IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 25, 2020, accepted May 6, 2020, date of publication May 19, 2020, date of current version June 15, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2995558

QROUTE: An Efficient Quality of Service (QoS) Routing Scheme for Software-Defined Overlay Networks

NITIN VARYANI[1], ZHI-LI ZHANG[1], (Fellow, IEEE), AND DAVID DAI[2]
[1]Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455, USA
[2]Huawei Futurewei Technologies Inc., Santa Clara, CA 95050, USA

Corresponding author: Nitin Varyani (varya001@umn.edu)

This work was supported in part by Huawei Futurewei Technologies Inc.

ABSTRACT Many computer network applications impose constraints for multiple quality of service (QoS) metrics, e.g., delay, packet loss, bandwidth, and jitter. These QoS constraints cannot be guaranteed by the Internet due to its best-effort service model. Overlay networks have been an effective technique at the application layer to support multiple QoS constraints of networking applications. In software-defined overlay networks, software-defined networking (SDN) paradigm is introduced in the overlay networks to enable centralized and efficient routing of traffic in the overlay networks, thus, enabling better QoS. One of the main challenges in software-defined overlay networks is the fast-changing overlay link QoS characteristics. However, the existing routing algorithms for satisfying multiple QoS constraints in software-defined overlay networks involve high route computation time and thus these routing algorithms cannot adapt to the fast-changing overlay link QoS characteristics. Moreover, as we scale the size of overlay networks, the size of forwarding tables increases exponentially. This is because the existing routing schemes for ensuring multiple QoS constraints use both the source and the destination address for data-plane forwarding. This leads to pushing a huge amount of forwarding table entries by the controller through the network and thus limiting the size of the overlay network. We propose an efficient routing scheme, QROUTE, for satisfying multiple QoS constraints in software-defined overlay networks. QROUTE consists of a control plane routing algorithm which has significantly low route computation time because of employing a novel directed-acyclic-graph (DAG) based approach. QROUTE also reduces the forwarding entries in the data plane by using a QoS-metric-based forwarding scheme. We extensively evaluate QROUTE using traces from a global overlay service provider. We also examine QROUTE on a testbed of P4-BMv2 switches controlled by the ONOS controller using P4Runtime protocol. We find that QROUTE outperforms other state-of-the-art QoS routing schemes in route computation time, size of the forwarding tables and meeting the QoS requirements of various applications.

INDEX TERMS QoS, routing, DAG, route computation time, forwarding table size, QoS-metrics-based forwarding, Lagrange relaxation, integer programming, P4, ONOS, BMv2.

I. INTRODUCTION

Many computer network applications such as video conferencing, interactive gaming, VoIP, virtual reality, telepresence, video-on-demand and live video streaming impose constraints for multiple quality of service (QoS) metrics, e.g., delay, packet loss, bandwidth, and jitter. This is usually expressed by a list of minimum/maximum bounds for each QoS metric and is commonly referred to as a QoS policy. For example, in [2] the QoS policy used for a 384-kbps video conferencing session is (150ms, 30ms, 460 kbps, 1%). The tuple is in the format (maximum delay bound, maximum jitter bound, minimum bandwidth requirement, maximum packet loss). These QoS constraints cannot be guaranteed by the Internet due to its best-effort service model. Overlay networks have been an effective technique at the application layer to

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Imran.

TABLE 1. Contributions of the paper.

Contribution	*Reduces route computation time*	*Reduces routing entries*	*Increases resiliency*	*Finding routes slightly deviating from QoS policy*	*Supports QoS*
DAG-based primary routing algorithm	Yes	Yes			Yes
QoS-metric-based forwarding		Yes			Yes
Backup DAG generating algorithm			Yes		Yes
Framework to evaluate path's conformance to QoS policy				Yes	Yes

support multiple QoS constraints of networking applications [3]–[7]. An overlay network is a virtual network over an existing physical network and is composed of software routers running on commodity servers connected using tunneling protocols like Virtual Extensible LAN (VXLAN) and Generic Routing Encapsulation (GRE). In software-defined overlay networks, software-defined networking (SDN) paradigm is introduced in the overlay networks to separate the control from the forwarding. This enables centralized and efficient routing of traffic in the overlay networks and thus helps achieve better QoS [8]–[20].

One of the main challenges in software-defined overlay networks is the fast-changing overlay link QoS characteristics because of various reasons like the change in the corresponding path in the physical network or the change in non-overlay back-ground Internet traffic [14]. However, the existing routing algorithms for satisfying multiple QoS constraints in software-defined overlay networks involve high route computation time and thus these routing algorithms cannot adapt to the fast-changing overlay link QoS characteristics [8]–[20]. Moreover, as we scale the size of overlay networks, the size of forwarding tables increases exponentially. This is because the existing routing schemes for ensuring multiple QoS constraints use both the source and the destination address for data-plane forwarding [8]–[20]. This leads to pushing a huge amount of forwarding table entries by the controller through the network and thus limiting the size of the overlay network [15].

The problem of finding the optimal routes that meet multiple QoS constraints is known to be NP-hard. Therefore, several polynomial time heuristics algorithms [8]–[30] have been proposed in the literature to find approximate solutions. However, none of them can efficiently reduce both the route computation time and number of routing entries while ensuring additive, multiplicative and concave QoS constraints.

To address the above limitations, we propose an efficient routing scheme, QROUTE, for satisfying multiple QoS constraints in software-defined overlay networks. We make the following contributions in the paper.

- **DAG-based primary routing algorithm:** QROUTE consists of a control plane routing algorithm which has significantly low route computation time because of employing a novel directed-acyclic-graph (DAG) based approach.
- **QoS-metric-based forwarding:** QROUTE reduces the forwarding entries in the data plane by using a QoS-metric-based forwarding scheme.
- **Backup DAG generating algorithm:** To improve resiliency, we generate a backup DAG using a topological sorting based algorithm in our QROUTE routing scheme. During failures, we use the backup DAG along with the original DAG to achieve a considerable fraction of QoS while ensuring loop-free routing.
- **Framework to evaluate path's conformance to QoS policy:** QROUTE also includes a framework to evaluate a path's conformance to a multi-constrained QoS policy. In the absence of paths that completely satisfy a given QoS policy, this framework enables the network operator to find paths that slightly deviate from the QoS policy.
- **Trace-based evaluation:** We extensively evaluate QROUTE using traces from a global overlay service provider.
- **Evaluation on P4 switches:** We also examine QROUTE on a testbed of P4-BMv2 switches controlled by the ONOS controller using the P4Runtime protocol.

We summarize our contributions in table 1 along with their respective benefits. We find that QROUTE outperforms other state-of-the-art QoS routing schemes in route computation time, size of the forwarding tables and meeting the QoS requirements of various applications.

In [1], we introduce our DAG-based primary routing algorithm and QoS metric-based forwarding and present its trace-based evaluation. In this paper, we also include a backup DAG generating algorithm and a framework to evaluate a path's conformance to QoS policy. We also include rigorous mathematical proofs to support our theory in this paper. This paper also includes the evaluation of QROUTE on a testbed of P4-BMv2 switches controlled by the ONOS controller using the P4Runtime protocol. We also present our evaluation of QROUTE on the topologies from the Internet Topology Zoo dataset [31] in this paper.

We organize the paper as follows. In section II, we give an overview of our system framework and our QROUTE routing scheme and mathematically formulate our QoS routing problem. Section III describes the control-plane of our QROUTE routing scheme. We explain our QROUTE data-plane in section IV. Section V provides the details of our framework for evaluating a path's performance. We present

the results from a trace-based and a real test-bed evaluation of QROUTE in Section VI. Section VII summarizes the related work. The paper ends with a conclusion in section VIII.

II. PROBLEM FORMULATION

This section explains our system framework, an overview of QROUTE and the mathematical formulation of our QoS routing problem.

A. SYSTEM FRAMEWORK

Our system framework consists of an overlay network connecting end-users to content servers and other end users as shown in figure 1. The overlay network is created using software routers deployed across multiple data centers all over the world. The routers are connected using an overlay tunneling protocol. A DNS system chooses the best ingress software router for an end-user. A hierarchical design multi-controller architecture [32] is used to handle more flows and reduce control plane latency in large networks. The entire overlay network is divided into domains where routing within each domain is managed completely by a domain controller. The root controller is required to synchronize between the domain controllers for setting routes for flows that originate and terminate at different domains. Various applications demand different kinds of QoS from the overlay network. A QoS requirement of an application is represented by a QoS policy, which is defined by a list of permissible ranges for the end to end delay, jitter, packet loss, and bandwidth. The bandwidth requirement for a QoS policy is determined based on the application and also the number of users using that application averaged over time. The link-wise delay, jitter, packet loss, and available bandwidth are monitored and reported to the domain controller using traffic measurement tools. With this information, the routing algorithm at each domain controller generates routing Directed Acyclic Graphs (DAGs) for all the QoS policies and destination routers in its domain. A DAG stitching algorithm stitches together the routing DAGs in different domains to create a global routing DAG. For simplicity, we omit the DAG stitching algorithm from this paper and consider only a single controller for the entire network.

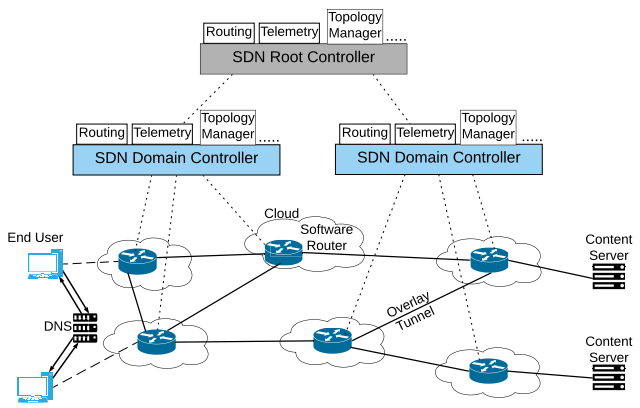


FIGURE 1. Our software-defined overlay network.

B. OVERVIEW OF QROUTE

In this section, we illustrate the control-plane and data-pane of the QROUTE routing scheme using an example.

We first provide an example of a primary routing DAG generated using our QROUTE control-plane routing algorithm. Consider the network graph in figure 2 where each edge between the nodes is bi-directed and the pair of value on each link represents the delay (ms) and jitter (ms) of the overlay link. For simplicity, we are considering here only two metrics but this can be extended to other metrics as explained in later sections. We do not depict the cost of the links in the figure. Given a QoS policy, we generate a routing DAG for every destination router instead of generating routes for every source-destination pair. In this example, we consider a QoS policy (40ms, 10ms) where the pair of values are maximum bounds for the delay (ms) and the jitter (ms) respectively. The dashed arrows in figure 2 depict the routing DAG generated using our QROUTE algorithm for the QoS policy (40ms, 10ms) and destination $R8$. From any node in the graph, the DAG contains at-least one path to the destination $R8$ that satisfies the QoS policy (40ms, 10 ms). Using breadth-first-search, we reduce the total number of route computations for a given destination router and QoS policy from $O(n)$ to $O(l)$

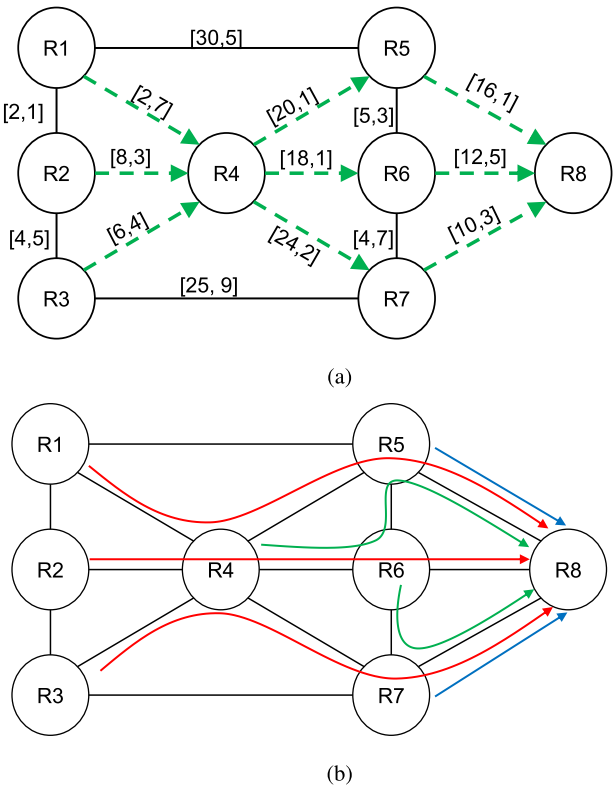


FIGURE 2. (a) The primary routing DAG generated by QROUTE for destination R8 and policy (40ms, 10ms) (represented by dashed arrows). All the links between network nodes are bi-directed. Each pair of values represents the delay (ms) and jitter (ms) of the corresponding link. The cost of the link is not indicated. (b) The paths generated from every source to destination R8 using the A*Prune algorithm. The number of route computations in path-based routing is 7 while QROUTE only performs 3 route computations.

where n is the number of nodes in the network and l is the number of leaves in a breadth-first-search traversal from the destination router. For m QoS policies and all destinations, the number of route computations reduces from $O(m \times n^2)$ to $O(m \times n \times l)$. In figure 2b, we depict the minimum cost paths generated by the A*Prune algorithm from every source to destination R8 that satisfy the QoS policy (40ms, 10ms). We can observe that for the given policy (40ms, 10ms) and destination router R8, the number of route computations using A*Prune is 7 while the number of route computations using QROUTE is only 3. Any path-based QoS routing algorithm like H_MCOP and MH_MCOP will also lead to the same number of route computations as that of A*Prune. QROUTE is thus useful for quickly computing routes in an overlay network with fast-changing traffic conditions.

To reduce the number of forwarding entries and to ensure that the packets for an application are forwarded along a route on the DAG that preserves the QoS policy of the application, we introduce a QoS-metrics-based forwarding scheme instead of forwarding based on source and destination IP address. We illustrate our QoS-metrics-based forwarding through the same example in figure 2. Table 2 shows the forwarding entries in router R4 for QoS policy (40ms, 10ms) and destination router R8. The policy (40ms, 10ms) is assigned an ID 1. We replace the source IP in the forwarding entries with the maximum amount of delay and jitter a packet can go through to be eligible to be forwarded to the corresponding outgoing link. These routing entries capture the QoS information of paths down the outgoing links. The packets carry the elapsed value of delay and jitter in their packet headers which are updated by each router by adding the delay and jitter of the outgoing link to the respective header field. These values are matched against the corresponding keys to determine where the packets should be forwarded to. For example, we can observe in the first entry in table 2 that a packet with QoS policy (40ms, 10ms) and destined to R8 can be forwarded to node R5 if the elapsed delay and the elapsed jitter of the packet is less than 4 ms and 8 ms respectively. We significantly reduce the forwarding entries by grouping them if they have the same next hop. Thus, we reduce the number of forwarding entries from $O(m \times n^2)$ to $O(m \times n \times deg)$ where deg is the maximum degree of the network graph. In the given example, we reduce the number of forwarding entries from 16 to 9 for the QoS policy (40ms, 10ms) and destination router R8. Thus, QROUTE alleviates the problem of the proliferation of routing entries in large overlay networks.

TABLE 2. Forwarding table for Router R4 using QROUTE.

Policy	*Destination*	*Maximum elapsed delay*	*Maximum elapsed jitter*	*Next hop*
1	R8	4	8	R5
1	R8	10	4	R6
1	R8	6	5	R7
1	R8	∞	∞	R7
2	R8	..	..	..

C. MATHEMATICAL FORMULATION

In this section, we provide a mathematical formulation of our QoS routing problem. Our QoS routing problem entails finding a directed acyclic graph (DAG) for every QoS policy and destination router such that from any node in the graph, the DAG contains at-least one path to the destination router that satisfies the QoS policy.

Network representation: A network graph $G = (V, E)$, where V is a set of overlay routers and E is the set of overlay tunnels.

Cost of edges: The column-vector of ''costs'' of the edges is denoted by c, $c \in R_+^{|E|}$. The ''cost'' denotes the price for using a link to deliver traffic and is determined by business relationships.

QoS policy: Let d, j, b and l, $\{d, j, b, l\} \in R_+$, denote the bounds for delay, jitter, bandwidth, and packet loss probability respectively.

QoS metric values of links: Let D, J, B and L, $\{D, J, B, L\} \in R_+^{|E|}$, denote the column-vectors of delay, jitter, bandwidth and packet loss of individual edges respectively.

QoS routing ''main problem'': Our QoS routing ''main problem'' involves finding a least-cost DAG for a destination node r and a QoS policy $Q = (d, j, b, l)$ such that the DAG has a path from every node s in the graph to the node r which satisfies all the constraints given in the QoS policy Q. We generate this routing DAG by composing the routes returned from the ''sub-problem'' as explained in section III-A2. We generate such routing DAGs for all destination nodes and QoS policies.

''Sub-problem'': We define the ''sub-problem'' as finding a path of minimum cost from a source vertex $s \in V$ to a destination vertex $r \in V$ which satisfies all the QoS constraints.

Vertex-edge incidence matrix: Let H denote the $|V| \times |E|$ vertex-edge incidence matrix such that for all $e = (u, v) \in E$ where $u, v \in V$, $H_{ue} = 1$ and $H_{ve} = -1$. Additionally, $H_{we} = 0$ for any $w \neq u, v$. Let K, $K \in R_+^{|V|}$, be a vector such that $K_s = 1$, $K_r = -1$ and $K_v = 0$ for all $v \in V \backslash \{s, r\}$.

Decision variables: Let y be a column-vector $(y_1, y_2 \ldots y_{|E|})$, $y_i \in \{0, 1\}$ of decision variables where $y_i = 1$ if the edge i belongs to the final routing path and 0 otherwise. The column vector y represents a path returned by the QROUTE routing algorithm.

We describe the ''sub-problem'' using the following optimization formulation.

$$p = \min_{y \in \{0,1\}^{|E|}} c^T y \tag{1}$$

$$s.t.\ H\, y = K \tag{1a}$$

$$D^T y \leq d \tag{1b}$$

$$\prod_{i=1}^{|E|} ((1 - L_i)^T y_i) \geq 1 - l \tag{1c}$$

$$J^T y \leq j \tag{1d}$$

$$\min_{i=1}^{|E|} (B_i^T y_i) \geq b \tag{1e}$$

The constraint (1a) restricts the value of y to a particular directed path from s to r. Constraints (1b) and (1d) ensure the additive metrics (delay and jitter) of the paths to be below their respective maximum bounds. Constraint (1c) checks if the product of success probability of the links of the paths is greater than the minimum bound on the success probability. Finally, the constraint (1e) limits the bottleneck bandwidth of the path to be greater than the minimum bound on the bandwidth. The ''sub-problem'' in equation 1 represents the formulation for an online QoS routing problem which adds the route for a single flow to a network such that the QoS requirements of the new and the existing flows are maintained. We use the solution from this ''sub-problem'' to find the solution for our QoS routing ''main problem'' of generating routing DAGs for all destination routers and QoS policies. To reduce the route computation time by availing the fast algorithms available for solving online QoS routing problems, we do not model the problem as a large and complex optimization problem that considers all the traffic flows to comprehensively determine the routes.

III. QROUTE CONTROL PLANE

In this section, we explain our QROUTE control-plane routing algorithm that generates the primary routing DAG for every destination router and QoS policy. We also explain our control-plane algorithm which is used to generate a backup DAG for each primary DAG.

A. PRIMARY ROUTING DAG

Our QoS routing problem entails finding a primary routing DAG for every QoS policy and destination router such that from any node in the graph, the DAG contains at-least one path to the destination router that satisfies the QoS policy. To compute a primary routing DAG per destination that satisfies all the constraints in a QoS policy, we decompose this problem to finding routes from a subset of nodes in the graph to the destination that satisfies all the QoS constraints. The task of finding every such route is referred to in this paper as the ''sub-problem''. We optimize this ''sub-problem'' to a polynomial-time algorithm using a Lagrange-relaxation-based technique as described in section III-A1. In section III-A2, we explain how we generate a routing DAG by composing the routes returned from the ''sub-problem''. In section III-A3, we describe how the bandwidth requirements of different QoS policies are ensured in our routing algorithm.

1) OPTIMIZING THE ''SUB-PROBLEM''

To find a near-to-optimal solution for a multi-constrained shortest path (MCSP) problem with only additive constraints, the Lagrange relaxation-based aggregated cost (LARAC) algorithm [33] is found to achieve one of the best performances [24]. To take advantage of the LARAC algorithm, we reduce our ''sub-problem'' described in equation 1 into a linear integer programming problem by converting the additive, multiplicative and concave constraints to only additive constraints.

The concave constraint bandwidth was eliminated from the network graph by removing links that do not meet the bandwidth constraint of the QoS policy. For more details about how we meet bandwidth requirements, refer to section III-A3. We accomplish this pruning by setting delay and jitter on those links to infinity and packet loss probability to 1. The multiplicative constraint, packet success probability, is simplified into an additive constraint through the negative logarithm of packet success probability of each link. The routing ''sub-problem'' then reduces to the following:

$$p = \min_{y\in\{0,1\}^{|E|}} c^T y \tag{2}$$
$$s.t.\ H\ y = K \tag{2a}$$
$$D^T y \leq d \tag{2b}$$
$$(L^T)' y \leq l' \tag{2c}$$
$$J^T y \leq j \tag{2d}$$

The column-vector L' is defined as $L'_e = -\log(1-L_e)$ for all $e \in E$. The bound l' is set to $-\log(1-l)$.

We relax the inequality constraints 2b-2d, by inserting the degree of violation of these constraints and their corresponding Lagrange variables into the objective function. Let $\lambda_1, \lambda_2, \lambda_3 \in R_+$ be the Lagrange multipliers for constraints 2b,2c, and 2d, respectively. Thus, the following Lagrange dual problem reduces from the above integer programming problem.

$$p_L = \max_{\lambda_1,\lambda_2,\lambda_3} LR(\lambda_1, \lambda_2, \lambda_3) \tag{3}$$
$$s.t. \lambda_1, \lambda_2, \lambda_3 \in R_+ \tag{3a}$$

where $LR(\lambda_1, \lambda_2, \lambda_3)$ is the Lagrangian dual function which is optimized subject to the non-dualized constraint.(Equation 4)

$$LR(\lambda_1, \lambda_2, \lambda_3) = \min_{y\in\{0,1\}^{|E|}} (c^T y + \lambda_1(Dy - d) + \lambda_2(L'y - l') + \lambda_3(Jy - j)) \tag{4}$$
$$s.t.\ H\ y = K \tag{4a}$$

The Lagrangian dual function 4 is reordered into the form:

$$LR(\lambda_1, \lambda_2, \lambda_3) = \min_{y\in\{0,1\}^{|E|}} ((c + \lambda_1 D + \lambda_2 L' + \lambda_3 J)y - (\lambda_1 d + \lambda_2 l' + \lambda_3 j)) \tag{5}$$
$$s.t.\ H\ y = K \tag{5a}$$

Since $\lambda_1\ d + \lambda_2\ l' + \lambda_3\ j$ is a constant, the value of y corresponding to the optimal solution of the Lagrangian dual function 5 is equal to that of Lagrangian dual function 6, which is equivalent to the shortest path between s and r with the cost of the links given by the vector $c+\lambda_1\ D+\lambda_2\ L'+\lambda_3\ J$.

$$LR(\lambda_1, \lambda_2, \lambda_3) = \min_{y\in\{0,1\}^{|E|}} ((c + \lambda_1 D + \lambda_2 L' + \lambda_3 J)y) \tag{6}$$
$$s.t.\ H\ y = K \tag{6a}$$

To find the shortest path for given node pairs, we can use single-source shortest path algorithms such as the Fibonacci based Dijkstra (with time complexity of $O(E + VlogV)$) or the faster A* algorithm.

Using a sub-gradient descent algorithm [34], we explore the dual problem's solution space. To start, we compute a solution path for the Lagrange dual problem (Line 9, Algorithm 1). If this solution is feasible, the cost of this path is set as the upper bound for the sub-gradient descent algorithm (Line 15, Algorithm 1). If this solution is non-feasible, we set the upper bound to the value stated in Theorem 1. During iteration, if a feasible path is discovered, the upper bound is updated using the cost of the found feasible path. The feasibility of a path is determined in Line 11, Algorithm 1. Theorem 2 is used to test if the solution to the Lagrangian dual function problem is optimal for the original problem or not (Lines 12, Algorithm 1).

Algorithm 1 Primary DAG Generating Algorithm

procedure PRIMARY_DAG($V, E, r, c, D, B, L, J, d, b, l, j$)

1: $E' = ReverseLink(E)$
2: $S = BFS(V, E', r)$
3: $M = \phi$
4: $P\_DAG = \{\}$
5: **while** $M\ != V$ **do**
6: Let $\lambda_1 = 3, \lambda_2 = 3, \lambda_3 = 3$
7: $s = S.pop()$
8: **for** $k \leftarrow 1$ to *iterations* **do**
9: $Y_{LR} = Dijkstra(s, c + \lambda_1\ D + \lambda_2\ L' + \lambda_3\ J)$
10: get shortest path from s to r, y_{sr}, from Y_{LR}
11: **if** $Dy_{sr} \leq d$ **and** $L'y_{sr} \leq l'$ **and** $Jy_{sr} \leq j$ **then**
12: **if** ($\lambda_1 = 0$ **or** $Dy_{sr} - d = 0$) **and** ($\lambda_2 = 0$ **or** $L'y_{sr} - l' = 0$) **and** ($\lambda_3 = 0$ **or** $Jy_{sr} - j = 0$) **then**
13: **break**
14: **else**
15: $UB = cy_{sr}$
16: **end if**
17: **end if**
18: **if** $UB == null$ **then**
19: $UB = ||c||_2\sqrt{MH}$
20: **end if**
21: $LB = cy_{sr} + \lambda_1(Dy_{sr} - d) + \lambda_2(L'y_{sr} - l') + \lambda_3(Jy_{sr} - j)$
22: $\theta = \frac{UB-LB}{(Dy_{sr}-d)^2+(L'y_{sr}-l')^2+(Jy_{sr}-j)^2}$
23: $\lambda_1 = max(0, \lambda_1 + \theta \times (Dy_{sr} - d))$
24: $\lambda_2 = max(0, \lambda_2 + \theta \times (L'y_{sr} - l'))$
25: $\lambda_3 = max(0, \lambda_3 + \theta \times (Jy_{sr} - j))$
26: **end for**
27: $M = M \cup nodes(y_{sr})$
28: $P\_DAG.add(y_{sr})$
29: **end while**
30: reset D, L, J

end procedure

We initialize all the Lagrange multipliers to 3. The subgradients for the relaxed constraints 2b, 2c and 2d are $(Dy_{sr} - d)$, $(L'y_{sr} - l')$ and $(Jy_{sr} - j)$ respectively.

Our scalar step size θ is given by

$$\theta = \frac{UB - LB}{(Dy_{sr} - d)^2 + (L'y_{sr} - l')^2 + (Jy_{sr} - j)^2}. \quad (7)$$

The difference of the current upper bound (UB) and the current lower bound (LB) and the scaling factor $(Dy_{sr} - d)^2 + (L'y_{sr} - l')^2 + (Jy_{sr} - j)^2$ informs the step size. Lines 23-25 of Algorithm 1 update the Lagrange multipliers and the dual function is then re-solved using the new set of multipliers. If a predefined number of iterations or the optimality condition (Theorem 2) is met, the algorithm will terminate.

Theorem 1: The square root of the solution to the minimum hop routing problem multiplied with 2-norm of cost vector is an upper bound to the solution of the shortest path routing problem.

Proof: A minimum hop routing problem involves finding a path of least hops between a source and destination node. We represent the problem in the following way: $MH = \min_{y \in P_{s,r}} \sum_{i=1}^{|E|} y_i$ where y represents a path as described in Section II-C and the objective function sums the number of links in the path y. The shortest path problem is represented by: SP $= \min_{y \in P_{s,r}} c^T y$ where c represents cost and y represents a path. We need to find an upper bound for shortest path problem (SP).

$$\rightarrow SP = \min_{y \in P_{s,r}} c^T y = \min_{y \in P_{s,r}} < c, y >$$

(By property of inner product, $< a, b >= a^T b$)

$$\rightarrow SP = \min_{y \in P_{s,r}} c^T y = \min_{y \in P_{s,r}} | < c, y > |$$

(Since c and y are positive vectors)

$$\rightarrow SP = \min_{y \in P_{s,r}} | < c, y > | \leq \min_{y \in P_{s,r}} ||c||_2 ||y||_2$$

(By Cauchy-Schwarz inequality, $| < a, b > | \leq ||a||_2 ||b||_2$)

$$\rightarrow SP \leq ||c||_2 \min_{y \in P_{s,r}} ||y||_2$$

$$\rightarrow SP \leq ||c||_2 \sqrt{\min_{y \in P_{s,r}} \sum_{i=1}^{|E|} {y_i}^2}$$

$$\rightarrow SP \leq ||c||_2 \sqrt{\min_{y \in P_{s,r}} \sum_{i=1}^{|E|} y_i}$$

(Since y_i takes a value of 1 or 0)

$$\rightarrow SP \leq ||c||_2 \sqrt{MH}$$

□

Theorem 2: A solution y_{sr} to a Lagrangian minimization problem is optimal for the original problem only if:

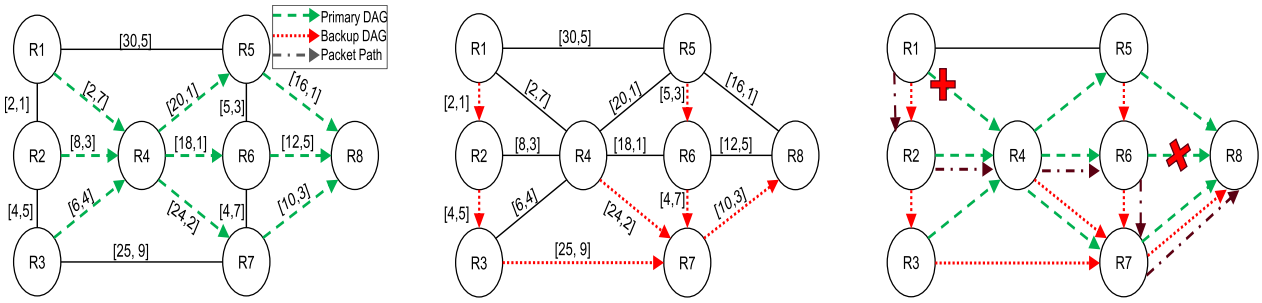


FIGURE 3. (a) The primary routing DAG for destination R8 and policy (40ms, 10ms). All the links between network nodes are bi-directed. Each pair of values represents the delay (ms) and jitter (ms) of the corresponding link. (b) Backup DAG for destination R8. (c) Path followed by packet from R1 to R8 in case of failures. If there is no matching entry in the primary DAG, the packet is forwarded along the backup DAG.

(a) y_{sr} *is feasible for the original problem*
(b) $cy_{sr} = [cy_{sr} + \lambda_1(Dy_{sr} - d) + \lambda_2(L'y_{sr} - l') + \lambda_3(Jy_{sr} - j)]$
i.e. $\lambda_1(Dy_{sr} - d) + \lambda_2(L'y_{sr} - l') + \lambda_3(Jy_{sr} - j) = 0$

Proof: This follows trivially from the Lagrangian sufficiency Theorem for inequality constraints. □

2) GENERATING A ROUTING DAG USING THE "SUB-PROBLEMS"

In this section, we elaborate on our approach for generating a directed-acyclic-graph (DAG) for a specific destination router and QoS policy by combining the solutions of "sub-problems".

For additive QoS metrics, we found that the feasible routes between an ingress router and an egress router also contain feasible routes between intermediary routers and the same egress router. Thus, if we compute feasible routes for only a subset of nodes, we can achieve full graph coverage.

In our approach we are trying to reduce the subset of nodes for which we need to compute a feasible path. We start by reversing the links in the original graph and then perform a breadth-first search (BFS) from the destination router (Lines 1-2 of Algorithm 1) while storing the nodes in decreasing order of distance (in hops) from the destination router. Then using this order, we find a feasible path between each router to the destination router. The further a router is from the destination, the more intermediary routers it will contain. We terminate the algorithm once all the nodes in the graph are covered (line 5&27, Algorithm 1). This approach significantly reduces the route computation time for the QROUTE algorithm. The BFS for all the destination routers can be done offline since it does not change with the changing traffic conditions. Only when the topology changes, we will need to re-perform BFS.

We illustrate routing DAG generation of QROUTE using the same example mentioned in section II-B with more detail. As explained in section II-B, the dashed arrows in figure 3a depicts the routing DAG generated using algorithm 1 for the QoS policy (40ms, 10ms) and destination *R*8. In this example, we perform breadth-first-search (BFS) traversal from the destination router R8 and arrange the nodes in the decreasing order of their distance from router R8. We then use algorithm 1 to compute feasible routes between routers R3-R8, R1-R8 and R2-R8 in the given order. By combining these routes, we create a routing DAG for the entire graph. Note that route generated for one pair of routers can be used for other pairs of routers if the QoS constraints are not violated. To ensure proper forwarding in data plane that ensures QoS constraints, we introduce QoS-metrics-based forwarding as explained in section IV-A.

Using breadth-first-search, we reduce the total number of route computations for a given destination router and QoS policy from $O(n)$ to $O(l)$ where n is the number of nodes in the network and l is the number of leaves in a breadth-first-search traversal from the destination router. For m QoS policies and all destinations, the number of route computations reduces from $O(m \times n^2)$ to $O(m \times n \times l)$. QROUTE is thus useful for quickly computing routes in an overlay network with fast-changing traffic conditions.

3) MEETING THE BANDWIDTH REQUIREMENTS

In this section, we explain how we meet the bandwidth requirements for different applications. For example, the QoS requirement for a single video streaming application used in our experimentation is (2000ms, 80ms, 0.5Mbps, 5%). We multiply the bandwidth requirement of an application by the number of flow requests corresponding to that application in a 30 second time interval. This time interval corresponds to the time between the last two route updates in the data plane. For example, if the number of flow requests corresponding to a video streaming application during that time interval is 500, we prune the links which have bandwidth less than $0.5 \times 500 = 250$ Mbps before computing DAGs for the video-streaming applications. The number of flows of a particular application in a given time interval is calculated by the controller using overlay header information of the flows sent by the ingress router to the controller when the first packet of a flow arrives in the network.

Once the routing DAG for a particular application and all destinations is calculated using algorithm 1, we subtract the bandwidth requirements of this application from the

available bandwidth of the network. For example, we subtract 250 Mbps from the available bandwidth of the links used in the DAGs for the video-streaming applications. We, then, find the routing DAGs for other applications. This ensures that the bandwidth requirements of all the applications are met.

B. BACKUP ROUTING DAG

In this section, we describe the algorithm for generating a backup DAG which we use along with the primary DAG to achieve a considerable fraction of QoS while ensuring loop-free routing.

Whenever an outgoing link is down or having congestion, the router should be able to forward packets to a backup link instead of incurring the delay of contacting the controller and waiting for the controller to install new rules. Installing backup paths for all pairs of source, destination and policy will again lead to the proliferation of routing entries and thus we need to generate a backup DAG for every destination router and policy that can be used during failures. The primary DAG should be as disjoint as possible to the backup DAG so that the congested or failed links of primary DAG do not affect the backup DAG. Moreover, using backup DAG, we should be able to provide sufficient QoS even during failures.

We use algorithm 2 to generate back-up DAG for the primary DAG generated by Algorithm 1. Algorithm 2 takes as input the primary DAG (*P_DAG*) and the adjacency list of the network graph (*Adj_G*) and generates a backup DAG (*B_DAG*) using the topologically sorted list of the *P_DAG* (*t_sort*) and the mechanism described below. We use an example to illustrate Algorithm 2. For the primary DAG in Fig. 3a, the backup DAG is shown in Fig. 3b. Algorithm 2 first sorts the nodes of the primary DAG using topological sort (line 2 of algorithm 2). A topological sort for a primary DAG in Fig. 3a is shown in Fig. 4.

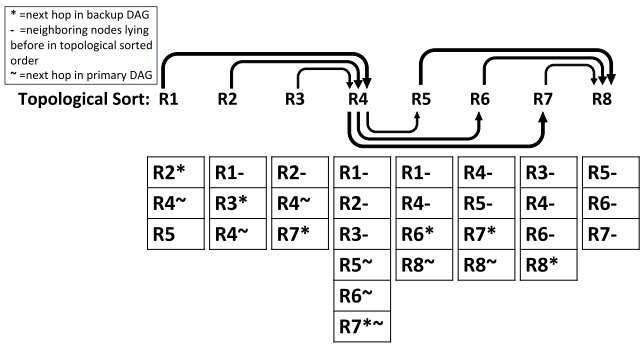


FIGURE 4. Topological sort of primary DAG in Fig. 3a along with adjacency list of the nodes in that order.

For a node, we do not use those neighboring nodes which lie before in the topologically sorted order (marked with a minus sign in Figure 4) as the next hop in the backup DAG. This was done so that the packets do not enter a forwarding loop while traversing links of both the primary and backup DAGs.For example, node R1 and R4 cannot serve as the next hop for node R5 in the backup DAG. We also avoid those nodes as next hops in backup DAG which are chosen as the next hops in the primary DAG (marked with a tilde sign in Figure 4). This was done to make the backup DAG as disjoint as possible to the primary DAG. This constraint is relaxed if there is no candidate for the next hop. For example, node R8 is avoided as the next hop for node R5 in the backup DAG. We then choose one of the remaining neighbors as the next hop in the backup DAG (marked with an asterisk sign in Figure 4). In the example, nodes R6 is used as the next hop for node R5 in the backup DAG. The backup DAG for destination router R8 is shown in Fig. 3b.

Algorithm 2 Back-up DAG Generating Algorithm

procedure BACKUP-DAG(*Adj_G*, *P_DAG*)

```
1:  B_DAG = {}
2:  t_sort = topological_sort(P_DAG)
3:  for i ∈ {1, ..., |V| − 1} do
4:    for j ∈ {i, i + 1, ..., |V|} do
5:      if t_sort[j] ∈ Adj_G[t_sort[i]] then
6:        last_neighbor_visited = t_sort[j]
7:        if t_sort[j] ∉ P_DAG then
8:          B_DAG[t_sort[i]] = t_sort[j]
9:          break
10:       else
11:         if t_sort[i] ∉ P_DAG[t_sort[j]] then
12:           B_DAG[t_sort[i]] = t_sort[j]
13:         end if
14:       end if
15:     end if
16:   end for
17:   if t_sort[i] ∉ B_DAG then
18:     B_DAG[t_sort[i]] = last_neighbor_visited
19:   end if
20: end for
```

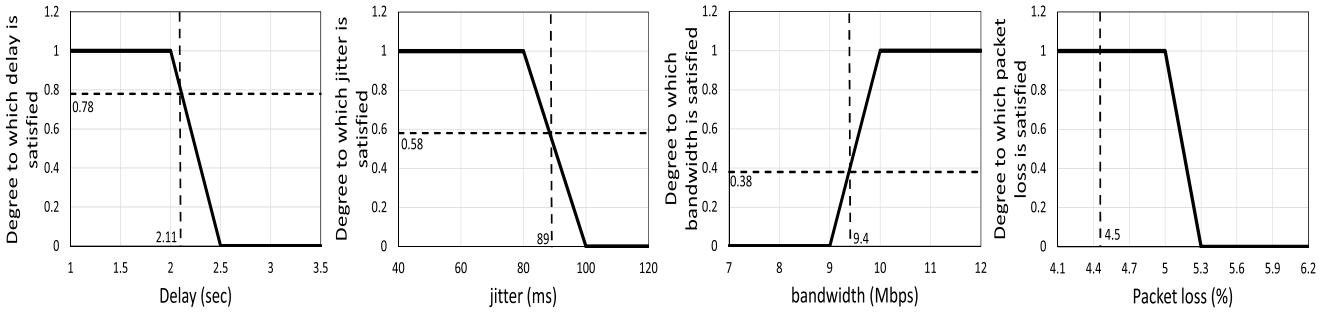
end procedure

IV. QROUTE DATA PLANE

In section IV-A, we explain our QoS-metrics-based forwarding which reduces the forwarding entries in the data plane while ensuring multiple QoS constraints. In section IV-B, we describe our data plane forwarding algorithm which uses the primary DAG entries along with the backup DAG entries to ensure QoS.

A. REDUCING FORWARDING ENTRIES IN THE DATAPLANE

We reduce the forwarding entries in the data-plane by using QoS-metric-based forwarding. We replace the source address in the forwarding entries of the switches with the maximum amount of QoS metrics a packet can spend in the network to be eligible to be forwarded to the corresponding outgoing link. We illustrate the QoS-metric-based forwarding using an example.



(a) Evaluation function for the delay metric

(b) Evaluation function for the jitter metric

(c) Evaluation function for the bandwidth metric

(d) Evaluation function for the packet-loss metric

FIGURE 5. Framework for evaluating performance of path generated by QROUTE.

Consider the network graph provided in Fig. 3. For simplicity each pair of values on the network links only represents delay and jitter for that bi-directed link measured in milliseconds. The following approach will hold for any number of additive and multiplicative constraints.

Figure 4 shows the DAG for a destination router R8, with a QoS policy of (40ms, 10ms) where 40ms is the maximum bound for delay and 10ms is the maximum bound for jitter. The only feasible path from $R1$ to $R8$ is $R1 \rightarrow R4 \rightarrow R5 \rightarrow R8$, from $R2$ it is $R2 \rightarrow R4 \rightarrow R6 \rightarrow R8$, and from R3 it is $R3 \rightarrow R4 \rightarrow R7 \rightarrow R8$. Packets from nodes R1, R2, and R3, arriving at R4 should be forwarded to R5, R6, and R7 respectively to maintain QoS requirements for each flow. For router R4 to route properly, one would need to utilize a source IP address, destination IP address, and policy ID as the key in forwarding tables or use a different key that satisfies the above requirement. However, using the source IP address leads to a proliferation of routing entries in the forwarding tables.

We replace the source IP in the routing entries of the forwarding table with the maximum amount of delay and jitter a packet can go through to be eligible to be forwarded to the corresponding outgoing link. The packets carry the elapsed value of delay and jitter in their packet headers which are updated by each router by adding the delay and jitter of the outgoing link to the respective header field. These values are matched against the corresponding keys to determine where the packets should be forwarded to.

For example, Table 3 depicts the forwarding table of router R4 with the forwarding entries corresponding to destination router R8 and QoS policy (40ms, 10ms). The delay and jitter along the path $R4 \rightarrow R5 \rightarrow R8$ is 36ms and 2ms respectively. Thus, if the maximum permissible delay and jitter that a packet has gone through before reaching R4 are 4ms (40-36) and 8ms(10-2) respectively then the packet can be forwarded to the router R5. This corresponds to the first entry in the forwarding table (Table 3). For the packets going from ingress router R1 to R4 directly, the elapsed delay and jitter will be 2ms and 7ms respectively. We can see that only the packet which is coming form R1 to R4 directly is eligible for forwarding to R5. Similarly we generate other forwarding entries for other feasible routes generated by algorithm 1 and passing through router R4. If any incoming packets headed for R8 do not match any routing entry or there is a failure, they will be routed through the default entry for policy 1 (represented by the 4th entry in Table 3). These default entries are created using the backup DAGs as described in Section III-A-D.

We significantly reduce the forwarding entries by grouping them if they have the same next hop. Given a QoS policy and destination, we have only one entry corresponding to an outgoing link which is used in a feasible route generated by Algorithm 1 and passing through that router. Thus, the maximum number of forwarding entries a router can have for a given QoS policy and destination is its degree in the network graph. In contrast, a source-destination based forwarding can have a forwarding entry for every source router. Thus, for a network of n nodes and m QoS policies, we reduce the number of forwarding entries from $O(m \times n^2)$ to $O(m \times n \times deg)$ where deg is the maximum degree of the network graph. Thus, QROUTE alleviates the problem of the proliferation of routing entries in large overlay networks.

TABLE 3. Forwarding table for Router R4.

Policy	*Destination*	*Maximum elapsed delay*	*Maximum elapsed jitter*	*Next hop*
1	R8	4	8	R5
1	R8	10	4	R6
1	R8	6	5	R7
1	R8	∞	∞	R7
2	R8	..	..	..

B. DATA-PLANE FORWARDING ALGORITHM

Our data plane forwarding algorithm uses the primary DAG entries along with the backup DAG entries to ensure QoS and provide resiliency. Algorithm 3 presents the forwarding algorithm of the switches. The switches first try to find a rule in the forwarding table corresponding to the primary DAG which matches all the QoS constraints. If it finds such

Algorithm 3 Data Plane Forwarding Algorithm

procedure FORWARDING(*Adj_G*, *Q_DAG*)

```
for rule in Primary_DAG do
    if elapsed_delay ≤ rule_max_elapsed_delay &&
       elapsed_jitter ≤ rule_max_elapsed_jitter &&
       elapsed_pkt_loss ≤ rule_max_elapsed_pkt_loss then
        if rule_output_port is up then
            add_delay(rule_output_link_delay)
            add_jitter(rule_output_link_jitter)
            add_pkt_loss(rule_output_link_pkt_loss)
            send_packet(rule_output_port)
        end if
    end if
end for
send_packet(Backup_DAG_output_port)
    {If no rule matches in Primary_DAG or the output ports
    for the matched rules are down}
```

end procedure

entry and the corresponding output port is up, the switch adds the QoS values of the forwarding link to the respective QoS values in the packet header and forwards the packet on the corresponding output port. If the switch does not find any entry or the output ports for the matched rules are down, it forwards the packet on the outgoing link along the backup DAG if it is up. If the outgoing link along the backup DAG is down, the switch drops the packet.

Fig. 3c shows the path followed by a packet in case of failures of two links. We can see from the figure that the packet traverses links of both the primary and the backup DAGs. The QoS is maximized by trying to re-route the packets to the primary DAG even if the packet has arrived using the backup DAG link.

V. FRAMEWORK FOR EVALUATING PATH PERFORMANCE

Some applications are not stringent about meeting their QoS requirements and thus the paths which satisfy their QoS policies to a reasonable fraction can be accepted. Moreover, in scenarios of network congestion or failures, the routing algorithm might not be able to find a path that satisfies the QoS policy for a particular application and thus the routing algorithm has to return a path that partially satisfies the QoS constraints. In such cases, the network manager needs to decide the degree to which the QoS should be satisfied for a path. We propose a framework to evaluate a path's performance in terms of meeting a multi-constrained QoS policy. Under the absence of paths that completely satisfy a given QoS policy, this framework enables the network operator to use the paths which slightly deviate from the QoS policy.

We represent each QoS constraint by a function which takes a value between 0 and 1. If the value of a particular QoS metric for a path returned by the routing algorithm is within its bound, then the function takes a value of 1. If it is outside its bound, then it takes a value less than 1. Beyond a certain threshold, the function takes a value of 0 which denotes that the QoS metric deviates a lot from its bound.

As an example, the QoS requirement used for video streaming in our experiments is (2000ms, 80ms, 0.5Mbps, 5%). The tuple is in the format (maximum delay bound, maximum jitter bound, minimum bandwidth requirement, maximum packet loss). Their respective functions are defined in Fig. 5. We can observe from Fig. 5a that the function for delay takes the value of 1 when the delay of a path is less than 2000ms. After that the function gradually drops to a value of 0. This function indicates the degree to which the delay metric is satisfied. Figure 5b to 5d can be interpreted along similar lines. A steeper slope for a QoS metric function indicates that the application is more stringent about meeting that QoS requirement. This slope is decided by the network operator and requires domain knowledge.

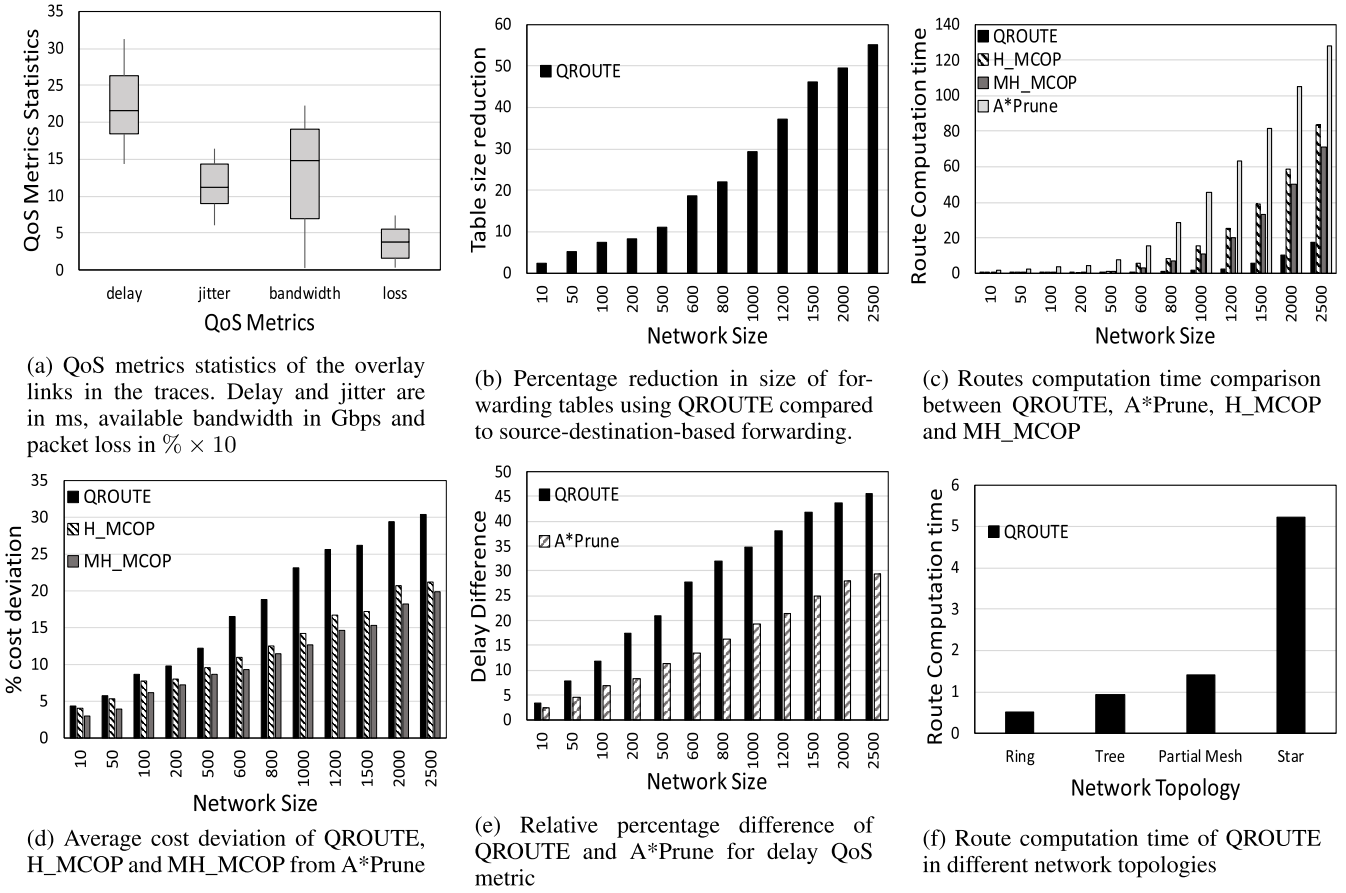
The vertical lines in the graphs denote the value of the corresponding QoS metrics for a path returned by the routing algorithm. The degree to which a path satisfies the QoS policy is equal to the smallest function value amongst all the QoS constraints. For example, in Fig. 5 the minimum function value is $min(.78, 0.58, 0.38, 1) = .38$. Thus this path satisfies the QoS policy by 38%. If the threshold for accepting a path for routing is 60%, this path will be rejected.

VI. EVALUATION

In this section, we describe the evaluation of the QROUTE routing scheme using traces from a global communications technology solutions provider's real overlay networks. We also explain the evaluation of QROUTE carried out in a test-bed of P4-BMv2 [35], [36] switches controlled with ONOS SDN controller [37] using P4Runtime [38]. All experiments were carried out in servers with Intel Xeon E5-1630 v4 @3.7GHz processors.

A. TRACE-DRIVEN EVALUATION

We determine the scalability and optimality of the QROUTE routing algorithm using trace-driven evaluation. We had access to 24-hour traces with measurements after every 30 seconds from a global communications technology solutions provider's real overlay networks. These traces consist of delay, jitter, packet loss, and bandwidth measurements of the links between the routers. The network topology the traces measured was comprised of routers spread across multiple data centers worldwide. The traces provided covered different sizes of overlay networks with different topologies (partial mesh, star, ring, and tree). We also perform an evaluation of QROUTE on the real topologies from the Internet Topology Zoo dataset [31] using synthetically generated traces. The delay between the network nodes in the topologies from the Internet Topology Zoo dataset was estimated based on their geographical locations. The bandwidth, jitter and packet loss values of the links in the synthetic generated traces were derived from the above traces. We wrote the QROUTE algorithm in Python using fast libraries like Dijikstar and used a CSV file of the traces for input.



(a) QoS metrics statistics of the overlay links in the traces. Delay and jitter are in ms, available bandwidth in Gbps and packet loss in $\% \times 10$

(b) Percentage reduction in size of forwarding tables using QROUTE compared to source-destination-based forwarding.

(c) Routes computation time comparison between QROUTE, A*Prune, H_MCOP and MH_MCOP

(d) Average cost deviation of QROUTE, H_MCOP and MH_MCOP from A*Prune

(e) Relative percentage difference of QROUTE and A*Prune for delay QoS metric

(f) Route computation time of QROUTE in different network topologies

FIGURE 6. Trace-driven evaluation.

TABLE 4. QoS requirements used for the experimentation.

Application type	*delay (ms)*	*jitter (ms)*	*bandwidth (Mbps)*	*packet loss (%)*
VoIP	150	30	0.1	1
Interactive Gaming	100	20	3	0.5
Video Conferencing	150	30	0.5	1
Video on Demand	2000	80	0.5	5
Telepresence	150	30	5	0.8
Virtual Reality	70	20	25	1
Live Video Streaming	150	30	2.5	1
Remote Desktop Connection	150	30	0.2	0.5
Telemetry	250	100	1.5	0.3
IPTV	150	30	1.5	1

For our evaluations, we select 10 different kinds of user-applications' QoS requirements as mentioned in Table 4.

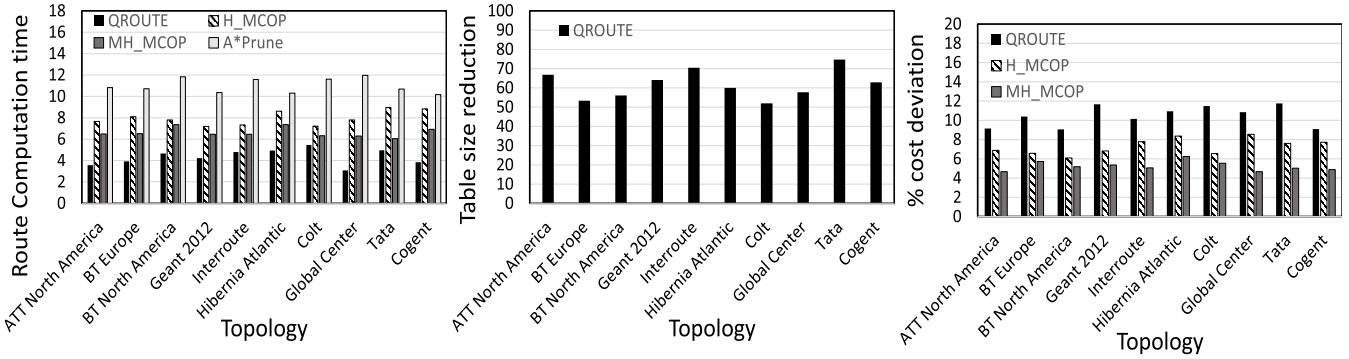
The resulting plots are informed by the average value of 28,800 ($24 \times 60 \times 2 \times 10$) instances of data points. Figure 6 shows the distribution of QoS metric values of the links in the traces using a box plot.

Using these traces we compared the scalability and optimality of the QROUTE algorithm to the current state-of-the-art multi-constrained shortest path (MCSP) algorithms, like A* Prune [21], H_MCOP [22], and MH_MCOP [23].

We measured four performance characteristics:

- **Percentage Reduction in Forwarding Entries**: The percentage reduction in forwarding entries compared to those generated by source-destination-based routing algorithms such as A*Prune, H_MCOP, and MH_MCOP.
- **Percentage Cost Deviation**: Indicates the percentage deviation of the cost of the approximate solution with that of the solution from A*Prune which is the optimal case.
- **Route Computation Time**: This includes generating the routes for all the destination routers and policies and the forwarding tables for the routers. Route computation time is measured in seconds.
- **Relative percentage difference**:The relative difference between the maximum QoS bound and the QoS of the paths found. The formula is: $(QoS\_bound - path\_QoS)/QoS\_bound \times 100$.

We obtain Fig. 6b, 6c, 6d and 6e with partial mesh topologies and Fig. 6f with networks of size 1000. From Fig. 6b we can observe that as the network size grows,



(a) Routes computation time comparison between QROUTE, A*Prune, H_MCOP and MH_MCOP

(b) Percentage reduction in size of forwarding tables using QROUTE compared to source-destination-based forwarding.

(c) Average cost deviation of QROUTE, H_MCOP and MH_MCOP from A*Prune

FIGURE 7. Trace-driven evaluation of QROUTE on the topologies from the Internet Topology Zoo dataset.

QROUTE achieves significant reduction in forwarding table sizes compared to other source-destination based routing algorithms. As depicted in Fig. 7b, we can observe the similar reduction in forwarding entries by QROUTE on the topologies from the Internet Topology Zoo dataset. This demonstrates the impact of replacing the source IP address with maximum elapsed QoS constraint values.

From Fig. 6c and Fig. 7a, we observe that QROUTE achieves a much lower route computation time than the other algorithms, like A*Prune, H_MCOP, and MH_MCOP. This is because of generating DAGs as described in Section III-A2.

This reduction in time complexity comes at a cost of solution optimality, and as a consequence, H_MCOP and MH_MCOP have much more optimal solution outcomes than QROUTE as depicted in Fig. 6d and Fig. 7c. However, unlike H_MCOP and MH_MCOP, QROUTE will always return a feasible route if one exists. If overlay providers do not consider a small increase in the cost of overlay links as a significant issue and desire a quick adaptive route computation algorithm for dynamic traffic, QROUTE is the ideal choice.

Fig. 6e shows the comparison of QROUTE and A*Prune in terms of relative percentage difference of delay QoS metric. For this comparison we did not include H_MCOP and MH_MCOP as they do not always generate a feasible path. We see that the relative percentage difference of delay of QROUTE increases quicker than that of A*Prune as the network size increases. This is because the feasible paths from some nodes to a destination node are contained in already computed routes.

QROUTE is well suited to ring, tree and mesh topologies and has fast route computations compared to that in the star topology (Fig. 6f). This is because the star topology does not benefit from our heuristic of a feasible route between two routers covering the feasible routes of many intermediary routers.

B. EVALUATION ON P4 SWITCHES

In this section, we describe our evaluation using a test-bed of P4-BMv2 [35], [36] switches controlled with ONOS SDN controller [37] using P4Runtime [38]. We evaluate the overhead incurred due to the addition and the range operation in the switches on the end-to-end delay. We also evaluate the degree to which the QoS requirements of the flows were satisfied as we increase the traffic in the network. We also measure the impact of failures on the QoS.

1) TEST-BED DESCRIPTION

P4 language [35] allows us to define custom forwarding behavior of a switch. Any forwarding behavior is composed of matches on header fields and based on that taking an action like changing header fields, forwarding on an output port, packet drop, etc. P4 allows us to define custom header fields and custom actions that are not supported on legacy or Openflow switches. The P4 codes can be compiled to many targets like FPGAs, ASICS, software switches, etc. We use the behavioral model (BMv2) [36], a software switch simulation which supports P4 language, in our experiments. We use P4 Runtime [38] API to install routing entries in the forwarding tables generated using the P4 code. We use the ONOS SDN controller [37] to control the network comprising of P4 BMv2 switches using P4 Runtime. To implement the QROUTE algorithm in a real overlay network, the routers should support the range and the addition operations on packet header fields along with the capabilities of OpenFlow switches [39], [40]. These operations are supported in P4, BMv2 switches and ONOS controller.

2) P4 PSEUDO-CODE FOR THE QROUTE OVERLAY HEADER AND ROUTERS

Algorithm 4 shows the P4 pseudo-code for our overlay header *QROUTE* which is inserted between the *Ethernet* header and the *IPv4* header. The *QROUTE* header contains the unique identifier for the QoS policy called *QoS_policy_ID*, the unique identifier for the egress router called *egress_router_id*, and the elapsed value of the QoS constraints called *elapsed_delay*, *elapsed_jitter* and *elapsed_pkt_loss*.

Algorithm 4 P4 Pseudo-Code for QROUTE Overlay Header

```
header QROUTE
    field QoS_policy_ID
    field egress_router_ID
    field elapsed_delay
    field elapsed_jitter
    field elapsed_pkt_loss

headers
    header Ethernet
    header QROUTE
    header IPv4
```

We describe the P4 pseudo-code for the forwarding table used in the ingress routers in algorithm 5. The ingress router matches the IPv4 address of the destination host of the incoming packets and encapsulates it with the *QROUTE* tunnel header. The encapsulation is performed using the *encapsulate_QROUTE_header* action.

Algorithm 5 P4 Pseudo-Code for Forwarding Table of Ingress Router

```
forwarding_table ingress_router
    match_fields
        (match_field_1, match_type)
        = (destination_host_IP, longest_prefix_match)
    actions
        action_1 = encapsulate_QROUTE_header
        action_2 = drop_packet
```

Algorithm 6 mentions the P4 pseudo-code for the forwarding table of intermediary and egress routers. To implement comparison operation on the elapsed value of delay, jitter and packet loss, we use the *range_match* supported by P4, P4Runtime, ONOS controller and BMv2 switches. A *range_match* match checks if a match field is between a lower and upper bound. For example, for the first entry in table 3, we use the range match to check if $0 \leq elpased\_delay \leq 4$ and $0 \leq elpased\_jitter \leq 8$. The intermediary routers match on all the 5 match fields mentioned in Algorithm 6 and accordingly forward the packet to an output port using *forward_QROUTE_packet* action described in Algorithm 7. The *forward_QROUTE_packet* action also adds to the respective headers of the packets the delay, jitter and packet loss of the link to which it is supposed to forward the packet. The delay, jitter and packet loss of the outgoing links are populated in the P4 switches by the ONOS controller. The action *decapsulate_QROUTE_header* mentioned in algorithm 6 is used by the egress router to remove the *QROUTE* tunnel header from the packet and forward the packet to its original destination host. A detailed description of our experimentation is given in Appendix A. We also provide an approach to measure available bandwidth in overlay networks having background traffic in Appendix B.

Algorithm 6 P4 Pseudo-Code for Forwarding Table of Intermediary and Egress Router

```
forwarding_table intermediary_and_egress_router
    match_fields
        (match_field_1, match_type)
        = (QoS_policy_ID, exact_match)
        (match_field_2, match_type)
        = (egress_router_ID, exact_match)
        (match_field_3, match_type)
        = (elapsed_delay, range_match)
        (match_field_4, match_type)
        = (elapsed_jitter, range_match)
        (match_field_5, match_type)
        = (elapsed_pkt_loss, range_match)
    actions
        action_1 = forward_QROUTE_packet
        action_2 = decapsulate_QROUTE_header
        action_3 = drop_packet
```

Algorithm 7 P4 Pseudo-Code for Forwarding a Packet With a QROUTE Header

```
action forward_QROUTE_packet(out_port, link_delay,
link_jitter, link_pkt_loss)
    add link_delay to header_elapsed_delay
    add link_jitter to header_elapsed_jitter
    add link_pkt_loss to header_elapsed_pkt_loss
    forward pkt on out_port
```

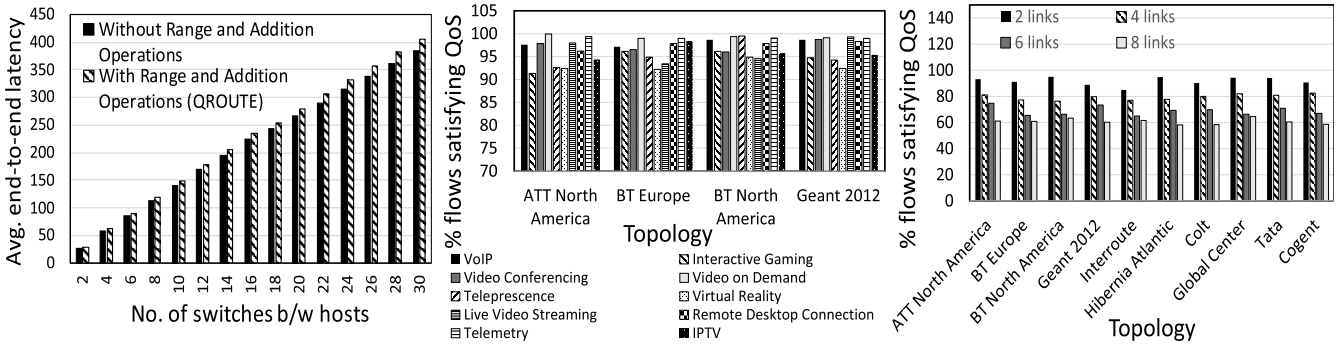
3) RESULTS

All the figures in this section are generated on a mesh network of 1000 P4-BMv2 switches.

We compare the average end-to-end delay observed by end-users with and without the overhead of range and addition operation on overlay headers (Fig. 8a). The paths taken by a packet between a pair of source and destination for both cases is the same. We observe that overhead imposed by QROUTE addition and range operation is not significant and as we increase the number of switches between end hosts, the overhead increases negligibly.

In Fig. 8b, we depict the percentage of flows that satisfy their QoS requirements using QROUTE in the absence of failures. We use the QoS policies for 10 different applications mentioned in Table 4 in this experiment. We can observe from Fig. 8b that QROUTE supports the QoS requirements for 90-99% of flows across different applications.

We also measure the percentage of flows that satisfy their QoS requirements using QROUTE after failures. This measurement is done until the SDN control responds to the change in the topology and installs new routes in the data plane. The QoS policies mentioned in Table 4 are used for this experiment also. As the number of failed links increases (Fig. 8c), we observe that the majority of the flows are still able to meet their QoS requirements. This resiliency is



(a) Average end to end delay observed by the application with/without addition and comparison operations.

(b) Percentage of flows which satisfy their QoS requirements using QROUTE in the absence of failures

(c) Percentage of flows which satisfy their QoS requirements using QROUTE after failures

FIGURE 8. Evaluation on P4 switches.

because we use the backup DAG along with the primary DAG and the forwarding scheme which maximizes QoS.

VII. RELATED WORK

A. LAGRANGE-RELAXATION BASED ROUTING SCHEMES

There are several works [12], [13], [21]–[24] which solve the multi-constrained shortest path problem using Lagrange relaxation but they generate routes for every source-destination pair and QoS policy instead of generating a routing directed-acyclic graph. Thus, their approaches lead to an exponential increase in route computation time and the proliferation of routing entries. A*Prune [21] is an optimal approach to solve the MCSP problem by assuming that there is a guess function available for the constraints and costs and the algorithm is made faster by pruning certain paths based on their projected constrained values. However, its run-time increases much faster with network size compared to heuristics based algorithms [21]–[23]. A heuristic and multi-constrained version of the LARAC algorithm, H_MCOP, proposed in [22], searches in the direction of all constraints and cost simultaneously, but this approach does not always return a feasible solution if it exists [23]. MH_MCOP [23] finds a closer solution to the optimal as compared to H_MCOP but also does not always find a feasible solution if it exists [24]. In [12], the authors model the QoS routing problem as a multi-commodity flow problem which is decomposed to simpler constrained shortest path problems. They use the GEN-LARAC algorithm [41] to solve the constrained shortest path problem that satisfies all QoS constraints. The authors in [13] model the overlay routing problem as a maximization of the Quality of Experience (QoE) instead of QoS and performed Lagrange decomposition of their original problem. The subproblems are solved using Lagrange relaxation and sub-gradient optimization. They used a k-shortest path algorithm to find the shortest paths in their sub-problems. All these approaches generate routes for every source-destination pair and QoS policy and thus leads to a significant route computation time and size of forwarding tables.

B. DIRECTED-ACYCLIC-GRAPH (DAG) BASED ROUTING SCHEMES

Directed-acyclic-graphs (DAGs) has been used in multi-constrained QoS routing [42]–[44]. However, none of them can efficiently reduce both the route computation time and the number of routing entries while ensuring additive, multiplicative and concave constraints. In [42], the authors use DAGs for routing in low-power and lossy networks. They consider the delay, jitter and packet loss while creating a destination-oriented DAG but they omit the bandwidth requirement (concave constraint) which is a very crucial network resource. They are using a distributed approach of exchanging messages between the nodes to construct a DAG rather than using a centralized routing algorithm. Moreover, they do not address the problem of the proliferation of routing entries in large networks. The authors in [43] consider all types of QoS constraints, that is, multiplicative, additive and concave. They prune the original network to a DAG that contains paths between a source and destination which satisfy all the constraints. They find the optimal path using this DAG. However, they do not use DAG for routing but only as an intermediary step to generate feasible paths. Thus, their approach leads to high route computation time and a large number of routing entries since they are generating paths for every source-destination pair. In [44], the authors use a DAG-based approach to provide resiliency against single arbitrary link failure. They also introduce delay and bandwidth constraints while generating DAGs. However, they do not consider jitter and packet loss constraints. Moreover, their approach leads to high route computation time and the proliferation of routing entries because of generating a DAG for every source-destination pair.

C. OTHER APPROACHES FOR ROUTING IN SOFTWARE-DEFINED OVERLAY NETWORKS

In [8], the authors propose a time-slot-based routing algorithm for finding multiple paths for a particular flow for ensuring QoS of multimedia applications in software-defined overlay networks. They model the cost of the overlay links

in terms of its availability and consider only bandwidth constraints. In [11], the authors use a routing engine based on random neural networks with reinforcement learning for their software-defined overlay networks. They only incorporate the latency of the paths while making their routing decisions. In [14], the authors propose a multipath routing algorithm that finds a path of least cost that satisfies the delay and bandwidth constraint. However, these approaches have high route computation time and a large number of forwarding entries because of generating routes for every source-destination pair and QoS policy.

VIII. CONCLUSION AND FUTURE WORK

We propose an efficient routing scheme, QROUTE, for satisfying multiple QoS constraints in software-defined overlay networks. QROUTE consists of a control plane routing algorithm which has significantly low route computation time because of employing a novel directed-acyclic-graph (DAG) based approach. QROUTE also reduces the forwarding entries in the data plane by using a QoS-metric-based forwarding scheme. QROUTE uses backup DAG combined with the primary DAG to provide sufficient QoS even during failures. We also provide a framework to evaluate a path's conformance to a multi-constrained QoS policy which enables the network operator to find paths slightly deviating from the QoS policy. Our experimental results demonstrate that the proposed QROUTE routing scheme not only significantly reduces the route computation time, but also decreases the forwarding table size considerably. Evaluations also show that the addition and range operations performed in the data plane do not incur significant overhead. Results demonstrate that QROUTE achieves QoS to a considerable degree with and without failures. QROUTE can be either implemented in software routers (e.g., BMv2 P4 switches) deployed in the cloud, or in hardware switches which support P4 (e.g., NetFPGA [45] and Barefoot Tofino [46]).

We are working on extending QROUTE to the hierarchical design multi-controller architecture. This involves designing a DAG stitching algorithm which stitches together the routing DAGs generated by QROUTE to create a global routing DAG.

According to the Cisco Annual Internet Report [47], the number of devices connected to Internet will be 29.3 billion by 2023. The majority of devices and the traffic that will dominate the Internet will be machine-to-machine communications, Internet of Things and enhanced 5G mobile broadband. These applications have significantly diverse QoS requirements [48]. To meet such diverse QoS requirements of huge network traffic requires more efficient QoS routing schemes.

APPENDIX A
DETAILED DISCUSSION OF OUR P4 SWITCHES TEST-BED

In this section, we provide a detailed description of our experimentation on test-bed of P4-BMv2 [35], [36] switches controlled with ONOS SDN controller [37] using P4Runtime [38].

A. INGRESS ROUTER OPERATION

Listings 1, 2, 3 and 4 show some important portions of our P4 code that implement the QROUTE algorithm. Listing 1 shows the overlay header *qroute_tunnel_t* which is inserted between the ethernet header and the ipv4 header and is used to carry the protocol ID of the next layer of header called *proto_id*, the QoS policy ID called *policyID*, the unique identifier of the egress router called *egress_router_id*, and the elapsed value of the QoS constraints called *elapsed_delay*, *elapsed_jitter* and *elapsed_pkt_loss*. Listing 2 describes the table used in the ingress router which matches the IPv4 address of the destination host for a packet using the longest prefix match (lpm) and then encapsulates the packet with the *qroute_tunnel_t* tunnel header using the *qroute_tunnel_ingress* action.

```
header qroute_tunnel_t{
    bit<16> proto_id;
    bit<32> egress_router_id;
    bit<16> policyID;
    bit<32> elapsed_delay;
    bit<32> elapsed_jitter;
    bit<32> elapsed_pkt_loss;
}
struct headers_t {
    ethernet_t      ethernet;
    qroute_tunnel_t qroute_tunnel;
    ipv4_t          ipv4;}
```

LISTING 1. The overlay header added to support QROUTE.

```
table qroute_tunnel_ingress {
    key = {hdr.ipv4.dst_addr: lpm;}
    actions = {
        qroute_tunnel_ingress;
        _drop();
    }
    default_action = _drop();}
```

LISTING 2. The P4 table for the ingress router.

```
table t_qroute_tunnel_fwd {
    key = {
        hdr.myTunnel.policyID:           exact;
        hdr.my_tunnel.egress_router_id: exact;
        hdr.myTunnel.elapsed_delay:      range;
        hdr.myTunnel.elapsed_jitter:     range;
        hdr.myTunnel.elapsed_pkt_loss: range;
    }
    actions = {
        qroute_tunnel_transit;
        qroute_tunnel_egress;
        _drop;
    }
    default_action = _drop();}
```

LISTING 3. The P4 table for the intermediary and egress router.

```
action qroute_tunnel_transit(port_tport, bit<32> link_delay, bit<32>
    link_jitter, bit<32> link_pkt_loss) {
    hdr.qroute_tunnel.elapsed_delay =
    hdr.qroute_tunnel.elapsed_delay + link_delay;
    hdr.qroute_tunnel.elapsed_jitter = hdr.qroute_tunnel.elapsed_jitter
    + link_jitter;
    hdr.qroute_tunnel.elapsed_pkt_loss =
    hdr.qroute_tunnel.elapsed_pkt_loss + link_pkt_loss;
    standard_metadata.egress_spec = port;}
```

LISTING 4. The P4 action for the intermediary routers.

B. INTERMEDIARY ROUTER OPERATION

To implement comparison on the elapsed value of delay, jitter and packet loss, we use the match type *range* supported by P4, P4Runtime and BMv2 switches. A *range* match checks if a specified header is between a *low* and a *high* value inclusively. ONOS supports a function *matchRange* (*PiMatchFieldId fieldId*, *byte*[ ] *low*, *byte*[ ] *high*) of type PiCriterion.Builder which adds a *range* field match for the given P4 header field ID *fieldId*, *low* value and *high* value of *range* match. In our scenario, we define three *range* matches for delay, jitter and packet loss (Listing 3) in our P4 code. We set the *low* limit to 0 and the *high* limit to the maximum permissible amount of delay, jitter or packet loss respectively in our *matchRange*() function call. Table *t_qroute_tunnel_fwd* (Listing 3) uses action *qroute_tunnel_transit* in the intermediate nodes and *qroute_tunnel_egress* in the egress nodes. The *qroute_tunnel_transit* action(Listing 4) is used in the intermediary nodes to forward the encapsulated packet on a output port based on the *egress_router_id*, *policyID* and the elapsed values of the QoS metrics. The BMv2 switch adds to the respective headers of the packets the delay, jitter and packet loss of the link to which it is supposed to forward the packet and then sets the output port of the packet. The delay, jitter and packet loss of the outgoing links are populated in the P4 switches by the ONOS controller.

C. EGRESS ROUTER OPERATION

The *qroute_tunnel_egress* action is used in the egress nodes to remove the *qroute_tunnel_t* header before forwarding the packet to the output port.

D. MEASURING QoS METRICS FOR THE LINKS

We use P4-BMv2 switches in integration with Mininet. Mininet provides us the options to specify the performance parameters of the links like delay, jitter, packet loss and maximum bandwidth. We need to start the mininet topology using *–link tc* command to set these performance parameters. This uses the TCLink class which is a wrapper around the Link class of mininet and allows us to specify the performance parameters like delay, jitter, etc. To create such a link in the custom topology python file, we need to use the command $self.addLink(switch1, switch2, bw = 10, delay = 10ms, jitter = 8ms, loss = 1)$. This creates a link with the maximum bandwidth of 10 Mbps, a delay of 10 ms, a jitter of 8 ms and a packet loss percentage of 1%. We have hard-coded the delay, jitter and packet loss percentage of the links in our test-bed. To measure the available bandwidth of the links, we create counters at every switch using the counter() function supported by P4. The switch counters count the bytes sent by the ports which is polled by an ONOS application at regular intervals and is stored in a CSV file. We compute the available bandwidth in the links using the bytes sent by the ports and the elapsed time using the approach mentioned in [49]. Using this telemetry information, the central controller computes new routes after every 30 seconds and installs the new rules in the forwarding tables of P4-BMv2 switches. A detailed discussion on measuring the QoS metrics in a real-overlay network is given in appendix B.

E. SIMULATING LINK FAILURES

Mininet provides a command *link s1 s2 up* and *link s1 s2 down* which activates and deactivates respectively the link between switch s1 and s2 of the network on the fly. We use this command to fail a link for testing our routing scheme during failures. Since there is no support in P4 switches to determine the link status, we store the status of the links in another table in the switches which are updated using the P4 switch's API used for accessing tables. When the outgoing link in the primary DAG is down, the packet is forwarded to the outgoing link in the backup DAG if it is up. If the outgoing link in the backup DAG is down, we drop the packet. Meanwhile, the ONOS controller detects that a link is down and the routing application calculates the new paths based on the new topology and installs new rules in the forwarding tables.

APPENDIX B
MEASURING AVAILABLE BANDWIDTH IN OVERLAY NETWORKS HAVING BACKGROUND TRAFFIC

We also provide an approach to measure available bandwidth in overlay networks having background traffic in this section. In a network with background traffic and unknown link capacity, available bandwidth can be computed using the values of delay and packet loss as described below. Since the throughput-intensive applications use either TCP or TCP like congestion control, the authors in [7] use the following equation for modeling throughput:

$$T = \frac{1}{rtt}\sqrt{\frac{1.5}{p}} \tag{8}$$

where p is the packet loss probability and rtt is the round trip time.

However, they have only considered last re-transmit and not timeout as the indicator for packet loss. Moreover, they have assumed that a received ACK is acknowledging at most 1 packet. They have also not considered that the congestion window size can be restricted. Thus, the below equation for throughput [50] will give a much accurate measurement of TCP throughput.

$$T = \min\left(\frac{W_{max}}{rtt}, \frac{1}{rtt\sqrt{\frac{2bp}{3}} + T_0 \min\left(1, 3\sqrt{\frac{3bp}{8}}\right) p(1+32p^2)}\right) \tag{9}$$

In equation (9), W_{max} is the maximum congestion window size, b is the number of packets that are acknowledged by a

received ACK, and T_0 is the initial time out. Current Windows and Linux operating systems often set $T_0 = 3\ sec$, $W_{max} = 64\ kb$ and $b = 2$ and putting them into Equation 9 will give a better estimate of TCP throughput in current networks. The delay, jitter and packet loss can be calculated by sending packet probes between the routers with timestamps.

ACKNOWLEDGEMENT

The authors would like to thank the anonymous reviewers for their valuable comments. This paper is revised and extended from our previous paper published in IFIP/IEEE IM 2019 [1].

REFERENCES

[1] 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. Integr. Netw. Service Manage. (IM)*, Apr. 2019, pp. 462–468.

[2] S. P. C. Lewis, "Implementing quality of service over Cisco MPLS VPNs," in *Selecting MPLS VPN Services*. San Jose, CA, USA: Cisco Systems, May 2006, ch. 5.

[3] Z. Li and P. Mohapatra, "QRON: QoS-aware routing in overlay networks," *IEEE J. Sel. Areas Commun.*, vol. 22, no. 1, pp. 29–40, Jan. 2004.

[4] R. K. Sitaraman, M. Kasbekar, W. Lichtenstein, and M. Jain, *Overlay Networks: An Akamai Perspective*, vol. 51. Hoboken, NJ, USA: Wiley, 2014, ch. 16, pp. 305–328.

[5] Z. Duan, Z.-L. Zhang, and Y. T. Hou, "Service overlay networks: Slas, QoS, and bandwidth provisioning," *IEEE/ACM Trans. Netw.*, vol. 11, no. 6, pp. 870–883, Dec. 2003.

[6] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proc. 18th ACM Symp. Oper. Syst. Princ.*, 2001, vol. 35, no. 5, pp. 131–145.

[7] J. Kurian and K. Sarac, "A survey on the design, applications, and enhancements of application-layer overlay networks," *ACM Comput. Surv.*, vol. 43, no. 1, pp. 1–34, Nov. 2010.

[8] S. Sahhaf, W. Tavernier, D. Colle, and M. Pickavet, "Adaptive and reliable multipath provisioning for media transfer in SDN-based overlay networks," *Comput. Commun.*, vol. 106, pp. 107–116, Jul. 2017.

[9] Y. Guan, W. Lei, W. Zhang, S. Liu, and H. Li, "Scalable orchestration of software defined service overlay network for multipath transmission," *Comput. Netw.*, vol. 137, pp. 132–146, Jun. 2018.

[10] P. Belzarena, G. G. Sena, I. Amigo, and S. Vaton, "SDN-based overlay networks for QoS-aware routing," in *Proc. Workshop Fostering Latin-Amer. Res. Data Commun. Netw. (LANCOMM)*, 2016, pp. 19–21.

[11] F. Francois and E. Gelenbe, "Optimizing secure SDN-enabled inter-data centre overlay networks through cognitive routing," in *Proc. IEEE 24th Int. Symp. Modeling, Anal. Simulation Comput. Telecommun. Syst. (MASCOTS)*, Sep. 2016, pp. 283–288.

[12] P. Medagliani, S. Paris, J. Leguay, L. Maggi, C. Xue, and H. Zhou, "Overlay routing for fast video transfers in CDN," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage. (IM)*, May 2017, pp. 531–536.

[13] G. Calvigioni, R. Aparicio-Pardo, L. Sassatelli, J. Leguay, P. Medagliani, and S. Paris, "Quality of experience-based routing of video traffic for overlay and ISP networks," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 935–943.

[14] W. Jiawei, Q. Xiuquan, and N. Guoshun, "Dynamic and adaptive multi-path routing algorithm based on software-defined network," *Int. J. Distrib. Sensor Netw.*, vol. 14, no. 10, pp. 1–10, 2018.

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

[16] T. Lin, T. Wen, W. Ren, Y. Zhang, and X. Zhang, "Table entry in software defined network," U.S. Patent 10 541 913, Jan. 21, 2020.

[17] Y. Guan, W. Lei, W. Zhang, H. Li, and S. Zhang, "SGMR: A spatial geometry-based multipath routing method on overlay networks," *Int. J. Commun. Syst.*, vol. 32, no. 5, p. e3894, Mar. 2019.

[18] S. Vaton, O. Brun, M. Mouchet, P. Belzarena, I. Amigo, B. J. Prabhu, and T. Chonavel, "Joint minimization of monitoring cost and delay in overlay networks: Optimal policies with a Markovian approach," *J. Netw. Syst. Manage.*, vol. 27, no. 1, pp. 188–232, Jan. 2019.

[19] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined wan," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, Sep. 2013.

[20] A. Rai, R. Singh, and E. Modiano, "A distributed algorithm for throughput optimal routing in overlay networks," in *Proc. IFIP Netw. Conf. (IFIP Netw.)*, May 2019, pp. 1–9.

[21] G. Liu and K. G. Ramakrishnan, "A*Prune: An algorithm for finding K shortest paths subject to multiple constraints," in *Proc. IEEE Conf. Comput. Commun., 20th Annu. Joint Conf. IEEE Comput. Commun. Soc. (INFOCOM)*, vol. 2, Apr. 2001, pp. 743–749.

[22] T. Korkmaz and M. Krunz, "Multi-constrained optimal path selection," in *Proc. IEEE Conf. Comput. Commun., 20th Annu. Joint Conf. IEEE Comput. Commun. Soc. (INFOCOM)*, vol. 2, Apr. 2001, pp. 834–843.

[23] G. Feng, K. Makki, N. Pissinou, and C. Douligeris, "Heuristic and exact algorithms for QoS routing with multiple constraints," *IEICE Trans. Commun.*, vol. E85-B, no. 12, pp. 2838–2850, Dec. 2002.

[24] J. W. Guck, A. Van Bemten, M. Reisslein, and W. Kellerer, "Unicast QoS routing algorithms for SDN: A comprehensive survey and performance evaluation," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 388–415, 1st Quart., 2018.

[25] P. Khadivi, S. Samavi, and T. D. Todd, "Multi-constraint QoS routing using a new single mixed metrics," *J. Netw. Comput. Appl.*, vol. 31, no. 4, pp. 656–676, Nov. 2008.

[26] P. T. A. Quang, J.-M. Sanner, C. Morin, and Y. Hadjadj-Aoul, "Multi-objective multi-constrained QoS routing in large-scale networks: A genetic algorithm approach," in *Proc. Int. Conf. Smart Commun. Netw. Technol. (SaCoNeT)*, Oct. 2018, pp. 55–60.

[27] S. Torkzadeh, H. Soltanizadeh, and A. A. Orouji, "Multi-constraint QoS routing using a customized lightweight evolutionary strategy," *Soft Comput.*, vol. 23, no. 2, pp. 693–706, 2019.

[28] D. Kalaiselvi and R. Radhakrishnan, "Multiconstrained QoS routing using a differentially guided krill herd algorithm in mobile ad hoc networks," *Math. Problems Eng.*, vol. 2015, pp. 1–10, Sep. 2015.

[29] X. Liu, A. Liu, T. Wang, K. Ota, M. Dong, Y. Liu, and Z. Cai, "Adaptive data and verified message disjoint security routing for gathering big data in energy harvesting networks," *J. Parallel Distrib. Comput.*, vol. 135, pp. 140–155, Jan. 2020.

[30] X. Zhang, W. Hou, L. Guo, Q. Zhang, P. Guo, and R. Li, "Joint optimization of latency monitoring and traffic scheduling in software defined heterogeneous networks," *Mobile Netw. Appl.*, vol. 25, no. 1, pp. 102–113, Feb. 2020.

[31] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet topology zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.

[32] T. Hu, Z. Guo, P. Yi, T. Baker, and J. Lan, "Multi-controller based software-defined networking: A survey," *IEEE Access*, vol. 6, pp. 15980–15996, 2018.

[33] A. Juttner, B. Szviatovski, I. Mecs, and Z. Rajko, "Lagrange relaxation based method for the QoS routing problem," in *Proc. IEEE Conf. Comput. Commun., 20th Annu. Joint Conf. IEEE Comput. Commun. Soc. (INFOCOM)*, vol. 2, Apr. 2001, pp. 859–868.

[34] S. Boyd, L. Xiao, and A. Mutapcic, "Subgradient methods," Stanford Univ., Stanford, CA, USA, Lecture Notes EE392o, Autumn Quart., 2003, vol. 2004, pp. 2004–2005.

[35] P. Bosshart, 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.

[36] P4 Language Consortium. (2020). *Behavioral Model Repository*. [Online]. Available: https://github.com/p4lang/behavioral-model

[37] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, and W. Snow, "ONOS: Towards an open, distributed SDN OS," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 1–6.

[38] P4 Language Consortium. (2020). *P4 Runtime*. [Online]. Available: https://p4.org/p4-runtime/

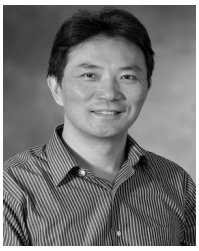
[39] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[40] (Jun. 25, 2012). *OpenFlow Switch Specification, Version 1.3.0 (Wire Protocol 0x04)*. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf

[41] Y. Xiao, K. Thulasiraman, and G. Xue, "GEN-LARAC: A generalized approach to the constrained shortest path problem under multiple additive constraints," in *Algorithms and Computation*, X. Deng and D.-Z. Du, Ed. Berlin, Germany: Springer, 2005, pp. 92–105.
[42] W. Khallef, M. Molnar, A. Benslimane, and S. Durand, "Multiple constrained QoS routing with RPL," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2017, pp. 1–6.
[43] X. Hu, K. Wang, J. Wang, K. Wang, Y. Hu, and S. Wang, "Multi-constrained routing optimization algorithm based on DAG," in *Proc. 44th Annu. Conf. IEEE Ind. Electron. Soc. (IECON)*, Oct. 2018, pp. 5906–5910.
[44] A. Pasic and P. Babarczi, "Delay aware survivable routing with network coding in software defined networks," in *Proc. 7th Int. Workshop Reliable Netw. Design Modeling (RNDM)*, Oct. 2015, pp. 41–47.
[45] N. Zilberman, Y. Audzevich, G. Kalogeridou, N. Manihatty-Bojan, J. Zhang, and A. Moore, "NetFPGA: Rapid prototyping of networking devices in open source," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 363–364, Sep. 2015.
[46] Barefoot Networks. (2020). *Barefoot Tofino Switch ASIC*. [Online]. Available: https://www.barefootnetworks.com/products/brief-tofino/
[47] Cisco. (2020). *Cisco Annual Internet Report (2018–2023) White Paper*. [Online]. Available: [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html
[48] *Service Requirements for Next Generation New Services and Markets*, document TS 22.261, Release 15, 3GPP, Aug. 2016. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3107
[49] M. Singh, N. Varyani, J. Singh, and K. Haribabu, "Estimation of end-to-end available bandwidth and link capacity in SDN," in *Ubiquitous Communications and Network Computing*, N. Kumar and A. Thakre, Ed. Cham, Switzerland: Springer, 2018, pp. 130–141.
[50] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 4, pp. 303–314, Oct. 1998.

NITIN VARYANI received the B.E. and M.E. degree in computer science from the Birla Institute of Technology and Science, India, in 2014 and 2016, respectively. He pursued a research internship in the area of software defined networking at the National Cybersecurity Research and Development Lab, NUS School of Computing, Singapore. He is currently pursuing the Ph.D. degree in computer science with the University of Minnesota, USA, under Prof. Zhi-li-Zhang.

His research interests include software defined networking, network function virtualization, mobile edge computing, and enabling quality of service in networking. He has published articles in top computer science conferences such as IFIP/IEEE IM, IEEE AINA, and EAI UBICNET. He is serving as a Reviewer for various journals such as *Future Generation Computer Systems*, the IEEE Transactions on Vehicular Technology, and the *IEEE Network Magazine*.



ZHI-LI ZHANG (Fellow, IEEE) received the B.S. degree in computer science from Nanjing University, China, and the M.S. and Ph.D. degrees in computer science from the University of Massachusetts.

He joined the faculty of the Department of Computer Science and Engineering, University of Minnesota, in 1997, where he is currently the McKnight Distinguished University Professor and a Qwest Chair Professor in Telecommunications. He also serves as the Associate Director for Research at the Digital Technology Center, University of Minnesota. He has published more than 100 journal and conference/workshop papers, many of them in top venues in networking and related fields. His research interests include broadly in computer and communication networks, Internet technology, multimedia systems, and content distribution networks, cyber-physical systems and Internet-of-Things, and (applied) machine learning and data mining.

Dr. Zhang was a co-recipient of several Best Papers awards including IEEE INFOCOM, ICNP, and ACM SIGMETRI.CS. He has Chaired the program committees of several major conferences in networking including IEEE INFOCOM, ACM SIGMETRICS, IEEE ICNP, and ACM Internet Measurement Conference (IMC), and served on the Editorial Board of several journals such as the IEEE/ACM Transactions on Networking, ACM TOMPECS, and PACM MACS.

DAVID DAI received the B.S. degree in mechanical engineering from Shanghai Jiao Tong University, in 1989, and the M.S. degree in civil engineering and computer engineering from the University of Missouri-Columbia, in 1995.

He is currently the Sr. Director of Engineering with Futurewei Technologies, Inc., Santa Clara, CA, USA. He leads a high-performance Research and Development team with expertise in the areas of cloud computing, edge computing, network virtualization, SDN, SD-WAN, overlay networking, and service chaining. His team prototypes and develops the infrastructure and technology needed for mobile broadband solutions. He provides expertise and direction instrumental to the definition and design of new architecture and technologies needed to drive adoption of NFV, SDN, cloud, and other related IT technologies into 5G and MEC architecture. He proposes technology research initiatives, from concept, analysis, detailed architecture definition, and specifications, to project planning, budgeting, execution, prototype development, and validation and transform technologies to products.