SmartChain: Enabling High-Performance Service Chain Partition between SmartNIC and CPU

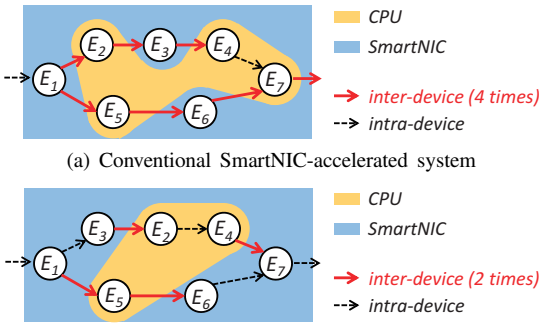
Shuhe Wang[1,2], Zili Meng[1,2], Chen Sun[1,2,6], Minhu Wang[1,2],
Mingwei Xu[1,2], Jun Bi[1,2], Tong Yang[3], Qun Huang[4], Hongxin Hu[5]
[1]Institute for Network Sciences and Cyberspace, Tsinghua University
[2]Beijing National Research Center for Information Science and Technology (BNRist)
[3]Peking University, [4]Chinese Academy of Sciences, [5]Clemson University, [6]Alibaba Group
{wangshuh18, wangmh19}@mails.tsinghua.edu.cn, zilim@ieee.org, qichen.sc@alibaba-inc.com
{xumw, junbi}@tsinghua.edu.cn, yangtongemail@gmail.com, huangqun@ict.ac.cn, hongxih@clemson.edu

***Abstract*—Smart Network Interface Cards (SmartNICs) have been widely used to accelerate software-based network functions (NFs). However, from the scope of a service chain, a careless selection of NFs to offload onto SmartNIC could severely degrade the performance due to frequent communications between CPU and SmartNIC. In this paper, we present SmartChain, a high performance and efficient framework that achieves optimal partition of service chains between SmartNIC and CPU. SmartChain consists of two logical steps. First, SmartChain analyzes the suitability of elements in a chain to run on SmartNIC to exploit its high performance. Besides, SmartChain also ensures the dependencies between elements. Second, as our key novelty, SmartChain models the service chain latency and resource constraints, and solves the partition problem with 0-1 integer linear programming. We implement a SmartChain prototype based on Netronome SmartNIC. Evaluation results show that when used in real world cases, SmartChain could reduce the service chain latency by up to 87% with throughput maintained compared with strawman solutions.**

I. Introduction

Network Function Virtualization have replaced *dedicated* hardware middleboxes with *virtualized* Network Functions (*vNFs*), providing flexibility to the development and management of network functions (NFs). In common NFV scenarios, network operators usually require flows to be processed by multiple NFs in a certain sequence, which is known as a *service chain* [1]. Meanwhile, recent research proposed to divide a NF into several *functional elements* [2] to further achieve individual scalability and reusability of elements. Sometimes the processing result of an element decides which downstream element should process the packet next, turning a sequential chain into an *element graph* with branches [2].

However, the major drawback of NFV is its low performance. Especially, software-based NF implementations have high processing latency (*e.g.*, Ananta Software Muxes running on commodity servers can add from 200 μs to 1 ms latency at 100 Kpps [3]). Applications, such as real-time analytics and Online Data-Intensive (OLDI) applications [4, 5], work under tight latency constraints(a few μs) so the latency above is hardly acceptable. To reduce the performance degradation, there is a recent trend of using Smart Network Interface Cards (SmartNICs) to apply in-network NF acceleration [5–7]. These researches proposed to offload *some specific kinds* of vNFs onto SmartNIC to make improvement.



(a) Conventional SmartNIC-accelerated system

(b) Redundant transmission avoidance with SmartChain

Figure 1. Conventional element graph datapath *v.s.* SmartChain datapath with optimal partition and placement. Inter-device transmissions (red arrows) between SmartNIC and CPU lead to additional element graph latency.

Unfortunately, existing works mainly focused on optimizing the performance of a *single* element of a service chain rather than the entire element graph. When some elements are offloaded to SmartNIC, the rest in the graph remain in CPU. Without carefully *placing* the elements, packets could be frequently transmitted between CPU and SmartNIC over Peripheral Component Interconnect Express (PCIe), incurring drastic performance degradation. For example, in Figure 1(a), the conventional smartNIC-accelerated way incurs *four* times of transmissions between CPU and SmartNIC (red arrows). Our preliminary evaluation in §II demonstrates that the round-trip transmission latency between SmartNIC and CPU could add up to 26μs, the same order of magnitude with element processing latency (tens or hundreds of μs in Table I). Therefore, 4 times of transmissions will quadruple the latency to more than 50 μs, which is unacceptable for those latency-sensitive applications under microsecond-level constraints.

To address this problem, we observe that redundant SmartNIC-CPU communications inside the service chain can be avoided by *optimizing the placement* of the elements. For the same element graph in Figure 1(a), if we adjust element placement into Figure 1(b), the total number of PCIe transmissions of each forwarding path can be decreased from 4 to 2. Therefore, orthogonal to current solutions that focus on accelerating a single element *within* SmartNIC devices, we consider from the scope of an *element graph* and exploit the opportunity to enhance the performance of element graphs

by carefully designing element placement mechanisms across SmartNIC and CPU to reduce packet transmissions.

However, there exist other *inherent factors* that also affect the performance, preventing us from achieving an optimal placement solution. First, elements have *inherent suitability* on SmartNIC. Naively placing *adjacent* elements onto SmartNIC to avoid redundant packet transmission may place *unsuitable* elements onto SmartNIC and drastically compromise performance. For example, offloading I/O-intensive elements, such as loggers, brings significant latency as PCIe will be frequently occupied for memory I/O. To address such a challenge, our *intuition* is to *reconstruct the element graph* to ensure that suitable elements offloaded to SmartNIC are placed together. However, this raises another challenge that reconstruction needs to respect the *inherent dependencies* between graph elements, *i.e.*, processing packets in sequence.

To address the above challenges, we propose SmartChain to optimize the element graph latency. SmartChain consists of two logical steps. First, we analyze the *suitability* of an element to be offloaded to SmartNIC, and identify inter-element dependency to quickly extract two types of *dependency* constraints between elements. Next, to minimize communication between devices with respect to element placement suitability, we carefully select the optimization objective as minimizing the *sum* of the processing latency and the transmission latency. We construct models to represent element dependency constraints and device resource constraints, and formulate the partition and placement problem using 0-1 linear programming. The placement may reconstruct the original element graph, but with the dependency constraints identified before, we can ensure packet processing consistency of the placement result. The overview and workflow of SmartChain framework is shown in Figure 2.

SmartChain makes the following contributions:

- We identify the critical performance problem in SmartNIC-based NFV systems: redundant packet transmissions at the scope of element graph. We then propose SmartChain to eliminate the redundancy with little overhead.
- We design a *Suitability Analyzer* as well as a *Dependency Analyzer* to retrieve the suitability of elements on SmartNIC and inter-element dependency, and an *Optimized Placer* to generate an optimal partition plan with respect to suitability and dependency.
- We implement the prototype of SmartChain framework on SmartNICs and 6 categories of elements. Extensive experimental results show that SmartChain can reduce the overall latency up to 87 %.

II. Motivation: Inter-Device Latency Breakdown

We understand the source of service chain latency by breaking down the packet transmission process between SmartNIC and CPU with a preliminary experiment based on a Netronome SmartNIC. We present the datapath of packets on SmartNIC in Figure 3(a) [8]. Packets go into SmartNIC from outside network via the Enhanced Network Interface (ENI). After checking checksum and parsing packets, ENI sends packets

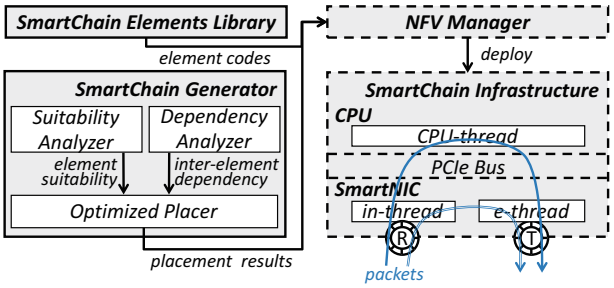
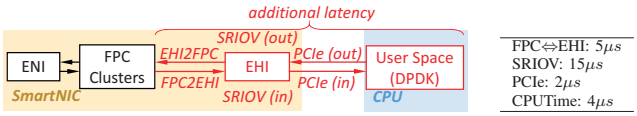


Figure 2. SmartChain Framework Overview.



(a) Datapath. (b) Latency.

Figure 3. Round-trip inter-device latency breakdown.

to Flow Processing Core (FPC) clusters for packet processing. If no vNF on the remaining service chain is placed onto CPU, packets will remain on FPC clusters with *run-to-completion*. Otherwise, packets will be transmitted to CPU. We implement a bouncer with DPDK on CPU to bounce all packets back to SmartNIC, and measure the latency of each stage through the data path of packets that visit CPU. We send packets of 64B and present the measurement results of per-packet latency in Figure 3(b). The average round-trip latency is 26μs in total, which comprises the following parts:

- **FPC↔EHI**. When packets need to be transmitted from SmartNIC to CPU, they are sent from FPC clusters to the Enhanced Host Interface (EHI) at first [8]. By timestamping the two components, the forwarding latency in the round trip between them on SmartNIC takes about 5μs.
- **SRIOV**. SmartNIC adopt Single Root I/O Virtualization (SRIOV) [9] to improve cooperation between CPU and NICs. The round-trip virtualization takes about 15μs.
- **PCIe**. Packets will then be transmitted over PCIe bus, which takes about 2 2μs [10]. Moreover, when there are multiple inter-device transmissions, the competition over PCIe bus between different vNFs can add insult to injury.
- **CPU Preprocessing**. Finally, before packets are fetched into user space, they still need to wait at the enqueue and batching stage [11]. The processing time on CPU is 4μs, which is the pre- and post-processing cost since we only implement a bouncer at the CPU side.

As we discussed in §I, this latency is unacceptable for latency-sensitive applications. This motivates us to reduce the inter-device transmission.

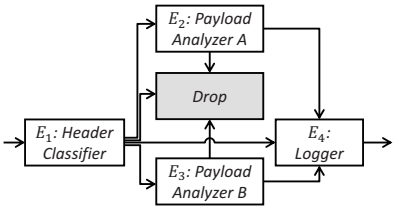
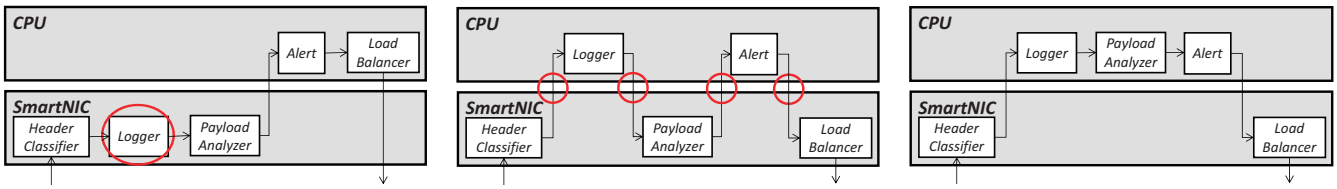


Figure 4. An example of element graph with branches that identifies and drops malicious packets.



(a) Sequential Algorithm (b) Suitability-greedy Algorithm (c) SmartChain *Optimized Placer*

Figure 5. Placement results of strawman solutions and SmartChain *Optimized Placer*.

III. Dependency and Suitability Analysis

This section efficiently make a dependency (§III-A) and suitability (§III-B) analysis to yield inputs for the placement algorithm in *Optimized Placer* (§IV).

A. Dependency Analyzer

Elements in the same graph may have dependencies on each other when processing packets [4]. Therefore, we need to identify element dependencies before graph reconstruction. We classify the dependencies into two categories:

Branch Dependency means that for an element with multiple downstream elements, which downstream element will be executed depends on the processing result of its upstream element. As shown in Figure 4, the classification result of E_1 determines whether E_2 or E_3 is going to be executed. Thus E_2 and E_3 must be placed after E_1. With $\trianglelefteq$ denoting the dependency between elements, we have $E_1 \trianglelefteq E_2,\ E_1 \trianglelefteq E_3$.

Action Dependency depicts those elements that have *read-write* dependencies on each others [4]. For example, NAT and Load Balancer both modify the IP address on packets. Switching the execution sequence of them will lead to the wrong output packets. For each pair of elements, we first identify the actions that elements exert on packets, including `read-only`, `write`, and `other` actions (*e.g.* packet drop or encapsulation). Then we find the packets fields the two actions operate, and refer to [4] for further dependency identification.

B. Suitability Analyzer

To decide whether element E_e is suitable to place onto SmartNIC, we select *latency* as our decision metric to satisfy latency-sensitive situations introduced in §I. Nonetheless, the following analysis can be easily extended to throughput-sensitive cases by changing the decision metric if needed.

To quantify the suitability of element E_e, we first separately *measure* the processing latency when placing it on CPU ($t_e^{\mathcal{C}}$) and on SmartNIC ($t_e^{\mathcal{S}}$). As the latency of the total element graph is the *sum* of each element's latency on the forwarding path, we intuitively say an element is more suitable to place on SmartNIC rather than CPU when less processing latency is required on SmartNIC, and vice versa. Therefore, we define the *suitability* of element E_e, denoted s_e, as the difference of processing latency between SmartNIC and CPU:

$$s_e = t_e^{\mathcal{C}} - t_e^{\mathcal{S}} \tag{1}$$

From the view of Equation (1), elements with a larger portion of processing latency difference emphatically have greater impacts on suitability.

$$\max \quad Benefit_{trans} + Benefit_{proc}$$

s.t.

$$\begin{array}{lllll}
(\mathbb{C}_1) & \forall (E_i \trianglelefteq E_j) \in \mathcal{D}\text{:} \sum_{p\in\mathcal{P}} \left(p \cdot x_j^p\right) & \geqslant & \sum_{p\in\mathcal{P}} \left(p \cdot x_i^p\right) \\
(\mathbb{C}_2) & \theta^{est} \cdot \sum_e \frac{1}{\theta_e^{\mathcal{S}}} \left(x_e^{\mathcal{S}_{ig}} + x_e^{\mathcal{S}_{eg}}\right) & \leqslant & 1 \\
(\mathbb{C}_3) & \theta^{est} \cdot \sum_e \frac{1}{\theta_e^{\mathcal{C}}} x_e^{\mathcal{C}} & \leqslant & 1 \\
(\mathbb{C}_4) & \forall e\text{:} \quad \sum_{p\in\mathcal{P}} x_e^p & = & 1 \\
(\mathbb{C}_5) & \forall e\text{:} \quad y & \geqslant & x_e^{\mathcal{C}}
\end{array}$$

Figure 6. SmartChain 0-1 ILP placement algorithm.

IV. Optimized Placer

The bump-in-the-wire SmartNICs [12] offer three natural positions to place elements including the ingress part of SmartNIC ($\mathcal{S}_{ig}$), CPU ($\mathcal{C}$), and the egress part of SmartNIC ($\mathcal{S}_{eg}$). Dumbly examining all n elements' positions one by one takes up $O(3^n)$ time, which becomes impractical as n grows up to a huge number. We first introduce two strawman solutions to the problem, and identify their limitations (§IV-A). We then present our key design of modelling the problem with 0-1 integer programming (§IV-B).

A. Strawman Solutions for Placement

Sequential Placement. A strawman idea is to place elements onto SmartNIC according to the processing sequence in the element graph until SmartNIC resources are exhausted. Other elements are then placed onto CPU. This avoids redundant packet transmissions. However, it may place elements to unsuitable places thus increase the processing latency. Figure 5(a) shows the placement result by the sequential algorithm of an example 5-element chain. It places the first three elements onto SmartNIC. However, *Logger* is I/O-intensive and writes back to memory frequently. Thus placing it onto SmartNIC leads to drastic performance degradation.

Suitability-greedy Placement. Another strawman solution is to place each element onto SmartNIC or CPU solely depending on its suitability. Elements are greedily checked in sequence. This can maximize the total suitability on SmartNIC. However, when the positions of those high-suitability elements are discontinuous in the element chain, this leads to redundant packet transmissions over PCIe. As shown in Figure 5(b), for example, packets have to be transmitted between CPU and SmartNIC for 4 times, which also increases the total latency.

Combining their strong assets and circumventing their weak points, we next explain our solution in detail.

B. Our solution: A 0-1 Programming Based Model

In contrast to the strawman solutions above, *Optimized Placer* aims at: 1) placing the most suitable elements onto

SmartNIC; and 2) avoiding redundant inter-device packet transmissions. To achieve *both* goals simultaneously, we model the problem into a 0-1 Integer Linear Programming (0-1 ILP) problem with dependency and suitability considered. We use a set of variables $x_e^p \in \{0,1\}$ to indicate whether element $e \in \{1, \cdots, N_e\}$ is placed onto $p \in \mathcal{P} = \{\mathcal{S}_{ig}, \mathcal{C}, \mathcal{S}_{eg}\}$ or not.

An overview of SmartChain placement algorithm is presented in Figure 6. There are three steps in optimizing the placement of the element graph. First, we need to analyze and model the optimization objective to achieve as much element graph latency benefits as possible. Next, we need to formulate the dependency and resource consumption depictions into solvable constraints. Finally, we need to interpret the placement results to a readily deployable element graph.

1) Optimization Objective: We set the objective of optimization as maximizing the latency benefits after partition and placement. The latency benefit contains two parts:

SmartNIC-CPU Transmission Benefit. To avoid redundant inter-device transmissions, there are at most one round-trip transmission between SmartNIC and CPU. But if all elements are placed onto SmartNIC, packets won't have to detour to CPU. This latency can then be expressed as:

$$Benefit_{trans} = t_{trans} \cdot \left[1 - sgn\left(\sum_e x_e^{\mathcal{C}}\right)\right] \tag{2}$$

$sgn(\cdot)$ denotes the sign of the value inside the parentheses.

We introduce an auxiliary variable for linearization:

$$y \in \{0,1\},\ s.t.\ \forall e,\ y \geqslant x_e^{\mathcal{C}} \tag{3}$$

$Benefit_{trans}$ can then be linearized as:

$$Benefit_{trans} = t_{trans} \cdot (1 - y) \tag{4}$$

Proof. If $\exists E_0\ s.t.\ x_0^{\mathcal{C}} = 1$, from Equation 3, $y \geqslant x_0^{\mathcal{C}} = 1$. As y is a 0-1 variable, we have $y = 1$. If $\forall E_e\ s.t.\ x_e^{\mathcal{C}} = 0$, y is feasible for both 0 and 1. However, as the coefficient of y in the objective is $\max(-t_{trans}y)$, y will equal to 0 to maximize the objective in the optimal solution. □

SmartNIC Processing Benefit. As we have discussed in §III-B, the processing ability of SmartNIC is different from CPU and varies across elements. Since suitability is defined as the difference of processing latency, the latency benefit gained by processing with SmartNIC can be expressed as:

$$Benefit_{proc} = \sum_e s_e \cdot \left(x_e^{\mathcal{S}_{ig}} + x_e^{\mathcal{S}_{eg}}\right) \tag{5}$$

Combining two parts of latency benefit together, we present the *linearized* optimization objective in Figure 6.

2) Constraints: We identify three types of constraints here:

Dependency Constraint ($\mathbb{C}_1$). We denote all dependency constraints extracted in §III-A as set $\mathcal{D}$. For each pair $(E_i \trianglelefteq E_j)$ in $\mathcal{D}$, element E_i should be placed at the upstream of or together with E_j. Thus E_j can be placed onto p_{j0} if and only if E_i has already been placed onto p_{i0} and $p_{i0} \leqslant p_{j0}$, *i.e.*

$$p_{i0} \leqslant p_{j0} \quad where \quad x_i^p = \delta(p - p_{i0}), x_j^p = \delta(p - p_{j0}) \tag{6}$$

$\delta(p)$ is the impulse function. with the shifting property of $\delta(p)$, we have:

$$p_{i0} = \sum_p (p \cdot \delta(p - p_{i0})) = \sum_p (p \cdot x_i^p) \tag{7}$$

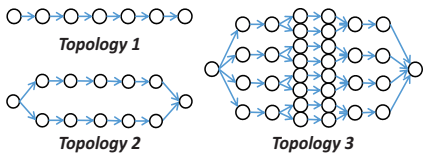


Figure 7. Topologies for the simulations of placement algorithms.

Resource Constraint ($\mathbb{C}_2, \mathbb{C}_3$). Inspired by discussions on the appliance capacity [13] and the linearity between throughput and CPU utilization [14], we use the *ratio of throughput* to estimate the resource utilization of elements on SmartNIC and CPU. For E_e, we measure its maximum throughput capacity when it is exclusively placed onto CPU ($\theta_e^{\mathcal{C}}$) and SmartNIC ($\theta_e^{\mathcal{S}}$). Thus for an estimated *graph* throughput θ^{est} (can be estimated with historical traffic or service agreements), the estimated ratios of resource consumed by E_e are:

$$r_e^{\mathcal{C}} = \frac{\theta^{est}}{\theta_e^{\mathcal{C}}}\left(x_e^{\mathcal{C}}\right), \quad r_e^{\mathcal{S}} = \frac{\theta^{est}}{\theta_e^{\mathcal{S}}}\left(x_e^{\mathcal{S}_{ig}} + x_e^{\mathcal{S}_{eg}}\right) \tag{8}$$

Thus we can sum up all the resource utilization of elements and get constraints ($\mathbb{C}_2$) and ($\mathbb{C}_3$) in Figure 6. The estimation here is the upper bound estimation of resource utilization because when there are branches in the element graph, throughput on each branch is less than the total estimated graph throughput. Meanwhile, previous work shows the bandwidth of PCIe won't be a limitation [7].

Variable Constraint ($\mathbb{C}_4, \mathbb{C}_5$). All elements should be placed only once. Moreover, for y, Equation (3) should be satisfied.

After formulating the model above, we can solve the 0-1 ILP problem and get the optimal placement results quickly with some state-of-the-art optimization toolboxes. If the solver is unable to find out a feasible solution, the network operator may relax the throughput estimation, or add additional devices (*e.g.* more CPU cores or SmartNICs).

3) Multiple Servers and Multiple Graphs: For simplicity, we have only discussed the single server scenario. For multiple servers, SmartChain could integrate with current inter-server service graph optimization solutions [15–17] to further improve the performance. Network operators could take state-of-the-art server placement solutions to decide the location of NF instances at the server level, and employ SmartChain to decide the target device inside each server. For multiple graphs, since SmartChain models at the level of elements rather than graphs, they could be optimized together as one big graph with dependencies inside each service graph constrained. We evaluate the scalability of SmartChain on multiple servers and multiple graphs in §V-D.

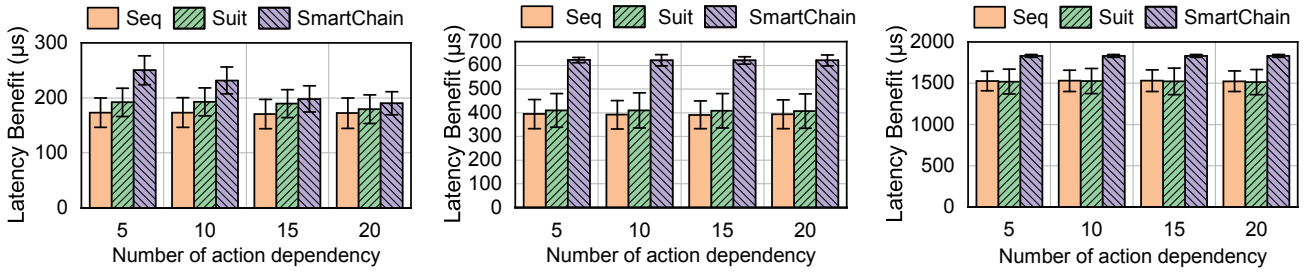
V. Implementation and Evaluation

In this section, we first introduce our testbed and the elements implemented and used for experiments [§V-A]. Then we evaluate SmartChain with the following five goals:

- We present the measurement methods and statistics of suitability and resource capacity of elements, and demonstrate the validity of our assumptions [§V-B].
- We demonstrate that SmartChain placement algorithm is *effective* with simulations under different topologies and element dependencies [§V-C].

Table I
ELEMENTS IMPLEMENTED WITH SMARTCHAIN. $\theta_e^{\mathcal{C}}, \theta_e^{\mathcal{S}}, t_e^{\mathcal{C}}$ AND $t_e^{\mathcal{S}}$ ARE THE CAPACITY ON SMARTNIC AND CPU, AND THE PROCESSING LATENCY ON SMARTNIC AND CPU. $s_e = t_e^{\mathcal{C}} - t_e^{\mathcal{S}}$ IS THE SUITABILITY.

Element	Descriptions	$\theta_e^{\mathcal{C}}$	$\theta_e^{\mathcal{S}}$	$t_e^{\mathcal{C}}$	$t_e^{\mathcal{S}}$	s_e
Header Classifier	Classify the packets based on their 5-tuples.	4Gbps	>10Gbps	14.8μs	0.8μs	14μs
Logger	Log the current flow processing information periodically.	4Gbps	2Gbps	15μs	245μs	-230μs
Monitor	Measure the statistics of flows with a k-ary sketch [18].	10Gbps	3.2Gbps	4μs	24μs	-20μs
Payload Analyzer	Match the payload in packets with a specific string.	200Mbps	5Gbps	800μs	65μs	735μs
Alert	Alert NFV manager with information about packets.	4Gbps	>10Gbps	15.1μs	2.1μs	13μs
Load Balancer	Balance flows to different ports by hashing packet headers.	4Gbps	>10Gbps	14.5μs	0.5μs	14μs



(a) Topology 1. (b) Topology 2. (c) Topology 3.

Figure 8. Latency benefit of SmartChain and two strawman solutions.

- We demonstrate that SmartChain is *efficient* and *scalable* in terms of computation overheads [§V-D].
- We demonstrate that the *robustness* of SmartChain placement algorithm to achieve a near-optimal solution even with inaccurate parameters [§V-E].
- We demonstrate that SmartChain could significantly improve performance with the implementation of two real world service chains based on our testbeds [§V-F].

A. SmartChain Infrastructure

We take the Netronome® NP-based SmartNIC as our implementation platform. The testbed consists of two servers. Each server is equipped with two Intel® Xeon® E5-2620 v2 CPUs (2.10 GHz, 6 physical cores) and 128G RAM. One server is used as packet sender and receiver. The other one is equipped with one Agilio CX 2×10 GbE *NP-based* SmartNIC [6] and used for packet processing.

We implement six elements in Table I, both on SmartNIC and CPU with DPDK [11], which cover most of element types classified in [2]. The element codes are written in C and P4 on SmartNIC and in C with DPDK library in CPU.

B. Suitability and Resource Capacity of Elements

We measure the suitability and resource capacity of the elements above with our testbed. For each element E_e, we use timestamps on SmartNIC and profiling tools on CPU to measure its latency and measure its maximum throughput as a conservative estimation of its capacity. See Table I for results.

We also verify the assumption of linearity between throughput and utilization on SmartNIC in Equation (8). Since SmartNIC does not provide tools to inspect its utilization, we cannot verify the assumption directly. Instead, we verify a deduction of the assumption: If we compose element E_1 and E_2 to an element chain and place them onto SmartNIC together, according to Equation (8) and the resource constraints ($\mathbb{C}_2$) in Figure 6, the capacity θ' of the chain $E_1 \Rightarrow E_2$ should satisfy:

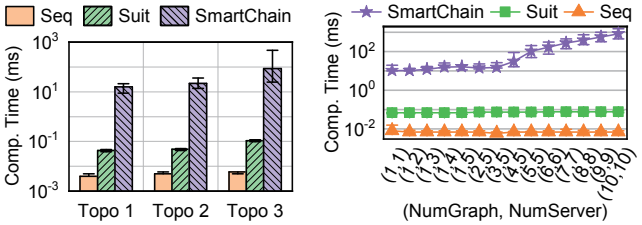
$$\frac{\theta'}{\theta_1^{\mathcal{S}}} + \frac{\theta'}{\theta_2^{\mathcal{S}}} = 1 \;\Rightarrow\; \theta' = \frac{\theta_1^{\mathcal{S}}\theta_2^{\mathcal{S}}}{\theta_1^{\mathcal{S}}+\theta_2^{\mathcal{S}}} \tag{9}$$

Thus we measure the capacity of placing "*Payload Analyzer* ⇒ *Monitor*" onto SmartNIC together. Our evaluation shows that the capacity of the element chain is 1.8Gbps, which is close to the theoretical value of 1.9Gbps calculated from Table I and Equation (9). This demonstrates the approximate linear relationship between throughput and utilization in SmartNIC. However, there is also a minor throughput degradation due to the coupling of multiple elements since SmartNIC does not provide mature isolation techniques.

Sometimes the processing latency and the resource capacity of software-based elements may fluctuate and affect the accuracy of measurement. In this case, we can send more packets and take the average values until the accuracy is acceptable.

C. Simulations of SmartChain Placement Algorithm

We measure the total latency benefits of element graphs. Three typical graph topologies(Figure 7), including a linear topology (Topo 1), a two-branch topology (Topo 2), and a nested multi-branch topology (Topo 3), are used with different numbers of elements and branches. We pick the elements from Table I randomly with the probability of their ratios in real world enterprise networks [4]. We compare the total latency benefit of SmartChain with the two strawman solutions. The 0-1 ILP problem is solved with the `intlinprog` function in MATLAB R2018a on our server. As the branch dependencies for topologies in Figure 7 have already been decided by the structure of graph, we vary the number of action dependencies from 5 to 20. We randomly select element pairs to follow action dependency. The experiment is repeated for 1000 times to eliminate the randomness.



(a) Three topologies. (b) Multiple servers and graphs.

Figure 9. The computation time of SmartChain placement algorithm. Error bars represent the 10^{th} and 90^{th} percentiles.

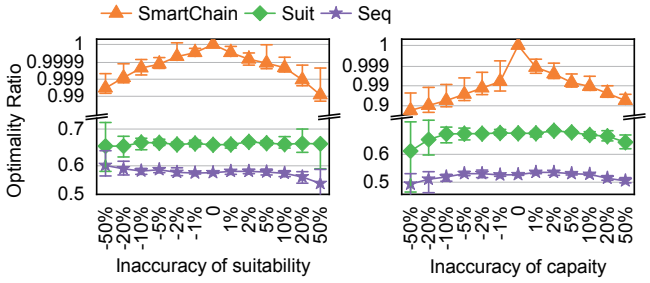


Figure 10. Sensitivity analysis of parameters.

As shown in Figure 8, SmartChain has earned at most 320 μs of the total latency benefit gain and up to 58% compared to two strawman solutions. Especially, the benefit in Topo 1 decreases as the number of action dependency increases. This is due to the scale of the element graph. Meanwhile, the improvement of SmartChain with Topo 3 is less than that with Topo 1. This is because the numerous branch dependencies in topology restrict the reconstruction of SmartChain, which is rare in real world service chains.

D. SmartChain Overhead and Scalability

We measure the computation time of the three placement algorithms under the simulation environments in §V-C, as shown in Figure 9(a). Even for the most complex topology, SmartChain can optimize most of situations within $0.5s$. As the placement calculation is done offline before element deployment, the time is negligible compared to the minute-level time for program compilation and element deployment.

We then evaluate the scalability of SmartChain placement algorithm on multiple-server, multiple-graph scenarios. We can merely extend the placement destination set $\mathcal{P}$ and consider multiple graphs as one big element graph. We vary the number of graphs from 1 to 10 by duplicating Topo 2 and vary the number of servers from 1 to 10. Results are shown in Figure 9(b). Even with 10 graphs and 10 servers, SmartChain can optimizes the placement at approximately $1s$. Since the algorithm is obliged to get executed *only once* before the deployment, the time is acceptable for *offline computation*.

E. Sensitivity Analysis

To demonstrate the robustness of SmartChain placement algorithm, we vary the inaccuracy of parameters. +5% inaccuracy indicates that if the measured (inaccurate) value is a, the accurate value is $a/(1+5\%)$. We define the optimality ratio of different algorithms $alg \in \{seq, suit, opt\}$ as:

$$OptRatio(alg) = \frac{Q(inaccurate, alg)}{Q(accurate, opt)} \quad (10)$$

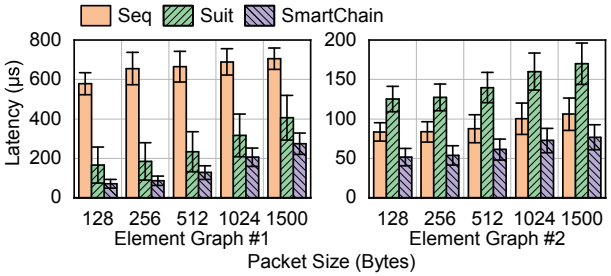


Figure 11. Average latency of element graphs.

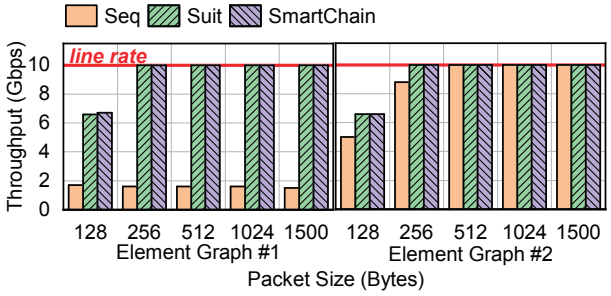


Figure 12. Average throughput of element graphs.

where $Q(inaccurate/accurate, alg)$ is the placement results of algorithm alg calculated with inaccurate / accurate parameters. $\{seq, suit, opt\}$ stands for sequential, suitability-greedy, and SmartChain placement algorithms respectively.

We measure the optimality ratio under inaccurate suitability (s_e) and capacity ($\theta_e^{\mathcal{C}}$ and $\theta_e^{\mathcal{S}}$) of elements. The inaccuracy is varied from -50% to 50%. We take Topo 1 and randomly select one out of the 7 elements to change its parameter. Especially, for capacity, we randomly choose one of $\theta_e^{\mathcal{C}}$ and $\theta_e^{\mathcal{S}}$ to change. We adopt the *push-aside migration* [14] when the inaccuracy leads to overload. For each element and inaccuracy, we repeat the experiment for 1000 times. As shown in Figure 10, we present the average, maximal and minimal results.

For suitability, the optimality ratio of SmartChain is greater than 99% under different inaccuracy. The latency benefit of SmartChain still outperforms the benefits of two strawman solutions significantly. For element capacity, as the inaccurate parameters may lead to overload and element migration, the optimality ratio of SmartChain is about 90% on average under different inaccuracy, which still achieve a higher optimality ratio than other solutions (50%-65%). The parameter inaccuracy of an element in the graph has very limited influence on the optimality of SmartChain placement algorithm.

F. Performance Improvement with Real World Service Chains

We pick two service chains in data centers and break them into element graphs according to [2]. Element graph #1 is from the west-east service chain in [4] for intrusion detection with 6 elements and 2 branches. Element graph #2 is from [20] with three levels of monitors and firewalls for different purposes and has 18 elements and 6 branches in total. We compare the performance of placement results on our testbed of SmartChain with two strawman solutions. We vary the packet size and measure the latency and throughput of the element graph. Results are shown in Figure 11 and 12.

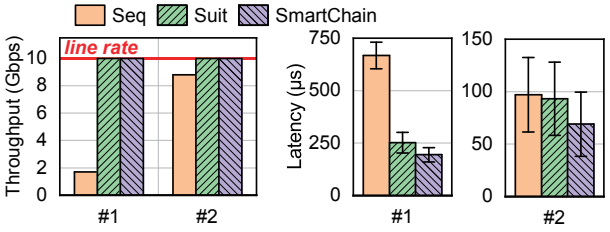


Figure 13. Performance with real-world traffic [19].

SmartChain decreases the element graph latency by at most 87% for the element graph #1 and by at most 38% for the element graph #2 compared to the sequential algorithm. Compared to the suitability-greedy algorithm, SmartChain also decreases the element graph latency by up to 56% for two element graphs. Meanwhile, SmartChain achieves the throughput of suitability-greedy algorithm for both graphs at different packet sizes.

We then measure the performance of two element graphs with *real-world* CAIDA traffic [19], as shown in Figure 13. SmartChain decreases the element graph latency by from 30% to 80% compared to strawman solutions, with line rate maintained in throughput.

VI. Related Work

NF hardware-acceleration. Many recent researches focused on accelerating NFV with GPUs [21, 22], SmartNIC [5, 7, 23] or even programmable switches [24, 25] to certain NFs for higher performance. Meanwhile, there are also some acceleration mechanisms with last-hop switches [26] or even conventional NICs [27]. However, all works above were mainly at the single NF level while SmartChain focuses on the coordination and partition at the scope of service chains.

Service chain placement. There are many researches on service chains or vNFs placement on different servers under different scenarios in NFV [15–17], discussing trade-off between link capacity, resource efficiency, and service chain performance. However, they focused on the placement at network-wide level across multiple servers instead of device-level across CPU and SmartNIC in a single server. [12] took a step on the placement between offloading devices and CPUs. But it focused on optimizing the energy efficiency without considering element dependencies in a service chain. In contrast, SmartChain is designed to optimize the latency benefit based on the dependency and suitability of elements.

VII. Conclusion

This paper presents SmartChain, a high performance element graph partition framework between SmartNIC and CPU. We identify the problem of redundant packet transmissions when applying hardware acceleration techniques to element graphs. SmartChain innovatively proposes a high-performance element graph partition and placement algorithm based on 0-1 programming. Evaluation shows that SmartChain could reduce the total latency by up to 87% with real world element graphs compared with strawman solutions. In the future, we plan to enhance SmartChain with more complicated scenarios such as vNF dynamic migration, priorities on different branches inside element graphs, and generality of SmartChain on other types of hardware devices.

VIII. Acknowledgement

We thank anonymous reviewers for their valuable comments. This research is supported by National Key R&D Program of China (2017YFB0801701) and the National Science Foundation of China (61625203, 61832013, 61872426). Mingwei Xu is the corresponding author.

References

[1] J. Halpern and C. Pignataro, "Service function chaining (sfc) architecture," *IETF RFC 7665*, 2015.
[2] A. Bremler-Barr, Y. Harchol, and D. Hay, "Openbox: a software-defined framework for developing, deploying, and managing network functions," in *Proc. ACM SIGCOMM*, 2016.
[3] R. Gandhi, H. H. Liu, Y. C. Hu *et al.*, "Duet: Cloud scale load balancing with hardware and software," in *Proc. ACM SIGCOMM*, 2014.
[4] C. Sun, J. Bi, Z. Zheng *et al.*, "Nfp: Enabling network function parallelism in nfv," in *Proc. ACM SIGCOMM*, 2017.
[5] B. Li, K. Tan, L. L. Luo *et al.*, "Clicknp: Highly flexible and high performance network processing with reconfigurable hardware," in *Proc. ACM SIGCOMM*, 2016.
[6] Netronome. Agilio cx dual-port 10 gigabit ethernet smartnic.
[7] Y. Le, H. Chang, S. Mukherjee *et al.*, "Uno: uniflying host and smart nic offload for flexible packet processing," in *Proc. ACM SoCC*, 2017.
[8] Netronome. Wihte paper: Nfp-4000 theory of operation. [Online]. Available: https://www.netronome.com/media/documents/WP_NFP4000_TOO.pdf
[9] J. Hwang, K. Ramakrishnan, and T. Wood, "Netvm: high performance and flexible networking using virtualization on commodity platforms," in *Proc. USENIX NSDI*, 2014.
[10] R. Neugebauer, G. Antichi, J. F. Zazo *et al.*, "Understanding pcie performance for end host networking," in *Proc. ACM SIGCOMM*, 2018.
[11] D. Intel. Data plane development kit. [Online]. Available: http://dpdk.org
[12] M. Liu, S. Peter *et al.*, "E3: Energy-efficient microservices on smartnic-accelerated servers," in *Proc. USENIX ATC*, 2019.
[13] S. Angel, H. Ballani, T. Karagiannis *et al.*, "End-to-end performance isolation through virtual datacenters." in *Proc. USENIX OSDI*, 2014.
[14] Z. Meng, J. Bi, H. Wang *et al.*, "Coco: Compact and optimized consolidation of modularized service function chains in nfv," in *Proc. IEEE ICC*, 2018.
[15] W. Ma, O. Sandoval, J. Beltran *et al.*, "Traffic aware placement of interdependent nfv middleboxes," in *Proc. IEEE INFOCOM*, 2017.
[16] J. Zhang, W. Wu, and J. Lui, "On the theory of function placement and chaining for network function virtualization," in *Proc. ACM MobiHoc*, 2018.
[17] J. Zheng, Q. Ma, C. Tian *et al.*, "Orchestrating service chain deployment with plutus in next generation cellular core," in *Proc. IEEE/ACM IWQoS*, 2019.
[18] B. Krishnamurthy, S. Sen *et al.*, "Sketch-based change detection: methods, evaluation, and applications," in *Proc. ACM IMC*, 2003.
[19] CAIDA. (2016) The caida anonymized internet traces 2016 dataset. https://www.caida.org/data/passive/passive_2016_dataset.xml.
[20] S. Kumar, M. Tufail, S. Majee *et al.*, "Service function chaining use cases in data centers," *draft-ietf-sfc-dc-use-cases-06*, 2017.
[21] S. Han, K. Jang, K. Park *et al.*, "Packetshader: a gpu-accelerated software router," in *Proc. ACM SIGCOMM*, 2010.
[22] K. Zhang, B. He, J. Hu *et al.*, "G-net: Effective gpu sharing in nfv systems," in *Proc. USENIX NSDI*, 2018.
[23] D. Firestone, A. Putnam, S. Mundkur *et al.*, "Azure accelerated networking: Smartnics in the public cloud," in *Proc. USENIX NSDI*, 2018.
[24] X. Jin, X. Li, H. Zhang *et al.*, "Netchain: Scale-free sub-rtt coordination," in *Proc. USENIX NSDI*, 2018.
[25] T. Jepsen, D. Alvarez, N. Foster *et al.*, "Fast string searching on pisa," in *Proc. ACM SOSR*, 2019.
[26] G. P. Katsikas, T. Barbette *et al.*, "Metron: Nfv service chains at the true speed of the underlying hardware," in *Proc. USENIX NSDI*, 2018.
[27] S. Radhakrishnan, Y. Geng, V. Jeyakumar *et al.*, "Senic: Scalable nic for end-host rate limiting." in *Proc. USENIX NSDI*, 2014.