P4KP: QoS-Aware Top-K Best Path Using Programmable Switch

DEBOBROTO DAS ROBIN AND JAVED I. KHAN
Department of Computer Science, Kent State University, Kent, OH 44240, USA
Corresponding author: Debobroto Das Robin (drobin@kent.edu)

ABSTRACT
Data center networks offer multiple parallel paths between a source-destination pair. But, due to the use of TCAM based single logical step ‘path-search and selection’ mechanism in the switches, most of the existing packet forwarding systems can utilize only a single-best path. It prevents them from utilizing the benefit of multiple paths for achieving application-specific QoS objectives. In this work, we propose P4KP, a programmable switch-based novel scheme for utilizing the top-k best paths toward a destination in a QoS-aware manner. P4KP complements TCAM’s path-selection capability using SRAM; decouples path-searching from path-selection and facilitates the path-selection using an SRAM-based data structure. Its SRAM-based data structure reduces costly TCAM usage and enables the efficient update of the top-k best paths in the data plane. P4KP also provides a bitmask-based API to express the QoS-based policy for selecting one-of-the-top-k best paths. We implemented P4KP using P4 language to demonstrate that it is realizable in currently available programmable switches while maintaining line-rate throughput. Our preliminary evaluation shows that P4KP significantly reduces the stateful memory consumption in programmable switches and achieves improved performance.

INDEX TERMS
QoS, programmable networks, P4, BMv2.

I. INTRODUCTION
Data center networks (DCN) typically contain multiple paths between a source-destination pair to carry traffic from various applications with diverse QoS objectives. Using multiple paths for traffic forwarding can significantly increase a system’s throughput [1]. At the end-host side, several transport layer protocols [1], [2] came into existence to rip this benefit. Innovation in end-host protocol stack design and implementation [3]–[5] has opened the opportunity to convey application-level QoS objectives to network switches using these protocol stacks. At the switch end, Protocol Independent Switch Architecture based programmable switches (a.k.a PISA switch) [6] along with the P4 programming language, have opened opportunities for including these application-level QoS objectives in forwarding path selection logic through data plane programming. Several PISA switch-based systems [7]–[9] focusing on various aspects of QoS-aware packet forwarding are proposed in recent times. However, most of them are inspired by the best-effort service model and consider only one best path at a time. Hence, they cannot make proper utilization of the network’s multipath for achieving application-level QoS objectives.

Consider the network of fig. 1a with four paths between L-1 and L-4. Assume a probe-based monitoring system measured each of the path’s delays as shown in the match-action-table (the main mechanism for path-search in PISA switches) of fig. 1b. Also, assume two different flows from 2 different traffic classes (source & destination are connected with L-1 & L-4) require their packet to be forwarded through a path with 5-10 ms and 25-40 ms delay. Now, both can use the best path (7 ms delay). But, that can make the best path over-utilized and other paths under-utilized. Alternatively, the utilization rate of the links can be probed in a regular interval (e.g., 50ms), and the paths in the forwarding table (fig. 1b) can be ordered based on both link utilization rate and path delay. With the increase in the number of fields used for filtering paths between source-destination pairs, the total number of entries in the forwarding table also increases. Data center networks can contain a large number of paths between source-destination pairs. Storing such a large number of entries in a table and updating them (in 50 ms timescale) is not scalable. Most importantly, this approach also selects the best path from all possible paths toward a destination.
An alternative way is to use the *path-delay range* (or any other QoS measurement in general) based aggregation technique in the forwarding table. For example, using *n* disjoint ranges for path delay and one best path for each range (fig. 1c) can use *n*-best paths at a time. But this approach increases the total number of entries in the forwarding table by at least *n* times (using range-based match fields can increase the table length even more [10]). A generalization of this approach is shown in fig. 1d; it maintains a group (may consume extra table and stage using P4_{16} ActionProfile [11]) of paths for each path delay range. After selecting the required range, one path is selected from the corresponding group using a QoS unaware policy (P4 Action Selector provides hash, random, or identity algorithm for selecting group members), similar to ECMP (shown in fig. 1e). Finding a path from these tables can be inefficient in various scenarios. Consider at a certain time; all the paths are in the 21-40 ms delay range, and a QoS-aware packet forwarding algorithm wants to find a path with a 15 ms delay. Searching such a path for a packet will lead to a ‘*miss*’ as there is no path in the 0-20 ms delay range. To avoid this, the delay ranges used in the forwarding table (fig. 1c and d) can be configured as 0-20ms and 0-40 ms. This overlapping range-based approach can avoid ‘miss’ while searching the path in the table. But for a matching query in the table, there will be multiple paths, and only the highest priority path will be selected. Therefore, this approach also suffers from the same problem as the approach shown in fig. 1b.

A TCAM stores separate flow-rule entries for various combinations of the fields used to find a path for a packet. All the entries remain active all the time, and a path is selected in a single clock cycle after searching over all of them. This performance comes at a higher cost of circuit area and power consumption [12]. Due to higher costs, TCAM storage space in switches is limited. On the other hand, the total number of entries in TCAM increases multiplicatively with an increase in the number of fields used for QoS aware path search. The rapid proliferation of network virtualization and multi-tenancy in the data center environment contributes to manifold increases in the number of TCAM entries. As a result, supporting QoS aware path selection algorithm using TCAM suffers from scalability problems also.

Considering these facts, most of the existing PISA switch-based works use a QoS aware *single-best-path* [13] or ECMP [14] like QoS unaware multipath capable policy-based packet forwarding schemes. However, the *single-best-path* policy cannot utilize the benefit from multiple paths in the network, and ECMP cannot support QoS-aware packet forwarding. A careful observation of the upper mentioned schemes reveals that TCAM centric ‘*path-search and selection*’ approach borrowed from traditional fixed protocol switches is the main reason behind this limitation. Traditional switches (routers) were designed for forwarding a packet based on some fixed header fields from selected protocols. In these switches, TCAMs were used for searching and selecting a path for a packet in a *single logical step*. This inherent tight coupling between *path-search and path-selection* was designed to achieve optimal performance. TCAMs are also the main instrument for path search and selection in PISA switches. But their multi-stage pipeline-based architecture [6] provides scope for complementing the capabilities of TCAMs. Moreover, traditional switches were designed for a fixed set of protocols with no or little support for utilizing multipath to achieve application-specific QoS objectives. This paradigm is not true anymore. Supporting application-specific QoS objectives is one of the most crucial design goals in futuristic data center networks.

Driven by this goal and the opportunities provided by the PISA switches, we propose **P4KP**: a QoS-aware scheme for managing *top-k* best paths and QoS policy-aware selection of *one-of-the-top-k* paths using P4 supported programmable switches. It decouples the single logical step ‘*path-search and selection*’ and divide them over two logical steps, P4KP contains an SRAM based data structure to store *top-k* best-paths (or top-k group of paths) toward a destination according to their QoS-based rank and efficiently updating them. It also provides a low-cost scalable alternative for storing a large number of flow-rule entries using SRAM. P4KP is designed to work as a building block for any QoS-aware forwarding algorithm. It can bridge the gap between QoS-aware *single-best-path* and ECMP like QoS unaware multipath capable policy.

The paper’s organization is as follows: in section II, we summarize some of the important related works. Then, in section III, we formally define the problem, and in section IV, we give an overview of a generalized system model with a detailed discussion on how P4KP can be used in a data center environment. In section V, we present an overview of P4KP; then in section V-B, we present the QoS-aware path-selection policies supported by P4KP. In section V-C, we explain the details of P4KP’s underlying data structure. In section V-D, we describe how path update and search works on this data structure. In section VI, we present the performance of P4KP and the total resource required to realize it in a PISA pipeline. In section VII, we discuss few related issues, and in section VIII we present the implementation of P4KP using P4_{16} [11] programming language and
evaluate its performance. Finally, we conclude the paper in section IX.

II. RELATED WORK

Several systems exist in the literature to improve overall network performance rather than achieve the application-specific QoS objectives. For example, Hedera [15], SMORE [16], MicroTE [17], DRILL [18], GreenDCN [19] etc., focuses on improving flow completion time, robustness, energy efficiency, etc. These systems do not focus on achieving QoS objectives of different types of applications. Another approach is to use QoS policy-directed probes to monitor a selected set of paths and provision suitable paths. Contra [20], PolicyCop [21], etc., follow this approach. But these systems are limited to only one path for each policy. As they use TCAM based single logical step ‘path-search and selection’ mechanism for selecting paths, they cannot accommodate multiple paths per policy.

Some of the research works focus on using multiple controllers in a hierarchical structure for scalable QoS-aware routing. Such as QROUTE [7] implements a scalable system targeted to achieve multiple QoS constraints aware packet forwarding in overlay networks. It relies on a hierarchical controller and uses table entries to provision QoS conforming paths. For large-scale data center networks, its forwarding table can still grow very large. Though, QROUTE provides a strong abstraction for representing QoS policy-aware path-selection using PISA switches, it is unable to use multiple paths for a single destination.

Few recent works have focused on using top-k paths toward a destination. For example, MP-HULA [22] splits traffic over top-k paths based on link utilization, but it needs k separate match-action tables. As the total number of stages and tables per stage available in a PISA pipeline is small, it can only support a limited number of paths using many stages. DASH [23] uses stateful ALUs to split traffic over multiple paths. It does not use table entries for storing path information. Hence it can adapt to changing traffic dynamics very quickly. But DASH does not provide any abstraction for using it to achieve QoS policy-aware packet forwarding.

This work focuses on designing abstractions and relevant data structures for utilizing the multipath capabilities of modern networks in a QoS aware manner.

III. PROBLEM STATEMENT

Consider an N criterion-based path-search query Q to find a QoS-aware forwarding path for a packet:

\[ Q = \text{if Criteria } C_1 \text{ AND } C_2 \text{ AND } \cdots \text{ AND } C_N \text{ matches Select a matching path} \]

Execution of Q in PISA switches can be divided into two steps: a) path-search: find the set of all path (PathSet) matching with the given criteria \( C = \bigcup_{i=1}^{N} C_i \) and b) path-selection: select a path from PathSet according to a Policy.

Match-action-table (MAT) abstraction is the main mechanism for criteria-based path selection in PISA switches. TCAM is the main hardware primitive for supporting MAT (as SRAMs only support exact matching, whereas TCAMs support exact, longest-prefix, ternary, and range matching we consider only the TCAMs here). The \textit{match} part of a MAT handles the path-search step of Q, and the \textit{action} part handles the path-selection step.

When a packet is matched with the rules in a MAT, a set of paths (PathSet) can satisfy the given criterion. PISA hardware selects the highest priority path (best-path) from the PathSet. But this single best-path policy is not suitable for utilizing multiple paths offered by the network for achieving various application-level QoS objectives. The goal is to design a scheme for PISA switches to utilize multiple paths for a QoS-aware path-selection query.

IV. PROPOSED SYSTEM MODEL

This section describes how a QoS-aware forwarding algorithm can use P4KP for QoS aware path selection.

A QoS-aware forwarding algorithm is characterized by QoS field-based matching rules (path-search criteria(s)) and the corresponding set of paths (section III) in the data plane. Usually forwarding algorithms store the rules and paths in a single TCAM based table (to facilitate IP prefix based searching). The single step ‘path-search and selection’ mechanism matches a packet against the path-search rules, and an appropriate path is selected according to the algorithm. As both the steps are logically coupled (actual mapping to hardware may differ) together in a TCAM-based implementation, it is hard to replace the single best-path policy-based path-selection mechanism. Besides this, the number of paths matching with a path-search query can be very large [24]–[26], and maintaining all of them may not be possible. P4KP decouples the rules used for path-search and corresponding paths. It stores top-k best paths (ranked from 0 to \( k - 1 \)) in a PathSet for each of the path-search rules and provides an API to select one-of-the-top-k paths. P4KP can be configured (at compile-time) to store a single path or a group of M paths in each rank. (later configuration consumes more stages and stateful memory in the PISA pipeline).

A. REQUIRED MODIFICATIONS IN EXISTING APPROACHES

Different rules for criteria-based path-search can result in different PathSet. Decoupling path-selection and path-search implies that maintaining a mapping from path-search rules to the candidate set of paths (PathSet) is required. Besides this, a policy for selecting a path from the PathSet is also required. For each of the PathSet, a unique index (PathSetID) can be maintained by CP. Several frameworks for generating a unique index for different paths exist in the literature [24]. These frameworks can be extended for generating the PathSetID for each PathSet. P4KP assumes that the control plane provides all the supported path-selection policies in the form of a k-bit bitmask (policyMask) (more on this in section V-B). Various proposals [21] exist in literature for negotiating the Service-Level-Agreements (SLA) between controller and the end-hosts. Based on the
Routing protocols like BGP [29], OSPF [30], etc. are widely systems are required. In DCN environment, standardized pathID case for assigning unique identifier for the paths (CriteriaMatcherMAT) the path in various scope. For example, port numbers can be used for uniquely identifying a path for hop-by-hop packet the paths toward the destinations. Wide range of schemes are avaialble for path discovery and configuration in data networks. As this is subject to use case, we do not discuss it in this work.

B. PROPOSED SYSTEM MODEL

Fig. 2 shows the general system model of a QoS-aware packet forwarding system that uses P4KP.

1) CONTROL PLANE

Role of the control plane in the proposed model can be divided into 2 logical parts (various protocols and algorithms can be used for relevant functionalities): network path manager and the P4KP control plane.

The network path manager can use different algorithms and protocols to discover, configure and monitor rank of the paths toward the destinations. Wide range of schemes are available for path discovery and configuration in data centers. These schemes assigns unique identifier for each of the path in various scope. For example, port numbers can be used for uniquely identifying a path for hop-by-hop packet forwarding algorithms; XPath [24] assigns unique identifier and configures paths at a scope of whole data center network; similarly, for tunnel based packet forwarding schemes [27], [28], tunnel-ID can be used as the PathID. An administrator can select any suitable scheme according to their use case for assigning unique identifier for the paths (pathID). These pathIDs are used by the P4KP.

Similarly, for finding the ranks of the paths, monitoring systems are required. In DCN environment, standardized routing protocols like BGP [29], OSPF [30], etc. are widely used [25], [31], [32] for determining relative ranks of the paths. Recently, various new schemes also came into existence for monitoring and ranking the paths. For example, HULA [13] or MP-HULA [22] monitors path utilization and use it as the metrics to determine rank of a path. In general, P4KP does not put any restriction on use of monitoring schemes to be used. The QoS-aware forwarding algorithm can use any kind of monitoring schemes depending on its requirements. The network path manager will use the results obtained from these schemes to calculate the top-k best path toward a destination and their ranks. After identifying the paths and their ranks, the network path manager feeds both pathID and their ranks to the P4KP control plane. And the P4KP control plane installs (section V-D1) them in data plane through control-messages.

For using the P4KP, the QoS aware forwarding algorithms also need to provide the path-selection policy. This is a shift from the single MAT abstraction based representation of the forwarding algorithm, where path-search rules and paths are configured together. In P4KP based forwarding schemes, whenever a path selection criteria/rule is needed to be inserted or updated, the network path manager have to configure path-selection rule, unique ID for the PathSet and path-selection policy (policyMask) in the CriteriaMatcherMAT. If the PathSet is configured to store top-k paths towards a destination then the policyMask is provided as a k-bit bitmask.

2) DATA PLANE

The data plane in P4KP based systems play three key roles: a) path-search: as part of the QoS-Aware Forwarding Algorithm, a packet is matched with the CriteriaMatcherMAT to find the PathSet’s ID (PathSetID) and policyMask for the matching destination b) P4KP data structure and path-selection: the P4KP Data Plane maintains the data structure for storing PathSet for each matching path-search query and facilitates path-selection (section V-D2) according to policyMask. On receiving control-messages for updating a PathSet, the data plane also executes the required logic (section V-D1) for updating the paths in the data structure c) executes any other functionalities (e.g., monitoring, handling packets for routing protocol and algorithm etc.) required for the system.

C. FOCUS OF THE WORK

In this work, we focus on designing and implementing data structures for the P4KP control plane and P4KP data plane functionalities.

V. P4KP

A. OVERVIEW

P4KP differs from the single MAT-based systems in 2 key aspects. Firstly, it extends the best path-selection policy to ‘one of the top-k’ path-selection policy. Secondly, instead of storing both path-search rules and corresponding PathSet in a single MAT, P4KP stores path-search rules in one
(or multiple) MAT (CriteriaMatcherMAT) and stores the PathSet in a later stage in the PISA pipeline. The PathSet is stored in a 2-D (k x M) array (PathStore) of stateful memory. All paths in the PathSet are ranked in the range [0 · · · (k − 1)] by CP (according to QoS value). Multiple paths can have the same QoS-based rank R. CP stores them in the R − 1-th row in PathStore. Maximum M paths can be of the same rank. Both M and k are compile-time configurable. Hence the size of the 2D-array can be customized according to use cases and resource availability. Instead of directly selecting a single best-path when C is matched, P4KP selects a k-bit bitmask (policyMask). Through the policyMask, a QoS-aware forwarding algorithm defines which path to select from the PathSet.

As a concrete example, consider the case of a QoS (path delay) aware path selection algorithm in a DCN switch. Delay of a path can be in one of 4 ranges (ranked in increasing order): 0-20ms, 21-40ms, 41-60ms, and 61-80 ms. Assume, P4KP maintains top-k best paths (k = 4) and the operator need to support 4 different traffic classes (shown in fig. 3a) having different QoS requirements. Fig. 3b shows required (delay) aware path-selection policies for these four traffic classes. The CriteriaMatcherMAT only shows the entries for a single destination (IP address = 10.0::1 and PathSetID = 1). In the action part of this MAT, the PathSetID and the policyMask is configured by the control plane. We discuss the policyMask generation mechanism in section V-B. How P4KP populates the paths in a QoS-aware manner is discussed in section V-C. The mechanism for updating rank of the paths are discussed in section V-D1, and the workflow for finding QoS-aware path for a flow is discussed in section V-D2.

B. PATH SELECTION POLICY

Regular expressions [33], [34] are one of the widely used choices by various frameworks for specifying packet processing policy. P4KP follows a similar approach and keeps the path open for future integration with those frameworks. The syntax for QoS-aware path-search queries in P4KP is shown in fig. 4. In this language, the QoS Aware Path Selection Criteria (C) part can be implemented using match-action-table in PISA switches. In the general system model shown in fig. 2, CriteriaMatcherMAT executes the matching. KPS represents policies supported by P4KP for selecting one-of-the-top-k best paths in this language. P4KP executes the task of selecting appropriate path based on policy.

P4KP supports six types of path-selection policies for selecting one of the paths from the top-k best paths (PathSet). Next, we describe how each of the six policies supported by P4KP can be represented using a bitmask. We used the network shown in fig. 1 and the four QoS level-based example of fig. 3 to explain the policies.

1) SELECT_BEST_PATH

Selects best (0-th) ranked path for a packet; if a valid path is not found in this rank, selects next best (1-st) ranked path and so on. To use this policy, set all the k bits of the packet’s policyMask to 1. For example, a packet is needed to be forwarded through the possible shortest delay path (policyMask = (1111)_b).

2) SELECT_n-th_RANKED_PATH

Selects the n-th (0 ≤ n < k) ranked path. It needs to set (n)-th bit to 1 in policyMask. This policy is used for selecting a path with a specific QoS bound. Example: a packet is needed to be forwarded through a path with delay in the range 41-60 ms (rank-2); the 2-th bit in policyMask is set to 1 (policyMask = (0100)_b).

3) SELECT_BEST_TO_n-th_RANKED_PATH

Selects best (0-th) ranked path; if a valid path is not found in this rank, select the next best (1-st) ranked path. Continue this process up to n-th rank (0 ≤ n < k). Need to set 1 in n + 1 bits in the policyMask, starting from 0-th to (n)-th bit. For example, a packet is needed to be forwarded through the possible shortest delay path, but the delay should not exceed 40ms (rank-1). The 0 and 1 bit in policyMask is set to 1 (policyMask = (0011)_b).

4) SELECT_n-th_TO_WORST_RANKED_PATH

Selects n-th ranked path; if a valid path is not found in this rank select next best (n + 1)-th ranked path. Continue this process up to (k − 1)-th rank (0 ≤ n < k). Need to set 1 in (K-n) bits in the policyMask, starting from n-th to (k − 1)-th bit. For example, a packet is needed to be forwarded through the possible shortest delay path, but it is not supposed to hamper the highest priority traffic that is using
the highest-ranked path (rank-0). To select this policy, 1st to 3rd bit policyMask is set to 1 (policyMask = (1110)_b).

5) SELECT_BEST_RANKED_FROM_EXCLUSIVE_PATHS
Selects a path from the best-rank out of p exclusively selected ranks. For example, assume a packet is needed to be forwarded through the possible shortest delay path, but rank-0 and rank-2 are already reserved for some high-priority traffic classes. Hence, rank-0 and 2 are excluded from the candidate set of paths (policyMask = (1010)_b).

6) SELECT_WORST_PATH
Selects the least priority path (rank-k - 1). For example, consider a flow has violated several traffic engineering constraints. Now all the packets from this flow are demoted and they are needed to be forwarded through the least priority path (policyMask = (1000)_b).

Selecting the appropriate rank for a policyMask is implemented through the use of a TCAM (RankFinderMAT) (section V-C). It is possible to enforce different policies by altering the priority of the entries in this MAT. For example, reversing priority of the entries will select the worst-ranked path in place of the best path. Instead of using only one TCAM for RankFinderMAT, parallelly using multiple TCAM to search multiple paths based on different policies, is also possible. But this approach needs extra information to select which MAT is to be used for rank selection. This step may consume an extra stage in the PISA pipeline.

Several frameworks [33], [35], [36] exist focusing on compiling packet forwarding policies to low-level programming languages like Openflow [37], P4 [38], etc. Their regular expression based languages can support P4KP’s path-selection policies and hide the low-level details of expressing the path-selection policy as the policyMask. We leave the goal of integrating P4KP with these frameworks as future research scope.

C. P4KP DATA STRUCTURE
At the core of P4KP is a data structure that spreads across 4 stages in the PISA pipeline.

1) STAGE-A
This stage contains a k-bit bitmask (PathSetMask) for a PathSet. The i-th bit in a PathSetMask corresponds to i-th ranked paths in the PathSet. The PathSet may not contain any path with rank i. The paths may be down; CP may decide to remove them as part of the update process (section V-D1). Or, in the case of QoS-based path ranking, no path can exist with respective QoS value. If i-th bit is 0, it indicates there is no valid path with rank i in the PathSet. And path with rank i exists if the bit is 1.

2) STAGE-B
This stage contains a k-bit wide TCAM based match-action-table (RankFinderMAT) for finding the rank of the desired path from PathSetMask based on a given k-bit policyMask. In this TCAM, k entries are inserted by CP during the bootstrapping process of the data plane program. Here, p-th (0 ≤ p ≤ (k - 1)) entry has p-th bit set to 1, and p is passed as parameter (pathRank) in its action. Assume, p and q-th entries have p and q-th bit set to 1; if p < q, then the p-th entry in the TCAM is assigned a higher priority than the q-th entry. This implies a lower-ranked path (p in this case) has a higher priority. During the path-search process, a packet’s policyMask is matched with RankFinderMAT. If it matches both the p and q-th entry, the TCAM will choose the entry corresponding to the p-th ranked path as it has a higher priority. Inserting entries in this TCAM is a one-time task, and the entries in this MAT remain unchanged over the whole lifetime of P4KP.

3) STAGE-C
This stage contains an array (RankMinMaxLocation) of k stateful memory cells. If P4KP is configured to store a total k × M paths of a PathSet in the PathStore, then D = ⌈log₂(k × M)⌉-bits are required for indexing them. Each cell in RankMinMaxLocation requires 2D-bits. The i-th (0 ≤ i ≤ (k - 1)) cell contains the starting (MinLocationIndex) and ending (MaxLocationIndex) location of all i ranked paths in the PathStore (stored in the next stage). Inside each cell, rightmost D bits represent MaxLocationIndex, and leftmost D bits represent MinLocationIndex. The MinLocationIndex for rank-i is initialized with M × i, and it remains fixed. On the other hand, MaxLocationIndex is also initialized with M × i but it changes with insertion and deletion (section V-D1) of a path in the rank. These variables are required to maintain the boundary between paths of two consecutive ranks in PathStore (v1model.p4 only supports a 1-D array of stateful memory/register).

4) STAGE-D
This stage contains an array (PathStore) of k × M stateful memory cells to store all the path information (port, pathID, etc.). Each cell is Q bits and able to store any of the 2^Q distinct paths toward a destination. In this array, all the paths with rank i are stored in location starting from M timesi to (M × (i + 1) - 1) (in no specific QoS based order). This array reserves M locations for rank-i path (0 ≤ i ≤ (k - 1)). But at a certain time, all M locations may not contain valid paths. The MaxLocationIndex and MinLocationIndex in the previous stage maintains the record of how many locations contain valid path information for each rank.

This multi-staged data structure is stored in the data plane. The P4KP control plane maintains a copy of the data structure in its own memory. Besides this, CP also keeps the record of a path’s current rank and index in the PathStore array. This information is stored in PathToLocationMap.

Example: Consider the 4 paths from L-1 to L-4 (through S-1 to S-4;) shown in fig. 1a. Each path’s delay can be in one of the 4 ranges (k = 4): 0-20ms, 21-40ms, 41-60ms and 61-80ms. Paths having a delay in the range of 0-20ms are assigned rank 0 (best path), 21-40ms are assigned rank 1
Updating the rank of a path \((X)\) in P4KP is a 2 step process. Assume \(X\)'s (path ID is \(pID\)) old rank was \(m\), and CP decided to update its rank to \(n\) \((0 \leq m, n \leq K - 1)\). At the first step, CP deletes \(X\) from rank \(m\) and then inserts it back to rank \(n\).

**Path-Deletion:** For deletion, the CP at first removes \(X\) from its own copy of the P4KP data structure. Deleting a path may need modification in three stages of P4KP data structure:

- a) If \(X\) is the last path with rank \(m\), then deleting \(X\) makes rank-\(m\) empty. To reflect that in the data structure, the \(m\)-th bit in \(PathSetMask\) is set 0; otherwise, the \(PathSetMask\) remains the same.

- b) Deleting \(X\) reduces the total number of the paths in rank-\(m\) by one. The \(MaxLocationIndex\) for rank-\(m\) is also reduced by one to reflect this reduction. Assume the current \(MaxLocationIndex\) for rank-\(m\) is \(OldMax\), and after reducing by one, the updated value is \(tempMax\). If \(X\) is the only path with rank-\(m\), this reduction is not needed. Because setting the \(m\)-th bit in \(PathSetMask\) to 0 (at step-a) rules out the scope of selecting any path (section V-D2) from rank-\(m\).

- c) Lastly, to remove the path from rank-\(m\), CP over-write \(X\)'s location in the \(PathStore\). Assume \(X\)'s current location is \(oldLoc\) (retrieved from \(PathToLocationMap\)). For deleting the path, the path ID \((pathAtOldMaxLoc)\) stored at \(OldMax\)-th location is written over at the \(oldLoc\)-th location in \(PathStore\). CP also updates \(pathAtOldMaxLoc\)'s location index in \(PathToLocationMap\) to \(oldLoc\).

After removing \(X\) from the internal copy of the data structure as described above, CP triggers the path-deletion in the P4KP data plane through sending a control message. CP encapsulates 5 information \((m, PathSetMask, tempMax, oldLoc and pathAtOldMaxLoc)\) in a control message (P4 Packet _In_) and send to DP. On receiving the control message, the DP replicates the process described above by writing the \(PathSetMask\) at stage-A, updating the \(m\)-th rank's \(MaxLocationIndex\) in stage-B with \(tempMax\) (writing \(tempMax\) at \((m - 1)\)-th location's leftmost \(D\) bits in \(RankMinMaxLocation\)) and writing \(pathAtOldMaxLoc\) at \(oldLoc\)-th index of \(PathStore\).

As example, consider the path 4 \((X = 4, pID = 4)\) shown in fig. 5 is to be deleted from rank-0. P4KP’s data structure after deleting path 4 is shown in fig. 6 (with the change highlighted). As shown in fig. 6a, removing path 4 does not empty the rank-0, and the \(PathSetMask\) remains the same (fig. 6c). Next, deleting path 4 reduces \(MaxLocationIndex\) for rank-0 by 1 and its new value is 0 (fig. 6d). And finally, the memory cell in \(PathStore\) containing path 4 is overwritten with 1 (rank-0’s \(pathAtOldMaxLoc\) (shown in fig.6e). The relevant control message sent by CP is also shown in fig.6b.

**Path-Insertion:** Inserting \(X\) in rank-\(n\) also needs modification in three stages of P4KP data structure.

- a) If there were no other path in rank-\(n\), then inserting \(X\) needs updating the \((n - 1)\)-th bit in \(PathSetMask\) to 1; otherwise, the \(PathSetMask\) remains the same.

- b) Inserting \(X\) increases the total number of paths (up to \(M\) paths per rank) with rank-\(n\). Hence, the \(maxLocationIndex\) for rank-\(n\) is increased by one. If \(X\) is the only member in
FIGURE 8. Steps for searching path for packet P with policyMask = 1010.

this rank, then the increase is not needed. In such cases, both maximum and minimum (minimum always remain fixed to maintain the boundary between 2 consecutive ranks) location index in RankMinMaxLocation contains same index.

c) The pID is written in the updated maxLocationIndex of PathStore. Besides this, CP updates the path’s rank and location in PathToLocationMap array for X with n and maxLocationIndex.

After updating its own internal copy of the data structure, CP sends a control message to DP with four information: (n, PathSetMask, maxLocationIndex, pID). Similar to deletion process, on receiving the control message, DP writes PathSetMask at stage-A, maxLocationIndex at (n-1)-th location’s left most D bits in RankMinMaxLocation in stage-B and finally writes pID at maxLocationIndex index of PathStore.

Fig. 7 shows the steps of inserting path 4 (X = 4, pID = 4) in P4KP’s data structure shows in fig. 6 (changes in the data structure are highlighted). As shown in fig. 7a, path 4 is the first path with rank-3, and the 3-rd bit in PathSetMask changed to 1 (fig. 7c). Next, inserting path 4 does not changes the MaxLocationIndex for rank-0 as it is the only path in this rank. Fig. 7e shows the status of PathStore after writing pID = 4 at 6’th location, and 7b shows the CP’s control message for inserting path 4 to rank 0.

• Default Path: While updating the paths in a PathSet, all the ranks can become empty (only two paths available in the PathSet, among them, one path is temporarily down, and another path is undergoing an update). To handle such a situation, the control plane can insert a default rule in the RankFinderMAT, and a default path will be selected when no active path is available in the PathSet.

2) PATH SEARCH

After a packet P is matched in CriteriaMatcherMAT corresponding k-bit policyMask and index for the PathSet is stored (as the action of the MAT) in its metadata (P.metadata.policiyMask and P.metadata.pathSetID). (Here, we omit discussion about P.metadata.pathSetID as explained earlier in section IV). Then the P.metadata.policyMask is used to search the packet’s appropriate path from P4KP’s data structure.

Path searching for a packet works as following:

At first, PathSetMask is loaded into tempMask from stateful memory. Next, tempMask is bitwise ANDed with P’s policyMask. If the resultant bitmask (tempMask) contains 1 in r bits, it implies among the available k set (ranked form 0 to K-1) of paths r set of paths are eligible for the destination. From these x set of paths, the highest priority ranked set is selected through RankFinderMAT.

For this, the tempMask is matched with the RankFinderMAT and corresponding path rank is stored in the packet metadata as P.metadata.pathRank if a matching entry is found. After that, the MaxLocationIndex and MinLocationIndex for P.metadata.pathRank-th ranked paths are loaded from (r-1)-th entry of RankMinMaxLocation. In the next stage, an index (pathLocation) between these 2 indexes are chosen based on the hashcode of the packet’s 5 tuple (random, round-robin, etc. policies also can be used). Finally, the pathLocation-th location entry is loaded from the PathStore. It gives the desired path for the packet.

As an example, consider we want to find the path for a packet P from the PathSet shown in fig. 7. Assume, the highest priority paths (rank-0: with 0-20 ms delay) are reserved for flows of other high priority traffic class and path with delay 41-60 ms (rank-2) is also not permitted for the flow due to some security reason. As both rank-0 and rank-2 are prohibited for the packet, its policyMask become 1010x. Steps for finding the appropriate path for this packet is shown in fig. 8.

E. ONE PATH PER RANK

Instead of maintaining M paths per rank, P4KP can be configured to store only one path per rank (M = 1). As only one path per rank is accommodated, a rank can contain either no path or a single valid path. Whether any path exists in a rank or not can be solely represented by the PathSetMask. Therefore the stage-c (fig. 5) in the data structure is not needed anymore. Moreover, it also establishes a direct one-to-one map between a rank r and the corresponding location in the PathStore. The rank of the appropriate path provided by the RankFinderMat gives the location of the desired path in PathStore. It also removes the necessity of hash-based calculation (stage 5 in fig. 8) to select one of the paths from rank r. This reduces the total number of stages required by P4KP to four (stage 4 and 5 in fig. 8 is not required anymore).

VI. PERFORMANCE & HARDWARE COST

A. PERFORMANCE

All the operations of P4KP discussed in section V-D require only one memory operation at each stage and one computation on only one field in the packet header. Hence P4KP is completely realizable at line rate in currently available PISA switches.
B. HARDWARE COST
As shown in fig. 8, searching path for a packet needs six stages. If instead of a group of \( M \) paths per rank, only one path per-rank is supported, then stages 4 and 5 shown in fig. 8 are no more necessary. Hence P4KP can be implemented using only four stages. This is small compared to the total number of stages [6] in currently available PISA switches.

The only bottleneck in P4KP is the number of bits used in PathSetMask and RankFinderMat. Currently, available PISA hardware has a \( M = 112 \) bit wide memory port and up to \( T = 640 \)-bit wide TCAM. Hence top-112 paths can be supported by P4KP using only one stage for PathSetMask and one stage for RankFinderMat. It can be extended up to top-640-paths using six stages for PathSetMask.

C. IMPROVING PATH-UPDATE COST
When \( M \) paths per rank are supported, updating a single path needs two control messages (section V-D1). But in the case of only one path per rank, the path deletion process is not necessary anymore V-E. The control plane can set (step a in path-insertion) and unset (step a in path-deletion) the 2 bits in the PathSetMask and send it to in the control message used for path-insertion. Hence, a path can be updated using only one control message leading to total \( k \) path control messages for updating the top-\( k \) best paths. This number can be reduced further using batch updates. If \( P \) bits are required for storing a path ID, then \( M \)-bit wide memory port can update \( \lceil M/P \rceil \) path information using one control message. This leads to a total of \( \lceil k/(M/P) \rceil \) control messages required for updating the top-\( k \) best paths.

D. MAXIMUM NUMBER OF PATH \((k)\) PER PATHSET SUPPORTED
P4KP can store top-\( k \) best paths for a path-search rule and it requires \( k \) bit memory for storing the path-selection policy (PolicyMask). Now, the PolicyMask and PathSetID are configured by the control plane as the action part of the CriteriaMatcherMAT. These two information are stored in SRAM, and they can share the SRAM in a single stage. In a PISA switch with \( M \)-bit wide memory port, if \( P \) bits are used for PathSetID then \( M – P \) bits can be allocated for the PolicyMask. Consider a PISA switch [6] with 112-bit wide memory port; if \( R = 20 \) bits are allocated for PathSetID, then 92-bits can be allocated for the PolicyMask. A 20-bit space can accommodate \( 2^{20} = 1048576 \) different PathSetID, which is enough for maintaining unique index for path between each ToR switch pair in a large scale data center network. If the PolicyMask and PathSetID are not stored in same stage in a PISA pipeline, another table can be used to store the mapping between these 2 information. This approach can increase the value of \( k \) even more. For large-scale networks, more than 92 paths per destination may not be a practical solution. As an example, consider a DCN with 10K ToR switches and top-92 paths are monitored at 50ms probe interval. It will require more than 18 million probe packets per second.

So many probe packets will cost huge bandwidth, and processing large number of monitoring packets in the control plane will also consume a large amount of processing power and time. Therefore, such kind of system may not be used in real life.

E. STATEFUL MEMORY CONSUMPTION
Assume top-\( k \) best paths (with \( M \) paths per rank) are supported in a PathSet, and \( Q \)-bits are required for storing a pathID. The total stateful memory resource required by different components of P4KP’s data structure (fig. 5) is following:

- \( k \) bits SRAM to store the PathSetMask in stage-A.
- \( a \times k \) TCAM for the RankFinderMAT and \( k \) entries in SRAM to store the corresponding PathRank in stage-B
- \( 2 \times D \times k \) bits SRAM to store the RankMinMaxLocation in stage-C
- If \( M \) paths per rank is stored, then total \( k \times M \times Q \) bits SRAM to store the PathIDs in PathStore in stage-D

F. MAXIMUM NUMBER OF PATH\(_{SET}\) SUPPORTED
P4KP does not impose any direct constraint on the maximum number of supported PathSets. As discussed in the previous paragraph, the stateful memory requirements of PathStore is the main deciding factor behind P4KP’s capacity. It is limited by the stateful memory capacity of each stage in PISA hardware. If MemoryCapacity bits of SRAM is available in a single stage of a PISA switch, it can accommodate total \( MemoryCapacity/(k \times M \times Q) \) number of PathSet in a single stage. Allocating \( S \) stages for PathStore increases this \( S \) times.

As a concrete configuration example, consider the case of a data center network with 10K ToR switches. Assume \( Q = 16 \) bits are allocated to represent a path ID (16 bits can accommodate unique ID for 65K paths), and only one stage is allocated for storing the PathStore. Using this configuration, \( k = 54 ((106 \times 1024 \times 5)/(10000) = 54) \) paths between each ToR switch pair can be maintained by P4KP using only one stage for the PathStore (one path per rank) in the PISA switches proposed in [6]. Over the period, PISA hardware’s stateful memory capacity has increased by 4-5x times [39]. This increases P4KP’s capability even more.

VII. DISCUSSION
A. HANDLING OUT-OF-ORDER PATH SELECTION
In multiple paths per rank configuration, a transient state can occur between deleting a path from the old rank and inserting it into a new rank. The path will not be in the P4KP data structure during this period, and no packet can be forwarded through the path. (this problem is common to any other TCAM or SRAM based path forwarding schemes)

To reduce the duration of transient configuration, control messages for insertion and deletion can be concatenated into the same control packet. On receiving, the control message, the P4KP data-plane will delete the path from its old rank using the first part of the control message. Then the packet
can be recirculated, and the second half of the packet will be used for inserting the path in its new rank. A flag can be maintained in the control message header to indicate which operation (path deletion or insertion) will occur and which part of the control message to use for the indicated operation. Besides this, keeping two copies of the P4KP data structure can also avoid this problem. The first copy will contain the old path configurations in this scheme, and all updates will be applied to the second copy. Once the path update process is complete, the forwarding path for the packets are selected from the second copy of the P4KP data structure.

During the transient configuration period, P4KP can select a path for a packet based on the old ranks of the paths and use a new path once the path ranks are updated. This can increase the number of out-of-order packet delivery to the destination of a flow. To reduce the chance of packet reordering, P4KP can be integrated with flowlet switching [40], where the path used for forwarding packets from a flow is kept fixed for a predefined time (flowlet interval). Usually, this time is large enough to reduce the probability of packet reordering (at Tbps speed, the flowlet interval is greater than the transient configuration period). Updating the forwarding path for a flow at flowlet granularity and selecting a new path at the start of a new flowlet interval can reduce out-of-order delivery of packets caused by the transient state of P4KP.

B. INTEGRATING WITH SINGLE TCAM BASED PACKET FORWARDING SCHEMES

P4KP can be configured to work with any single TCAM based packet-forwarding algorithm. In reconfigurable match-action-table [6] based PISA switches, on finding a match with the packet header fields, a path is selected for the packet through executing an action (section III). These switches can be configured to select different actions based on the matching header fields. Leveraging this feature, the CriteriaMatcherMAT used in QoS-aware packet forwarding systems can be configured to execute actions for P4KP (example is discussed in section V-A) for some of the the match-action-table entries. This same match-action-table can also be configured to directly select the forwarding path in one logical step through the actions. It allows the network operators to apply either P4KP based policy or any other customized packet forwarding policy for different types of workloads in the same switch.

C. P4KP IN VIRTUAL NETWORKING

Data center networks implement QoS, network isolation for multi-tenancy, access control, etc., through virtual networking. Various tunneling protocols (e.g., VXLAN [27], GRE [28], etc.) are designed for creating overlay networks over the substrate networks. Hypervisor switches implement these protocols and encapsulate the packets received from the tenants according to these protocols before forwarding to the switches. The network administrator of a multi-tenant data center network can provision different QoS policies for each of the tenants by configuring different PolicyMask in the CriteriaMatcherMAT (section V). Alternatively, centralized or distributed admission control modules [21], [41] can be deployed, and the tenants can negotiate the QoS policy with these modules. After the initial negotiations, hypervisors can tag the packets [7] with PolicyMask according to their QoS policies. The programmable switches can interpret the encapsulated packet according to the tunneling protocol formats. After that, the flows are treated based on the virtual network identifier (VNI) [42] in the packet and the administrator configured QoS profiles using P4KP.

For example, assume one of the tenant’s QoS profiles demands a dedicated path between two ToR switches. All packets from the tenant can be marked by the hypervisor with corresponding virtual network identifier (VNI), and a rule for reserving the path will be inserted in the CriteriaMatcherMAT of a switch by the control plane. The control plane will also mark the path as excluded (select_best_ranked_from_exclusive_paths policy in section V-B) for other tenants or flows. This will ensure the path is not shared by flow from other tenants.

The tunneling protocols used for overlay networking also provide extension mechanisms, which can be used to add customized QoS features. Recent works [43], [44] provide functionality for customized extension in the virtual switches used in the hypervisors. Together with the programmable switches, they can implement different customized classes of QoS policy and P4KP to provide the necessary QoS aware forwarding capability in the switches.

One of the main challenges in overlay networks is, QoS properties of virtual links can change very quickly due to changes in substrate links or other traffic running over the same substrate link. The controller needs to recompute paths and push forwarding table entries to each host over the same substrate network to cope with these changes. With the increase in network size, the fast-changing nature of the data center substrate network can trigger pushing many [7], [45] updated forwarding table entries. Propagating them to the hypervisors at a fine time scale is challenging, and they can create traffic bursts in the substrate network switches. Besides this, TCAMS are costly and power-hungry; hence, the amount of TCAM memory available in switches is limited. It brings a limit on the total number of forwarding table entries in the switch for overlay networks. All these factors limit the size of overlay networks.

Using P4KP can reduce these overheads by avoiding the use of TCAM for QoS-aware path selection in the switches. In the P4KP centric workflow, the control plane can advertise (either periodically or on receiving a request from the hypervisors) the QoS-based ranks supported by a switch. On finding changes in substrate network conditions, the P4KP enabled control plane will recompute the paths and update them in the data plane. The hypervisor switches will use the QoS ranks advertised by the control plane to generate the PolicyMask and add it to the packets. On receiving the packets, the P4KP enabled switch will forward the packets based on their PolicyMask. In this scheme, a controller does
not need to propagate updated forwarding table entries each time to the hypervisors. The controller only updates the path and their ranks in the P4KP data plane. It will advertise only the updated rank information to the hypervisors. It can reduce costly bandwidth consumption for propagating updated forwarding table entries to the hypervisors. On the other hand, storing forwarding table entries in SRAM (instead of TCAM) increases a switch’s ability to store the maximum number of forwarding table entries.

D. POTENTIAL USAGE IN APPLICATION OBJECTIVE-AWARE NETWORK INTERFACE (ANI)

Implementing ANI is an important requirement for future application-aware networking [3], [46]. P4KP can be used to express the application’s QoS objectives in a manner that is actionable to the network switches. Consider the case of a multipath TCP communication where different subflows of a TCP connection can follow different paths toward the destination. As these paths can have a different end-to-end delay, forwarding packets of a subflow through a path with the same end-to-end delay is important to reduce packet reordering. In this case, packets of a subflow can be tagged with their required end-to-end delay, and the switches can map these requirements to the ranks of the paths used in P4KP. Alternatively, the approach discussed in section VII-C also can be followed, and the hypervisors can tag the packets with PolicyMask representing the end-to-end delay requirement of the subflow. Finally, the switches can forward the packets through the specific ranked path.

VIII. EVALUATION

Virtual networking is the key mechanism used in data center environments to achieve QoS. At first, we evaluate P4KP’s ability to correctly select QoS-aware paths in a virtualized environment. Then using a real-life data center workload, we evaluate how the use of P4KP leads to performance improvement. We implemented P4KP’s data plane program using P4_16 programming language [11]. The P4KP control plane program was implemented using the Python programming language. A modified version of P4RuntimeShell [48] (it uses P4Runtime [49] API) was used for communication between the data and control plane. We have evaluated the prototype on a software switch behavioral model version 2 (BMv2) [50] of P4_16 and Mininet [51] based test-bed. We used the v1model.p4 (most commonly available architecture of PISA switches) [52] as the reference target architecture of PISA switches. All experiments were conducted on an HP laptop with 6 Intel Core i7-9750H (all cores configured at 2.60GHz), 24 GB RAM, running Ubuntu 20.04.

A. TEST-BED DESCRIPTION

We simulated a two-tier fat-tree topology (a.k.a. leaf-spine topology) based virtual network over a substrate network in the test-bed (fig. 9). As the focus of this work is on the QoS aware packet forwarding, we did not consider the end host virtualization and virtual network embedding algorithm here. The virtual network was statically mapped to the substrate network.

The QoS performances of a virtual network can change due to: a change in QoS characteristics of substrate links due to background Internet traffic, substrate link failure [53], change in the virtual network to substrate network mapping [54] for optimization purposes, etc. Currently, there is no openly available dataset existing in the literature for modeling the QoS performance of virtual networks. Moreover, extensive simulation of virtual network link performance goes beyond the scope of this work. Hence, like other existing works in the domain [55] we simulated the change in virtual network performance by assigning different bandwidth capacities to the virtual links. Change in virtual network link speed was modeled using a discrete event generator. The time gap between two successive changes in link bandwidth capacity and the link capacities were assigned using different strategies (described in section VIII-B and VIII-C) to evaluate P4KP’s performance in different scenarios.

B. ACCURACY AND SCALABILITY

1) TRAFFIC DESIGN AND MEASUREMENT

Here, we experimentally evaluate P4KP’s capability to select QoS aware path and compare it with the fully TCAM-based scheme (as the baseline version). We also compare the scalability of both schemes. In the test-bed, we started three flows from a host (H-1) connected with the switch L-1 to three different hosts connected to L-2, L-3, and L-4. All three flows

---

1 The P4KP prototype is available as an open-source project [47].
flows were configured to transfer 2MB of data using 1500B TCP segment size (BMv2 supports a maximum 1500B packet size). They were tagged with three different traffic classes with the following QoS requirements: a) traffic class-0 × 14: needs to forward packets through the highest capacity path (best-ranked path). b) traffic class-0 × 0A: needs to forward packets through a path with at least 6 packets per second rate (specific QoS bounded path selection policy). (BMv2 allows configuring a link capacity in terms of packets per second processing rate (pps). Hence, instead of using in bits/sec. we used pps as the unit of measurement.) In this case, we configured it to use rank-2 path and c) traffic class-0 × 12: needs to forward traffic through other paths to not hamper the other two high-priority flows. All three flows pass through the virtual switch L-1 and transfer packets to end hosts connected with other leaf switches. Therefore any change in L-1’s upward links performance impacts the desired QoS performance of the flows.

To simulate changing network behavior and corresponding monitoring results, we created a random event generator that replicates a monitoring component’s behavior and reports different remaining capacities for L-1’s upward links at every 0.5 seconds interval. Among the four links, it reports one of the randomly selected ports as down. For the other three ports, it reports three fixed bandwidth capacities: 4, 6 and, 10 pps (packet per second), and the remaining one port is configured as ‘down’. But at every interval, a link is assigned a different bandwidth capacity from these three. All the links were arranged in four ranks (k = 4 ranks and one path per rank M = 1) as following: a) rank-0: link capacity of 0-10 pps, b) rank-1: link capacity of 0-6 pps, c) rank-2: link capacity of 0-4 pps, and d) rank-3: all other links. The P4KP control-plane reacts on these emulated monitoring reports and configures the port in the data plane according to their ranks. It also removes the link configured as ‘down’ from the data structure (in the P4KP data-plane). If the path is up again in the next interval, the P4KP control-plane reinserts the path and assigns it the corresponding rank. All other switch’s bandwidth capacity was set to a large value not to create any bottleneck link for any of the flow.

In the QoS-aware forwarding algorithm, rank-0,1 and 2 were assigned to forward packets from traffic class 0 × 14, 0xA and 0 × 12 to meet their QoS requirements. Here we assumed that the traffic class to link capacity-based rank is negotiated between the controller and the end hosts using any existing admission control systems [21]. The hypervisor tags each packet with their desired throughput rate (in case of range match-based TCAM) and PolicyMask (in case of P4KP). The forwarding tables of range match based TCAM and the QoS-aware forwarding algorithm that uses P4KP are shown in fig. 10.

2) ACCURACY ANALYSIS
Now, the range match-based TCAM always explicitly selects a path with desired link capacity. Therefore, packets from the flow tagged with traffic class 0 × 14 should always be forwarded through the path with link bandwidth 10 pps (rank-0) and utilize the full link capacity. Similarly, packets from the flow tagged with traffic class 0 × 0A and 0 × 12 should always be forwarded through the path with link bandwidth 6 pps (rank-0) and 4 pps (rank-2 path). Ideally, all three flows should achieve the throughput rate equal to the QoS value of their assigned path (link bandwidth). If the P4KP works properly, both TCAM and P4KP based systems should achieve similar throughput for all three flows. To verify this, we compared the TCP throughput achieved by these three flows under both schemes. We converted the bitrate to packets per second as the link bandwidths were configured at a unit of packets per second. Fig. 11 shows each flow’s TCP throughput when P4KP and TCAM based scheme is used and compare with the expected ideal throughput. It shows that all three flows achieve near ideal throughput under both range match-based TCAM and P4KP. Minor deviation from the ideal throughput can be attributed to wrong path selection by the schemes while the control plane updates the paths in the data plane. This behavior shows P4KP’s ability to select QoS aware path correctly and match the QoS performance provided by range match based TCAM.

3) STATEFUL MEMORY CONSUMPTION ANALYSIS
The range match based TCAM (fig. 10a) needs to store k (k = 4 in the example) link capacity based range rules for top-k destinations. In the worst-case scenario, a range based rule defined over a W-bit (W = 4 in fig. 10a) range-match field (used for link capacity demand) and L bit destination prefix (for IPv6 address L = 128) requires W TCAM entries [10], [56]; where each entry is W + L bits. For P destinations and k range rules, total k × W × P TCAM entries and k × P × Q SRAM entries to store corresponding ports as actions [6] are required; here, Q = total bits required to store a path (port) information. Whereas the P4KP based scheme of fig. 10b requires P entries of L bit width in TCAM for the destination lookup table and P SRAM entries to store the P bit PathSetID for each destination. Besides this, k entries in a k bit wide TCAM to store the RankFinderMAT and k × P+kk+D+2 × k × P × D+k × P × M × Q bit (here, M = 1,
TABLE 1. Stateful memory (in KB) required by range match TCAM and P4KP based schemes.

| Total Destinations (P) | Range Match TCAM | P4KP |
|------------------------|------------------|------|
|                        | TCAM | SRAM | TCAM | SRAM |
| 1024                   | 1056 | 20   | 16.03| 40.5 |
| 2048                   | 2112 | 40   | 32.03| 81   |
| 4096                   | 4224 | 80   | 64.03| 162  |
| 8192                   | 8448 | 160  | 128.03| 324 |
| 10240                  | 10560| 200  | 160.03| 405 |

because one path per rank is stored, and $D = [\log_2(k)]$ bits are required to index locations for $k$ paths) bit space in SRAM are required (section VI). We have simulated the TCAM and SRAM storage consumption by P4KP and range match based TCAM for different values of $k$ and $D$ using a Python script (BMv2 implements the range match-based tables using simple C++ data structures which do not reflect the actual storage consumption in a TCAM. Moreover, BMv2 does not provide any mechanism to measure total TCAM and SRAM consumption by a P4 program). Table 1 shows the total TCAM and SRAM memory consumption by these two schemes for selecting a QoS based path from top-$16$ paths for a different number of destinations. Here, we assumed that, $R = 20$ bits are used for uniquely indexing each PathSet, $Q = 10$ bits are used for uniquely identifying a path in each PathSet. The table shows that, for selecting a QoS aware path, P4KP uses a much smaller amount (only 2%) of TCAM memory (columns 3 and 4) and two times SRAM memory (columns 3 and 5) compared to the fully TCAM based scheme. Compared to the SRAMs, TCAMs consume approximately two times more circuit area and six times more power. Therefore P4KP provides a cheaper scheme for top-$k$ QoS aware path management and selection.

4) SCALABILITY ANALYSIS
The total amount available stateful memory in PISA switches is limited. This limitation restricts the maximum number of rules allowed for QoS-aware routing in a switch. To compare the scalability of the two schemes, we benchmarked the total number of range-based rules supported by them on the PISA switch described in [6]. In the fully TCAM based scheme, each rule (128b IPv6 address of 128b prefix length combined with $W = 4b$ range match field) in the forwarding table of fig. 10a requires four entries in a 160b wide TCAM block, and each stage of the PISA hardware can accommodate 8K of these entries (2K rules). In the P4KP based scheme, the Destination Lookup Table of fig. 10b also needs to store a 128b entry for each destination IPv6 prefix, and these rules have no range match field. Hence each rule needs one entry in a 160b wide TCAM block, and each stage of the PISA hardware can accommodate 8K of these entries (8k rules also). Besides this, the PathStore (stage-D in fig. 5) used in the P4KP Data Structure (fig. 10b) requires $Q = 10$ bits for each rule in SRAM, and each stage can accommodate 848K of these entries. In the PISA switches, all the TCAM based tables and SRAM based data structures can be expanded to multiple stages to accommodate a large number of rules. We allocated the same number of stages ($S$) to store the forwarding table for range match TCAM and the data structures (along with the Destination Lookup Table) for P4KP based scheme. Fig. 12 compares the total number of destinations with information for their top-$16$ paths (defined over $W = 4$ bit range match field and able to address maximum top-$k$($= 16$) paths) supported by both the schemes. The P4KP based scheme needs at least 4 stages in PISA switch; hence it cannot perform QoS aware routing when 1,2, or 3 stages are allocated. But starting from 4 stages, it completely outperforms the fully TCAM based scheme and provides a high level of scalability.

C. PERFORMANCE IMPROVEMENTS
1) TRAFFIC DESIGN AND MEASUREMENT
Now, we evaluate how P4KP’s QoS aware packet forwarding feature leads to performance improvements (in terms of flow completion time) in a virtual network. Here we simulated the real-life web search workload [57] observed in production data center networks over the leaf-spine topology-based virtual network of fig. 9. The workload contains a diverse mix of small and large flows, with more than 90% of the bytes, are carried by large flows. Packet from each type of flows was tagged with different traffic classes in their headers. In the simulation, the source and destinations of the flows were selected according to stride pattern, and the flow arrival rates were selected from Poisson distribution to obtain the desired level of load in the virtual network. The host-to-leaf and Leaf-to-spine links of the substrate links were configured with 64 packets/sec. and 32 packets/sec. bandwidth to achieve a 2:1 oversubscription ratio. The bandwidth demand of the virtual links was configured with 16 pps.

To simulate the change in QoS performance of the virtual links, we followed existing works in literature [55], [58] and varied the bandwidth capacity of the virtual links between 40%-90% (selected using uniform probability) of their original demands. The interval between two successive changes was selected from an exponential distribution. We have tested three different packet forwarding schemes: a) ECMP which is QoS unaware and forwards a packet based on its flow hash code b) HULA, which uses path utilization to select the best path (least utilized) for a packet and c) P4KP, which arranges the paths in three ranks based on their bandwidth allocation.
FIGURE 13. P4KP based scheme achieves improved average flow completion time.

D. D. Robin, J. I. Khan: P4KP: QoS-Aware Top-K Best Path Using Programmable Switch

to demand ratio: \( \geq 60\% \) rank 0, \( \geq 45\% \) rank-1, and the rest are rank-2. At the beginning of each interval, the P4KP control plane updates the paths in the data plane according to their ranks. Then we implemented P4KP based packet forwarding scheme where large flows were forwarded using select\_best\_path policy; the rest of the flows were divided into two categories based on their flow size and forwarded using select\_kth\_ranked\_path \((k = 1, 2)\). The range match TCAM based scheme requires many TCAM entries for a large number of destinations. Hence it is not suitable for practical use and we have not evaluated it here.

2) RESULT ANALYSIS

Fig. 13 shows that P4KP achieves 4-11% shorter average flow completion time compared to ECMP. Because ECMP is agnostic to change in the virtual to substrate link mappings and unable to consider the variations in capacities of the virtual links. In comparison, P4KP ranks the virtual links according to their allocated capacity over the substrate link and makes link capacity aware decision making for packet forwarding. HULA can indirectly identify the change in virtual link capacities as it monitors the utilization of the links. However, it only considers a single best path toward a destination, making the single best path overutilized. As a result, other paths remain underutilized, and HULA does not perform well. At different workloads, P4KP achieves 21-30% improved average flow completion time compared to HULA.

IX. CONCLUSION

We have presented P4KP, a scheme for organizing top-\(k\) best paths for a path-search query and selecting a path from them based on a QoS-aware policy. P4KP relies on cheaper SRAM for storing the top-\(k\) best paths for a path-search query and reduces costly TCAM usage. It provides a scalable QoS aware path-search and selection mechanism that can be used as a building block for advanced routing algorithms. Moreover, P4KP’s control plane-centric path-update scheme can efficiently update the path’s rank in the data plane using a small number of control messages. P4KP’s configurable nature allows varying the number of paths to consider for a path-search rule and makes it suitable for use in a wide range of applications. P4KP’s simple but powerful bitmask-based mechanism to express path-selection policy can express diverse application’s QoS requirements. Currently, P4KP supports six types of path-selection policies that can express a wide range of QoS-aware forwarding algorithms. Moreover, it provides the scope for integration with existing frameworks for path-query compilation. In the future, we plan to work in this direction and supporting other path-selection policies (load-balancing, security-aware, application-state aware, etc.) too.

REFERENCES

[1] C. Raiciu, S. Barre, C. Phuntke, A. Greenhalgh, D. Wischik, and M. Allweyer, “Improving data center performance and robustness with multipath TCP,” ACM SIGCOMM Comput. Commun. Rev., vol. 41, no. 4, pp. 266–277, 2011.
[2] Q. De Coninck and O. Bonaventure, “Multipath QUIC: Design and evaluation,” in Proc. 13th Int. Conf. Emerg. Netw. Experiments Technol., Nov. 2017, pp. 160–166.
[3] S. A. Iyotthi, S. H. Hashemi, R. Campbell, and B. Godfrey, “Towards an application objective-aware network interface,” in Proc. 12th USENIX Workshop Hot Topics Cloud Comput. (HotCloud), 2020, pp. 1–8.
[4] Q. De Coninck, F. Michel, M. Piraux, F. Rochet, T. Given-Wilson, A. Legay, O. Pereira, and O. Bonaventure, “Plugmizing QUIC,” in Proc. ACM Special Interest Group Data Commun., Aug. 2019, pp. 59–74.
[5] A. Bujari, C. E. Paluzzi, G. Quadrito, and D. Ronzani, “Emerging interactive applications over QUIC,” in Proc. IEEE 17th Annu. Consum. Commun. Netw. Conf. (CCNC), Jan. 2020, pp. 1–4.
[6] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izard, F. Muija, and M. Horowitz, “Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN,” ACM SIGCOMM Comput. Commun. Rev., vol. 43, no. 4, pp. 99–110, Oct. 2013.
[7] N. Varyani, Z.-L. Zhang, and D. Dai, “QROUTE: An efficient quality of service (QoS) routing scheme for software-defined overlay networks,” IEEE Access, vol. 8, pp. 104109–104126, 2020.
[8] N. Varyani, Z.-L. Zhang, M. Rangachari, and D. Dai, “LADEQ: A fast Lagrangian relaxation based algorithm for destination-based QoS routing,” in Proc. IFIP/IEEE Symp. Integ. Netw. Service Manage. (IM), Apr. 2019, pp. 462–468.
[9] D. Bhat, J. Anderson, P. Ruth, M. Zink, and K. Keay, “Application-based QoS support with P4 and OpenFlow,” in Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS), Apr. 2019, pp. 817–823.
[10] O. Rottenstreich, R. Cohen, D. Raz, and I. Keslassy, “Exact worst case TCAM rule expansion,” IEEE Trans. Netw., vol. 62, no. 6, pp. 1127–1140, Jun. 2012.
[11] The P4 Language Consortium. P416 Language Specification. [Online]. Available: https://p4lang.github.io/p4-spec/docs/P4-16-working-spec.html
[12] B. Agrawal and T. Sherwood, “Ternary CAM power and delay model: Extensions and uses,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 16, no. 5, pp. 554–564, May 2008.
[13] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, “HULA: Scalable load balancing using programmable data planes,” in Proc. Symp. SDN Res., Mar. 2016, pp. 1–12.
[14] A. Sivaraman, C. Kim, R. Krishnamoorthy, A. Dixit, and M. Budiu, “DC-p4: Programming the forwarding plane of a data-center switch,” in Proc. 1st ACM SIGCOMM Symp. Softw. Defined Netw. Res., Jun. 2015, pp. 1–8.
[15] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, “Hedera: Dynamic flow scheduling for data center networks,” in Proc. NSDI, San Jose, CA, USA, 2010, vol. 10, no. 8, pp. 89–92.
[16] P. Kumar, Y. Yuan, C. Yu, N. Foster, R. Kleineberg, P. Lapukhov, C. L. Lim, and R. Soulé, “Semi-oblivious traffic engineering: The road not taken,” in Proc. 15th USENIX Symp. Netw. Syst. Design Implement. (NSDI), 2018, pp. 157–170.
[17] T. Benson, A. Anand, A. Akella, and M. Zhang, “MicroTE: Fine grained traffic engineering for data centers,” in Proc. 7th Conf. Emerg. Netw. Experiments Technol. (CoNEXT), 2011, pp. 1–12.
[18] S. Ghobrani, B. Godfrey, Y. Ganjali, and A. Firoozshahian, “Micro load balancing in data centers with DRILL,” in Proc. 14th ACM Workshop Hot Topics Netw., Nov. 2015, pp. 1–7.
[19] L. Wang, F. Zhang, J. A. Aroca, A. V. Vasilakos, K. Zheng, C. Hou, D. Li, and Z. Liu, “GreenDCN: A general framework for achieving energy efficiency in data center networks,” IEEE J. Sel. Areas Commun., vol. 32, no. 1, pp. 4–15, Jan. 2013.

[20] K.-F. Hu, R. Beckett, A. Chen, J. Rexford, and D. Walker, “Contra: A programmable system for performance-aware routing,” in Proc. 17th USENIX Symp. Syst. Design Implement. (NVD), 2020, pp. 701–721.

[21] M. F. Bara, S. R. Chowdhury, R. Ahmed, and R. Boutaba, “Policy-Cop: An autonomic QoS policy enforcement framework for software defined networks,” in Proc. IEEE SDN Future Netw. Services (SDN FNS), 2013, pp. 149–156.

[22] C. H. Benet, A. J. Kassler, T. Benson, and G. Pongracz, “MP-HULA: Multipath transport aware load balancing using programmable data planes,” in Proc. Morning Workshop Network Comput., Aug. 2018, pp. 7–13.

[23] K.-F. Hu, P. Tammana, R. Beckett, A. Chen, J. Rexford, and D. Walker, “Adaptive weighted traffic splitting in programmable data planes,” in Proc. NVD, Mar. 2020, pp. 103–109.

[24] S. Hu, K. Chen, H. Wu, B. Cai, L. Han, W. Hao, Z. Zhao, and C. Guo, “Explicit path control in commodity data centers: Design and applications,” in Proc. 12th USENIX Symp. Syst. Design Implement. (NVD), 2015, pp. 15–28.

[25] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, S. Hu, K. Chen, H. Wu, W. Bai, C. Lan, H. Wang, H. Zhao, and C. Guo, “Optimal in/out TCAM encodings of ranges,” in Proc. 11th USENIX Symp. Syst. Design Implement. (NVD), 2010, pp. 113–126.

[26] M. Mahalingam, D. G. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, Virtual Extensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks Over Layer 3 Networks, RFC 7348, 2014, pp. 1–22.

[27] D. Parinci, T. Li, S. Hanks, D. Meyer, and P. Traina, Generic Routing Encapsulation (GRE), document RFC 2784, Mar. 2000.

[28] Y. Rekhter and T. Li, A Border Gateway Protocol 4 (BGP-4), document RFC1771, 1994.

[29] J. Moy, OSPF Version 2, document RFC2328, 1998.

[30] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, and A. Kanagal, “Jupiter rising: A decade of clos topologies and centralized control in Google’s datacenter network,” ACM SIGCOMM Comput. Commun. Rev., vol. 45, no. 4, pp. 183–197, 2015.

[31] N. Bitar, S. Gringeri, and T. J. Xia, “Technologies and protocols for data center and cloud networking,” IEEE Commun. Mag., vol. 51, no. 9, pp. 24–31, Sep. 2013.

[32] R. Jap, S. Bonn, R. Ahuja, S. Basu, S. Jack, V. J. Kleinberg, E. G. Sirer, and N. Foster, “Merlin: A language for provisioning network resources,” in Proc. 10th ACM Int. Conf. Emerg. Netw. Experiments Technol., Dec. 2014, pp. 213–226.

[33] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, “Modular SDN programming with Pyretic,” Tech. Rep. USENIX, Oct. 2013, p. 30.

[34] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak, “Maple: Simplifying SDN programming using algorithmic policies,” ACM SIGCOMM Comput. Commun. Rev., vol. 43, no. 4, pp. 87–98, 2013.

[35] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: Enabling innovation in campus networks,” ACM SIGCOMM Comput. Commun. Rev., vol. 38, no. 2, pp. 69–74, Apr. 2008.

[36] P. Bossart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, “P4: Programming protocol-independent packet processors,” SIGCOMM Comput. Commun. Rev., vol. 44, no. 3, pp. 87–95, Jul. 2014.

[37] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, “SilkRoad: Making stateful layer-4 load balancing fast and cheap using switching ASICs,” in Proc. ACM Special Interest Group Data Commun., Aug. 2017, pp. 15–28.

[38] S. Kandula, D. Katabi, S. Sinha, and A. Berger, “Dynamic load balancing without packet reordering,” ACM SIGCOMM Comput. Commun. Rev., vol. 37, no. 2, pp. 51–62, Apr. 2007.

[39] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, S.-J. Lee, and P. Yalagandula, “Automated and scalable QoS control for network convergence,” INProWREN, vol. 10, no. 1, p. 1, 2010.

[40] V. Moreno, “VXLAN deployment models—A practical perspective,” Cisco Live, 2015.

[41] M. Shahbaz, S. Choi, B. Pfaff, C. Kim, N. Feamster, N. McKeown, and J. Rexford, “PISCES: A programmable, protocol-independent software switch,” in Proc. ACM SIGCOMM Conf., Aug. 2016, pp. 525–538.

[42] T. Ouisiki, H. Tarasov, P. Chaigammon, and M. Kossakowski, “Part-OVS: Programming protocol-independent, runtime extensions for open vswitch with P4,” in Proc. IFIP Netw. Conf. (Netw.), 2020, pp. 413–421.

[43] N. Yadav and S. Merchant, “Forwarding tables for virtual networking devices,” U.S. Patent 9,755,965, Sep. 5, 2017.

[44] M. Jarschel, F. Wamser, T. Hohn, T. Zinner, and P. Tran-Gia, “SDN-based application-aware networking on the example of YouTube video streaming,” in Proc. 2nd Eur. Workshop Softw. Defined Netw., Oct. 2013, pp. 87–92.

[45] P4ipk Prototype: Github Repository. Accessed: Jul. 10, 2021. [Online]. Available: https://github.com/drorbink/TOP-K-PaTH-ReLase.git

[46] The P4 Language Consortium. P4runtime Specification Version 1.3.0. GitHub. [Online]. Available: https://github.com/p4lang/p4runtime-shell

[47] The P4.org API Working Group. P4runtime Specification Version 1.3.0. GitHub. [Online]. Available: https://p4.org/p4runtime/spec/main/p4Runtime-Spec.html

[48] DEBOBROTO DAS ROBIN received the B.Sc. degree in computer science and engineering from Bangladesh University of Engineering and Technology (BUET), in 2012. He is currently pursuing the Ph.D. degree in computer science with Kent State University. His research interests include software-defined networks, data center networking, cross-layer optimization, and in-network computation.

[49] JAVED I. KHAN received the B.Sc. degree from Bangladesh University of Engineering and Technology (BUET) and the M.S. and Ph.D. degrees in computer science from the University of Hawaii, Manoa. He is currently a Professor and the Chair of the Computer Science Department, Kent State University. He has a large experience in computer networks, in particular, next-generation network architecture, cross-layer communication, semantic design and composition, complex community, and active and programmable networking. His current research interests include future internet architecture, software-defined networks, and video communications.

[50] P4ipk Prototype: Github Repository. Accessed: Jul. 10, 2021. [Online]. Available: https://github.com/drorbink/TOP-K-PaTH-ReLase.git

[51] The P4 Language Consortium. P4runtime Specification Version 1.3.0. GitHub. [Online]. Available: https://github.com/p4lang/p4runtime-shell

[52] The P4.org API Working Group. P4runtime Specification Version 1.3.0. GitHub. [Online]. Available: https://p4.org/p4runtime/spec/main/p4Runtime-Spec.html