Hybrid Flow Table Installation: Optimizing Remote Placements of Flow Tables on Servers to Enhance PDP Switches for In-Network Computing

Yuhan Xue and Zuqing Zhu, *Senior Member, IEEE*

***Abstract*—Recently, the programmable data plane (PDP) switches have been considered as the key enablers for in-network computing. However, the limited memory resources in them for flow tables might restrict their performance. This work addresses this challenge by studying how to optimize the placements of flow tables in the external memory on multiple servers, and to access them with remote direct memory access (RDMA) for ensuring low latency. Specifically, we consider a data-center network (DCN) that uses PDP switches as top-of-rack (ToR) switches, and propose and optimize the hybrid flow table installation (hFT-INST) on each ToR switch. With hFT-INST, the switch can either store flow tables in its local memory or use RDMA to install and access them remotely in its rack servers. We first design the protocol and operation procedure of hFT-INST. Then, regarding the key problem of hFT-INST, *i.e.*, how to place the flow tables on the external memory on different servers, we take a few practical parameters into account, and formulate a mixed integer linear programming (MILP) model to tackle it. Next, the optimization in the MILP is transformed into a capacitated facility location problem (CFLP) with additional constraints. We further transform it into a k-median problem through pre-processing, and design a polynomial-time approximation algorithm to solve the problem. Extensive simulations confirm the performance of our proposed algorithm. We also prototype our design of the hFT-INST, and conduct experiments to demonstrate its feasibility.**

***Index Terms*—Software-defined networking (SDN), In-network computing, P4, Programmable data plane (PDP), Remote direct memory access (RDMA), Approximation algorithm.**

I. Introduction

NOWADAYS, the fast developments of cloud computing, big data analytics, and immersive multimedia applications have driven the Internet to go through revolutionary changes [1, 2]. For instance, the advances on software-defined networking (SDN) [3–5], network virtualization [6–10], and physical-layer technologies [11–15] have paved the road to realize network function virtualization (NFV), for flexible, cost-effective, and short time-to-market network service provisioning [16–18]. NFV instantiates virtual network functions (vNFs) and composes various network services with them. Recent studies have suggested that the cost-effectiveness and power-efficiency of NFV can be further improved with in-network computing [19, 20]. Specifically, in-network computing refers to the execution of vNFs within the network devices that are already used to forward traffic [20]. Because it terminates transactions as they traverse a network, in-network computing can significantly reduce the load on the network.

In-network computing can normally be realized with three types of devices, *i.e.*, the field programmable gate arrays (FPGAs), SmartNICs and programmable data plane (PDP) switches [20]. Among them, PDP switches [21, 22] are the key enablers. With domain specific languages (*e.g.*, P4 [23]), one can program the data plane logic of PDP switches, *i.e.*, defining packet header fields and packet processing pipelines, and specifying the match fields and actions of each flow table in the pipelines. Compared with traditional ones, PDP switches have similar costs and power consumption, but they offer improved programmability without sacrificing packet processing performance. Hence, they have the potential to realize high performance in-network computing [24–28].

Although PDP switches are promising for implementing in-network computing, there are still a few challenges to address. Among them, a major one is that due to the power consumption and cost of ternary content addressable memory (TCAM) and static random-access memory (SRAM), a PDP switch usually has very limited memory resources, which might not be sufficient to accommodate all the active flow tables for in-network computing [29]. For instance, the memory resources in a state-of-art PDP switch can only store a few thousands of flow tables at most [22, 30]. To address the difficulty caused by the limited memory capacity, people have considered to expand the memory in PDP switches with external memory and enable them to access the external memory with remote direct memory access (RDMA) [30]. Specifically, RDMA was developed to enable the direct access from the memory of one computer into that of another one without the intervention of either one's operating system [31]. Hence, it eliminates the overheads of memory copying and context switching, and thus allows high-throughput and low-latency data transfers.

Inspired by the idea in [30], we consider a practical and more complex scenario than the one in that work, *i.e.*, a data-center network (DCN) that uses PDP switches as top-of-rack (ToR) switches, and study how to optimize the placements of flow tables in the external memory on multiple servers to improve the performance of the PDP switches for in-network computing. Specifically, each ToR switch supports hybrid flow table installation (hFT-INST), *i.e.*, it can either store flow tables in its local memory, or use RDMA to install and access them in the external memory on the servers in its rack. Note that, RDMA can be supported with either the RDMA-based network adapters (RDMA-NICs) or normal

Y. Xue and Z. Zhu are with the School of Information Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, P. R. China (email: zqzhu@ieee.org).

Manuscript received on Jun. 4, 2020.

network adapters that enable soft RDMA over converged Ethernet (Soft-RoCE-NICs). RDMA-NICs are more expensive and thus less common in a network, while Soft-RoCE provides a software-based alternative solution to enable RDMA with normal network adapters. Meanwhile, in addition to RDMA, one can also use the remote produce call (RPC) [32] to realize remote memory access. However, both the existing studies in [33, 34] and our own experimental investigation suggested that compared with the generic RPC, RDMA can achieve more efficient and faster remote memory operations. Hence, we model a practical situation where the servers can be equipped with RDMA-NICs or Soft-RoCE-NICs, and consider the different data transfer latencies of the hFT-INSTs through RDMA-NICs and Soft-RoCE-NICs.

Based on this network environment, we first design the protocol for hFT-INST, and lay out the operation procedure to implement it. Note that, the RDMA-based design in [30] mainly considered the scheme that tries to perform the related tasks purely in the data plane. Although the data plane implementation was compact, it also suffers from the restrictions due to the limited computing and storing capability of the data plane. Hence, each PDP switch can only use the external memory on a single location, *i.e.*, organizing and indexing flow tables with different match fields on a single memory. However, as match fields can have various lengths and RDMA operations also consume bandwidth, the one-to-one scheme has to use complex flow table indexing/accessing methods and causes "hot-spots" of bandwidth usage in some extreme cases. Meanwhile, using a single server to provide the remote memory will make our proposal unreliable, because the entire system will stop functioning if the server breaks down.

Moreover, the design in [30] cannot be expanded to support multiple servers with simple modifications, because an effective algorithm is then needed to manage the mapping between flow tables and external memory locations, which was not addressed in [30]. This motivates us to study the related network modeling and algorithm design together with system implementation. In other words, as this work addresses both algorithm design and system implementation, its coverage is more comprehensive than that of [30]. We design the system to involve the control plane in the process, which has enhanced computing and storing capability to manage the mapping between flow tables and external memory locations. Therefore, the flow tables can be organized and indexed more efficiently and the bandwidth overheads of RDMA can be shared among multiple servers, which is similar to anycast [35].

It can be seen that in this new design, it is essential to determine how to place the flow tables on the external memory on different servers, which is the problem of hFT-INST considered in this work. More specifically, we take the widths of flow tables, computing and networking loads on servers, and RDMA latencies of RDMA-NICs and Soft-RoCE-NICs into account, and formulate an optimization model to tackle the problem of hFT-INST. To the best of our knowledge, such algorithm design has not been addressed in the existing studies on this topic. Next, we first transform the problem of hFT-INST into a capacitated facility location problem (CFLP) with additional constraints [36], further transform it into a k-median problem through pre-processing, and then design a polynomial-time approximation algorithm to solve it with performance guarantee. Finally, we prototype our design of hFT-INST with Soft-RoCE and BMv2 [37], and conduct experiments to demonstrate the feasibility of our proposal.

The rest of this paper is organized as follows. Section II presents the related work. We describe the network model and related definitions in Section III. The mixed integer linear programming (MILP) model for optimizing hFT-INST is formulated in Section IV. Then, Section V designs the approximation algorithm, and its simulation results are shown in Section VI. Next, we introduce the implementation of the prototype system and discuss the experimental demonstrations in Section VII. Finally, Section VIII summarizes the paper.

II. Related Work

In-network computing [20], also known as in-network computation [19], tries to extend the functionalities of network devices beyond traffic forwarding, to cover computing tasks in the application layer. The studies in [24–26] leveraged PDP switches to accomplish the computing tasks for key-value store. Sonchack *et al.* [38] introduced Turboflow, which was an in-network computing system for network telemetry. Specifically, they designed and implemented two types of flow record generators for network telemetry, based on the PDP switch using Tofino application-specific integrated circuit (ASIC) [22] and a prototype switch built with SmartNICs, respectively. Note that, SmartNICs usually consist of network processors whose architectures are similar to the run-to-completion model, and thus they can also be used for in-network computing [39, 40]. The authors of [39] utilized SmartNICs to extend RDMA primitives for offloading key-value store. IPipe [40] was an actor-based framework for offloading distributed applications onto SmartNICs.

Although in-network computing is promising, it has to consume a noticeable amount of memory in programmable network devices, and thus will bring new challenges since the memory resources in these network devices are usually very limited [22, 30]. For instance, in [28], the authors designed NetHCF to realize the filtering of spoofed IP traffic with PDP switches, while they had to take the shortage of memory resources in PDP switches into account and considered flow table aggregation. These new challenges actually motivate us to design and optimize hFT-INST.

Previously, there were also studies to address the shortage of memory resources in SDN/PDP switches. Bosshart *et al.* [21] investigated how to optimize the hardware design of SDN switches to relieve the memory fragmentation in them, and proposed two chip design techniques. Even though their results confirmed that the design could achieve efficient memory utilization over a wide range of network configurations, the proposals were still hardware-specific. The study in [41] proposed to replace part of hardware TCAM with the table resources realized by software. However, the high-speed search capability of hardware is also lost. The lookup time introduced by software-based table resources is much longer. The authors of [42] tried to delay the installation of flow tables

and expedite their evictions to adapt to the limited memory in SDN switches, but this might affect the packet processing performance. The shortage of memory resources in SDN switches could also be relieved by deploying aggregated default paths [43] or leveraging flow rule multiplexing [44]. However, these schemes might not be applicable to in-network computing, because the flow tables would be much more complicated than those used for packet forwarding. Previously, we also tackled the problem with table resource virtualization [45, 46], and tried to leverage the memory resources in directly-connected PDP switches as the supplement to address the limited memory in one of them. Nevertheless, the performance of the proposed scheme depended on the actual network topology.

Nowadays, the RDMA technology [31] has already been widely used in DCNs. However, most of its applications are still about realizing fast memory sharing among servers, *e.g.*, in FaRM [47]. The study in [30] proposed to expand the memory in PDP switches with external memory by leveraging RDMA. Specifically, the authors designed three remote memory primitives, *i.e.*, packet buffer primitive, lookup table primitive and state-store primitive, to show the feasibility of accessing remote memory from PDP switches. Nevertheless, they only considered to accomplish the remote table lookup purely in the data plane, which restricts the memory sharing in the one-to-one manner (*i.e.*, each PDP switch can only leverage the external memory on a single location). In this work, we also utilize external memory with RDMA to enhance PDP switches for in-network computing, but we improve the design by adding control plane operations to realize hFT-INST in the one-to-many way. Specifically, we take real system parameters into consideration, formulate a model to optimize hFT-INST, and design a polynomial-time approximation algorithm to solve it with performance guarantee. To the best of our knowledge, such algorithm design has not been addressed in the existing work on this topic.

III. Problem Description

In this section, we first explain the memory in PDP switches for storing flow tables, then introduce the RDMA scheme to access remote memory on servers, present our system design, and finally define the problem of hFT-INST.

A. Memory Resources in PDP Switches

Fig. 1 shows the the general architecture of a PDP switch that is based on a P4-enabled programmable ASIC [21, 29]. It processes packets with a pipeline model, where each packet needs to traverse a fixed sequence of flow tables. Each flow table stores its flow entries in TCAM, and uses SRAM for the counters and other states of processed packets. Meanwhile, the flow table also needs to occupy a small number of processing units in the ASIC, for executing operations while accessing the SRAM. The major benefit of such pipeline-based architecture is that the processing delay on each packet is almost the same, regardless of the number of flow tables that it needs to go through [21]. This is extremely useful for in-network computing, because it enables a PDP switch to handle complex operations with fixed latency. Meanwhile, we can see that the

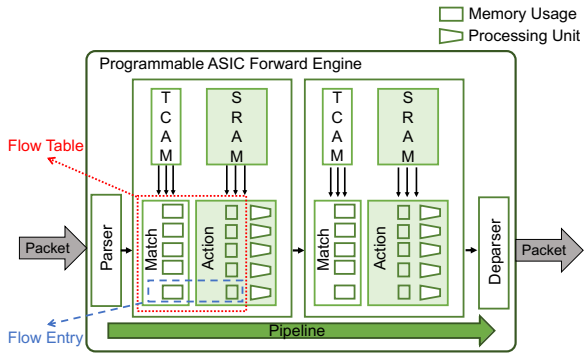


Fig. 1. General architecture of a P4-enabled PDP switch.

amount of memory resources provided by TCAM/SRAM will affect the PDP switch's performance, since the limited capacity will make it difficult to accommodate necessary flow tables.

B. Remote Access of Memory in Servers

We explain our proposal of hFT-INST in Fig. 2. Here, we consider a DCN environment, where PDP switches are used as the top-of-rack (ToR) switches, and assume that they can leverage external memory in its rack servers to store the flow tables for in-network computing, by utilizing RDMA. In a practical situation, the rack servers can be equipped with either RDMA-NICs or Soft-RoCE-NICs to share their memory with the ToR switch. Note that, the remote memory access will introduce extra bandwidth usages between the ToR switch and its rack servers. However, according to a recent measurement of the traffic characteristics in real DCNs [48], the traffic between a ToR switch and its rack servers is usually sparse. Therefore, we will consider the bandwidth usages between the ToR switch and its rack servers when solving the problem of hFT-INST, and try to find the servers whose bandwidth to/from the ToR switch is enough for the RDMA operations.

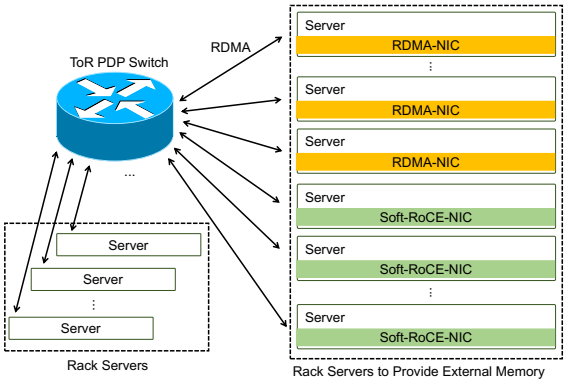


Fig. 2. Network model of hFT-INST.

C. System Design

To realize the hFT-INST in Fig. 2, we design the system architecture as shown in Fig. 3, which includes implementations in both the PDP switch and rack servers. On the switch side, the function modules are distributed in the control and data

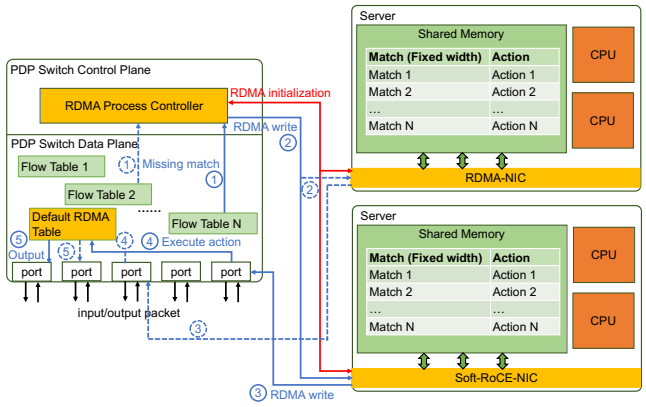


Fig. 3. Overall system architecture to realize hFT-INST.

planes. We insert a RDMA process controller in the control plane to 1) determine the scheme of hFT-INST according to network status, and 2) send *RDMA-Write* messages to servers to implement the scheme. The data plane still processes and forwards packets according to the pipelines built with flow tables. To overcome the shortage of TCAM/SRAM in the PDP switch, hFT-INST places and accesses flow entries remotely on rack servers. To achieve this, we program the PDP switch to enable low-latency remote table lookup with RDMA. When a packet comes in, if its flow entries are stored locally, it is processed as usual in normal PDP switches. Otherwise, the data plane of the PDP switch first asks the control plane to do a remote table lookup, which reads the external memory on servers with RDMA to get the flow entries for the packet, and then performs the actions in the obtained flow entries by leveraging the default RDMA table. Hence, instead of relying on the data plane to locate flow entries in external memory on servers, we design the control plane to accomplish the task.

Specifically, the overall procedure is explained in Fig. 3. When a packet encounters a table-miss, the PDP switch escalates it from the data plane to the RDMA process controller (**Step 1**). Then, the controller finds the remote location of its flow entry based on the match field of the flow table, and generates an *RDMA-Write* message to the corresponding server for remote table lookup (**Step 2**). Here, we include the packet in the *RDMA-Write* message to avoid caching it on the PDP switch. When the RDMA-NIC/Soft-RoCE-NIC on the target server receives the message, it locates the packet's flow entry in its local memory, encodes another *RDMA-Write* message to include both the packet and the flow entry, and sends the message back to the PDP switch (**Step 3**). On the server side, we store flow entries in a storage structure that is similar to the key-value store [26], where the key is the match field and the value is the corresponding action. Note that, the principle of RDMA ensures that the operations in **Steps 2** and **3** will not cause any overheads on the CPU on the server [31]. Finally, with the *RDMA-Write* message, the PDP switch executes the needed actions on the packet in the default RDMA table, and sends it out if necessary (**Steps 4** and **5**).

Note that, the system design in Fig. 3 assumes that the control plane of the PDP switch can communicate with servers using RDMA. This assumption is reasonable for the following two reasons. Firstly, the control plane should be able to talk to the servers, because such communications are needed even we do not consider remote memory access (*e.g.*, collecting working status of the servers). Secondly, the assumption does not necessarily mean that the control plane has to be equipped with a RDMA-NIC, since RDMA can also be realized on normal adapters by leveraging Soft-RoCE. Meanwhile, considering the fast development and wide usage of RDMA, it could be possible for us to equip a RDMA-NIC in a PDP switch just for the control plane in the future.

D. Problem Description of hFT-INST

It can be seen that the system design of hFT-INST described above sacrifices packet processing latency and bandwidth for larger memory capacities to store the flow tables for in-network computing. Since the rack servers can be equipped with different kinds of NICs and the bandwidth usages between them and the ToR switch can be various, how to allocate the flow entries to the servers (*i.e.*, the problem of hFT-INST) will be an interesting but necessary problem to investigate, especially when the PDP switch needs to satisfy different quality-of-service (QoS) requirements in in-network computing. Meanwhile, we should also pay attention to the length of the match field (*i.e.*, the key) in each flow entry, when organizing the entries in the servers. This is because for the remote table lookup with RDMA, it will be much easier and faster if the RDMA process controller searches in fixed-length keys. Hence, it would be beneficial to store flow entries whose keys have the same or similar lengths in a same server.

We define the problem of hFT-INST as follows. Given a set of flow tables that should be accommodated in external memory (C) and a set of rack servers (F), we need to map the tables in C to the servers in F, such that the overall cost of the mapping is minimized. Here, the RDMA latency, the bandwidth overhead, and the key length differences among the entries stored in each server all contribute to the cost.

IV. MILP Formulation for hFT-INST

In this section, we formulate an MILP model to optimize hFT-INST, based on the system design explained in Section III-C. Note that, RDMA-NICs and Soft-RoCE-NICs provide different RDMA latencies. Specifically, the latency from RDMA-NICs is shorter and usually does not change with packet length, while that from Soft-RoCE-NICs can be longer and increase with packet length. Hence, to be generic, we model the RDMA latency of an NIC as

$$T = \begin{cases} \alpha + \gamma \cdot l, & \text{enough bandwidth for RDMA,} \\ \beta + \gamma \cdot l, & \text{insufficient bandwidth for RDMA,} \end{cases} \quad (1)$$

where α and β are the fixed latencies for the cases where the available bandwidth between the PDP switch and a server is enough and not enough for RDMA operations, respectively, γ is the factor for calculating the dynamic latency based on packet length, and l is the packet length. The α, β and γ of RDMA-NICs and Soft-RoCE-NICs take different values.

Notations:

- C: the set of flow tables that need to be stored in external memory in servers[1], where we assume that each flow table can be indexed with an integer $j \in [1, |C|]$.
- F: the set of rack servers that can provide external memory for flow tables, where we assume that each server can be indexed with an integer $i \in [1, |F|]$.
- $c_{i,j}$: the cost when the entries in the *Table* j get stored in *Server* i, and the detailed formulation will be given later.
- l_j: the average length of the packets that match to flow entries in *Table* j.
- s_j: the average arrival rate of the packets that match to flow entries in *Table* j.
- B_i: the bandwidth left for RDMA on *Server* i.
- α_i: the fixed RDMA latency if the switch and *Server* i have enough bandwidth for RDMA operations.
- β_i: the fixed RDMA latency if the switch and *Server* i do not have enough bandwidth for RDMA operations[2].
- γ_i: the factor for calculating the dynamic RDMA latency on *Server* i, based on packet length.
- δ: the normalization factor for length difference between the key in a server and the match field in a table.
- cw_j: the length of the match field in *Table* j.
- fw_i: the key length of the remote memory in *Server* i.
- M: a large positive constant.
- $b_{\max}$: the maximum bandwidth usage of packets, *i.e.*, $b_{\max} = \max_{\forall j}(s_j \cdot l_j)$.
- $b_{\min}$: the minimum bandwidth usage of packets, *i.e.*, $b_{\min} = \min_{\forall j}(s_j \cdot l_j)$.

Variables:

- $x_{i,j}$: the boolean variable that equals 1 if the entries in *Table* j are stored on *Server* i, and 0 otherwise.
- y_i: the boolean variable that equals 1 if *Server* i stores flow entries, and 0 otherwise.
- f_i: the real variable that indicates the fixed RDMA latency on *Server* i.
- $f_{i,1}$, $f_{i,2}$, $p_{i,1}$ and $p_{i,2}$: the boolean auxiliary variables that are introduced for linearization.
- $q_{i,1}$ and $q_{i,2}$: the integer auxiliary variables that are introduced for linearization.

Objective:

The optimization tries to minimize the overall cost of mapping the flow tables to rack servers. Therefore, the optimization objective can be designed as

$$\text{Minimize} \quad \sum_{\forall i,j} c_{i,j} \cdot x_{i,j} + \sum_{\forall i} f_i \cdot y_i, \tag{2}$$

where the first term includes the costs caused by the dynamic RDMA latencies and the key length difference, while the second term denotes the total fixed RDMA latency. Hence, we formulate the cost $c_{i,j}$ as follows.

$$c_{i,j} = \gamma_i \cdot l_j + \delta \cdot (fw_i - cw_j). \tag{3}$$

Note that, the second term in Eq. (2) is nonlinear, and in the following, we will show the method to linearize it.

Constraints:

$$\sum_{\forall i} x_{i,j} = 1, \quad \forall j. \tag{4}$$

Eq. (4) ensures that each flow table is allocated to one and only one server, for storing its entries remotely.

$$y_i - x_{i,j} \geq 0, \quad \forall i, j. \tag{5}$$

Eq. (5) ensures that the value of y_i is set correctly.

$$f_i = \begin{cases} \alpha_i, & B_i \geq \sum_{\forall j} s_j \cdot l_j \cdot x_{i,j}, \\ \beta_i, & B_i < \sum_{\forall j} s_j \cdot l_j \cdot x_{i,j}. \end{cases} \tag{6}$$

Eq. (6) determines the value of the fixed RDMA latency f_i on *Server* i according to the bandwidth usage[3]. However, this is a nonlinear constraint and needs to be linearized.

$$f_{i,1} + f_{i,2} = 1, \tag{7}$$

$$\alpha_i \cdot f_{i,1} + \beta_i \cdot f_{i,2} = f_i, \tag{8}$$

$$B_i \geq f_{i,1} \cdot \sum_{\forall j} s_j \cdot l_j \cdot x_{i,j}, \quad \forall i, \tag{9}$$

$$B_i < f_{i,2} \cdot \sum_{\forall j} s_j \cdot l_j \cdot x_{i,j} + M \cdot fi_1, \quad \forall i. \tag{10}$$

Eqs. (7)-(10) leverage variables $f_{i,1}$ and $f_{i,2}$ to linearize Eq. (6), while Eqs. (9)-(10) need to be further linearized.

$$B_i \geq q_{i,1}, \quad \forall i, \tag{11}$$

$$q_{i,1} \leq \sum_{\forall j} s_j \cdot l_j \cdot x_{i,j}, \quad \forall i, \tag{12}$$

$$q_{i,1} \geq \left(\sum_{\forall j} s_j \cdot l_j \cdot x_{i,j} \right) - b_{\max} \cdot (1 - f_{i,1}), \quad \forall i, \tag{13}$$

$$b_{\min} \cdot f_{i,1} \leq q_{i,1} \leq b_{\max} \cdot f_{i,1}, \quad \forall i. \tag{14}$$

We use Eqs. (11)-(14) to linearize Eq. (9) with variable $q_{i,1}$.

$$B_i < q_{i,2} + M \cdot f_{i,1}, \quad \forall i, \tag{15}$$

$$q_{i,2} \leq \sum_{\forall j} s_j \cdot l_j \cdot x_{i,j}, \quad \forall i, \tag{16}$$

$$q_{i,2} \geq \left(\sum_{\forall j} s_j \cdot l_j \cdot x_{i,j} \right) - b_{\max} \cdot (1 - f_{i,2}), \quad \forall i, \tag{17}$$

$$b_{\min} \cdot f_{i,2} \leq q_{i,2} \leq b_{\max} \cdot f_{i,2}, \quad \forall i. \tag{18}$$

Similarly, we use Eqs. (15)-(18) to linearize Eq. (10) with variable $q_{i,2}$. Then, we can linearize the objective in Eq. (2) with variables $f_{i,1}$ and $f_{i,2}$ as

$$\text{Minimize} \ \sum_{\forall i,j} c_{i,j} \cdot x_{i,j} + \sum_{\forall i} \left(\alpha_i \cdot f_{i,1} \cdot y_i + \beta_i \cdot f_{i,2} \cdot y_i \right), \tag{19}$$

[1]Note that, flow tables should only be placed on external memory in servers when the local memory on a PDP switch is used up. This is because RDMA brings extra memory access latency and thus will affect the packet processing performance of the PDP switch. Hence, C just contains all the flow tables that cannot be stored locally in a PDP switch, and can be easily obtained.

[2]We normally have $\beta_i \gg \alpha_i$, and thus the packet processing in a PDP switch will be severely affected if the bandwidth left for RDMA operations is insufficient.

[3]Here, we assume that the average length and average arrival rate of the packets, which match to the flow entries in each flow table in C, should be known *a prior*. This assumption is valid, because for in-network computing, a flow table normally contains the flow entries for the same application, while the traffic statistics of applications in DCNs can be estimated [48, 49].

which can be further linearized as follows.

$$\text{Minimize} \quad \sum_{\forall i,j} c_{i,j} \cdot x_{i,j} + \sum_{\forall i} \left(\alpha_i \cdot p_{i,1} + \beta_i \cdot p_{i,2}\right), \tag{20}$$

which represents the final formulation of the objective. Here, $p_{i,1}$ and $p_{i,2}$ are the auxiliary integer variables introduced to assist the linearization of the objective, and the following constraints should be added for them.

$$\begin{cases} p_{i,1} \leq f_{i,1}, \\ p_{i,2} \leq f_{i,2}, \end{cases} \quad \forall i, \tag{21}$$

$$\begin{cases} p_{i,1} \leq y_i, \\ p_{i,2} \leq y_i, \end{cases} \quad \forall i, \tag{22}$$

$$\begin{cases} p_{i,1} \geq f_{i,1} + y_i - 1, \\ p_{i,2} \geq f_{i,2} + y_i - 1, \end{cases} \quad \forall i. \tag{23}$$

By treating the flow tables in C as demand points and the servers in F as potential facility sites, we can easily verify that the optimization described by the aforementioned MILP can be transformed into a capacitated facility location problem (CFLP) with additional constraints, which is $\mathcal{NP}$-hard according to [50]. Therefore, in the next section, we will leverage some existing techniques in [50, 51] to design a polynomial-time approximation algorithm for it.

V. Approximation Algorithm

In this section, we design a polynomial-time approximation algorithm to solve the optimization in Section IV with performance guarantee, and thus it can be implemented in the RDMA process controller in Fig. 3. Specifically, as the optimization is an instance of CFLP with additional constraints, we first design a preprocessing procedure to transform it into a k-median problem, and then we leverage the local search approach in [51] to design the approximation algorithm for it.

A. Preprocessing

By looking into the MILP model in Section IV, we can see that compared with the standard CFLP, it includes a few additional constraints, which are mainly related to bandwidth usages. More specifically, because the relation between fixed RDMA latency and bandwidth usage is modeled as a piecewise function in Eq. (6), we introduce several variables and constraints to linearize it. Hence, we design the preprocessing to simplify the constraints, reduce the scale of the problem, and transform it into a standard k-median problem. The main idea of the preprocessing is to aggregate tables and filter out infeasible servers, such that all the selected servers should have enough bandwidth for the RDMA operations of hFT-INST.

Algorithm 1 explains the procedure of the preprocessing. *Lines* 1-2 is for the initialization. Note that, the mapping of flow tables to servers is not conducted in the preprocessing, but it is just for organizing the flow tables in a reasonable set of grouped tables and removing the servers whose bandwidth left for RDMA is too small. Hence, *Line* 2 selects the server whose bandwidth is median among all the servers, and records its bandwidth in $\bar{B}$ as an empirical threshold. Then, the for-loop covering *Lines* 3-12 organizes the flow tables in C in grouped tables. Specifically, the flow tables in each grouped table have the same length of match field, and the total bandwidth usage of the packets that match to them does not exceed $\bar{B}$. We insert all the obtained grouped tables in set C' (*Line* 13), record the number of grouped tables in k (*Line* 14), and find the grouped table whose total bandwidth usage from packets is the maximum and store the bandwidth in $\hat{b}$ (*Line* 15). Finally, we use the for-loop that covers *Lines* 16-20 to filter out the servers whose bandwidth left for RDMA is less than $\hat{b}$ and insert the remaining ones in the set of selected servers F'. The time complexity of *Algorithm* 1 is $O(|C| + |F|)$.

Algorithm 1: Preprocessing

Input: the set of servers F, and set of flow tables C.
Output: the set of selected servers F', set of grouped tables C', and number of grouped tables k.

1 $F' = F$, $C' = C$, $p = 0$, $gb = 0$, $cw = 0$;
2 sort servers in F based on bandwidth left for RDMA, select the median one, and store its bandwidth in $\bar{B}$;
3 **for** *each Table* j *in* C *in ascending order of* cw_j **do**
4 **if** $(cw_j = cw)$ *AND* $(gb + s_j \cdot l_j \leq \bar{B})$ **then**
5 add *Table* j into *Table Group* G_p;
6 $gb = gb + s_j \cdot l_j$;
7 **else**
8 $p = p + 1$;
9 add *Table* j into *Table Group* G_p;
10 $gb = s_j \cdot l_j$, $cw = cw_j$;
11 **end**
12 **end**
13 insert all the grouped tables in C';
14 $k = p$;
15 store the maximum bandwidth of all grouped tables in $\hat{b}$;
16 **for** *each Server* i *in* F **do**
17 **if** $B_i \geq \hat{b}$ **then**
18 add *Server* i in the set of selected servers F';
19 **end**
20 **end**
21 **return**(F', C', k);

B. Approximation Algorithm to Solve k-Median Problem

Since the preprocessing makes sure that all the selected servers in F' can provide sufficient bandwidth for RDMA, the second term in the optimization objective in Eq. (2) becomes a constant multiplied by k. Therefore, the optimization in Section IV gets transformed into the following problem.

$$\text{Minimize} \quad \sum_{\forall i \in F',\ \forall j \in C'} c_{i,j} \cdot x_{i,j}. \tag{24}$$

Constraints:

$$\sum_{\forall i \in F'} x_{i,j} = 1, \quad \forall j \in C', \tag{25}$$

$$y_i - x_{i,j} \geq 0, \quad \forall i \in F',\ \forall j \in C', \tag{26}$$

$$\sum_{i \in F'} y_i \leq k. \tag{27}$$

Algorithm 2: Local Search Algorithm

Input: the set of selected servers F', set of grouped tables C', and number of grouped tables k.
Output: $\mathcal{M}$: the mapping of $C' \rightarrow F'$.

1 $F'' = \emptyset$;
2 **while** $|F''| < k$ **do**
3 move a server from F' to F'', such that mapping grouped tables in C' to servers in F'' greedily leads to the smallest cost with Eq. (24);
4 record the mapping and its cost in $\mathcal{M}$ and $\mathcal{C}$;
5 **end**
6 swap n servers in F'' with those in F', and store all the possible new server sets in $\mathcal{F}$;
7 **for** *each* $F'' \in \mathcal{F}$ **do**
8 map grouped tables in C' to servers in F'' greedily to minimize the cost in Eq. (24);
9 record the new cost in $\mathcal{C}'$;
10 **if** $\mathcal{C}' < \mathcal{C}$ **then**
11 update $\mathcal{M}$ with the new mapping for F'';
12 $\mathcal{C} = \mathcal{C}'$;
13 **end**
14 **end**
15 **return**($\mathcal{M}$);

We can see that the optimization described with Eqs. (24)-(27) is just a standard k-median problem [51]. Note that, there have already been a few approximation algorithms to solve this problem [51–53], and the local search approach in [51] can provides the best approximation ratio. Therefore, we leverage it to design *Algorithm* 2 and solve the optimization of Eqs. (24)-(27) with performance guarantee. *Line* 1 is for the initialization. Then, we use a while-loop to move one server from F' to F'' in each iteration with a forward greedy process, *i.e.*, the mapping cost is minimized after each operation, until there are k servers in F'' (*Lines* 2-5). In *Line* 6, we try to swap n servers in F'' with those in F', obtain all the possible new server sets, and store them in set $\mathcal{F}$. Finally, the for-loop covering 7-14 checks all the new server sets in $\mathcal{F}$ to find the best mapping scheme, which provides the smallest mapping cost. The time complexity of *Algorithm* 2 is $O((|C'|+|F'|)^n)$.

C. Approximation Ratio

We assume that there are sufficient servers with enough bandwidth for RDMA to accommodate the flow tables. This is reasonable because if the servers are insufficient, the packet processing in the PDP switch will be severely affected, and we should not offload flow tables in this case. Hence, by denoting the optimal solution from the MILP as $\mathcal{C}_{\text{MILP}}$, we have

$$\mathcal{C}_{\text{MILP}} \geq \sum_{j \in C} \min_{i \in F}(c_{i,j}) = \mathcal{C}_1. \tag{28}$$

Meanwhile, we denote the optimal solution obtained by solving the k-median problem of Eqs. (24)-(27) exactly as $\mathcal{C}_{\text{kMP}}$,

$$\mathcal{C}_{\text{kMP}} \leq \sum_{j \in C} \max_{i \in F}(c_{i,j}) + k \cdot \max_{i \in F}(\alpha_i) = \mathcal{C}_2. \tag{29}$$

It can be seen that $\mathcal{C}_1$ and $\mathcal{C}_2$ are constants for each instance of the hFT-INST problem, and the ratio of $\mathcal{C}_{\text{kMP}}$ to $\mathcal{C}_{\text{MILP}}$ satisfies

$$\frac{\mathcal{C}_{\text{MILP}}}{\mathcal{C}_{\text{kMP}}} \geq \frac{\mathcal{C}_1}{\mathcal{C}_2}. \tag{30}$$

Then, we define the solution from *Algorithm* 2 as $\mathcal{C}_{\text{APP}}$, and according to [51], we have

$$\frac{\mathcal{C}_{\text{APP}}}{\mathcal{C}_{\text{kMP}}} \leq 3 + \frac{2}{n}. \tag{31}$$

Finally, by combining Eqs. (28)-(32), we get the upper-bound of the approximation ratio as

$$\frac{\mathcal{C}_{\text{APP}}}{\mathcal{C}_{\text{MILP}}} \leq \frac{\mathcal{C}_2}{\mathcal{C}_1} \cdot \frac{\mathcal{C}_{\text{APP}}}{\mathcal{C}_{\text{kMP}}} \leq \frac{\mathcal{C}_2}{\mathcal{C}_1} \cdot \left(3 + \frac{2}{n}\right). \tag{32}$$

VI. Numerical Simulations

In this section, we evaluate the performance of our approximation algorithm (*Algorithms* 1 and 2) with extensive numerical simulations. Specifically, we consider both small- and large-scale problems of hFT-INST. Due to the MILP's complexity, we only compare it with the approximation algorithm in the small-scale scenario. In the large-scale scenario, we consider more flow tables and rack servers to further study the performance of the approximation algorithm. Note that, other than the MILP and approximation algorithm, we do not consider heuristics in the simulations. This is because for an optimization problem, a heuristic can only obtain feasible solutions but can never guarantee a fixed approximation ratio. In other words, it is theoretically impossible to have the heuristics that could better approximate the optimal solutions than an approximation algorithm in all the scenarios. The simulations obtain each data point by averaging the results from 20 independent runs, to ensure sufficient statistical accuracy.

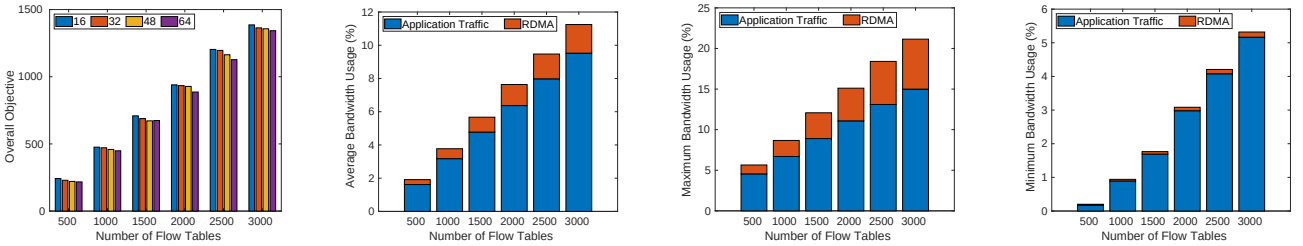
As we will explain in Section VII-A, the system implementation uses the software-based BMv2 [37] to emulate a P4-enabled PDP switch. Therefore, the settings of α, β and γ in Eq. (1) should follow the assumption that the PDP switch is a software-based one using BMv2. Then, based on the experimental results about RDMA-NICs in [30], the simulations choose $\alpha = 0.4$ ms, $\beta = 1000$ ms, and $\gamma = 0$ ms/byte for RDMA-NICs. On the other hand, we measure the RDMA latency on a Soft-RoCE-NIC in the experiments that will be described in Section VII-C. According to the experimental results, we set $\alpha = 0.5655$ ms, $\beta = 1000$ ms, and $\gamma = 4.5 \times 10^{-4}$ ms/byte for Soft-RoCE-NICs.

A. Small-Scale Simulations

In this scenario, we choose the number of rack servers from $\{16, 32\}$. The key length of the remote memory on each server is randomly selected from $\{32, 48, 64, 128\}$ bits with probabilities of $\{0.15, 0.1, 0.6, 0.15\}$, respectively. The bandwidth capacity of the connection between each server to the PDP switch is chosen from $\{1, 10, 40\}$ Gbps with probabilities of $\{0.75, 0.1875, 0.0625\}$. Also, we assume that servers with 10 and 40 Gbps connection speeds are equipped with RDMA-NICs, while those with 1 Gbps connection speed have Soft-RoCE-NICs. The number of flow tables in C is fixed as 500 or 1000, and the lengths of their match fields are

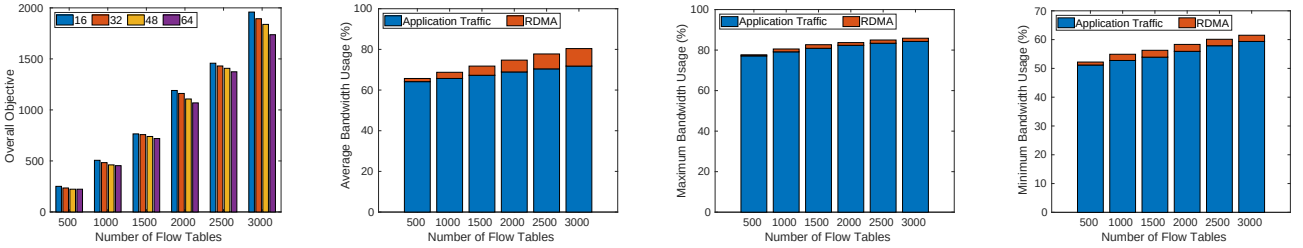
TABLE I
PERFORMANCE COMPARISONS BETWEEN MILP AND APPROXIMATION ALGORITHM

		MILP				Approximation Algorithm			
Servers	Flow Tables	First Term	Second Term	Overall	Running	First Term	Second Term	Overall	Running
$\|F\|$	$\|C\|$	in Eq. (2)	in Eq. (2)	Objective	Time (s)	in Eq. (2)	in Eq. (2)	Objective	Time (s)
16	500	231.25	1.20	232.45	0.800	249.35	1.20	250.55	0.003
16	1000	471.75	1.20	472.95	1.300	495.25	1.60	496.85	0.013
32	500	217.50	2.40	219.90	13.600	233.50	1.60	235.10	0.011
32	1000	455.00	1.60	456.60	366.400	471.25	1.60	472.85	0.100



(a) Overall cost (b) Average bandwidth usage (c) Maximum bandwidth usage (d) Minimum bandwidth usage

Fig. 4. Large-scale simulation results (First scenario: light-loaded DCN).



(a) Overall cost (b) Average bandwidth usage (c) Maximum bandwidth usage (d) Minimum bandwidth usage

Fig. 5. Large-scale simulation results (Second scenario: heavy-loaded DCN).

randomly selected within $\{16, 32, 48, 64, 96, 128\}$ bits[4]. The length of the packets that match to the flow tables is chosen within $\{64, 128, 256, 512, 1024, 1500\}$ bytes with probabilities of $\{0.2, 0.2, 0.2, 0.2, 0.1, 0.1\}$, respectively. The packet rate is randomly selected from $[1, 10]$ kilo-packets per second (kpps).

Table I summarizes the simulation results to compare the performance of the MILP and approximation algorithm. We can see that the objectives from the approximation algorithm are always very close to those from the MILP, and the maximum relative gap is only 7.78%. Meanwhile, the running time of the approximation algorithm can be three magnitudes shorter than that of the MILP. Hence, the results verify the time-efficiency and effectiveness of our proposed algorithm, and confirm that it can be implemented in the RDMA process controller for dynamic operations.

B. Large-Scale Simulations

In this scenario, we increase the numbers of rack servers and flow tables. Specifically, the number of servers is chosen from $\{16, 32, 48, 64\}$, the number of flow tables is selected within $[500, 3000]$, and the packet rate is selected from $[1, 20]$ kpps. While the remaining parameters are unchanged. We consider two scenarios in the simulations. The first scenario emulates a light-loaded DCN, where the bandwidth between the ToR switch and rack servers is abundant. The second one considers a relatively heavy-loaded scheme, where $[0.5, 0.75]$ of the bandwidth between the ToR switch and rack servers has already been used for carrying application traffic. We only simulate the approximation algorithm, and compare the performance of hFT-INST in terms of the following metrics:

- *Overall Objective*: the cost of hFT-INST with Eq. (2).
- *Metrics about Bandwidth Usage*: the average, maximum and minimum bandwidth usages on each server over the time of each simulation.

Fig. 4 shows the simulation results of the first scenario. In Fig. 4(a), we can see that when the number of flow tables is fixed, the overall objective does not change much if the number of rack servers varies. Specifically, the overall objective only decreases slightly with the number of servers. This is because when the DCN is empty, the hFT-INST schemes with fewer or more servers do not make much difference in terms of the overall cost since the bandwidth resources are abundant.

[4]In a practical DCN, the distribution of flow tables can change when different types of in-network computing tasks are considered. Hence, uniform distribution is the most statistically-suitable assumption to use. Meanwhile, as our proposals do not depend on the distribution of flow tables, this assumption will not restrict the generality of our numerical evaluations.

However, as when the number of servers increases, the number of servers with RDMA-NICs also increases, and this helps to reduce the overall cost slightly. Then, we fix the number of servers as 64 and obtain the bandwidth usages in Figs. 4(b)-4(d). The average bandwidth in Fig. 4(b) shows that the extra bandwidth consumed by RDMA operations is much smaller than that for normal packet processing of application traffic. The similar trend can be observed in Figs. 4(c) and 4(d).

For the second scenario, the results in Fig. 5(a) show that the overall costs are generally higher than their counterparts in Fig. 4(a). This is because the fact that the available bandwidth between the ToR switch and rack servers is smaller makes *Algorithm* 1 generate more grouped tables, which will in turn let the hFT-INST distribute the grouped tables on more servers. When more servers are used, the probability of placing flow tables on the servers with Soft-RoCE-NICs increases, and this leads to higher overall costs. Meanwhile, we also observe that the overall cost decreases more noticeably with the number of servers, and this is also due to the limited available bandwidth. Other than these, Figs. 5(b)-5(d) still indicate that the extra bandwidth consumed by RDMA operations is much smaller than that for normal packet processing of application traffic.

VII. System Implementation and Experimental Demonstrations

In this section, we provide the detailed design of our hFT-INST system and discuss its experimental demonstrations.

A. System Implementation

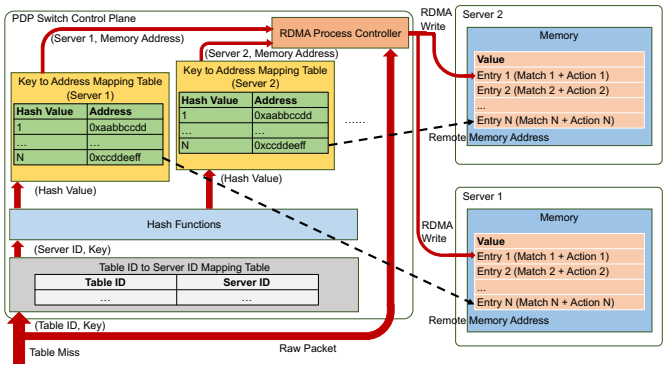
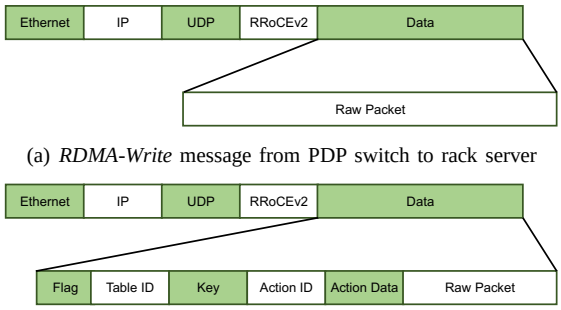


Fig. 6. Control plane design of PDP switch for hFT-INST.

Fig. 6 shows our design of the control plane in the PDP switch for hFT-INST. In our system, the control plane in the PDP switch is in charge of indexing/accessing flow tables stored in the external memory on multiple rack servers. As different servers may use the same memory address range, we create a hash table to store the memory addresses of each server that provides external memory for hFT-INST. Specifically, the hash table is obtained by applying the preset hash functions on the tuples of <*Server ID*, *Key*>, where *Key* is the match field in a flow table. We program the control plane with Python, while the RDMA process controller is written in the C language and works as a RDMA client to use the interfaces provided by Soft-RoCE. The PDP switch



(a) *RDMA-Write* message from PDP switch to rack server

(b) *RDMA-Write* message from rack server to PDP switch

Fig. 7. Packet formats designed for RDMA messages.

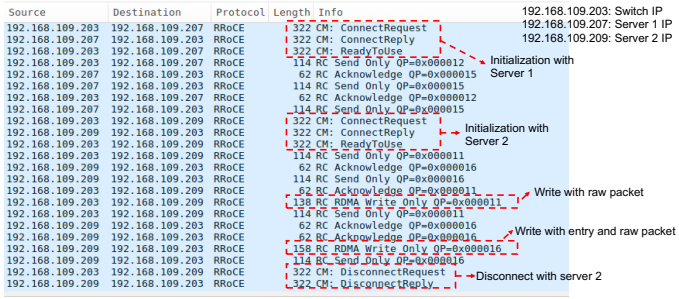
is emulated with the software-based BMv2 [37], and we program its data plane with the P4 language. The control and data planes of the PDP switch talk with each other using the interface of remote procedure call (RPC). On each rack server, we program a RDMA server on it, also in the C language, to communicate with the RDMA process controller for accomplishing RDMA operations.

The operation of our system involves two phases, *i.e.*, the initialization and operational phases. The initialization implements a few key settings of hFT-INST, *e.g.*, the key length and addresses of the remote memory on each related rack server, and the hash functions for memory indexing. Then, the system enters its operational phase to let the PDP switch process packets according to local/remote flow tables. As shown in Fig. 6, when a packet encounters a table-miss, the data plane will first generate a key based on the match field of the flow table, and then send the packet to the control plane along with the table ID and key. The control plane first determines the ID of the server that stores the flow table remotely based on the table ID, and then applies the corresponding hash function to the tuple of <*Server ID*, *Key*> to obtain the memory address that points to the required flow entry. Next, the remote memory address and the raw packet are sent to the RDMA process controller, which constructs an *RDMA-Write* message with the format in Fig. 7(a).

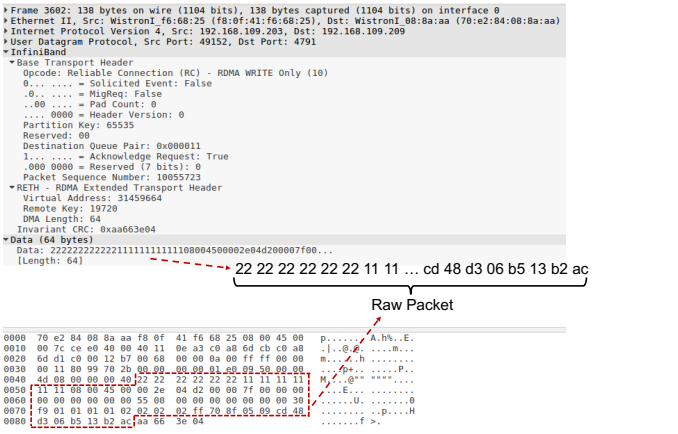
When the target server receives the *RDMA-Write* message, it performs the remote table lookup to determine the action for the packet, generates another *RDMA-Write* message with the format in Fig. 7(b), and sends it back to the PDP switch. With the content in the *RDMA-Write* message, the PDP switch executes the needed action on the packet in the default RDMA table. After the packet having been processed in the PDP switch, the data plane sends the table ID, key, action ID, and action data about it to the control plane, which can use the information to generate a flow entry and insert it locally in the data plane. Then, the processing of the subsequent packets can be completed locally on the PDP switch. The caching of flow entries is optional in our design, and should only be invoked when there are sufficient memory resources on the PDP switch.

B. Function Verification

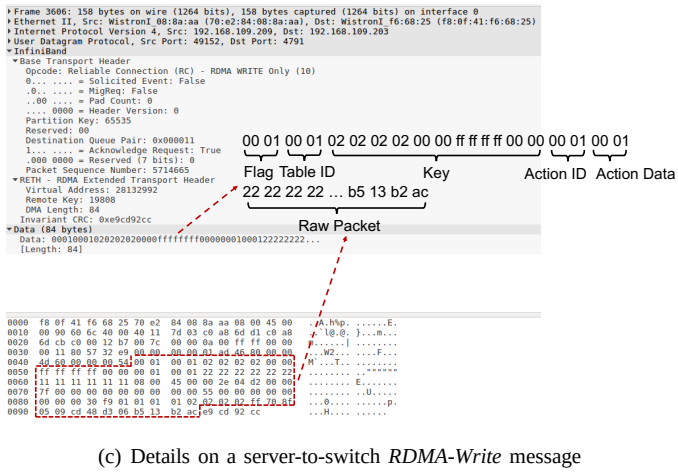
To verify the functionality of our hFT-INST system, we capture RDMA messages with Wireshark on the PDP switch. Fig.



(a) RDMA messages received on PDP switch



(b) Details on a switch-to-server *RDMA-Write* message



(c) Details on a server-to-switch *RDMA-Write* message

Fig. 8. Wireshark captures on PDP switch.

8(a) shows the operation procedure, which includes the RDMA messages for the initializations on two rack servers, and the switch-to-server and server-to-switch *RDMA-Write* messages to accomplish a remote table lookup. The details about the switch-to-server *RDMA-Write* message are illustrated in Fig. 8(b), which only includes the packet that encounters a table-miss. Shortly after receiving the *RDMA-Write* message, the server returns the server-to-switch *RDMA-Write* message in Fig. 8(c) to the PDP switch. This time, the message includes both the packet and the information about its matched entry.

C. Latency Measurements

With our implementation, we first compare the packet processing latency with and without remote table lookup. In Fig. 9, we can see that for each packet, the remote table lookup with RDMA can cause an extra latency of around 200 μs. Note that, we implement the hFT-INST with a software-based scheme, *i.e.*, using BMv2 to emulate the PDP switch and realizing RDMA operations on a Soft-RoCE-NIC. Hence, we expect that the RDMA latency can be significantly reduced if the entire hFT-INST system is implemented in a hardware-based environment (*e.g.*, a PDP switch with Tofino ASIC [22] and RDMA-NICs). Meanwhile, we observe that the packet size does affect the packet processing latency when there is remote table lookup. The linear regression in Fig. 10 suggests that the latency T changes with the packet length l as $T = \alpha + \gamma \cdot l$, where $\alpha = 0.5655$ ms and $\gamma = 4.5 \times 10^{-4}$ ms/byte.

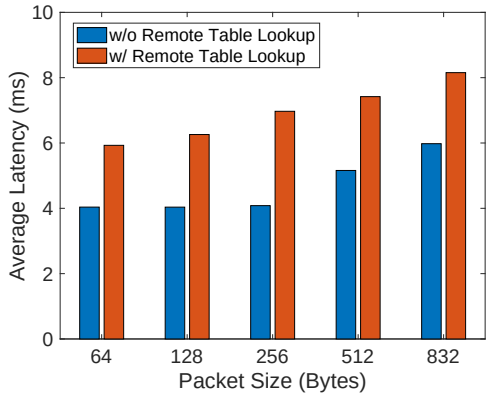


Fig. 9. Comparison of packet processing latencies for with and without remote table lookup.

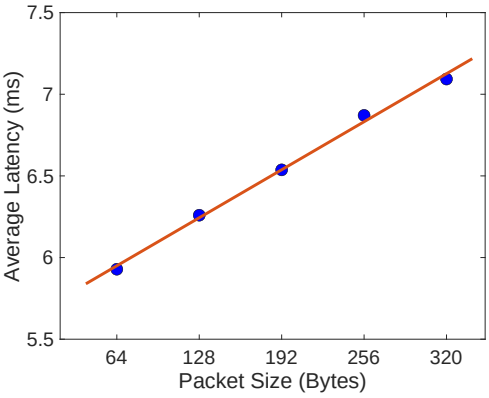


Fig. 10. Relation between packet length and processing latency (with remote table lookup).

D. HD Video Streaming

To further verify the practicalness of our proposal, we utilize high-definition (HD) video streaming as the application to test whether the remote table lookup with RDMA can affect its QoS. Specifically, we stream an HD video with an average throughput of 2 Mbps and the resolution of 1280×720 through the PDP switch, and let the switch store the flow table to process its traffic on a remote server. We measure the luminance component's peak signal-to-noise ratio (Y-PSNR)

of the video's playback on the receiver end, to quantify the QoS of the video streaming. The experimental results are shown in Fig. 11, which indicate that the Y-PSNR always stays at relatively high values, *i.e.*, the RDMA latency does not cause noticeable impacts on the HD video streaming.

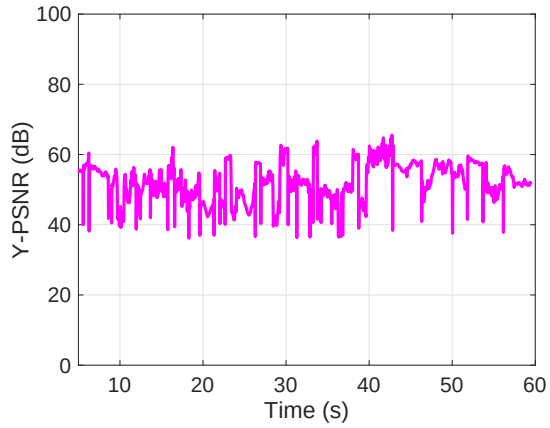


Fig. 11. Y-PSNR of video streaming (with remote table lookup).

VIII. Conclusion

This paper considered a DCN that uses PDP switches as ToR switches, and studied how to leverage the external memory in rack servers to improve the performance of PDP switches for in-network computing. Specifically, each ToR switch facilitated hFT-INST, which means that it can either store flow tables in its local memory or use RDMA to install and access them remotely in the rack servers. We designed the protocol for hFT-INST, and laid out the operation procedure to implement it. Regarding the key problem of hFT-INST, *i.e.*, how to place the flow tables on the external memory on different servers, we took the widths of flow tables, computing and networking loads on servers, and RDMA latencies of RDMA-NICs and Soft-RoCE-NICs into account, and formulated an MILP model to tackle it. The optimization in the MILP was first transformed into a CFLP with additional constraints, and then we transformed it into a k-median problem through pre-processing, and designed a polynomial-time approximation algorithm to solve it. Extensive simulations verified the performance of our proposal. Finally, we prototyped our design of the hFT-INST with Soft-RoCE and BMv2, and conducted experiments to demonstrate its feasibility.

Acknowledgments

This work was supported in part by the NSFC projects 61871357, 61771445 and 61701472, Zhejiang Lab Research Fund 2019LE0AB01, CAS Key Project (QYZDY-SSW-JSC003), SPR Program of CAS (XDC02070300), and Fundamental Funds for Central Universities (WK3500000006).

References

[1] Cisco Global Cloud Index: Forecast and Methodology, 2016-2021. [Online]. Available: https://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html.

[2] P. Lu *et al.*, "Highly efficient data migration and backup for Big Data applications in elastic optical inter-data-center networks," *IEEE Netw.*, vol. 29, pp. 36–42, Sept./Oct. 2015.

[3] N. Xue *et al.*, "Demonstration of OpenFlow-controlled network orchestration for adaptive SVC video manycast," *IEEE Trans. Multimedia*, vol. 17, pp. 1617–1629, Sept. 2015.

[4] Z. Zhu *et al.*, "Demonstration of cooperative resource allocation in an OpenFlow-controlled multidomain and multinational SD-EON testbed," *J. Lightw. Technol.*, vol. 33, pp. 1508–1514, Apr. 2015.

[5] S. Li *et al.*, "Protocol oblivious forwarding (POF): Software-defined networking with enhanced programmability," *IEEE Netw.*, vol. 31, pp. 12–20, Mar./Apr. 2017.

[6] L. Gong, Y. Wen, Z. Zhu, and T. Lee, "Toward profit-seeking virtual network embedding algorithm via global resource capacity," in *Proc. of INFOCOM 2014*, pp. 1–9, Apr. 2014.

[7] L. Gong and Z. Zhu, "Virtual optical network embedding (VONE) over elastic optical networks," *J. Lightw. Technol.*, vol. 32, pp. 450–460, Feb. 2014.

[8] H. Jiang, Y. Wang, L. Gong, and Z. Zhu, "Availability-aware survivable virtual network embedding (A-SVNE) in optical datacenter networks," *J. Opt. Commun. Netw.*, vol. 7, pp. 1160–1171, Dec. 2015.

[9] J. Liu *et al.*, "On dynamic service function chain deployment and readjustment," *IEEE Trans. Netw. Serv. Manag.*, vol. 14, pp. 543–553, Sept. 2017.

[10] L. Gong, H. Jiang, Y. Wang, and Z. Zhu, "Novel location-constrained virtual network embedding (LC-VNE) algorithms towards integrated node and link mapping," *IEEE/ACM Trans. Netw.*, vol. 24, pp. 3648–3661, Dec. 2016.

[11] Z. Zhu *et al.*, "Jitter and amplitude noise accumulations in cascaded all-optical regenerators," *J. Lightw. Technol.*, vol. 26, pp. 1640–1652, Jun. 2008.

[12] Z. Zhu, W. Lu, L. Zhang, and N. Ansari, "Dynamic service provisioning in elastic optical networks with hybrid single-/multi-path routing," *J. Lightw. Technol.*, vol. 31, pp. 15–22, Jan. 2013.

[13] L. Gong *et al.*, "Efficient resource allocation for all-optical multicasting over spectrum-sliced elastic optical networks," *J. Opt. Commun. Netw.*, vol. 5, pp. 836–847, Aug. 2013.

[14] Y. Yin *et al.*, "Spectral and spatial 2D fragmentation-aware routing and spectrum assignment algorithms in elastic optical networks," *J. Opt. Commun. Netw.*, vol. 5, pp. A100–A106, Oct. 2013.

[15] M. Agiwal, A. Roy, and N. Saxena, "Next generation 5G wireless networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 18, pp. 1617–1655, Third Quarter 2016.

[16] W. Fang *et al.*, "Joint spectrum and IT resource allocation for efficient vNF service chaining in inter-datacenter elastic optical networks," *IEEE Commun. Lett.*, vol. 20, pp. 1539–1542, Aug. 2016.

[17] M. Zeng, W. Fang, and Z. Zhu, "Orchestrating tree-type VNF forwarding graphs in inter-DC elastic optical networks," *J. Lightw. Technol.*, vol. 34, pp. 3330–3341, Jul. 2016.

[18] B. Li, W. Lu, S. Liu, and Z. Zhu, "Deep-learning-assisted network orchestration for on-demand and cost-effective vNF service chaining in inter-DC elastic optical networks," *J. Opt. Commun. Netw.*, vol. 10, pp. D29–D41, Oct. 2018.

[19] A. Sapio *et al.*, "In-network computation is a dumb idea whose time has come," in *Proc. of HotNets-XVI 2017*, pp. 150–156, Nov. 2017.

[20] N. Zilberman, "In-network computing," Apr. 2019. [Online]. Available: https://www.sigarch.org/in-network-computing-draft/.

[21] P. Bosshart *et al.*, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, pp. 99–110, Oct. 2013.

[22] Tofino switch. [Online]. Available: https://www.barefootnetworks.com/products/brief-tofino/.

[23] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.

[24] X. Li *et al.*, "Be fast, cheap and in control with SwitchKV," in *Proc. of NSDI 2016*, pp. 31–44, Mar. 2016.

[25] E. Cidon, S. Choi, S. Katti, and N. McKeown, "AppSwitch: Application-layer load balancing within a software switch," in *Proc. of APNet 2017*, pp. 64–70, Aug. 2017.

[26] X. Jin *et al.*, "NetCache: Balancing key-value stores with fast in-network caching," in *Proc. of SOSP 2017*, pp. 121–136, Oct. 2017.

[27] Z. Liu *et al.*, "DistCache: Provable load balancing for large-scale storage systems with distributed caching," in *Proc. of FAST 2019*, pp. 143–157, Feb. 2019.

[28] G. Li *et al.*, "NETHCF: Enabling line-rate and adaptive spoofed IP traffic filtering," in *Proc. of ICNP 2019*, pp. 1–12, Oct. 2019.

[29] L. Jose, L. Yan, G. Varghese, and N. McKeown, "Compiling packet programs to reconfigurable switches," in *Proc. of NSDI 2015*, pp. 103–115, May 2015.

[30] D. Kim *et al.*, "Generic external memory for switch data planes," in *Proc. of ACM HotNets 2018*, pp. 1–7, Nov. 2018.

[31] Remote direct memory access (RDMA). [Online]. Available: https://en.wikipedia.org/wiki/Remote_direct_memory_access.

[32] gRPC: an open-source universal RPC framework. [Online]. Available: http.//www.gRPC.io/.

[33] M. Su *et al.*, "RFP: When RPC is faster than server-bypass with RDMA," in *Proc. of EuroSys 2017*, pp. 1–15, Apr. 2017.

[34] R. Biswas, X. Lu, and D. Panda, "Accelerating TensorFlow with adaptive RDMA-based gRPC," in *Proc. of HiPC 2018*, pp. 2–11, Dec. 2018.

[35] L. Zhang and Z. Zhu, "Spectrum-efficient anycast in elastic optical inter-datacenter networks," *Opt. Switch. Netw.*, vol. 14, pp. 250–259, Aug. 2014.

[36] Facility location problem. [Online]. Available: https://en.wikipedia.org/wiki/Facility_location_problem.

[37] P4 behavioral model. [Online]. Available: https://github.com/p4lang.

[38] J. Sonchack, J. Aviv, E. Keller, and M. Smith, "Turboflow: Information rich flow record generation on commodity switches," in *Proc. of EuroSys 2018*, pp. 1–16, Apr. 2018.

[39] B. Li *et al.*, "KV-Direct: High-performance in-memory key-value store with programmable NIC," in *Proc. of SOSP 2017*, pp. 137–152, Oct. 2017.

[40] M. Liu *et al.*, "Offloading distributed applications onto SmartNICs using IPipe," in *Proc. of ACM SIGCOMM 2019*, pp. 318–333, Aug. 2019.

[41] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Cacheflow: Dependency-aware rule-caching for software-defined networks," in *Proc. of SOSR 2016*, pp. 1–12, Mar. 2016.

[42] S. Shirali-Shahreza and Y. Ganjali, "Delayed installation and expedited eviction: An alternative approach to reduce flow table occupancy in SDN switches," *IEEE/ACM Trans. Netw.*, vol. 26, pp. 1547–1561, Aug. 2018.

[43] G. Zhao *et al.*, "Joint optimization of flow table and group table for default paths in SDNs," *IEEE/ACM Trans. Netw.*, vol. 26, pp. 1837–1850, Aug. 2018.

[44] H. Huang *et al.*, "Joint optimization of rule placement and traffic engineering for QoS provisioning in software defined network," *IEEE Trans. Comput.*, vol. 64, pp. 3488–3499, Feb. 2015.

[45] Y. Xue *et al.*, "Virtualization of table resources in programmable data plane with global consideration," in *Proc. of GLOBECOM 2018*, pp. 1–6, Dec. 2018.

[46] Y. Xue, J. Peng, K. Han, and Z. Zhu, "On table resource virtualization and network slicing in programmable data plane," *IEEE Trans. Netw. Serv. Manag.*, vol. 17, no. 1, pp. 319–331, 2020.

[47] A. Dragojević, D. Narayanan, O. Hodson, and M. Castro, "FaRM: Fast remote memory," in *Proc. of NSDI 2014*, pp. 401–414, Apr. 2014.

[48] Q. Zhang, V. Liu, H. Zeng, and A. Krishnamurthy, "High-resolution measurement of data center microbursts," in *Proc. of IMC 2017*, pp. 78–85, Nov. 2017.

[49] Y. Han *et al.*, "Flow-level traffic matrix generation for various data center networks," in *Proc. of NOMS 2014*, pp. 1–6, May 2014.

[50] Y. Bejerano, "Efficient integration of multihop wireless and wired networks with QoS constraints," *IEEE/ACM Trans. Netw.*, vol. 12, pp. 1064–1078, Dec. 2004.

[51] V. Arya *et al.*, "Local search heuristic for k-median and facility location problems," in *Proc. of STOC 2001*, pp. 21–29, May 2001.

[52] K. Jain, M. Mahdian, and A. Saberi, "A new greedy approach for facility location problems," in *Proc. of STOC 2002*, pp. 731–740, May 2002.

[53] K. Jain and V. Vazirani, "Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation," *J. ACM*, vol. 48, pp. 274–296, Mar. 2001.