A Flexible vCPE Framework to Enable Dynamic Service Function Chaining Using P4 Switches

Muthuraman Elangovan
EECS IGP
National Yang Ming Chiao Tung University
Hsinchu, Taiwan
muthuraman.e@hotmail.com

Chien Chen
Department of CS
National Yang Ming Chiao Tung University
Hsinchu, Taiwan
chienchen@cs.nctu.edu.tw

Jyh-Cheng Chen
Department of CS
National Yang Ming Chiao Tung University
Hsinchu, Taiwan
jcc@cs.nctu.edu.tw

***Abstract* – The Virtual Customer Premises Equipment (vCPE) technology has recently emerged to reduce telecom operators' OPEX and CAPEX. It evolved from Network Function Virtualization (NFV) and Software Defined Networking (SDN) technologies. This paper proposes a flexible vCPE framework to enable dynamic chaining of Virtual Network Functions (VNF) by using Programmable Protocol-independent Packet Processors (P4) switches. It can obtain a much better packet processing performance compared to the pure software vCPE solution. The OpenFlow switch provides similar hardware acceleration, but it executes the flow tables in a fixed order, and it is not possible to skip the flow tables, which are not subscribed to by a customer. However, P4, the domain-specific language used to describe how to process packets on a data plane (DP), gives more flexibility than the OpenFlow. It provides a way to avoid the fixed flow table execution order and can introduce a new service on the fly. Our flexible vCPE framework can be achieved by the synergies between an NFV controller on the cloud and a P4 switch at the edge. In this paper, an Open Network Operating System (ONOS) controller with P4Runtime is used as a VNF control plane and a P4 Behavioral Model (BMv2) software switch is used as a DP. Finally, dynamic function chaining is realized using three possible implementations, viz. multi-instance, clone, and resubmit. P4 language is used for the implementations. Experimental results show that a multi-instance based solution is better than a resubmit and clone.**

Keywords – SDN, NFV, P4, ONOS, vCPE, VNF, BMv2

I. INTRODUCTION

Recent research focuses on evolving the network infrastructures based on Software Defined Networking (SDN) [1]-[4] and Network Function Virtualization (NFV) [7]-[9]. The core concept of SDN is separating the control and data plane operations that enable smart control on the switch and giving a vantage point for innovation in network industries.

NFV was introduced by Telco operators. It is a marvelous choice for innovation in the service delivery arena. It is used to reduce the coupling between Network Function (NF) and hardware devices. For example, legacy network services at customer premises are based on a hardware appliance called Physical Customer Premises Equipment (P-CPE). This P-CPE includes all functions that are required to provide services for an end-user such as Network Address Translation (NAT), Firewall (FW), Quality-of-Service (QoS), etc. But the demands and requirements from the end-user are more diverse and unique. Adding new services to fulfill end-user needs requires that the service providers replace new P-CPEs. It also needs extra deployment labor cost, which takes a lot of supply chain effort and time. All these operations increase their OPEX and CAPEX. Due to high competition, the service providers can't increase their subscription fees for those additional expenses. Thus, they are forced to find a way to reduce their OPEX and CAPEX.

The vCPE [10] concept has emerged as a solution for improved delivery of NF, which helps to move hardware functionality to the cloud as a software module. It also helps service providers introduce new services faster without rolling out existing hardware appliances located on the customer premises. It can provide Dynamic Service Function Chaining (DSFC) using SDN. However, pure software-based vCPE cannot achieve the needed throughput and latency for today's network services. Therefore, people [12] have explored the possibility of separating the control and data functions of vCPE and using OpenFlow hardware switches to improve the packets' processing speed in the data plan of vCPE.

The authors in [12] proposed a multiple OpenFlow [5][6] tables mechanism to enable VNFs in vCPE with faster packet processing in the data plane (DP). OpenFlow has provided programmability of the network control plane, but the network DP is still rigid, without the flexibility that vCPE is looking for. In [12], each table represents one VNF service. Each table has fixed header match fields like a 5-tuple entity. By default, the packet processing pipeline of OpenFlow starts with the first VNF table to the end of VNF in order. In [12], the dynamic chaining of VNF is achieved by adding a high-priority rule such as Goto action in such a way that we can disable the number of VNFs for a customer. In case a large number of customers like to design it by their own packet processing order of VNF, then it requires a numbered Goto action rule on each enabled VNF table. Furthermore, at a later time, suppose a customer wants to either enable some more VNFs or change their execution order; this requires a number of table entries. It will consume a lot of memory, and it makes for a complex packet processing pipeline of VNF.

Instead of using OpenFlow hardware switches in the DP of vCPE, this paper proposes a flexible framework of Programmable Protocol-independent Packet Processors (P4) [16], [17] switches to DSFC in vCPE. P4 switches give a way to reconfigurable NF tables in the DP of vCPE. P4, a high-level domain-specific language, describes how to process packets on a DP, and gives more flexibility than the OpenFlow protocol. We use the programmability of P4 switches to provide the high-performance and flexibility to vCPE. For example, a customer chooses the list of VNF from a number of VNF supported by the vCPE device, and they can specify the execution order of chosen VNF. In this way, the DP can apply a number of VNF based on the customer's chosen in order. Moreover, P4 provides a way to reconfigure the forwarding pipeline table in a P4-capable device and introduce a new protocol parser at runtime. Therefore, we can add or remove a list of VNF tables to achieve faster rolling out of the new services of vCPE in the field.

There are a number of related works done to support DSFC using multiple P4 programs in a single physical DP. It is achieved by virtualization, which involves virtualizing the multiple network contexts with different VNFs chaining. Hyper4 [13] proposed portable virtualization of DP in order to run independently. However, this approach requires more match-action stages and limits the number of primitive actions. P4Visor [14] proposed a method to merge different P4 programs into one and deploy them as a single pipeline of a device. This approach [14] only focused on reducing resource consumption by removing the redundancy of the parser and tables while merging different P4 programs into one. Unfortunately, all customers' network traffic has to go through all tables. P4SC [15] proposed a method to convert the given service function chaining request into a P4 Program. It leveraged a Least Common Subsequence (LCS) algorithm to merge multiple NF chains into a single chain to minimize the number of pipeline table instances. FASE [23] proposed a method to reduce redundant SFs by using re-circulation and it will take a longer completion time in the worst case.

To provide DSFC in vCPE, for the n number of VNFs, $\{v_1, v_2, v_3, \ldots, v_n\}$, a limited set of combinations of VNF chaining can be predefined. Then, the customers have to choose their packet processing pipelines from among them. However, the customer is limited to designing any chaining of packet processing pipelines from n VNFs. Therefore, in this paper we assume that our flexible vCPE framework will support any combinations of n VNFs. Since the number of combinations could reach at most $S(n) = n!\ (1/0! + 1/1! + \ldots + 1/(n-1)!)$ with n NFs, such as 3 VNFs $\{v_1, v_2, v_3\}$, there exist six possible combinations of three VNFs: $\{v_1, v_2, v_3\}$, $\{v_1, v_3, v_2\}$, $\{v_2, v_1, v_3\}$, $\{v_2, v_3, v_1\}$, $\{v_3, v_1, v_2\}$, $\{v_3, v_2, v_1\}$; six possible combinations of two VNFs: $\{v_1, v_2\}$, $\{v_1, v_3\}$, $\{v_2, v_1\}$, $\{v_2, v_3\}$, $\{v_3, v_1\}$, $\{v_3, v_2\}$; and three possible combinations of a single VNF: $\{v1\}$, $\{v_2\}$, $\{v_3\}$. Therefore, the parallel requires $S(3) = 15$ pipeline table instances in order to satisfy all customer requests. To merge pipeline tables with $S(n)$ combination of n VNFs, the P4SC could produce a very complex P4 control flow to chain them. For example, merging the chaining of combination of three VNFs, the result of LCS of $\{v_1, v_2, v_3\}$ and $\{v_1, v_3, v_2\}$ is either $v_1\, v_2$ or $v_1\, v_3$, then the pipeline table in P4 after merging could be $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow d_2$ or $v_1 \rightarrow v_3 \rightarrow v_2 \rightarrow d_3$ where d denotes the duplicate table and the suffix represents a VNF. Furthermore, merging $\{v_2, v_1, v_3\}$ with the chain $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow d_2$, the result of LCS could be either $v_2\, v_3$ or $v_1\, v_3$, then the resultant chain would be $v_1 \rightarrow v_2 \rightarrow d_1 \rightarrow v_3 \rightarrow d_2$ and there exist three duplicate tables. Likewise, if we merge more combination of VNFs then the output of P4SC P4 program [22] is more complex control blocks, and the length of the P4 pipeline table increases due to chaining the multiple combinations of VNFs based on LCS. Since the P4 grammar rule doesn't allow calling the same table multiple times in the P4 packet processing pipeline, it will force P4SC to produce the overhead in terms of duplicate P4 tables which will increase the length of pipeline tables. Therefore, each VNF chain could experience more packet processing delay while traveling through the pipeline.

Motivated by the inefficient complex chaining of VNFs in Openflow and P4SC, we aim to chain the number of VNFs more effectively under the vCPE environment. In general, we need at least n^2 VNF table instances to satisfy all possible combinations of n VNFs. In our design, with n^2 VNF table instances, we assign a unique bitmap identifier for each VNF. The design of our own scheme called *multi-instance,* depicted in Fig. 1, requires 9 table instances for 15 DSFCs of 3 VNFs. Since the P4 grammar rule doesn't allow calling the same table multiple times in the P4 packet processing pipeline, we duplicate two more table instances with different names for each VNF. Each customer can subscribe to a list of VNFs and specify the execution order of VNFs. Each customer's network traffic is applied over multiple VNF tables based on the stream of bitmap value on the user-defined metadata in P4. The packet processing pipeline of a customer will be based on their subscription of NFs, as shown in Fig. 1. In this way, we can skip the execution of unsubscribed VNF tables. Therefore, the pipeline delay will be exactly same as the number of NFs each customer is subscribed to. The customer can enable/disable one or more of their subscriptions and also change the execution order at runtime without modifying the running program in DP.

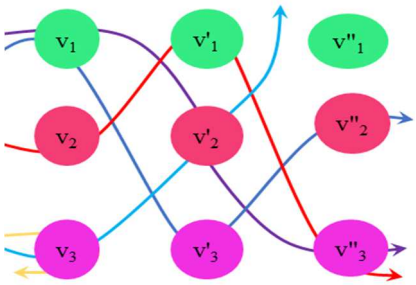


Fig. 1. Example of multi-instance approach

To further reduce the number of tables, we invest the DSFC in vCPE implementations based on the P4 primitives *resubmit,* and *clone* which require only n table instances with n NFs.

This paper evaluates the proposed framework model by using P4 BMv2 [18] with P4Runtime software switch as a DP. P4Runtime is a communication protocol between control and DP that is used to manipulate the DP operations. The ONOS [11], [19] is used as the SDN control plane. It offers high-availability, scale-out, and distributed management of network elements. Experimental results demonstrate that the *multi-instance* pipeline model achieves better throughput and pipeline latency of DSFC than the *resubmit* and *clone* pipeline model.

The rest of this paper is organized as follows. Section II discusses the system design and implementation. Section III introduces the DSFC. Section IV presents the relevant experimental results. Section V draws conclusions and describes future works.

II. System Design And Implementation

In this section, we present a brief overview of our system design and implementation of the control plane application.

The control plane application of the proposed vCPE is written in java on top of the ONOS SDN controller core module. The service provider can define the forwarding pipeline of the vCPE device, and compile it using the P4 compiler (p4c). The compiled output for the BMv2 software switch files p4info and json are given to the ONOS controller, either ONOS compile-time or run time through a vCPE CLI command. The ONOS controller will initiate a gRPC connection over TCP/P4Runtime protocol to the vCPE device. On successful connection establishment, the ONOS will update the forwarding pipeline of that device. The simple P4 vCPE dynamic service function deployment model is depicted

in Fig. 2. The vCPE device can be deployed at the edge of the service provider's network.

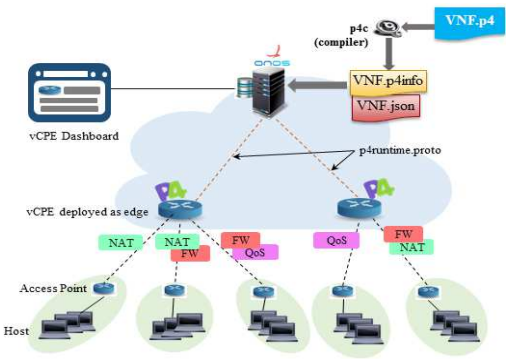


Fig. 2. Simple vCPE Dynamic service function deployment model

The customer can subscribe to the list of VNF services and also specify the order of VNF execution through the dashboard. Once the subscription is done, the ONOS controller will update the required P4 flow entries on to the corresponding edge device. In this way, each customer has their subscription of VNF services. The VNF packet processing is based on each customer's subscriptions. Each green area in Fig. 2 is considered a customer local network domain, and their subscriptions and execution orders are shown on top of the link.

The service provider can easily implement new services or remove existing services to/from the vCPE device through our vCPE CLI command. The service provider has to rewrite the forwarding pipeline of a vCPE device and pass it to the vCPE CLI command. The vCPE control plane application will update the bitmap identifier for a newly added VNF and the disabled VNF service bitmap identifier will be kept as reserve. Finally, the vCPE control plane application will reconfigure the forwarding pipeline of the vCPE device. The reconfiguration and populating of the flow table entries takes a reasonable amount of network downtime.

III. Dynamic Service Function Chaining

In this section, we first present the classification of customer traffic in P4 for different NFV subscriptions. After that, we propose a dynamic service function chaining for packet pipeline processing. Finally, we show the various possibilities of the dynamic service function chaining model in P4.

A. Classification of Customer Traffic

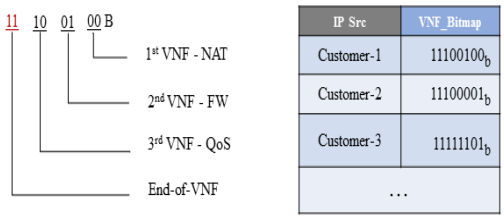


Fig. 3. VNF bitmap value and classification table format

Before discussing the classification of customer traffic, we explain VNF bitmap identifiers and table execution orders. As displayed in Fig. 3, there are a total of three VNFs, which are NAT, FW, and QoS, and each can be subscribed to by customers. Each VNF has a unique bitmap identifier, which is used to make a big-chunk of bitmap value for each customer traffic classification. The VNF table execution always starts from the right to the left of the bitmap value. We also define one more bitmap identifier 11b (*End-of-VNF),* to terminate the execution of VNF. In this example with three VNFs, we require 2bits of value to encode a VNF uniquely.

The classification of customer traffic is done based on the source IPv4 address of incoming network traffic, depicted in Fig. 3. The flow entry of the classification table includes the IPv4 address of the customer as a match field, and the subscribed VNF bitmap value *VNF_Bitmap* as an action parameter to set the user-defined metadata for each customer's packet accordingly. For example, in Fig. 3, the Customer-1's *VNF_Bitmap* value is 11100100b, which denotes that the customer is subscribed to NAT, FW, and QoS, and the table execution order is also the same as a subscription order.

B. DSFC

Fig. 4 depicts the overall DSFC for packet pipeline processing over multiple VNF tables.

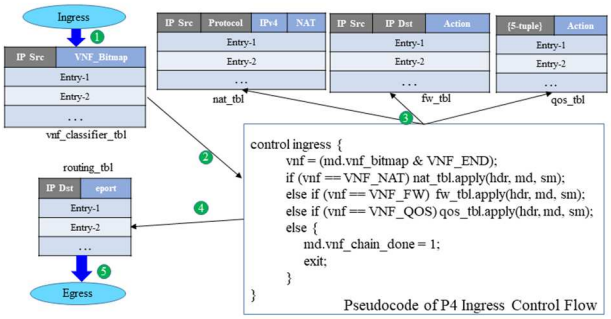


Fig. 4. Example of DSFC packet processing pipeline flow

In this vCPE implementation, we have three VNFs, which are NAT, FW, and QoS. First, we explain the flow tables shown in Fig. 4 to evaluate DSFC. The in-built table *vnf_classifier_tbl* is used to get the customer's VNF subscription bitmap value. The *nat_tbl* is a stateless NAT table that translates the source or destination IPV4 address. The *fw_tbl* is a matched packet marked as drop that then terminates the execution of the remaining list of VNFs. The *qos_tbl* is a state full QoS that implements QoS based on the P4 *meter* primitive. The in-built table *routing_tbl* is a simple L3 routing table. The VNF validation process gets the VNF by right-shifting the shift bits from *VNF_Bitmap*. The right-shifted value could be any one VNF and applies the corresponding VNF table. This process will continue until we met a condition like VNF_END, dropping a packet, etc.

C. VNF Pipeline

In this paper, we propose three VNF pipeline models called *resubmit*, *clone*, and *multi-instance* using the P4 switch. We used P4 V1Model architecture to evaluate our proposed DSFC. P4 V1Model has six pipeline controls, namely *Parser*, *VerifyChecksum*, *Ingress*, *Egress*, *ComputeChecksum,* and *Deparser*. In P4 V1model, the programmable Match+Action pipeline can be done at the *Ingress* and *Egress* control. There is a challenge to implement the packet processing pipeline over multiple VNF tables. Currently, P4 grammar doesn't allow invoking the same table more than one time in the Match+Action pipeline because of potential looping issues while doing the grammatical checks. We can solve this issue by using P4 primitives like *resubmit, clone,* and *recirculate*. The *recirculate* primitive is used to recirculate the packet to the *Parser*, *Ingress* control and so on once the packet finishes

the pipeline processing which is not fit for DSFC. We define a model called *multi-instance* model which creates multiple table instances for a VNF, where each instance is uniquely identified by its P4 table identifier. The P4 table identifier is used to update the flow rule entry on the specific table.

1) DSFC based on resubmit: The P4 *resubmit* primitive function should be called from *Ingress* control and accepts a list of parameters like user-defined metadata. In this pipeline model, we use a single table instance for a VNF. The DSFC is implemented in *Ingress* control, as illustrated in Fig. 5.

In Fig. 5, *Classification* denotes the *vnf_classifier_tbl;* the "C" indicates conditional DSFC algorithm; and *nat1*, *fw1*, and *qos1* represent the VNF tables of NAT, FW, and QoS respectively. When a packet is entered into the *Ingress*, we check the packet's standard metadata, whether that packet instance is marked with a resubmit flag or not. If the packet wasn't marked, we then apply the *Classification* table to mark the customer's VNF subscription bitmap. The DSFC algorithm will right-shift the shift-bits from the bitmap and then apply the corresponding VNF table. Once the table execution is done, we then check the remaining bitmap, whether it reaches VNF_END or not. If the bitmap reaches VNF_END, then the packet will move to the *Egress* control. Otherwise, the packet will be resubmitted to process the remaining list of VNFs. The user-defined metadata keeps that packet's updated VNF bitmap value in the cycle.

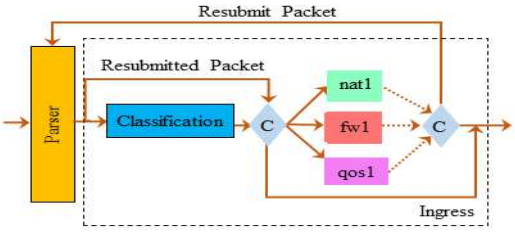


Fig. 5. DSFC based on P4 resubmit primitive function

2) DSFC based on clone: P4 has two cloning primitives, namely *clone,* and *clone3*. Both primitive functions are used to create a new copy of the packet. The original and cloned packets are processed independently. There are two types of cloning supported by P4 specifications, which are *Ingress-to-Egress* (I2E) and *Egress-to-Egress* (E2E). We rely on *clone3* and E2E type of cloning, which is used to execute the *Egress* control, repeatedly, as in Fig. 6. This pipeline model is also similar to resubmit, but the major difference is that it avoids reparsing the header fields.

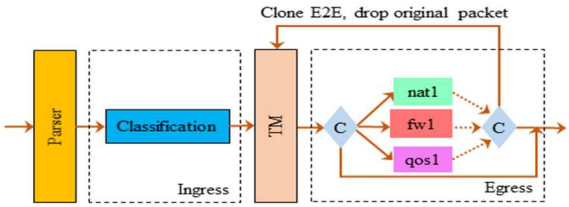


Fig. 6. DSFC based on P4 clone primitive function

In this model, the VNF subscription bitmap, routing, and cloning specific information get it in the *Ingress* control. The VNF bitmap processing is the same as the *resubmit* pipeline model. If more VNFs are to be processed, then the packet will be cloned and marked the original packet to drop. In this pipeline model, we use a single table instance for each VNF.

3) DSFC based on multi-instance: The DSFC placed in the *Ingress* Match+Action pipeline is illustrated in Fig. 7. We create multiple table instances of each VNF table like *nat1*, *nat2*, and so on. The traversal of the VNF subscription bitmap is the same as *resubmit* and *clone*, but the VNF execution is different. In this model, we define multiple instances of a VNF and execute sequential but separate table instances of a VNF. For example, if the value of the VNF bitmap is FW, NAT, and QoS, then the table executions will be *fw1*, *nat2*, and *qos3*.

This pipeline model will apply a number of VNFs sequentially unlike the *resubmit* and *clone* pipeline models. However, the drawback of the *multi-instance* model is that the number of VNF tables needed will be increased by *n*-fold, where *n* is the number of VNFs supported by the service provider. For example, in Fig. 7, where the service provider provisions three VNFs: NAT, FW, and QoS, we need to create 9 table instances in total. Since we can't predict the customer subscription and execution order, we can't assume different table instances will have different table sizes. We have to assign the maximal table size for all of the VNF table instances initially. We will end up with unusable memory in some table instances. Eventually it will introduce table fragmentation issues. Fortunately, we can still take advantage of the field-reconfigurable P4 switch to adjust the table size as necessary.

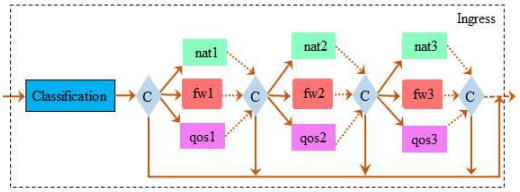


Fig. 7. DSFC based on multi-instance

IV. Performance Evaluation

Our development environment for vCPE includes a P4 BMv2 software switch as a DP, and the control plane is a suite with an ONOS SDN controller. We explain our results with a discussion about the three proposed DSFC pipeline models of vCPE.

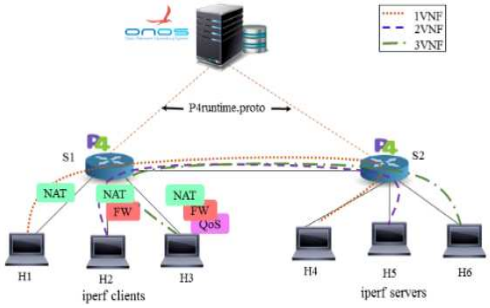
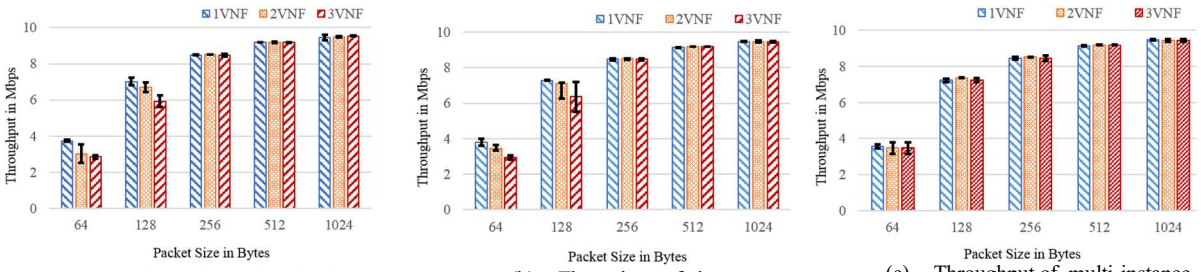


Fig. 8. Evaluation setup

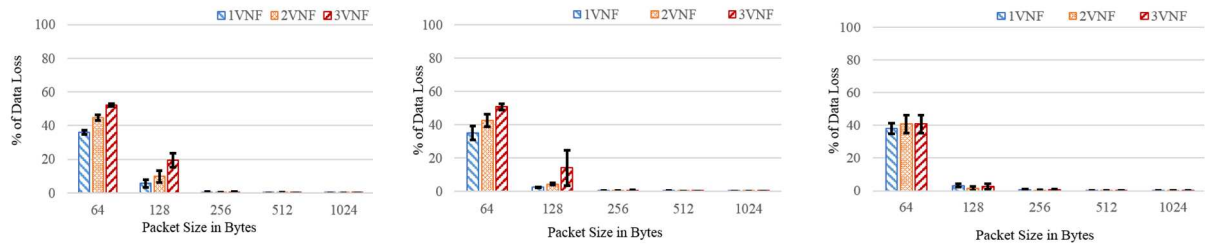
A. Evaluation Setup

We run the vCPE in Ubuntu 16.04 OS on a commodity Intel PC. On that server, the P4 BMv2 software switch with P4Runtime support was installed and run. We configured the BMv2 switch without logging support. The switches, hosts,



(a) Throughput of resubmit (b) Throughput of clone (c) Throughput of multi-instance

Fig. 9. Throughput of DSFC



(a) Packet loss of resubmit (b) Packet loss of clone (c) Packet loss of multi-instance

Fig. 10. Packet loss of DSFC

and links are constructed using Mininet [20] running on the same server. The source code version of ONOS is 2.1.0 [19], which is also running on the same server.

We adopt a simple topology and flow entry for our Proof of Concept (PoC) implementation with link speed of hosts and switch set as 10Mbps, displayed in Fig. 8. The two P4 nodes S1 and S2 in the access region are gateways that connect the hosts. The forwarding pipeline of S1 and S2 is written in the P4_16 programming language and compiled using the p4c compiler. The node S1 has a DSFC forwarding pipeline that connects with iperf3 [21] client nodes H1-H3. The node S1 forwarding pipelines like the *resubmit*, *clone*, and *multi-instance* VNF pipeline models were configured at runtime by using the vCPEApp CLI command *vcpe-update*. The command *vcpe-update* takes vCPE device identifier, p4info, and json as input parameters. The node S2 has a simple routing forwarding pipeline that connects with iperf3 server nodes H4~H6. Both nodes S1 and S2 have the same type of packet header fields parsing (*Ethernet*, *IPv4*, and *UDP*), but the implementation of the *Ingress* and *Egress* controls is different based on the forwarding pipeline model.

B. Throughput and Packet loss

In this section, three performance metrics– 1VNF pipeline (from H1 to H4), 2VNFs pipeline (from H2 to H5), and 3 VNFs pipeline (from H3 to H6) – are evaluated using three proposed pipeline models. The configuration of test cases consists of frame size and transmission rate. UDP traffic is used in all test cases of the three pipeline models. The frame sizes are 64, 128, 256, 512, and 1024 bytes, and the transmission rates at 10Mbps. Here we report the mean throughput in Megabits per second (Mbps) and packet loss with 95% confidence interval over 5 runs.

Fig. 9 and Fig. 10 shows the throughput and packet loss rate of the proposed three DSFC pipeline models. The throughput increases as packet size increases. On the contrary, the packet loss rate decreases as packet size increases. Since we use a BMv2 software switch with a fixed packet processing speed, with a constant transmission rate of 10Mbps, the number of packets to be processed increases as the packet size increases. Therefore, the smaller packet size cases would experience a great amount of packet loss and affect the average throughput. Since the 3VNF case needs more processing power than 2VNF, and 2VNF needs more processing power than 1VNF, the mean throughput for a packet size less than 256 shows the decrease of throughput from 1VNF to 3VNF. With respect to the throughput, the packets over 256 bytes and three pipeline models achieve the same line-rate, which shows that software switch is able to process a number of packets with packet size 256 bytes or larger.

For the packet with size of 64 bytes and one VNF test, all of the three pipeline models show different throughput, but not much difference because the number of conditional statements is used differently on each pipeline model which is poorly translated by *p4c*. For the packet with size of 64 bytes and three VNF tests, the *multi-instance* pipeline model has better throughput than *resubmit* (~19%) and *clone* (~16%). Regarding data loss, for the packet with size of 64 bytes and three VNF tests, the *multi-instance* pipeline model has lost rate less than *resubmit* (~37%) and *clone* (~33%). For the packet with size 128 bytes and three VNF tests, the *multi-instance* pipeline model still has better throughput than *resubmit* (~20%) and *clone* (~7%).

For the *resubmit* VNF pipeline model and three VNF tests, a packet is required to go through *resubmit* two times, reparse the packet header fields in the *Parser* control three times, and process the *Ingress* control three times. The resubmitted packet is queued to the resubmit queue, and it has a high-priority and is executed first. There is a long delay in processing the newly received packet at the ingress queue. In this pipeline model test, we observed the number of packets dropped on both resubmit and ingress queues with a large amount of traffic. Thus, the performance penalty is due to the packet processing delay, which happened an increased number of times while processing the *Parser* and *Ingress* controls. If the number of subscribed VNFs increases with multiple different chaining of VNF, then this pipeline model can't handle the heavy network

traffic due to pipeline processing delay, and the performance will be dropped further.

For *clone* VNF pipeline model, in the case of three VNF tests, a packet is cloned two times, which results in executing the *Egress* control two times. In this method, the starvation will happen at the egress queue like the *resubmit* pipeline model. In the experiment herein, the results demonstrate that the *multi-instance* based DSFC pipeline gives better performance. The *multi-instance* pipeline model can support dynamic chaining of multiple VNFs without extra overheads like *resubmit* and *clone*.

C. Pipeline Latency

Here we report the results of the packet processing pipeline latency of the proposed DSFC pipeline model using iperf3 and UDP traffic. We chose the frame size as 1024 bytes because we experience a large number of packets drops while doing the measurement of frame size 64 and 128 bytes.

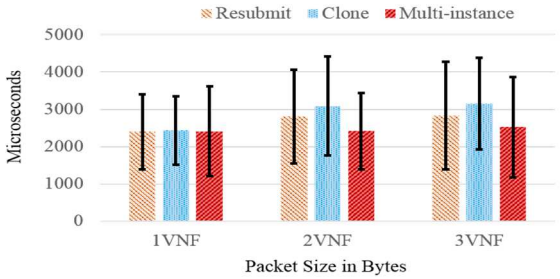


Fig. 11. Packet processing pipeline latency of DSFC

Fig. 11 shows the mean packet processing pipeline latency with 95% confidence interval over 5 runs for three DSFC VNF pipeline models. The 3VNF table processing takes higher latency than the 2VNF and 1VNF, and likewise for 2VNF and 1VNF. In the test of 1VNF, *multi-instance*'s latency is better than that of *resubmit* (~3%) and *clone* (~5%) because the *multi-instance* pipeline model required minimal conditional statements to traverse the *VNF_Bitmap*. But, the *resubmit* and *clone* models need more conditional statements than the *multi-instance* pipeline model. The p4c compiler will poorly translate the written P4 code into DP representation if more conditional statements are used. With respect to the measurement of efficiency for the 3VNF table pipeline operation, the *multi-instance* pipeline model gives better latency compared to *resubmit* (~13%) and *clone* (~27%) since the *multi-instance* pipeline model doesn't have to process the *Parser*, *Ingress*, and *Egress* controls multiple times compared to *resubmit* and *clone*.

V. Conclusion And Future Work

We have described the concept, design, implementation, and evaluation of our proposed flexible vCPE framework. We can enable multiple VNFs and dynamically chain each VNF via the programmability of P4. The service provider can easily place the vCPE device at the edge of the network. This implementation focuses on P4-capable devices with respect to the following aspects: DSFC based on multiple VNF tables, handling thousands of customer network traffic with different subscription of services, saving the number of table entries, and getting better throughput and latency with hardware DP. Three pipeline test cases are considered to evaluate three VNF tables. According to the test results, the *multi-instance* pipeline model gives better performance than *resubmit* and *clone*. In the near future, the proposed framework model will be implemented in P4-capable hardware switch platforms and expand the number of VNFs on real-time network traffic.

References

[1] O. N. Foundation, "Software-defined networking: The new norm for networks", ONF White Paper, vol. 2, pp. 2-6, 2012.

[2] N.McKeown, "Software-defined networking", INFOCOM keynote talk, vol. 17, no.2, pp. 30-32, 2009.

[3] N.Feamaster, J.Rexfore, and E.zegura, "The road to SDN", ACM SIGCOMM Computer communication Review, vol. 44, pp. 87-98, Apr 2014.

[4] Kreutz, F.M.V.Ramos, P.E. Verissimo, C.E. Rothenberg, S.Azodolmoky, and S. Uhlig, "Software-defined networking: A comprehensive survey", Proceeding of the IEEE, vol. 103, pp. 14-76, Jan 2015.

[5] N.McKeown, T.Anderson, H.Balakrishnan, G.Parulkar, L.Peterson, J. Rexford, S.Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks", ACM SIGCOMM Computer Communication Review, vol. 38, pp. 69-74, Mar. 2008.

[6] B.Pfaff, B. Lantz, B. Heller, et al., "Openflow Switch Specification, version 1.3.0", Open Networking Foundation, 2012.

[7] M. Chiosi, D. Calrke, P. Willis, A. Reid, J. Feger, M. Bugenhagen, W. Khan, M.Faragano, C. Cui, H. Deng, et al,. "Network functions virtualization: An introduction, benefits, enablers, challenges and call for action", SDN and OpenFlow world Congress, pp. 22-24, 2012.

[8] NFV ISG, "Network Functions Virtualization (NFV); Virtual Network Functions Architecture", Tech. Rep. GS NFV-SWQ 001 V1.1.1, ETSI, Dec. 2014.

[9] R. Mijumbi, J. Serrat, J.L. Gorricho, N.Bouten, F.D.Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges", IEEE Comunications Surveys & Tutorials, vol. 18, no. 1, pp. 236-262, 2016.

[10] P. Minoves, O.Frendved, B.Peng, A. Mackarel, and D. Wilson, "Virtual CPE: Enhancing CPE's deployment and operations through virtualization", 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings, IEEE, Dec. 2012.

[11] P. Berde et al., "ONOS: Towards an open distributed SDN OS", Proceeding of 3rd Workshop Hot Topics Software Defined Networking (HotSDN), pp. 1-6, 2014.

[12] N.-F. Huang, C.-H. Li, C.-C. Chen, I.-H. Hsu, C.-C. Li, and C.-H. Chen, "A novel vCPE framework for enabling virtual network functions with multiple flow tables architecture in SDN switches", Network Operations and Management Symposium (APNOMS), 2017 19th AsiaPacific, IEEE, 2017.

[13] David Hancock and Jacobus van der Merwe, "Hyper4: Using P4 to virtualize the programmable data plane", In CoNEXT '16, pages 35-49, 2016.

[14] Peng Zheng, Theophilus Benson, and Chengchen Hu, "P4Visor: Lightweight Virtualization and Composition Primitives for Building and Testing Modular Programs", in Proceedings of the 14th International Conference on Emerging Networking Experiments and Technologies, ser. CoNEXT '18. New York, NY, USA: ACM, pp. 98-111, 2018.

[15] X. Chen, D. Zhang, X. Wang, K. Zhu, and H.Zhou, "P4SC: Towards high-performance service function chain implementation on the P4-capable device", in Proceedings IFIP/IEEE International Symposium on Integrated Network Management (IM), pp. 1-9, Apr. 2019.

[16] The P4 Language Consortium. The P4 Language Specification. https://p4.org/p4-spec/docs/P4-16-v1.1.0-spec.pdf

[17] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford,C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, & D. Walker "Programming Protocol-Independent Packet Processors", CoRR abs/1312.1719 (2013).

[18] BMv2. https://gitbub.com/p4lang/behavioral-model

[19] ONOS Source. https://github.com/opennetworkinglab/onos

[20] Mininet. http://mininet.org

[21] ESnet, Lawrence Berkeley National Laboratory. iPerf3. http://software.es.net/iperf

[22] P4SC. https://github.com/P4SC/p4sc

[23] Lee.J, Ko.H, Lee. H, & Pack.S "Flow-Aware Service Function Embedding Algorithm in Programmable Data Plane", IEEE Access 2021.