HyperSFP: Fault-Tolerant Service Function Chain Provision on Programmable Switches in Data Centers

Hongyi Huang
Tsinghua University
hhy.hongyi@outlook.com

Wenfei Wu
Peking University
wenfeiwu@pku.edu.cn

***Abstract*—With the cloud networks being equipped with programmable switch, Service Function Chain (SFC) provision has started to be migrated to the switches for better performance and manageability. In this paper, we design HyperSFP which places multiple SFCs to a data center network (DCN). In the placement algorithm, HyperSFP builds an integer programming (IP) model to achieve functionality, fault tolerance, and load balance. To support large-scale networks, HyperSFP IP model is relaxed to two approximate approaches: Stage-Separated IP model and linear programming (LP) model. Both approaches can improve the algorithm efficiency. HyperSFP's data plane is designed to deploy the active and backup NFs in the control-plane plan, and migrate traffic from failed active NFs to its backup NFs. Our prototype and evaluation shows that HyperSFP achieves performance gain by implementing NFs on programmable switches, its control plane achieves fault tolerance, load balance, and scalability, and its data plane can handle network failures promptly.**

***Index Terms*—NFV, SFC, fault tolerance, programmable switch**

I. Introduction

Network functions (NFs) in modern clouds transparently process tenant traffic, which provides security enhancement or performance acceleration. A tenant's requirements in traffic processing, e.g., load balancing and blacklisting, would be satisfied by chaining multiple NFs in the custom sequence, which is named *service function chaining (SFC)*. A recent trend of implementing SFC as software [1]–[6] on commodity servers, namely network function virtualization (NFV), provides beneficial features such as deployment flexibility and management simplicity. But NFV also suffers from the low efficiency compared with the hardware middlebox approach (i.e., CPU v.s. ASIC).

Recent data centers have started to be empowered by *programmable switches* [7]–[9], which provides new opportunities for SFC provision. The programmable switch has memory to store persistent states, and can load and execute user specified programs for packet processing. By implementing NFs on programmable switches, the cloud operator achieves both management flexibility and high-speed processing (at the line rate). Such programmable switch-based NFs have been proved the feasibility in examples of load balancer, firewall, gateway, VPN and rate limiters [10]–[14].

However, data center networks (DCN) operators have been long upset by switch failure problems [15]–[17]. With thousands of switches deployed in DCN, it is quite difficult (even impossible) to guarantee all switches can persistently run without errors. Commercial DCNs also occasionally report outage (e.g., [18], [19]). If SFCs are implemented on switches, they should also allow for *fault tolerance*.

In this paper, we design a switch-based SFC solution for clouds, namely HyperSFP. HyperSFP implements fault-tolerant SFCs on programmable switches. In the control plane, HyperSFP places SFCs on multiple switches in the DCN topology. The placement algorithm models an integer programming (IP) problem, which takes fault tolerance, load balancing, and resource availability into consideration. The IP approach is also transformed to approximate approaches for efficient execution. HyperSFP's data plane is to set up SFC instances (both active NFs and backup ones) and switching traffic between instances in case of switch failure.

We build HyperSFP's data plane on Tofino switches and the controller in Python. Our evaluation demonstrates that HyperSFP can provide SFC with fault tolerance (e.g., no traffic compromised when there is one switch failure), the SFC placement algorithm can be executed efficiently (about 100 seconds for 1000 custom SFCs in 96-node Fat-Tree) and near-optimal (on average 11.4% gap to the optimal result).

II. Background

A. SFCs on PISA Switches

Programmable switches [20] become a new platform that can host NFs. Several NFs are ported to the programmable switches [10], [11], [21], [22]. Typical types are ACL, firewall, load balancer, NAT, etc. Implementing NFs on programmable switches brings about two advantages. First, the processing throughput can be improved to line rate. NFs on servers can process traffic at a rate of 10X Gbps, but cloud services could generate traffic at 1X Tbps [23]. As for programmable switches, a single programmable pipeline can handle traffic at 100X Gbps with its dedicated chip, and the whole switch capacity can reach 1X Tbps. Second, switch NFs have ultra low processing latency. It does not need to get through the software network stack like server NFs. The per-hop latency for switch processing is within 100X ns. In addition, the switch NF is on-path, saving hops to the server (and to return) compared with server NFs. Thus, the total processing latency can be significantly reduced on switch NFs.

Wenfei Wu is the corresponding author.

978-1-6654-0601-7/22$31.00

Resource Consumption. When an NF is implemented on PISA, it consumes two kinds of resources. The NF's logic is written in PISA specific language (e.g., P4 in Tofino [24] and NPL in Trident [25]), and the code would consume "computation resources" when compiled to the PISA chip, which includes PHV, ALU, gateway tables and so on. A more complex NF usually consumes more computation resources. [26], [27]

In the runtime, NFs (on PISA) need to be configured with runtime rules, which consume switch memories (i.e., SRAM and TCAM). For example, a load balancer needs to be configured with backend servers, and a firewall needs to be configured with ACL rules. These rules can be dynamically changed by the controller in the runtime.

B. Cloud Switch Failures

Switch failure is not an occasional event but prevalent in DCN [15], [17]. Switch failures could lead to network congestion [28] and link failures, affecting users' quality of experience (QoE) [15], [16] and even lead to catastrophic cloud outage [29].

When the switches are hosting SFCs, switch failures would cause SFC unavailability, disrupting traffic processing. Although routing protocols such as OSPF can rebuild the route, the SFCs on the original (failure) path cannot be immediately ready on the new path. Thus, it is not trivial to build an SFC solution resilient to network failures.

C. Goal and Intuition

In this paper, our goal is to build a *switch-based SFC solution*, which satisfies the following requirements. We name our system HyperSFP.

- **(R1) Performance Gain.** HyperSFP implements NFs on PISA, which is expected to inherit its advantage of high throughput and low latency.
- **(R2) Fault Tolerance.** While DCN switches could suffer from occasional failure, HyperSFP should be resistant to switch failures, i.e., provide availability even when the switch is down.
- **(R3) Load Balance.** When multiple SFCs are placed in a topology (with multiple programmable switches), they had better be load balanced so that the system can be resilient to bursty workload (both new SFCs and new traffic).
- **(R4) Scalability.** The solution should be applicable to large-scale networks.
- **(R5) Quick Failover.** When failure happens, HyperSFP should recover the SFCs service promptly.

III. Design Overview

Assumptions. We make the following assumptions about the DCN scenario.

- **SFC ID for Traffic Classification.** Each SFC has a unique ID, and its traffic has that ID `SFC ID` as well. This ID is used to distinguish different tenants or application in management. The ID can be implemented as VLAN, VxLan, GRE, or even IP subnet. The programmable switch parses the SFC ID and recognizes SFC traffic.

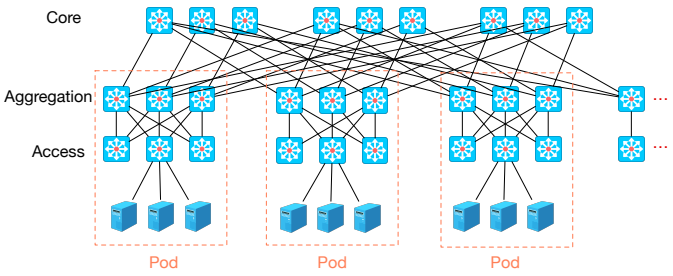

Fig. 1: The Topology of a Typical 3-layer Fat-Tree Hierarchical Data Center Network

- **Statically Prepared NF Types.** There are limited types of NFs (some listed in § VI). We assume the operator pre-developed a few NF types for the tenants, and a tenant would select NFs from the NF types to construct its own SFC. While the types of NFs can increase with long-term evolution, we regard the types of NFs at each deployment cycle (a few days) to be static and fixed.
- **Switch NF Serving SFC Instances.** NFs are deployed on switches, serving (possibly) multiple SFCs. NFs on SFCs are instances, and NF instances of the same type can be installed on the same switch NFs to share the computation resources. NFs instances' traffic is isolated by the `SFC ID`, processed by its own rules but the same NF processing logic.
- **SFC Pre-configuration.** An SFC's traffic volume and the number of rules in each NFs are determined in advance for HyperSFP. Because these are configuration parameters from the tenant (e.g., reserved bandwidth, pre-configured rules) before the SFC is launched.
- **DCN Topology.** Typical data centers use a Fat-Tree topology with multiple paths between servers. There are three-layer in the topology — access switches (a.k.a. TOR switches), aggregation switches, and core switches. The access switches and aggregation switches are grouped as pods that serve different servers. Different paths among servers are homogeneous with the same number of hops and each hop at the same layer. Fig. 1 is an example of Fat-Tree topology.

Design Rationale. First, HyperSFP first implements NFs on programmable switches so as to achieve performance gain (R1).

Second, HyperSFP makes redundant NF deployment on multi-paths of the DCN topology so as to achieve fault tolerance (R2). When HyperSFP sets up multiple SFC instances, a few of them serve the runtime traffic, named "active NFs", and others stand by to take over the traffic in case of failure, named "backup NFs".[1] A backup NF is deployed on the same position of another path of its active NF, which is called *intra-layer* redundancy. All backup NFs are pre-installed because the prolonged traffic interruption would be caused by the new

[1]A switch can deploy both active NFs and backup NFs simultaneously, e.g., an active NF from one SFC and a backup NF from another SFC.

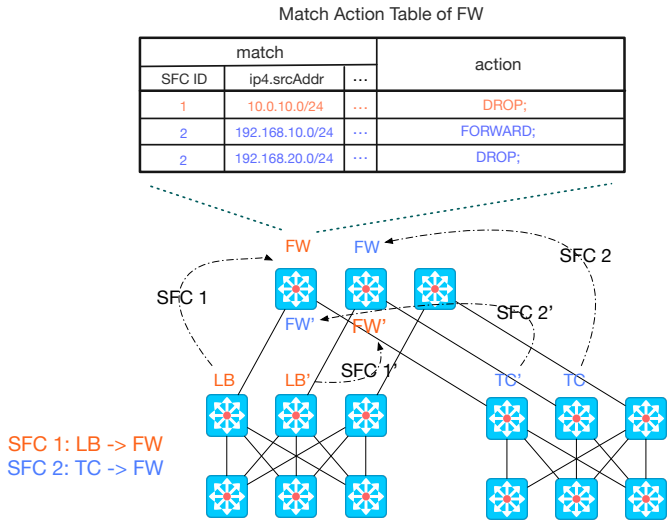

Fig. 2: An Example of HyperSFP Placement

setup of NF, where programmable switches take a while to necessarily reboot, compile NF code and set up rules [30].

Third, HyperSFP formulates an IP model to place SFCs onto DCN, with the objective of minimizing the maximum load on switches (R3), and relaxes the problem to approximate ones for efficiency (R4). Finally, HyperSFP data plane sets up SFCs according to the control-plane plan and makes quick failover whenever failure happens (R5).

Example. As is shown in Fig. 2, two SFCs are distributed in the DCN. Both the SFCs span different layers of switches to meet resource requirements. For example, the load balancer (LB) of $SFC1$ is deployed on aggregation switches while the firewall (FW) is deployed in a core switch. The backup instances of all these NFs (LB, FW, TC) are created with intra-layer redundancy, marked as LB', FW' and TC'. The active FW of $SFC1$ and the backup FW' of $SFC2$ are deployed in the same core switch, sharing computation resources. In the match-action table of the shared firewall, the `SFC ID` justifies using one switch NF to isolate the traffic from multiple SFCs.

IV. Control Plane Design

HyperSFP uses Integer Programming to model the SFC deployment and relaxes it to approximate approaches for efficiency.

A. Preliminaries

A topology in DCN consists of N switches $S = \{s_1, s_2, \cdots, s_N\}$. In a Fat-Tree topology, the path between any pair of end hosts has the fixed number of hops; if two hosts are local (within the same pod), we let the traffic traverse to the core and then be reflected back, using more hops in the aggregation layer and the core layer to deploy SFC. We also assume all switches are programmable switches.

One SFC is denoted as c_l, which carries a flow f_l with the traffic rate T_l. The m-th NF on c_l requires Mem_{ml} to deploy its rules. An NF type is denoted by i (e.g., load balancer, firewall), and its processing logic would consume computation resource of Com_i once newly deployed on a switch. For one SFC, HyperSFP keeps its traffic traverse physical NFs in the same order as specified by the SFC.

TABLE I: Symbols Used in Modeling

Symbol	Type	Explanation
f_l	const	the l-th flow
c_l	const	the SFC that processes the flow f_l
s_n	const	the n-th switch
T_l	const	the bandwidth requirement of the flow f_l
Mem_{ml}	const	the memory requirement of m-th NF of SFC c_l
i	const	the type of NF
Com_i	const	the computation requirement of the NF in type i
NF_{ml}	const	the type of the m-th NF of SFC c_l
C_{COM_n}	const	maximal computation capacity for switch s_n
C_{MEM_n}	const	maximal memory capacity for switch s_n
K	const	the number of all access switches
R	const	the number of NF instances; larger R within range leads to more fault tolerance
α	const	the switch failure rate
M	const	the maximal length of all service chains
X_{ni}	BIN var	whether to deploy the NF of type i in switch s_n
Z_{nml}	BIN var	whether to "prepare" the m-th NF of SFC c_l in switch s_n
Y_{nml}	BIN var	whether to deploy the m-th "active" NF of SFC c_l in switch s_n
D_{ml}	INT var	the layer number of the m-th NF of SFC c_l
B_n	derived	the overall traffic in switch s_n

B. Variables

We use binary variable X (TABLE I) to denote whether an NF type is deployed on a switch. HyperSFP needs to consider both active and backup NF deployment. Thus, we use binary variable Z to denote whether one SFC's NF is prepared on a specific switch, i.e., both active and backup NFs; [2] we use binary variable Y to denote whether one SFC's NF is active on a specific switch, i.e., active NFs.

C. Constraints

The SFC placement is constrained by the fault tolerance requirement, the switch resources (memory and computation) and the bandwidth capacity of the switch board.

1) Deployment Constraint: If an SFC's NF is deployed on a switch, the switch must have that type of NF set up.

$$Z_{nml} \leq X_{ni}, i = NF_{ml}, \forall n, m, l. \quad (1)$$

NFs within the same SFC should retain a sequential order to ensure correct packet processing. When one of the NFs is deployed, the order should be taken care of during the placement of other NFs. In other words, NFs in an SFC should be placed from the access layer to the aggregation layer, and to the core layer in sequence.

$$\lceil \frac{\sum_n n \times Y_{n,m,l}}{K} \rceil \leq \lceil \frac{\sum_n n \times Y_{n,m+1,l}}{K} \rceil, \forall l, m \quad (2)$$

If Y is 0, the equation is always true; if both Ys are 1, $n \times Y$ is the switch index. We index switch from bottom to the top, and use K to denote the number of switches in each layer (except the core layer). Thus, $\lceil \frac{n \times Y}{K} \rceil$ denotes the layer of a switch. [3]

[2]To prevent SFCs from being deployed in other pods, Z is variable only with the range of switches in the same pod; otherwise, it's 0.

[3]The access switches are in the first layer (denoted as 1), and so on and so forth.

2) Switch Resource Constraint: Each new setup of certain NF type consumes the computation resource of the switch, and the total consumption should not exceed the switch computation capacity.

$$\sum_i Com_i \times X_{ni} \leq C_{Com_n}, \forall n. \tag{3}$$

Deploying an NF instance needs to configure its rules (e.g., the firewall rules in Fig. 2), which consumes switch memory. All NF instances' memory consumption should not exceed the switch memory capacity.

$$\sum_{m,l} Mem_{ml} \times Z_{nml} \leq C_{Mem_n}, \forall n. \tag{4}$$

3) Fault Tolerance: One active instance should be allocated to each NF.

$$\sum_n Y_{nml} = 1, \forall m, l \tag{5}$$

Each SFC's NF should have at least several instances prepared (denoted as R), including one active instance and $(R-1)$ backup instances.

$$\sum_n Z_{nml} = R, \forall m, l \tag{6}$$

The active instance belongs to the prepared instances.

$$Y_{nml} \leq Z_{nml}, \forall n, m, l. \tag{7}$$

We assume an active NF and its backup NF(s) are placed in the same layer by HyperSFP, which guarantees the NF orders when switching to backup NFs. Fixing m and l and varying n in Z_{nml}, if for all Z_{nml}s that is 1, their layer number $\lceil \frac{n \times Z_{nml}}{K} \rceil$ should be equal. We use D_{ml} to denote this layer number, and $D_{ml} = \lceil \frac{n \times Z_{nml}}{K} \rceil$, $\forall Z_{nml} = 1$. Thus, we have the constraint

$$D_{ml} = (1 - Z_{nml}) \times D_{ml} + \lceil \frac{n \times Z_{nml}}{K} \rceil, \forall n, m, l. \tag{8}$$

If Z is 1, the last expression is constrained to be equal to the same D_{ml}; otherwise, if Z is 0, the equation permanently holds.

4) Switch Bandwidth Capacity.: Each switch has a maximum processing capacity, which is used to handle traffic in active NFs in normal runtime and handle extra fail-over traffic of backup NFs in case of failure recovery. For a switch s_n, it handles active NF traffic with the volume of $\sum_l \lceil \frac{\sum_m Y_{nml}}{M} \rceil \times T_l$, where $\lceil \frac{\sum_m Y_{nml}}{M} \rceil$ is a binary indicator (0 or 1) whether the SFC c_l has any NFs on s_n.

We use α to denote the switch failure rate, e.g., reported as 2% in [17] and 5% in [29]. Since reserving full bandwidth for all backup NFs is wasteful in practice, we require the switch to only reserve α times of normal traffic to handle failover traffic.[4] The failover traffic is $(\lceil \frac{\sum_m Z_{nml}}{M} \rceil - \lceil \frac{\sum_m Y_{nml}}{M} \rceil) \times T_l$. Thus, we have switch s_n's traffic

[4]The reservation ratio can be tuned in different scenarios.

$$B_n = (1-\alpha) \sum_l \left\lceil \frac{\sum_m Y_{nml}}{M} \right\rceil T_l + \alpha \sum_l \left\lceil \frac{\sum_m Z_{nml}}{M} \right\rceil T_l. \tag{9}$$

And we have the constraint

$$B_n \leq B, \forall n. \tag{10}$$

D. Objective

The objective of the SFC placement is to achieve load balance among switches processing capacity. This objective can ensure the system resilience in case of bursty workload by minimizing the maximal traffic among all n switches.

$$Objective = \text{ minimize } \max(B_n). \tag{11}$$

E. Approximations

In cloud computing scenarios, the tenants join and leave dynamically. Their requests need timely response. But solving integer programming is time consuming, which is not practical. We consider two approaches to accelerate the placement computation.

1) Stage Separated Approach: The IP complexity increases with the number of constraints and variables. We try to break the whole IP problem into two smaller ones. This approach has two steps. In the first step, we solve the problem of finding locations for all NFs (both active ones and backup ones), assuming all NFs have traffic. The objective is still (11), but the constraint (5) and (7) are deactivated, and the equation (9) is changed to

$$B_n = \sum_l \lceil \frac{\sum_m Z_{nml}}{M} \rceil \times \frac{T_l}{R}. \tag{12}$$

For example, if R is 2, that is to have 2 instances for each NF, we divide the bandwidth requirement on each switch to half.

In the second step, we solve the problem of choosing active NFs among the prepared NF positions, where the objective is still (11). The constraints are the same as the original solution, with Equation (1) to (10), but X and Z are constants as a result of the first step instead of variables.

2) LP Relaxation: We first relax the integer constraints to linear constraints since linear programming (LP) can be solved in polynomial time. The integer constraints are changed to real numbers in the same range, e.g., binary variables $X, Y, Z \in \{0, 1\}$ are transformed to $0 \leq X, Y, Z \leq 1$.

Second, integer variables are rounded to a nearby integer with probability. For example, a value of $X.Y$ (X is integer and Y is decimal) is rounded to X with probability $1-Y$ and to $X+1$ with probability Y.

Finally, the rounded variables are put to the original IP problems to validate the constraints. If the rounded variables cannot satisfy the constraints, the algorithm will withdraw one SFC which requires most resources but least bandwidth from the hot spot, and go back to the second step for another trial; otherwise, rounded results are output. Randomized rounding

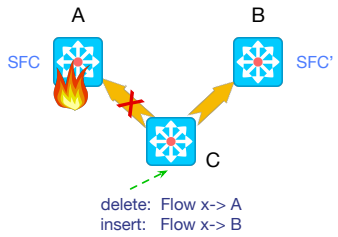

Fig. 3: An Example of Failure Takeover

can guarantee the expected result of each rounding step to be the optimal (i.e., E(Objective with rounded variable) = Objective of LP).

V. Data Plane Design

The data plane prepares NFs during boot-up and also migrates traffic when the failure takes place.

A. Set Up SFC with Redundancy

As discussed in §III, setting up an NF during the failover is not a first-class choice due to the time cost in code compilation and rule installation. Thus, HyperSFP prepares both active and backup NFs during boot-up.

While tenants may wait for active NFs, they are set up and provided firstly to tenants. Then the backup NFs are setup asynchronously after active NFs with lazy rule installation.

B. Failure Takeover

When the network controller is notified (e.g., SNMP) of the network failure, e.g., a link/switch failure, it takes actions to redirect the traffic from the compromised NFs to its backup NFs, i.e., switching from Y to $Z - Y$ in §IV-B.

HyperSFP installs static routing rules for the NF switching. Illustrated in Fig. 3, once a switch failure is reported, the controller would rapidly locate the redundancy of all SFCs in fault switch A (e.g., SFC whose redundant one is SFC' in switch B). The controller decides on changing the routing table (i.e., insert or delete) in predecessor switch C to route original traffic from A to B. This routine is applied to other neighbors of switch A as well.

VI. Evaluation

We conduct experiments to answer the following key questions:

- What throughput can HyperSFP achieve in various SFC provision scenarios? What's the gap between HyperSFP and strict IP solution? (§VI-B)
- Is HyperSFP scalable to complex network topology and large number of SFCs? (§VI-C)
- To what extent can HyperSFP mitigate the traffic interruption after switch/link failures happen? (§**??**)

A. Experiment Setup

Implementation. We implement HyperSFP SFC placement solver using optimizer Gurobi [31]. We implement HyperSFP routing strategy using OpenFlow APIs [32] and manage the SFC rules via gRPC (i.e., BRI) with the switch. HyperSFP's controller has 892 lines of code in Python, and its SFCs with 377 lines of code in P4.

Testbed Setup. We evaluate HyperSFP's data plane on a testbed consisting of one commodity switch, one programmable switch (32-port Tofino switch, each port with 100Gbps bandwidth), and 2 mid-range servers (4 Intel(R) Xeon(R) Gold 5120T CPU @ 2.20GHz 16 cores, 192RAM with Ubuntu 18.04, 100Gbps Mellanox ConnectX-5 NICs, 22TB disk) that run the sender and the receiver. We use libpcap [33] to generate traffic with traces [34].

Simulation Topology. We also run a simulation for large-scale networks according to the settings in [35]. We use two kinds of topology varying in its complexity. We set the switch failure rate α to be 5%.

1) A small topology with moderate number of flows. The topology is 3-tier, consisting of 4 core switches, 4 aggregation switches and 4 access switches. The switches are fully connected to those in the adjacent layer.
2) A complex 96-node Fat-Tree topology, including 16 core switches, 40 aggregation switches and 40 access switches. The first and second switches are divided into 10 pods, each with 4 access and 4 aggregation switches.

NFs. We implement several NFs on Tofino Switches, including firewall, load balancer, traffic classifier and router.

SFCs. In the simulation, we randomly generate SFCs of a sequence of diverse NFs and assign memory and computation requirements to them according to existing measurement reports [36], [37]. For the small topology, we generate SFCs with the number ranging from 10 to 100; for the large topology, the number can reach the order of hundreds to thousands. The average volume of the traffic is 10G bps.

Baselines. We compare three variants of HyperSFP. The "HyperSFP" means the approach with linear programming relaxation (§IV-E2); "Separate" means the approach of HyperSFP with integer programming in separated stages (§IV-E1); and "IP" means the approach of integer programming without any optimizations.

Metrics. We measure the throughput of the system and the execution time of the algorithm to evaluate HyperSFP's efficiency and scalability. We measure the compromised traffic and failover time during failure recovery to evaluate HyperSFP's effectiveness.

B. Comparison of HyperSFP Variants

We run experiments in simulation with the small topology (because some variants cannot finish for the large topology). We tune the number of SFCs and measure the execution time of HyperSFP and two baselines varying in the number of SFCs. As shown in Fig. 4, a combined searching could sharply increase the runtime and reach the runtime limitation (600s) with 40 SFCs. Separate searching does perform much better, which gives us hints that stage-separated optimization does not essentially eliminate the complexity of IP solution. HyperSFP applies linear relaxation to offer new near-optimal solutions, which could decrease the execution time to great extent, all less than 35s.

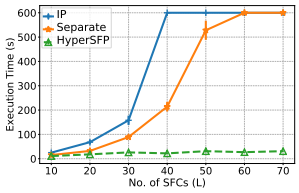

Fig. 4: Execution Time of HyperSFP Compared with Baselines

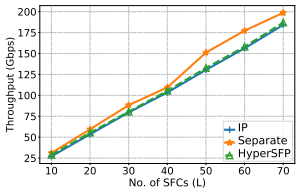

Fig. 5: The Objective Throughput of HyperSFP with Low Bandwidth Capacity

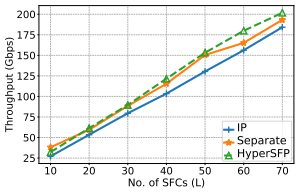

Fig. 6: The Objective Throughput of HyperSFP with High Bandwidth Capacity

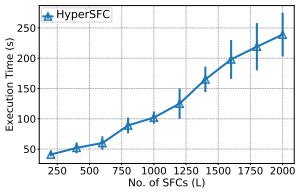

Fig. 7: Execution Time of HyperSFP Varying in the Number of SFCs

The approximation might raise some concerns on precision loss. Hence, we output the objective value (maximal throughput among all switches) to show the differences. To distinct the 'Separate' method, we conduct two experiments under different bandwidth capacities.

Undergoing low bandwidth capacity of switches, SFCs may contend for the bandwidth resources, which forces 'Separate' to re-run the decision-making process of SFC placement no matter how to choose active/backup. Before the 'Separate' method enters a new iteration, some decision has been made and added to the model, which might affect the overall objective. Thus, as shown in Fig.5, 'Separate' would perform worse under these circumstances; 'IP' has no difference since it takes all these factors into consideration; HyperSFP shows a little increase in the maximal throughput.

With high bandwidth capacity, as shown in Fig.6, 'Separate' can run the whole procedure within one iteration. The throughput is close to the 'IP', which is better than HyperSFP, typically as we increase the number of SFCs.

Despite that HyperSFP shows some throughput gap behind to the optimal solution 'IP', the error is narrowed within 15.1% in the worst case and 11.4% on average, which is acceptable to achieve comparatively balanced workload within feasible execution time.

C. Scalability

To testify the feasibility of scaling HyperSFP to large-scale data center network, we adopt a more complex topology and increase the number of SFCs to more than 1K. As illustrated in Fig. 7, HyperSFP shows that the execution time goes up as the number of SFCs increases and reaches around 240s when 2000 SFCs are provisioned. We omit to show the result of baselines since they have reached the limitations in small scale experiments. Although the number of SFCs is no more than the order of thousands, it can satisfy a moderate cloud network. [36]

VII. Related Work

P4-Acclerated NFs. Since the IC vendors enable the programmability atop traditional switches [26], [27], some works emerge to utilize the fast switch and offload their functionalities from end-host to in-network. SilkRoad and HULA [10], [38] design fast layer-4 load balancing using programmable data plane. [14], [39] propose heterogeneous gateways to alleviate the burden on servers. [40], [41] provide in-network cache for large-scale storage systems. [42] performs in-network aggregation to accelerate neural network training. [43] enables data aggregation to accelerate data processing applications. HyperSFP is a control plane that can manage these NFs.

Fault-Tolerance in NFV and P4. Fault-tolerance is important issue in large scale system [44]–[49]. Ananta and Maglev [50], [51] are distributed software NFs, which could mitigate the impact of failures. FTMB [52], REINFORCE [53], FTC [54], FTvNF [55] and [56] support chain-level failover to provide fault tolerance in NFV systems. For example, [57] uses checkpoints to recover the states. Some work [58]–[60] proposes NF modularization to improve fault resiliency.

SwiShmem [61] and RedPlane [62] provide state backup mechanisms to prevent NF state from switch failures. They are alternative approaches compared with HyperSFP. HyperSFP uses heterogeneous devices (all switches) instead of switch-server mixed platforms. Many work [63]–[68] propose state management interfaces, but they do not target unplanned downtime. They are complementary to HyperSFP.

SFC Deployment in Programmable Switches. Recent efforts propose offloading SFC to programmable switches. P4SC [69] provides primitives to deploy SFCs in switches, but it lacks the ability to ensure resource efficiency. Dejavu [70] mainly proposes data plane design to enable multi-tenancy in switches, but our work focuses more on control plane design. LightNF [22] carries out analysis on the feasibility of porting NFs to programmable switches and proposes naive approaches to utilize both the hardware and software.

Other works [35], [71]–[79] also alleviate the progress of offloading SFCs to programmable switches but they all lack support for fault tolerance.

VIII. Conclusion

We built HyperSFP, which can place multiple SFCs in DCN and handle network failures. HyperSFP implements NFs on programmable switches, which provides performance acceleration. HyperSFP builds IP model, stage-separated model, and LP model to place multiple SFCs, and it achieves fault tolerance, load balance, and scalability. HyperSFP's data plane installs NFs according to the control-plane plan and handles failure by rerouting traffic to backup NFs. Our prototype and evaluation shows that HyperSFP achieves good properties of fault tolerance, scalability, load balance, prompt failover, and performance gain when deploying multiple SFCs in DCN.

REFERENCES

[1] W. Zhang, G. Liu, W. Zhang, N. Shah, P. Lopreiato, G. Todeschi, K. K. Ramakrishnan, and T. Wood, "Opennetvm: A platform for high performance network service chains," in *Proceedings of the ACM SIGCOMM Workshop on Hot topics in Middleboxes and Network Function Virtualization, HotMiddlebox@SIGCOMM 2016, Florianopolis, Brazil, August, 2016*, 2016, pp. 26–31. [Online]. Available: http://doi.acm.org/10.1145/2940147.2940155

[2] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu, "Nfp: Enabling network function parallelism in nfv," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 43–56.

[3] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "Clickos and the art of network function virtualization," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*. USENIX Association, 2014, pp. 459–473.

[4] S. G. Kulkarni, W. Zhang, J. Hwang, S. Rajagopalan, K. Ramakrishnan, T. Wood, M. Arumaithurai, and X. Fu, "Nfvnice: Dynamic backpressure and scheduling for nfv service chains," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 71–84.

[5] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker, "E2: A framework for nfv applications," in *Proceedings of the 25th Symposium on Operating Systems Principles*, ser. SOSP '15. New York, NY, USA: ACM, 2015, pp. 121–136. [Online]. Available: http://doi.acm.org/10.1145/2815400.2815423

[6] J. Hwang, K. K. Ramakrishnan, and T. Wood, "Netvm: high performance and flexible networking using virtualization on commodity platforms," *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 34–47, 2015.

[7] "https://www.alibabacloud.com/blog/the-network-architecture-and-network-management-system-behind-this-years-double-11_595615."

[8] Facebook. Disaggregate: Networking recap. [Online]. Available: https://engineering.fb.com/2017/01/30/data-center-engineering/disaggregate-networking-recap/

[9] "https://scholar.harvard.edu/srivatsan-krishnan/publications/accelerating-recurrent-neural-networks-analytics-servers-comparison."

[10] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 15–28. [Online]. Available: https://doi.org/10.1145/3098822.3098824

[11] Y. He, W. Wu, X. Wen, H. Li, and Y. Yang, "Scalable on-switch rate limiters for the cloud," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.

[12] R. Datta, S. Choi, A. Chowdhary, and Y. Park, "P4guard: Designing p4 based firewall," in *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*. IEEE, 2018, pp. 1–6.

[13] F. Hauser, M. Häberle, M. Schmidt, and M. Menth, "P4-ipsec: Site-to-site and host-to-site vpn with ipsec in p4-based sdn," *IEEE Access*, vol. 8, pp. 139 567–139 586, 2020.

[14] K. Qian, S. Ma, M. Miao, J. Lu, T. Zhang, P. Wang, C. Sun, and F. Ren, "Flexgate: High-performance heterogeneous gateway in data centers," in *Proceedings of the 3rd Asia-Pacific Workshop on Networking 2019*, ser. APNet '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 36–42. [Online]. Available: https://doi.org/10.1145/3343180.3343182

[15] H. H. Liu, Y. Zhu, J. Padhye, J. Cao, S. Tallapragada, N. P. Lopes, A. Rybalchenko, G. Lu, and L. Yuan, "Crystalnet: Faithfully emulating large production networks," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 599–613. [Online]. Available: https://doi.org/10.1145/3132747.3132759

[16] J. Meza, T. Xu, K. Veeraraghavan, and O. Mutlu, "A large scale study of data center network reliability," in *Proceedings of the Internet Measurement Conference 2018*, ser. IMC '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 393–407. [Online]. Available: https://doi.org/10.1145/3278532.3278566

[17] R. Singh, M. Mukhtar, A. Krishna, A. Parkhi, J. Padhye, and D. Maltz, "Surviving switch failures in cloud datacenters," *SIGCOMM Comput. Commun. Rev.*, vol. 51, no. 2, p. 2–9, May 2021. [Online]. Available: https://doi.org/10.1145/3464994.3464996

[18] No fooling: Microsoft cloud outage takes azure, teams and office 365 offline. [Online]. Available: https://siliconangle.com/2021/04/01/no-foolin-microsoft-cloud-outage-takes-azure-teams-office-365-offline/

[19] Aws outage report. [Online]. Available: https://aws.amazon.com/message/11201/

[20] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[21] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "Netcache: Balancing key-value stores with fast in-network caching," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 121–136. [Online]. Available: https://doi.org/10.1145/3132747.3132764

[22] X. Chen, Q. Huang, P. Wang, Z. Meng, H. Liu, Y. Chen, D. Zhang, H. Zhou, B. Zhou, and C. Wu, "Lightnf: Simplifying network function offloading in programmable networks," in *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, 2021, pp. 1–10.

[23] Akamai cdn hits 72tbps data traffic record. [Online]. Available: https://www.capacitymedia.com/articles/3822904/akamai-cdn-hits-72tbps-data-traffic-record

[24] The p4 language specification. [Online]. Available: https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html

[25] K. Gao, T. Nojima, and Y. R. Yang, "Trident: Toward a unified sdn programming framework with automatic updates," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 386–401. [Online]. Available: https://doi.org/10.1145/3230543.3230562

[26] L. Jose, L. Yan, G. Varghese, and N. McKeown, "Compiling packet programs to reconfigurable switches," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, 2015, pp. 103–115. [Online]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/jose

[27] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 99–110. [Online]. Available: https://doi.org/10.1145/2486001.2486011

[28] S. Jha, A. Patke, J. Brandt, A. Gentile, B. Lim, M. Showerman, G. Bauer, L. Kaplan, Z. Kalbarczyk, W. Kramer, and R. Iyer, "Measuring congestion in high-performance datacenter interconnects," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 37–57. [Online]. Available: https://www.usenix.org/conference/nsdi20/presentation/jha

[29] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, p. 350–361, Aug. 2011. [Online]. Available: https://doi.org/10.1145/2043164.2018477

[30] Leveraging stratum and tofino fast refresh for software upgrades. [Online]. Available: https://opennetworking.org/wp-content/uploads/2018/12/Tofino_Fast_Refresh.pdf

[31] Gurobi. [Online]. Available: https://www.gurobi.com

[32] [Online]. Available: https://docs.openvswitch.org/en/latest/faq/openflow/

[33] libpcap. [Online]. Available: https://www.tcpdump.org

[34] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 267–280. [Online]. Available: https://doi.org/10.1145/1879141.1879175

[35] J. Gao, E. Zhai, H. H. Liu, R. Miao, Y. Zhou, B. Tian, C. Sun, D. Cai, M. Zhang, and M. Yu, "Lyra: A cross-platform language and compiler for data plane programming on heterogeneous asics," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 435–450. [Online]. Available: https://doi.org/10.1145/3387514.3405879

[36] R. Potharaju and N. Jain, "Demystifying the dark side of the middle: A field study of middlebox failures in datacenters," in *Proceedings of the 2013 Conference on Internet Measurement Conference*, ser. IMC '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 9–22. [Online]. Available: https://doi.org/10.1145/2504730.2504737

[37] C. Prakash, J. Lee, Y. Turner, J.-M. Kang, A. Akella, S. Banerjee, C. Clark, Y. Ma, P. Sharma, and Y. Zhang, "Pga: Using graphs to express and automatically reconcile network policies," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 29–42. [Online]. Available: https://doi.org/10.1145/2785956.2787506

[38] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "Hula: Scalable load balancing using programmable data planes," in *Proceedings of the Symposium on SDN Research*, 2016, pp. 1–12.

[39] T. Pan, N. Yu, C. Jia, J. Pi, L. Xu, Y. Qiao, Z. Li, K. Liu, J. Lu, J. Lu, E. Song, J. Zhang, T. Huang, and S. Zhu, "Sailfish: Accelerating cloud-scale multi-tenant multi-service gateways with programmable switches," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, ser. SIGCOMM '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 194–206. [Online]. Available: https://doi.org/10.1145/3452296.3472889

[40] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "Netcache: Balancing key-value stores with fast in-network caching," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 121–136. [Online]. Available: https://doi.org/10.1145/3132747.3132764

[41] Z. Liu, Z. Bai, Z. Liu, X. Li, C. Kim, V. Braverman, X. Jin, and I. Stoica, "Distcache: Provable load balancing for large-scale storage systems with distributed caching," in *17th USENIX Conference on File and Storage Technologies (FAST 19)*. Boston, MA: USENIX Association, Feb. 2019, pp. 143–157. [Online]. Available: https://www.usenix.org/conference/fast19/presentation/liu

[42] C. Lao, Y. Le, K. Mahajan, Y. Chen, W. Wu, A. Akella, and M. Swift, "ATP: In-network aggregation for multi-tenant learning," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, Apr. 2021, pp. 741–761. [Online]. Available: https://www.usenix.org/conference/nsdi21/presentation/lao

[43] L. Mai, L. Rupprecht, A. Alim, P. Costa, M. Migliavacca, P. Pietzuch, and A. L. Wolf, "Netagg: Using middleboxes for application-specific on-path aggregation in data centres," in *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 249–262. [Online]. Available: https://doi.org/10.1145/2674005.2674996

[44] B. Yang, Z. Xu, W. K. Chai, W. Liang, D. Tuncer, A. Galis, and G. Pavlou, "Algorithms for fault-tolerant placement of stateful virtualized network functions," in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–7.

[45] A. Katsarakis, Y. Ma, Z. Tan, A. Bainbridge, M. Balkwill, A. Dragojevic, B. Grot, B. Radunovic, and Y. Zhang, "Zeus: Locality-aware distributed transactions," in *Proceedings of the Sixteenth European Conference on Computer Systems*, ser. EuroSys '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 145–161. [Online]. Available: https://doi.org/10.1145/3447786.3456234

[46] Y. Harchol, A. Mushtaq, V. Fang, J. McCauley, A. Panda, and S. Shenker, "Making edge-computing resilient," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, ser. SoCC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 253–266. [Online]. Available: https://doi.org/10.1145/3419111.3421278

[47] N. Katta, H. Zhang, M. Freedman, and J. Rexford, "Ravana: Controller fault-tolerance in software-defined networking," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, ser. SOSR '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2774993.2774996

[48] Y. Yu, C. Qian, W. Wu, and Y. Zhang, "Netcp: Consistent, non-interruptive and efficient checkpointing and rollback of sdn," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, 2018, pp. 1–10.

[49] H. Zhu, Z. Bai, J. Li, E. Michael, D. R. K. Ports, I. Stoica, and X. Jin, "Harmonia: Near-linear scalability for replicated storage with in-network conflict detection," *Proc. VLDB Endow.*, vol. 13, no. 3, p. 376–389, Nov. 2019. [Online]. Available: https://doi.org/10.14778/3368289.3368301

[50] D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, B. Cheyney, W. Shang, and J. D. Hosein, "Maglev: A fast and reliable software network load balancer," in *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, 2016, pp. 523–535.

[51] P. Patel, D. Bansal, L. Yuan, A. Murthy, A. Greenberg, D. A. Maltz, R. Kern, H. Kumar, M. Zikos, H. Wu, C. Kim, and N. Karri, "Ananta: Cloud scale load balancing," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 207–218. [Online]. Available: https://doi.org/10.1145/2486001.2486026

[52] J. Sherry, P. X. Gao, S. Basu, A. Panda, A. Krishnamurthy, C. Maciocco, M. Manesh, J. a. Martins, S. Ratnasamy, L. Rizzo, and S. Shenker, "Rollback-recovery for middleboxes," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, p. 227–240, Aug. 2015. [Online]. Available: https://doi.org/10.1145/2829988.2787501

[53] S. G. Kulkarni, G. Liu, K. K. Ramakrishnan, M. Arumaithurai, T. Wood, and X. Fu, "Reinforce: Achieving efficient failure resiliency for network function virtualization-based services," *IEEE/ACM Transactions on Networking*, vol. 28, no. 2, pp. 695–708, 2020.

[54] M. Ghaznavi, E. Jalalpour, B. Wong, R. Boutaba, and A. J. Mashtizadeh, "Fault tolerant service function chaining," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 198–210. [Online]. Available: https://doi.org/10.1145/3387514.3405863

[55] Y. Harchol, D. Hay, and T. Orenstein, "Ftvnf: Fault tolerant virtual network functions," in *Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 141–147. [Online]. Available: https://doi.org/10.1145/3230718.3230731

[56] N. Gray, C. Lorenz, A. Müssig, S. Gebert, T. Zinner, and P. Tran-Gia, "A priori state synchronization for fast failover of stateful firewall vnfs," in *2017 International Conference on Networked Systems (NetSys)*, 2017, pp. 1–6.

[57] S. Rajagopalan, D. Williams, and H. Jamjoom, "Pico replication: A high availability framework for middleboxes," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13. New York, NY, USA: Association for Computing Machinery, 2013. [Online]. Available: https://doi.org/10.1145/2523616.2523635

[58] A. Bremler-Barr, Y. Harchol, and D. Hay, "Openbox: A software-defined framework for developing, deploying, and managing network functions," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 511–524. [Online]. Available: https://doi.org/10.1145/2934872.2934875

[59] S. R. Chowdhury, Anthony, H. Bian, T. Bai, and R. Boutaba, "A disaggregated packet processing architecture for network function virtualization," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1075–1088, 2020.

[60] Y. Jiang, Y. Cui, W. Wu, Z. Xu, J. Gu, K. K. Ramakrishnan, Y. He, and X. Qian, "Speedybox: Low-latency nfv service chains with cross-nf runtime consolidation," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2019, pp. 68–79.

[61] L. Zeno, D. R. K. Ports, J. Nelson, and M. Silberstein, "Swishmem: Distributed shared state abstractions for programmable switches," in *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, ser. HotNets '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 160–167. [Online]. Available: https://doi.org/10.1145/3422604.3425946

[62] D. Kim, J. Nelson, D. R. K. Ports, V. Sekar, and S. Seshan, "Redplane: Enabling fault-tolerant stateful in-switch applications," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, ser. SIGCOMM '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 223–244. [Online]. Available: https://doi.org/10.1145/3452296.3472905

[63] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "Opennf: Enabling innovation in network function control," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: Association

for Computing Machinery, 2014, p. 163–174. [Online]. Available: https://doi.org/10.1145/2619239.2626313

[64] M. Pozza, A. Rao, D. F. Lugones, and S. Tarkoma, "Flexstate: Flexible state management of network functions," *IEEE Access*, vol. 9, pp. 46 837–46 850, 2021.

[65] S. Woo, J. Sherry, S. Han, S. Moon, S. Ratnasamy, and S. Shenker, "Elastic scaling of stateful network functions," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, Apr. 2018, pp. 299–312. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/woo

[66] T. V. Doan, C. Ding, G. T. Nguyen, D. You, and F. H. P. Fitzek, "Fast: Flexible and low-latency state transfer in mobile edge computing," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 2020, pp. 1–6.

[67] J. Khalid and A. Akella, "Correctness and performance for stateful chained network functions," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. Boston, MA: USENIX Association, Feb. 2019, pp. 501–516. [Online]. Available: https://www.usenix.org/conference/nsdi19/presentation/khalid

[68] D. Kim, Z. Liu, Y. Zhu, C. Kim, J. Lee, V. Sekar, and S. Seshan, "Tea: Enabling state-intensive network functions on programmable switches," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 90–106.

[69] 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 *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2019, pp. 1–9.

[70] D. Wu, A. Chen, T. S. E. Ng, G. Wang, and H. Wang, "Accelerated service chaining on a single switch asic," in *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, ser. HotNets '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 141–149. [Online]. Available: https://doi.org/10.1145/3365609.3365849

[71] K. Zhang, D. Zhuo, and A. Krishnamurthy, "Gallium: Automated software middlebox offloading to programmable switches," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 283–295. [Online]. Available: https://doi.org/10.1145/3387514.3405869

[72] N. Sultana, J. Sonchack, H. Giesen, I. Pedisich, Z. Han, N. Shyamkumar, S. Burad, A. DeHon, and B. T. Loo, "Flightplan: Dataplane disaggregation and placement for p4 programs," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, Apr. 2021, pp. 571–592. [Online]. Available: https://www.usenix.org/conference/nsdi21/presentation/sultana

[73] H. Liu, X. Chen, Q. Huang, H. Zhou, D. Zhang, and C. Wu, "Sra: Switch resource aggregation for application offloading in programmable networks," in *GLOBECOM 2020*, 2020, pp. 1–6.

[74] X. Chen, H. Liu, Q. Huang, P. Wang, D. Zhang, H. Zhou, and C. Wu, "Speed: Resource-efficient and high-performance deployment for data plane programs," in *ICNP' 20*, 2020, pp. 1–12.

[75] J. Ma, S. Xie, and J. Zhao, "P4sfc: Service function chain offloading with programmable switches," in *2020 IEEE 39th International Performance Computing and Communications Conference (IPCCC)*, 2020, pp. 1–6.

[76] D. Moro, G. Verticale, and A. Capone, "A framework for network function decomposition and deployment," in *2020 16th International Conference on the Design of Reliable Communication Networks DRCN 2020*, 2020, pp. 1–6.

[77] Y. Zhou, J. Bi, C. Zhang, M. Xu, and J. Wu, "Flexmesh: Flexibly chaining network functions on programmable data planes at runtime," in *2020 IFIP Networking Conference*, 2020, pp. 73–81.

[78] Y. Xue and Z. Zhu, "Leveraging heterogeneous nfv platforms to upgrade service function chains in dcns," in *NetSoft' 21*, 2021, pp. 283–287.

[79] D. Moro, G. Verticale, and A. Capone, "Network function decomposition and offloading on heterogeneous networks with programmable data planes," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 1874–1885, 2021.