Software-Defined Flow Table Pipeline

Xiaoye Steven Sun
Rice University

T. S. Eugene Ng
Rice University

Guohui Wang
Facebook Inc.

***Abstract*—Software-Defined Networking (SDN) is revolutionizing data center networks for cloud computing with its ability to enable network virtualization and powerful network resource management that are crucial in any multi-tenant environment. In order to support sophisticated network control logic, the data plane of a switch should have a flexible Flow Table Pipeline (FTP). However, the FTP on state-of-the-art SDN switches is hardware-defined, which greatly limits the advantages of using FTP in cloud computing systems. This paper removes this limitation by introducing software-defined FTP (SDFTP), which provides an extremely flexible FTP as the southbound interface of the SDN control plane. SDFTP offers arbitrary number of pipeline stages and adaptive flow table sizing at runtime by building Software-Defined Flow Tables (SDFTs). Our analysis shows that SDFTP could create 138 times more adaptively sized pipeline stages than the hardware-defined data plane while maintaining comparable performance.**

I. Introduction

The data plane of the state-of-the-art SDN enabled switches has a Flow Table Pipeline (FTP). The pipeline consists of multiple flow tables. The incoming packets start with matching the first flow table; the actions in the matched entry could direct the packets to another flow table for the next step processing of the packets. With this packet redirection mechanism between flow tables, the control plane could build a logical single source directed acyclic graph on the FTP for packet processing.

FTP is a critical data plane feature to support flexible and efficient network control in cloud computing systems. Comparing with the data plane that has a single flow table, FTP offers several advantages at three different levels of network control, from network virtualization to individual control application design:

1. Virtualized network share: In a multi-tenant cloud computing system, each tenant creates its own network slice on top of the shared data center infrastructure. Network virtualization techniques are required in order to achieve the isolation and efficient sharing of the network resources, with which each tenant could independently control its own slice and run its control applications on the shared infrastructure without interfering with other tenants. However, achieving network virtualization in a large scale cloud is extremely difficult due to the complexity of network configuration on a large number of network devices for many tenants.

FTP makes network virtualization easier in the following ways [1]. Each network slice could occupy a sub-pipeline containing one or multiple flow tables in the FTP. The first flow table can be used as an arbiter which directs the packets to the sub-pipeline of the network slice that the packets belong to. Comparing with the control application installing the entries for all the network slices into a single flow table, the control applications built upon FTP can be independently developed without the interference with the control applications of other network slices. In other words, the tenant could design the control applications independently as if they exclusively own the network.

2. Modularized control application design: Modularization, the ability to decouple network control logic into individual application modules and compose them to construct more complex functions, is a key enabler to build sophisticated and scalable SDN control software. Recent works point out the importance of modularized SDN control application design and the necessity of composing control modules [2] [3] [4] [5] [6].

FTP supports the modularized control application design by giving each application module a flow table (or even a sub-pipeline), and organizing the flow tables (sub-pipelines) into a logical matching graph so as to achieve sequential and parallel composition of the application modules [4]. Should the data plane only have a single flow table, then the control plane must compile the network policies from multiple application modules into the entires in the single flow table. Using FTP in the data plane overcomes the following difficulties of using single flow table for application modularization:

1) Complex compilation logic: In a simple example where all the modules match different match fields of the packet header, the compilation logic needs to operate Cartesian product among the set of the policies in each module. In a more general case where the match fields of the modules overlap with each other, the compilation will be even more complex. As the sizes of polices in the modules and the number of modules increase, the compilation complexity will explode and become unmanageable, which makes it very hard to reason about the intended logic among application modules and introduces extra long delay during a network update.

2) Clumsy flow table management: With a single flow table, there is no isolation between the flow entries installed by different application modules. Any single entry installed by the complex compilation algorithm could inference the logic of all application modules. On the other hand, a policy in one module does not map to only one or isolated subset of flow table entries. This introduces complicated inter-dependency between flow table entries and application modules. Thus, it is cumbersome to add, modify or delete a policy in a module, since this change could involve the changes to multiple entries.

3) Huge flow table consumption: As discussed earlier, to compose the functions across multiple application modules, the compilation logic has to operate Cartesian product among all the policies from multiple modules, which causes factorial explosion of flow table consumption with the number of modules supported. For example, if there are N modules that process packets based on different match fields and each

module has M_n policies, the size of the Cartesian product, i.e. the number of entries, is $O(\Pi_{n=0}^{N-1} M_n)$, which scales poorly.

3. Hierarchical network management: Since the data center infrastructure is shared, a sophisticated network management control application is necessary in order to allocate the network resources and control the accessibility with the guarantee of performance, fairness, utilization and security. Such kind of network management logic can be achieved in a hierarchical manner [7], which makes network management more flexible and the application design more structured. The hierarchical packet matching tree can be constructed with the FTP by giving a flow table to the children belonging to the same parent, and building the forwarding branches among these flow tables. It is impossible to achieve such a hierarchical flow table matching on a single flow table.

A. Limits of Hardware-Defined Flow Table Pipeline

To fully realize the potentials of FTP, the FTP has to be made more flexible and software-driven to accommodate various application requirements and run-time dynamics of network control software. This is because 1) if the flow table size and the number of stages are fixed, the control plane could need more pipeline stages and larger flow table than what the FTP has; 2) if the data center network were to support dynamic policy changes, the needed number of pipeline stages, the number of entries in the flow table, and matching order between the flow tables could also vary accordingly at runtime. For example: 1) Cloud computing tenants might come and go, so the network slicing could change from time to time. Thus, the flow tables in sub-pipeline for a tenant should be acquired and released as needed. 2) When a new tenant joins, it might require that the virtual resources (link and bandwidth etc.) of existing tenants to be reallocated to different physical devices in order to accommodate the need of the network slice of the new tenant. In such a case, the flow table entries in the involved switches will also be changed drastically. 3) The network workloads and traffic patterns might vary a lot, so there might be needs to re-route the traffic in order to achieve better performance. In this case, the control applications will add a series of new entries into the flow tables of the switches on the new forwarding path of the traffic.

In summary, the number of pipeline stages, the number of flow table entries in each stage, and the forwarding among the flow tables should be reconfigurable at runtime so as to accommodate the varying needs of control applications.

However, the state-of-the-art FTP on SDN switches is hardware-defined, which means the pipeline has a fixed number of flow tables and fixed flow table sizes. Even worse, the control plane software cannot build a flexible enough FTP on top of such Hardware-Defined FTP (HDFTP) because of the restriction on the matching order of a packet among the flow tables. More specifically, the flow tables in the pipeline are numbered and a packet can only be forwarded to a flow table having a larger index number than the current flow table.

B. Software-Defined Flow Table Pipeline

In this paper, we propose software-defined FTP (SDFTP). SDFTP provides an extremely flexible FTP as the southbound interface of the SDN control plane. SDFTP offers arbitrary number of pipeline stages and adaptive flow table sizing at runtime by building Software-Defined Flow Tables (SDFTs). The remainder of this paper presents how we realize SDFTP over generic Hardware Flow Tables (HFTs). In order to fully utilize the available flow table entries in the data plane, we design an efficient Software-to-Hardware Mapping Logic (SHML) which makes any entry in the HFTs available to any SDFTs. Upon the SHML, we also design a fast and efficient entry insertion and deletion algorithm. Our analysis shows that SDFTP could create 138 times more pipeline stages than existing HDFTP while maintaining comparable performance.

II. Software-Defined Flow Table Pipeline

SDFTP acts as a southbound interface of the SDN control plane, which exposes a much more flexible logical pipeline. The logical pipeline is composed of arbitrary number of SDFTs each with adaptive table size. The control plane could arrange the SDFTs into arbitrary flow table matching orders. This section introduces the functions that SDFTP supports; how to achieve the SDFTP will be discussed in Sec. III.

SDFTP adds the following three representative flexibilities that the HDFTP cannot support:

1. Flexible software-defined matching order between SDFTs: SDFTP treats the HFTs in the data plane as general packet matching resources. In SDFTP, instead of requiring that the packets can only be forwarded to the flow table having larger indexes, the HFT supporting SDFTP does not pre-define nor restrict packet matching order between the HFTs. Specifically, the packet matching order among the SDFTs can be arbitrarily configured by the control plane. With such flexibility, once the control plane wants to create additional flow tables or even a sub-pipeline at runtime, these new flow tables could be integrated into the existing SDFTP.

2. Arbitrary number of SDFTP stages: For various data center networks, the numbers of needed flow table stages are different, since the networks may be different in sizes and the control applications may achieve different control logic. SDFTP is flexible in the number of flow table stages in the pipeline, since it is not limited by the number of HFTs on the switch. The control plane could even acquire new SDFTs at runtime, when the control plane loads a new set of applications or an existing application needs more flow tables. Once a control application terminates, the control plane could also release the SDFTs used by that application, and the associated hardware resources will become available for future use.

3. Adaptive size of SDFT: At runtime, the control applications may dynamically insert or delete flow table entries from the SDFTs in order to react to network changes or achieve better network performance. In SDFTP, the entry in an SDFT is adaptively allocated and deallocated, In other words, the size of an SDFT is the same as the number of existing entries in the SDFT. All the available entries in any HFTs can be used by any SDFT, so that the achievable size of an SDFT is only constrained by the aggregated total capacity of the HFTs.

III. Software-to-Hardware Mapping Logic

In this section, we propose SHML, which addresses the issue of how to map the SDFTs into the HFTs in the switches and how to insert and delete entries from an SDFT.

A. Arbitrary Flow Table Matching Order

On an SDN switch, such as the switch supporting OpenFlow v1.1 and above, there could be multiple HDFTs with the restricted matching order between the flow tables. However, in the data plane supporting SDFTP, an HFT is able to forward packets to any HFT (including itself) for the next step packet processing. As we will show in this section, this arbitrary flow table matching order capability fundamentally enables the flexibility in SDFTP.

In the concrete example in Fig. 1, the control plane runs two sequentially composed applications (shown in Fig. 1a), each of which occupies and configures its own SDFT. Incoming packets are firstly processed by the source address monitor and then forwarded by the routing application. The entries of these two applications are inserted to one HFT show in Fig. 1b. To avoid a matching loop under the arbitrary matching order, each incoming packet is tagged with an *sdft_id* as a *metadata* throughout the matching process in a switch. (The metadata is supported by latest OpenFlow protocol.) In an HFT entry, the *match fields* include the *sdft_id* in addition to the packet header fields and the *action fields* could include an action that changes the *sdft_id* of the matched packets to the *sdft_id* of the next SDFT, so that a packet can only match with the entries having the same *sdft_id* as the *sdft_id* of the packet.

B. SDFT segment and HFT fragment.

An SDFT can be divided into one or more SDFT segments. Each segment will be inserted into one HFT fragment, and an HFT fragment contains only one SDFT segment. The fragments containing the segments from the same SDFT are concatenated, so that all these segments can be virtually constructed into an entire SDFT. Finally, a "link" between the SDFTs could forward a packet to the next SDFT after the packet finds a matched entry in the current SDFT. To illustrate SHML, we use the example in Fig. 2, in which flow entries from four control applications are kept the same in SDFTs and are mapped into three HFTs, each of which have 100 entries.

SDFT Segment

To overcome the constraint of fixed HFT sizes, an SDFT is divided into multiple sorted segments according to the priorities of the entries, so that the priority of any entry in a segment is not lower than the priority of any entry in the following segments of the same SDFT. The index of each segment will be denoted as *seg_idx*. With this division, the segments of an SDFT can be stored in different HFTs, and hence the SDFT could always use the free entries in any HFT without being limited by the size of a particular HFT.

Fig. 2a shows an example. The rules in an SDFT is simply represented by a rectangle with different size and marked with the number of entries in that SDFT. In this example, SDFT1 and SDFT3 are both partitioned into two segments.In these two SDFTs, the entries in the light color region have priority higher than or equal to that in the dark color region underneath.

HFT Fragment

To overcome the constraint of fixed number of HFT stages, an HFT can also be divided into multiple logical fragments. A fragment contains all the entries of an SDFT segment. The *sdft_id*s in these entries are the same as the *sdft_id* of the SDFT that the segment belongs to (an HFT cannot contain multiple segments from the same SDFT). A packet tagged with a specific *sdft_id* can only match with entries among the fragments having the same *sdft_id* in *match fields* of the entries. This fragment representation provides the following three benefits. First, the entries in one fragment are not required to be installed in a contiguous HFT region, leading to great flexibility. Secondly, no HFT entry is wasted. Finally, the size of a fragment can be adaptively changed with no overhead.

Fig. 2b gives an example. In the figure, entries for different SDFTs are marked with different patterns. The original *match fields*, *priority* and *actions* of the SDFT entries are not shown, since they remain the same in the HFTs. In this figure, HFT0 and HFT2 are partitioned into 2 and 3 fragments respectively. For example, in HFT0, one fragment is assigned to the whole SDFT0 and the other fragment is given to the lower priority segment of SDFT3. With this approach, even a single HFT could support an enormous number of SDFTs by giving the fragments to the SDFT segments.

Concatenate Fragments

It is easy to see that matching against the segments of an SDFT one-by-one in a priority non-increasing order until a packet finds a matched entry is the same as matching against the entire SDFT. Based on this observation, the segments of an SDFT are concatenated in a priority non-increasing order (*seg_idx* increasing order) by installing an auxiliary entry in the HFT fragment. This auxiliary entry matches the same *sdft_id* and has the lowest priority, so it could forward the mismatched packets of the current segment to the HFT containing the next segment. The fragment of the last segment of an SDFT does not have the auxiliary entry.

The two black entries in Fig. 2b are the auxiliary entries. HFT1 contains the first segment of SDFT1 and an auxiliary entry, while the second segment of SDFT1 resides in HFT2. If a packet tagged with *sdft_id*=1 cannot find any matched entry in the first segment, it will match with the low priority auxiliary entry and be forwarded to HFT2 where the second segment of SDFT1 resides (indicated by a hollow arrow). That packet will keep matching against the fragment with *sdft_id*=1, since the *sdft_id* of that packet is unchanged.

Link between SDFTs

A packet may need to be matched against the next pipeline stage after it finds a matched entry from the current SDFT. In an HFT, matching a packet against an SDFT starts by matching it against the first segment. Thus, the regular entries (not including auxiliary entry) need to change the *sdft_id* of the packet to that of the next SDFT and forward the packet to the HFT where the first segment of next SDFT resides.

As we can see in Fig. 2b, the entries for both segments of SDFT1 change the *sdft_id* to 2, which is the *sdft_id* of SDFT2 and then forward the packet to HFT2, where the first segment of SDFT2 resides.

C. Flow Table Update

Although SHML shuffles the placement of the entries by dividing an SDFT into segments, it still can perform entry

SDFT 0: Monitoring Application		
Match Fields	Priority	Actions
src_ip=192.168.1.0/24	2	count, goto_table:1
src_ip=192.168.0.0/22	1	count, goto_table:1

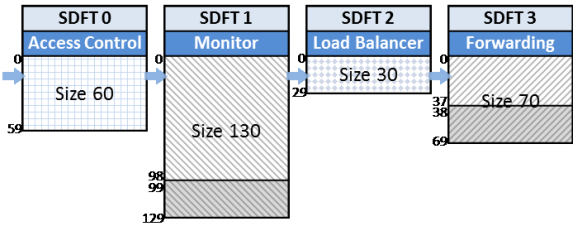
SDFT 1: Packet Forwarding Application		
Match Fields	Priority	Actions
dst_ip=192.168.9.0/24	2	fwd:2
dst_ip=192.168.8.0/22	1	fwd:1

(a) Application policies

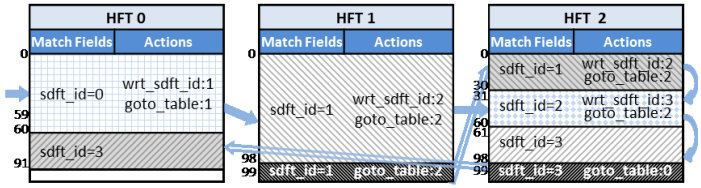
HFT 0: with Arbitrary Flow Table Matching Order		
Match Fields	Priority	Actions
sdft_id=0, src_ip=192.168.1.0/24	2	count, wrt_sdft_id:1, goto_table:0
sdft_id=0, src_ip=192.168.0.0/22	1	count, wrt_sdft_id:1, goto_table:0
sdft_id=1, dst_ip=192.168.9.0/24	2	fwd:2
sdft_id=1, dst_ip=192.168.8.0/22	1	fwd:1

(b) Entries in an HFT supporting arbitrary matching order

Fig. 1. Arbitrary matching order illustration



(a) Software-defined flow tables



(b) Hardware flow tables

Fig. 2. An example for SHML

insertion and deletion efficiently. To do that, the SHML must keep recording the following three mapping informations: 1) the SDFT segment and HFT fragment mapping pair for each SDFT, i.e., the pair of the *seg_idx* of a segment in an SDFT and the *hft_id* of the HFT where that segment is stored. 2) the priority range, namely the highest and the lowest entry priorities of each segment. 3) the number of free entires in each HFT.

Insertion Algorithm

When the control application inserts a new entry to an SDFT, in most cases, choosing the segment can be done quickly by looking up the mapping information, which adds ignorable delay to the entry insertion. The new entry could be directly inserted to the segment of the SDFT satisfying the following criteria: 1) the allowed lowest entry priority of the segment should be no larger than the priority of the new entry. 2) this segment should have the smallest *seg_idx* among the segments satisfying the above criterion. 3) the HFT containing this segment has an available entry.

The segment satisfying the first two criteria is denoted as **target segment**. There are three different cases under the situation where the target segment cannot meet the third criterion. The condition of these three different cases can be easily got from the recorded mapping informations.

Case 1: the HFTs containing other segments in the same SDFT have a free entry. In this case, free space can be created in the target segment by moving entires among the segments. In order to reduce the number of entries involved in this movement, the algorithm can choose the segment closest to the target segment, and the HFT containing this chosen segment should have free space. By saying 'closest' we mean that the absolute value of the difference between the *seg_idx* of the chosen segment and that of the target segment should be minimum. If the chosen segment is prior to the target segment, the algorithm starts from the chosen segment, and moves the highest priority entry from the following segment to the previous segment recursively until it reaches the target segment. If the chosen segment is subsequent to the target segment, it recursively moves the lowest priority entry from the previous segment to its following segment until it reaches the target segment.

Case 2: none of the HFTs containing the segments in the same SDFT has free entry, while there is another HFT has at least two free entries. In this case, the algorithm will create a new segment following the last segment.Choosing the HFT could involve certain optimization (discussed in Sec. V), but at least the chosen HFT should satisfy the following requirements: 1) the HFT doesn't contain any other segment of that SDFT. 2) the HFT has at least two available entries. The algorithm will move two of the lowest priority entries from the last segment to the new segment since an auxiliary entry will be inserted to the last segment in order to forward the mismatched packet to the new segment. Then, algorithm will move the corresponding entries as if there is free space in the last segment in case 2. The time complexity of the worst case is $O(H*t_{move} + 2*t_{insert})=O(H)$, where H is the number of HFTs.

Case 3: none of the above cases. In this case, there is no free space in the HFTs. Thus, the new entry cannot be inserted to the HFTs unless some of the existing entries are removed.

Deletion Algorithm

Deletion is much simpler than insertion. When a control application deletes an entry in an SDFT, the *hft_id* of the HFT containing the entry can be retrieved from recorded mapping informations, and then the entry can be simply deleted.

IV. Performance of SDFTP

A. Maximizing Throughput

If an HFT supports arbitrary matching order, multiple packets could contend for the same HFT. This is because that the fragments in an HFT belong to the segments of different SDFTs and a packet might match against these SDFT segments throughout the processing of a packet. It is easy to proof that, given the throughput of an HFT is T_h, i.e., each HFT has the ability of processing T_h packets per second, the throughput of the SDFTP with SHML is at least $\frac{T_h}{N}$, where N is the maximum number of fragments among HFTs.

To maximize the throughput of the data plane of SDFTP under SHML, the switch could use the first HFT to store the results of the packet matchings. The first HFT can be used in the following way. After a packet finishes matching the pipeline, an exact-match rule specifying all the match fields will be inserted to the first HFT. The action field of this exact-match rule is the list of actions accumulated from all the matched entries. The packets mismatched with the first

HFT will match with a wildcard auxiliary entry in the first HFT. The auxiliary entry sets the *sdft_id* to the first SDFT and forwards the packets to the HFT where the first segment of the first SDFT resides.

Using the first HFT in this way could greatly improve the throughput of the data plane by several orders of magnitude, since it exploits the temporal locality of the network traffic. In principle, if the mismatch rate of the first HFT (number of packets match the auxiliary entry per second) can be suppressed below $\frac{1}{N}$ of the throughput of an HFT, the throughput of the data plane will not be affected. We implement a simulator computing the mismatch rate at each $1ms$ time slot, based on the LRU entry replacement mechanism. We run the simulation with an HFT having only 1000 entries and a data center packet trace from [8]. The CDF of the per-$1ms$ mismatch rate shows that such a small HFT could make the 99.9th percentile of the mismatch rate under 6000 mismatch per second. However, a 10Gbps switch could process the packets at 833k packets-per-second at each port, which is already 138 times more than the mismatch rate. This implies that an HFT could contain at least 138 fragments without hurting the performance of the data plane. In addition, as we will show in Sec. V that, in reality, the throughput could be further improved, since not all the SDFT segments will be matched during the processing of a packet.

B. Minimizing Latency

In SDFTP, an SDFT could be divided into multiple segments. When a packet matches against a partitioned SDFT, it actually need s HFT matching operations, if the first matched entry is in the sth SDFT segment. As a consequence, comparing with a single HFT matching in HDFT, these $s-1$ extra matching operations will add latency to the packet processing. However, adding packet processing stages is a common choices in data plane design [9], [10], since the TCAM lookup takes no more than a few tens of nanoseconds, which is very small relative to many other sources of delays (e.g. packet queuing and end host packet processing). In Sec. V, we will show that the partition in SDFT only adds few stages during the processing of a packet. Moreover, most of the packets will be processed by the first HFT without being processed by the fragmented HFTs.

V. SDFT PLACEMENT

A. SDFT Placement Strategies

Although a data center network could experience frequent changes which can be reactively responded by the control applications, many of the control applications could also proactively control the network in order to set an initial configuration for the network or to save the time in runtime network configuration. This section answers the question about given a set of SDFTs, how to place them into the HFTs so as to achieve minimum impact on the data plane performance.

With the two metrics discussed in Sec. IV to be optimized, placing the SDFTs in HFTs is a hard problem. We propose a series of SDFT placement strategies working under the framework shown in Alg. 1. The policies of choosing an SDFT candidate (Line 2 in Alg. 1) can be that either pick the SDFT that has the maximum (**s-max**) or the minimum (**s-min**) number of entries not being installed to HFTs. The policies of choosing an HFT candidate (Line 3 in Alg. 1) can be that either pick the HFT that has the maximum (**h-max**) or minimum (**h-min**) number of remaining capacity. Thus, each strategy has a unique combination of the SDFT and HFT candidate choosing policies, so there are four different strategies in total.

Algorithm 1 SDFT placement algorithm framework

1: **while** not all SDFT entries are placed into HFTs **do**
2: Choose an SDFT candidate
3: Choose an HFT candidate
4: Insert the top N highest priority entires in the SDFT candidate to the HFT candidate, where N is the smaller one among the number of remaining entries in the SDFT and the remaining capacity of the HFT
5: **end while**

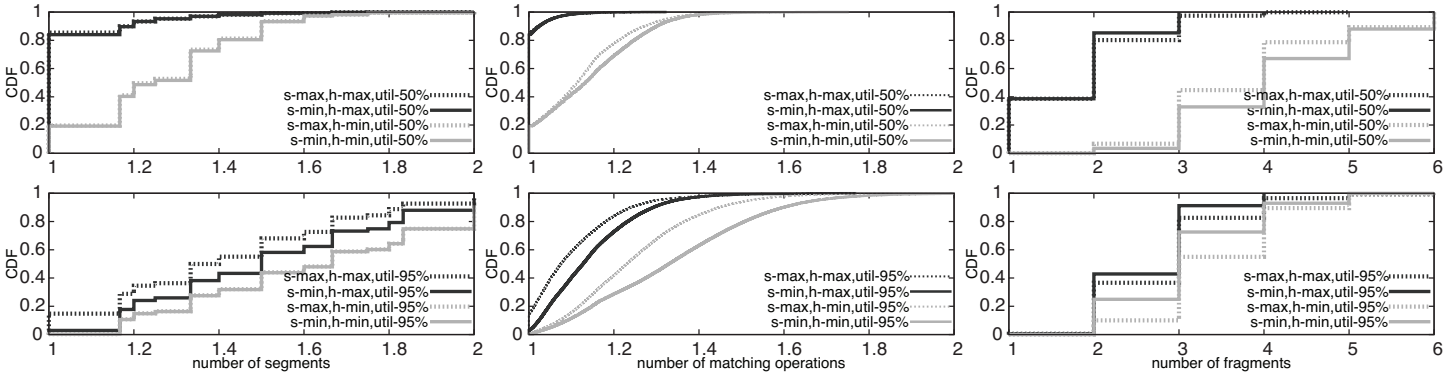
B. Strategy Evaluation

1) Evaluation Setup: We evaluate the SDFT placement strategies on s SDFTs and h HFTs, where s ranges from 3 to 6 and h ranges from 2 to 7. So there are 24 different combinations of the numbers of SDFTs and the numbers of HFTs. The strategies are insensitive to the absolute sizes of SDFTs while the relative sizes among the SDFTs and the utilization of the HFTs could affect the placement. For s SDFTs, the size of each SDFT can be any number chosen from the list of $\{$1k, 2k, ..., 10k$\}$, and we test the strategies on all unique combinations of the s SDFT sizes. For h HFTs, we vary the HFT sizes so that the overall HFT utilization rate is either 50% or 95%; for each case, the HFT capacity is equally distributed into all of the h HFTs. In summary, each case has different numbers of SDFTs and HFTs, SDFT sizes, and HFT utilization rates.

2) Results: The evaluation on different strategies is based on the comparison on the two metrics defined in Sec. IV. We show the CDFs of the total numbers of the SDFT segments and the maximum numbers of fragments among HFTs in each case. Besides that, we also plot the CDF of the average numbers of matching operations needed for packets with the assumption that all the entries in an SDFT have the same probability to be matched and all packets need to match all SDFTs. The total number of SDFT segments and the average number of matchings are normalized by the number of SDFTs in the corresponding case, so they represent how many segments an SDFT has and how many matchings needed for an SDFT respectively. These CDF plots are shown in Fig. 3 with different HFT utilization rates shown separately. We make the following conclusions:

1. Strategies that adopt h-max has much better placement than those use h-min. This is because that the less remaining capacity in an HFT usually implies that the HFT could already contain multiple fragments and the less remaining capacity could also result in a further partitioning in the SDFT being placed. Thus, we will only consider the h-max policy in the following discussion.

2. The strategy of s-max/h-max could create less segments than s-min/h-max, while the later strategy could create less maximum number of fragments. This is because that always choosing the largest SDFT could reduce the chance of partitioning, since at the beginning of the placement, the remaining capacity in each individual HFT is larger than later;



(a) (b) (c)

Fig. 3. Strategy evaluation (a)normalized total number of segments (b)normalized average number of matchings (c)maximum number of fragments in HFTs

however, in the case that the SDFT sizes are highly imbalanced and there are more small SDFTs, choosing the largest SDFT could make a lots of the small SDFTs placed in a single HFT.

3. The average number of matching operations is usually much less than the number of segments. For example, even in the cases utilizing 95% capacity of the HFTs, the 90th percentile of the number of segments in an SDFT is 2, while the 90th percentile of the average number of matching operations per SDFT is only no more than 1.3.

4. In lower HFT utilization, the quality of the placement is better than that in high HFT utilization. This is because the much larger free space of the HFT could reduce the change of dividing the SDFTs, since in most cases, the HFT is able to contain the whole SDFT.

VI. Related Work

Monsanto et al. [4] and Foster et al. [11] propose programming language abstractions as the northbound API of the SDN control plane to support the composition of modularized control applications. Although SDFTP also supports control application modularization as part of its function and provides advantages over the language compilation, we don't view these as mutually exclusive works for the following reasons. 1) The control plane using the programming language abstraction could still use SDFTP as the southbound interface that provides flexible FTP to the control plane so that the control plane does not tightly constrained by the HDFTP. 2) In data center networks, the tenant could have the flexibility to choose his own network programming language to achieve the network control. In this case, the SDFTP could provide each programming platform with a sub-pipeline.

FlowAdapter [12] is a middleware layer on switches designed to enable OpenFlow multi-table processing on legacy hardware. However, the software flow table model and the software-to-hardware table conversion algorithm in FlowAdapter is tightly coupled to its particular way of using *metadata* field in OpenFlow protocol, which limits the flexibility of using multiple tables for independent application modules. Similar to FlowAdapter, HAL [13] also provides the hardware resource abstraction for legacy hardware switch. However, SDFTP is built on top of the general TCAM flow table enabled by the state-of-the-art technology.

Bosshart et al. designed reconfigurable FTP in [14], in which the number of flow table stages and the size of the flow tables can be reconfigured. However, the reconfiguration cannot be done at runtime so it still an HDFTP lacking of the flexibility the SDFTP could provide.

References

[1] W. Wendong, Y. Hu, X. Que, and G. Xiangyang, "Autonomicity design in openflow based software defined networking," ser. Globecom Workshops (GC Wkshps), 2012 IEEE.

[2] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Can the production network be the testbed?" ser. OSDI'10.

[3] S. Gutz, A. Story, C. Schlesinger, and N. Foster, "Splendid isolation: a slice abstraction for software-defined networks," ser. HotSDN'12.

[4] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing software-defined networks," ser. NSDI'13.

[5] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," ser. ICFP'11.

[6] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson, "Fresco: Modular composable security services for software-defined networks," ser. NDSS'13.

[7] I. Stoica, H. Zhang, and T. S. E. Ng, "A hierarchical fair service curve algorithm for link-sharing, real-time and priority services," ser. SIGCOMM '97.

[8] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," ser. IMC '10.

[9] Y. Ma and S. Banerjee, "A smart pre-classifier to reduce power consumption of tcams for multi-dimensional packet classification," ser. SIGCOMM'12.

[10] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with difane," ser. SIGCOMM '10.

[11] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: a network programming language," ser. ICFP'11.

[12] H. Pan, H. Guan, J. Liu, W. Ding, C. Lin, and G. Xie, "The flowadapter: Enable flexible multi-table processing on legacy hardware," ser. HotSDN'13.

[13] D. Parniewicz, R. Doriguzzi Corin, L. Ogrodowczyk, M. Rashidi Fard, J. Matias, M. Gerola, V. Fuentes, U. Toseef, A. Zaalouk, B. Belter, E. Jacob, and K. Pentikousis, "Design and implementation of an openflow hardware abstraction layer," ser. DCC '14.

[14] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," ser. SIGCOMM '13.