumption by up to 64%. The bandwidth saving increases as the prime reuse rate α decreases due to the fact that a smaller reuse rate cuts a path into more smaller segments, increasing the opportunities of dropping sub-labels in the middle of the path. It is also worth noting that, while the initial size of *CRT-Chain*'s header is mostly larger than the legacy NSH length (32 bits), as shown in Fig. 5(b), *CRT-Chain*'s header bandwidth consumption is however much smaller than that required by the legacy NSH when α is less than or equal to 0.8. We hence conclude that, by enabling prime reuse and segmentation, *CRT-Chain* not only saves flowtable entries significantly but also reduces the header bandwidth consumption required by service chaining, i.e., total overhead of NSHs.

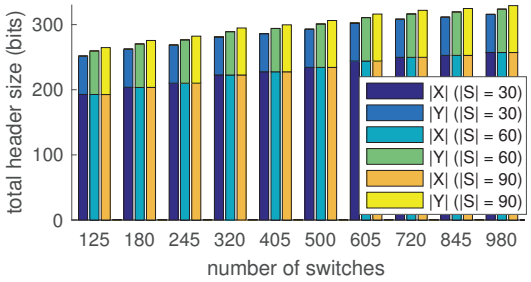


Fig. 8: Overall CRT-based label size.

F. Overall Overhead

We finally examine the total header size when we combine our CRT-based SFP forwarding with CRT-based routing, i.e., the size of $(\mathcal{X}, \mathcal{Y})$. In this simulation, we disable segmentation and change the port of each switch from 10 to 28, which corresponds to the number of switches from 125 to 980 in a fat-tree topology. Fig. 8 plots the average header size for different network scales when the system supports 30, 60, and 90 SF types, respectively. The results show that the path label size $|\mathcal{X}|$ grows slightly as the number of switches/SFFs increases since each one is more likely to use a large prime. In addition, an increasing number of switches (pods) in the fat-tree topology also increases the chances of having a path across different pods, as a result further increasing the path length and enlarging its path label. However, since the SFP length is independent of the path length, the SFP label size $|\mathcal{Y}|$ hence stays constant for the same number of SF types. That is, the SFP label length $|\mathcal{Y}|$ is mainly determined by the number of SF types.

We can also see that the path label size $|\mathcal{X}|$ is much larger than the SFP label size $|\mathcal{Y}|$. This is because the label size is closely related to the maximum prime used for calculating the label. As the number of forwarders is typically much larger than the number of SF types, the maximum prime assigned to forwarders is much larger than the maximum prime assigned to SF types, therefore leading to a larger path label. That is to say, for each SFC, enabling CRT-based SFP requires less overhead and is, hence, more efficient than enabling CRT-based routing. Therefore, when a forwarder is short of flowtable entries, we should favor replacing SFP forwarding rules with SFP labels $\mathcal{Y}$ over replacing routing rults with path labels $\mathcal{X}$.

VII. Conclusion

In this paper, we have presented *CRT-Chain*, a CRT-based service function chaining protocol. *CRT-Chain* leverages the Chinese remainder theory to encode both the routing path and the SFP of an SFC into small labels. Switches and SFFs can forward a packet by simply extracting the forwarding rules from the labels attached in its NSH, without requiring to insert forwarding rules for every individual SFC request. While *CRT-Chain* only inserts the information about the primes allocated to a forwarder and its associated SFs into its flowtable, it only needs a constant number forwarding rules, regardless of how many SFC requests served in a system. We present extensive results comparing the protocol efficiency of *CRT-Chain* to that of the legacy scheme, and demonstrate that *CRT-Chain* can efficiently support flowtable-free chaining at a fairly small cost of additional header overhead. By further enabling path segmentation, we further reduce the overall bandwidth consumption to a level smaller than the legacy NSH, showing that *CRT-Chain* not only saves flowtable usage but also reduces the signaling overhead.

Acknowledgement

This work was supported in part by the Ministry of Science and Technology of Taiwan under grant numbers MOST 106-2628-E-009-004-MY3 and Inventec.

References

[1] X. Zhao, Y. Liu, L. Wang, and B. Zhang, "On the aggregatability of router forwarding tables," in *Proc. IEEE INFOCOM*, 2010.

[2] Q. Li, D. Wang, M. Xu, and J. Yang, "On the scalability of router forwarding tables: Nexthop-selectable FIB aggregation," in *Proc. IEEE INFOCOM*, 2011.

[3] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: Distributing tables in software-defined networks," in *Proc. IEEE INFOCOM*, 2013.

[4] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "CacheFlow: Dependency-aware rule-caching for software-defined networks," in *Proc. ACM SOSR*, 2016.

[5] A. Viswanathan, N. Feldman, Z. Wang, and R. Callon, "Evolution of multiprotocol label switching," *IEEE Commun. Mag.*, vol. 36, no. 5, pp. 165 – 173, May 1998.

[6] P. B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica, "Pathlet routing," in *Proc. ACM SIGCOMM*, 2009.

[7] O. Rottenstreich, M. Radan, Y. Cassuto, I. Keslassy, C. Arad, T. Mizrahi, Y. Revah, and A. Hassidim, "Compressing forwarding tables for datacenter scalability," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 1, pp. 138–151, 2014.

[8] K. Agarwal, C. Dixon, E. Rozner, and J. Carter, "Shadow MACs: Scalable label-switching for commodity ethernet," in *Proc. ACM HotSDN*, 2014.

[9] A. Schwabe and H. Karl, "Using MAC addresses as efficient routing labels in data centers," in *Proc. ACM HotSDN*, 2014.

[10] H. Wessing, H. Christiansen, T. Fjelde, and L. Dittmann, "Novel scheme for packet forwarding without header modifications in optical networks," *IEEE/OSA J. Lightw. Technol.*, vol. 20, no. 8, pp. 1277 – 1283, Aug. 2002.

[11] M. Martinello, M. Ribeiro, R. E. Z. de Oliveira, and R. de Angelis Vitoi, "KeyFlow: a prototype for evolving SDN toward core network fabrics," *IEEE Network*, vol. 28, no. 2, pp. 12 – 19, 2014.

[12] Y. Ren, T.-H. Tsai, J.-C. Huang, C.-W. Wu, and Y.-C. Tseng, "Flowtable-free routing for data center networks: A software-defined approach," in *Proc. IEEE GLOBECOM*, 2017.

[13] J. M. Halpern and C. Pignataro, "Service Function Chaining (SFC) Architecture," RFC 7665, October 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc7665.txt

[14] M. C. Luizelli, W. L. da Costa Cordeiro, L. S. Buriol, and L. P. Gaspary, "A fix-and-optimize approach for efficient and large scale virtual network function placement and chaining," *Comput. Commun.*, vol. 102, pp. 67–77, apr 2017.

[15] H. Huang, S. Guo, J. Wu, and J. Li, "Service chaining for hybrid network function," *IEEE Trans. Cloud Comput.*, vol. PP, no. 99, pp. 1–1, 2017.

[16] Z. Xu, W. Liang, M. Huang, M. Jia, S. Guo, and A. Galis, "Approximation and online algorithms for NFV-enabled multicasting in SDNs," in *Proc. IEEE ICDCS*, 2017.

[17] T. W. Kuo, B. H. Liou, K. C. J. Lin, and M. J. Tsai, "Deploying chains of virtual network functions: On the relation between link and server usage," in *Proc. IEEE INFOCOM*, 2016.

[18] A. Gushchin, A. Walid, and A. Tang, "Enabling service function chaining through routing optimization in software defined networks," in *Proc. IEEE Annu. Allert. Conf.*, 2016.

[19] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, 2nd ed. McGraw-Hill Higher Education, 2001.