DEMUR: Dependable Multipath Routing in Software Defined Networking for ISP Backbone

Chenfei Gao*, Weiyi Zhang†, Jian Tang*, Rakesh K Sinha†, Kostas N Oikonomou†

* Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY 13244
† AT&T Labs Research, Middletown, NJ 07748

***Abstract*—Robustness and reliability are critical issues in network management. To provide resiliency, a popular protection scheme against network failures is simultaneous routing along *multiple paths*. The use of Software-Defined Networking (SDN), with fine-grained forwarding rules and the limited forwarding table size of commodity switches, brings a new set of constraints and challenges. In this work, we consider the problem of routing demands with dependability in a software-defined ISP backbone. We propose splitting large demands over multiple paths to reduce fragmentation and achieve service reliability under failures. We also minimize the delay differential among these paths to reduce the number of out-of-order packets. Finally, our routing algorithm works with forwarding tables of bounded size, commonly required by low-cost SDN switches. We report emulation and simulation results confirming the practicality of our solutions.**

***Index Terms*—Software Defined Networking; ISP backbone; forwarding table size; multipath routing; differential delay.**

I. Introduction

Google [8] deployed the first global software-defined WAN, B4, interconnecting their datacenters. The routing strategy used in B4 delivers max-min fair allocation to applications using multiple paths with the objective of maximizing link utilizations. In [7], Hong et al. proposed SWAN to boost the utilization of inter-datacenter networks by centrally scheduling the traffic from each service, and frequently re-configuring the network data plane to match the traffic demand. While the benefits of SDN have been well-justified in works dealing with data centers, such clean-slate designs cannot be directly applied to carrier/ISP networks, whose requirements and properties create different challenges. Early SDN designs such as B4 are used in *an environment where the network has the privilege to schedule or throttle certain types of traffic when necessary*. For example, many large data transfers in a data center or a private WAN are not delay- or bandwidth-sensitive, and thus can be scheduled to avoid peak hours. Unfortunately, *ISP networks, servicing numerous customers of different needs do not have this privilege, in that they must ensure service-level agreements (SLAs) for heterogeneous customer applications*. Delay-sensitive traffic must be routed accordingly, to avoid congestion. Pre-determined bandwidth must be guaranteed under both normal and failure conditions. And disruption due to by failures must be restored within a pre-determined interval.

Given the scalability of the routing in ISP backbones, a critical issue for SDN routing is the restriction on the *forwarding table size* [3]. While SDN brings the benefit of cheaper and simpler switches, the forwarding tables of commodity switches are smaller than those of traditional routers [11]. For example, Pica8 open switches usually can store a few thousand forwarding entries [12], which may be exceeded even in small data centers [3]. Thus limiting the forwarding table size is an important consideration in WANs using SDN switches.

In this paper, we study a challenging problem with several practical routing concerns, coupled with the restriction on the forwarding table size.

II. Related work

Earlier work has shown that splitting demands over multiple paths leads to better resiliency due to statistical multiplexing [8], [15]. In [1], the authors proposed a scheme named *PESO* which aims to route the connections in a manner such that link (or node failure) does not shut down the entire stream but allows significant amount of traffic to still continue to flow.

Ability to split and route traffic separately however introduces a unique problem. When traffic is routed over different physical paths, they may incur different amounts of delay and thus, reach at different times. This difference of delay of paths is popularly called *differential delay*. Presence of differential delay requires increased memory in the destination node to buffer the traffic until the data arrives on all the paths [5], [13], and consequently forces increased requirements for memory and bandwidth at the destination node in order to compensate for the transmission. In turn, this potentially raises the network element cost, making deployments more expensive. Moreover, in the worst case, buffer overflows can cause data corruption and bring down the service. Clearly, it is necessary bound the delay differential among these paths so as to minimize out-of-order packets. [14] argued the benefit of measuring differential delay values in reverse engineering the accurate link delays. [2] introduced the concept of differential delay minimization when adding a new circuit to an existing group of circuits. It showed the hardness of the problem and proposed heuristics.

In [8], the authors assumed a few dozens of datacenters in the field and each switch in the network is unusual to have the forwarding table with maximum size. Thus, they did not model the forwarding table size and further consider it as a constraint in Traffic Engineering optimization. And also, the paper does not take practical reliability and differential delay constraints into account. In [7], Hong et al., proposed a system SWAN to boost the utilization of inter-datacenter networks by centrally scheduling the traffic from each service, and frequently re-configuring the network's data plane to match current traffic demand. A greedy scheme was proposed to tackle the scalability issue of SWAN in the face of limited forwarding table size.

978-1-5090-5019-2/17/$31.00 ©2017 IEEE

However, their research problem is different from ours on the objective and reliability requirement. Other than that, we also bound the differential delay for multipath routing, which was rarely considered in SDN papers.

III. DEMUR Routing Problem

We model the network by a weighted directed graph $G(V, E, c, d, \mathcal{X})$, where V is the set of n nodes, representing switches, and E is the set of m edges representing pairs of connected switches. Each edge $e = (u, v)$ has capacity $c(e) > 0$ and delay $d(e) \geq 0$. Each node v has a maximum forwarding table size $\mathcal{X}(v)$.

Definition 1 (**DE**pendable **MU**ltipath **R**outing - **S**ingle **D**emand[DEMUR-SD])**.** Let $G = (V, E, c, d, \mathcal{X})$ be a graph, as defined above and we get an $s-t$ demand of size B, reliability requirement r and delay constraint (d_{min}, d_{max}), The DEMUR-SD problem seeks a set of $s-t$ flow paths $\mathcal{P}$ and associated flow $b(p)$ for each path $p \in \mathcal{P}$ such that:

feasibility: $\mathcal{P}$ is a feasible set of paths for this demand. The aggregated bandwidth of all paths in $\mathcal{P}$ is at least B (Eqn 3.1) and the sum of bandwidth allocated from all flow-paths to a link is at most equal to the link capacity (Eqn 3.2).

$$\sum_{p \in \mathcal{P}} b(p) \geq B \tag{3.1}$$

$$\forall e \in E, \sum_{p \in \mathcal{P}, e \in p} b(p) \leq c(e) \tag{3.2}$$

reliability: Any single link failure does not affect more than $r\%$ of the total bandwidth. This reflects the case when the traffic is provisioned for the peak rate but operator needs to ensure that the average rate, say $r\%$ below the peak rate, is maintained even after failures. Notice that it is not enough for each path to carry less than $r\%$ of the demand as it is possible for multiple paths to share a link and thus a single link failure may fail many paths.

delay: Any $p \in \mathcal{P}$ satisfies the delay constraint:

$$d_{min} \leq d(p) \leq d_{max}. \tag{3.3}$$

forwarding table size: The number of flow paths via a node v cannot exceed the available forwarding table size:

$$\forall v \in V, \sum_{\{p \in \mathcal{P} | v \in p\}} x(p) \leq \mathcal{X}(v). \tag{3.4}$$

We can generalize the DEMUR-SD problem where, instead of a single demand, we are given a set of demands to be routed. Each demand has an associated utility value. If all demands can not be routed with the specified constraints, we seek to maximize the cumulative utility of routed demands.

Definition 2 (**DE**pendable **MU**ltipath **R**outing - **M**ultiple **D**emands [DEMUR-MD])**.** This is a generalization of the [DEMUR-SD] problem, where we are given a set of demands $\mathcal{F} = \{f_1, \ldots, f_k\}$. Demand f_i has size B_i, reliability requirement r_i, and delay constraint (d^i_{min}, d^i_{max}). In addition, demand f_i has *utility* W_i.

The DEMUR-MD problem seeks to maximize the total utility of the demands that are routed. The reliability and delay constraints from DEMUR-MD have to be satisfied by each flow-path. Equation 3.2 for feasibility changes so that for any link, we are now ensuring that the link capacity can accommodate all flow-paths across *all* demands. The new equation is

$$\forall e \in E, \sum_{p \in \cup \mathcal{P}_i, e \in p} b(p) \leq c(e) \tag{3.5}$$

Similarly the forwarding table constraint has to account for flow-paths across *all* demands. Equation 3.4 changes to

$$\forall v \in V, \sum_{\{p \in \cup \mathcal{P}_i | v \in p\}} x(p) \leq \mathcal{X}(v). \tag{3.6}$$

Several problems related to DEMUR-MD are known to be NP-**hard**. The closely-related problem PESO is NP-hard, as shown in [1]. The differential delay problem has also been shown to be NP-hard [14]. Finally, routing with forwarding table size (in terms of number of flow paths) constraint is also NP-hard [3].

IV. Pseudo-Polynomial Solution for IDEMUR

We first study the single-flow DEMUR-SD problem, and then extend our solution to DEMUR-MD. To find a provably good solution for DEMUR-SD, first we look into a special case of the DEMUR-SD problem, where link delays and differential delay bounds *are integer values*. We denote the special case by IDEMUR problem in the rest of the paper. To solve IDEMUR, we present a pseudo-polynomial time solution based on a novel graph transformation.

A. Graph Transformation

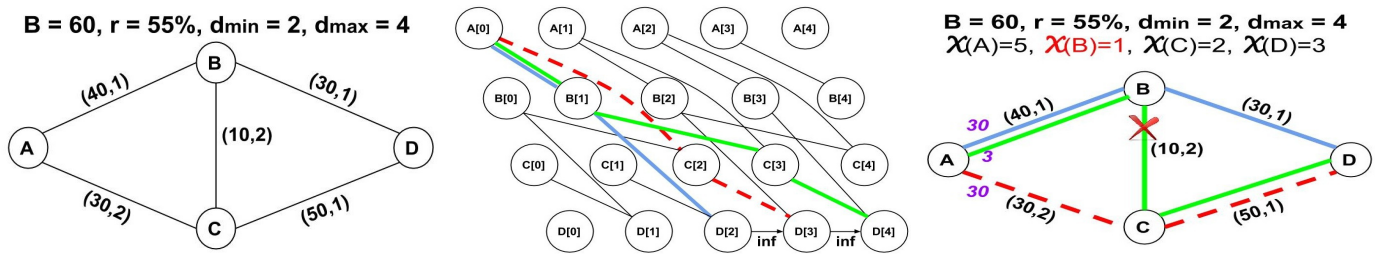
Let an instance of IDEMUR be given by graph $G(V, E)$, reliability requirement $r\%$, bandwidth request B, differential delay bounds d_{min} and d_{max}, available forwarding table size $\mathcal{X}$, and source-destination node pair (s, t).

We construct a layered graph $G^R = (V^R, E^R)$ from G as follows.

1) For each node $u \in V$, V_R contains $(d_{max} + 1)$ copies: $u_{[0]}, u_{[1]}, \ldots, u_{[d_{max}]}$.
2) For each link (u, v) in G, E^R contains $d_{max} - d(u, v) + 1$ links in the form $(u_{[i]}, v_{[i+d(u,v)]}), i = 0, 1, ..., (d_{max} - d(u, v))$. The logic for creating link $(u_{[i]}, v_{[i+d(u,v)]})$ in G^R is to demonstrate that if at node u a path has delay i, and the path is extended to v by using link (u, v), then the path delay will be increased to $i + d(u, v)$ when reaching node v.
3) E^R also contains $(d_{max} - d_{min})$ links (of infinite bandwidth) in the form of $(t_{[i]}, t_{[i+1]}), i = d_{min}, ..., d_{max} - 1$.
4) If $c(u, v) \geq r\% \cdot B$, for all edges $(u_{[i]}, v_{[i+d(u,v)]})$ in E^R, set aggregated bandwidth of these edges to $r\% \cdot B$. Otherwise, aggregated bandwidth is $c(u, v)$ for these edges.

We use an example to illustrate our graph transformation technique. In Fig. 1(a), an instance of IDEMUR with $s = A, t = D, B = 60, r = 55\%, d_{min} = 2$ and $d_{max} = 4$ is presented. The number pair associated with each link is the (capacity,delay) pair. The corresponding layered graph G^R is represented in Fig. 1(b).

Since the reliability requirement is 55%, and the bandwidth request is 60, the bandwidth that can be allocated on each link in G is at most 33. Correspondingly, in Fig. 1(b), all links



(a) Graph showing $(c(e), d(e))$ on each link e (b) Graph G^R corresponding to the graph G in Fig. 1(a) (c) UPS illustration

Fig. 1: IDEMUR graph transformation and UPS illustrations

in the form $(u_{[i]}, v_{i+d(u,v)}), i = 0, 1, ..., (d_{max} - d(u,v))$ have an aggregated bandwidth of 33. For example, link (C, D) has bandwidth 50 in Fig. 1(a), but at most bandwidth 33 can be used for demand $A - D$. Thus, the corresponding links in G^R, $(C_{[0]}, D_{[1]})$, $(C_{[1]}, D_{[2]})$, $(C_{[2]}, D_{[3]})$, $(C_{[3]}, D_{[4]})$, have aggregated bandwidth 33, shown in Fig. 1(b).

Algorithm 1 IDEMUR $(G, s, t, r, B, d_{min}, d_{max}, \mathcal{X})$

Input: G, s, t, r, B, d_{min}, d_{max}, $\mathcal{X} =< \mathcal{X}(v) >$
Output: $\mathcal{P}$, $\mathbf{b}$
Construct corresponding G^R using proposed graph transformation;
Find a maximum restricted flow from $s_{[0]}$ to $t_{[d_{max}]}$;
if flow value is less than B **then**
  Drop the request.
else
  Decompose the flow to generate the path set $\mathcal{P}^{\mathcal{R}}$;
  Find the corresponding path set $\mathcal{P}$ in G;
  $< \mathcal{P}, \mathbf{b} >:=$ UPS$(G, \mathcal{P}, \mathbf{b}, \mathcal{X})$;
  if $(b(\mathcal{P}) \geq B)$ **then**
    return $< \mathcal{P}, \mathbf{b} >$;
  else
    return Drop the request.
  end if
end if

The constructed graph G^R is acyclic since for each link $(u_{[i]}, v_{[j]}) \in E^R$, $i < j$. Finding a solution for an instance of IDEMUR from s to t in G is equivalent to searching paths in G^R from $s_{[0]}$ to $t_{[d_{max}]}$ in Algorithm 1. By adding the links in Graph Transformation, any path p^R from $s_{[0]}$ to $t_{d_{max}}$ has guaranteed delay between d_{min} and d_{max}. For any path p^R from from $s_{[0]}$ to $t_{[d_{max}]}$in G^R, there is a corresponding path p from s to t in G whose delay is also bounded by d_{min} and d_{max}. Since the aggregated flow on links $(u_{[i]}, v_{[i+d(u,v)]})$ is no more than $r\% \cdot B$, link (u, v) has no more than $r\% \cdot B$ flow value on it in the corresponding $s - t$ flow in G.

B. Satisfying flow table size constraint

From previous section, Routing Engine will find a feasible solution. However, not all computed paths will be validated and installed in SDN switches due to the limited flow table size constraint. Based on the forwarding table size model defined in Section III, the number of flow rules installed in an SDN switch is equal to the number of flow paths routed via it. If the number of flow paths needed for a new request exceeds a switch's available flow table size, not all the flow paths can be accommodated and the controller (Routing engine) needs to determine which subset of paths will be installed before sending solutions to the flow configuration agent.

Algorithm 2 UPS$(G, \mathcal{P}, \mathbf{b}, \mathcal{X})$

1: **Input:** G, $\mathcal{P}$, $\mathbf{b}$, $\mathcal{X} =< \mathcal{X}(v) >$
2: **Output:** $\mathcal{P}$, $\mathbf{b}$
3: Sort the paths in $\mathcal{P}$ in descending order of their corresponding bandwidth allocation $\mathbf{b}$;
4: **for** each flow path $p \in \mathcal{P}$ **do**
5: **if** $(\mathcal{X}(v) = 0, \quad \exists v \in p)$ **then**
6: Remove p from $\mathcal{P}$; $b(\mathcal{P}) := b(\mathcal{P}) - b(p)$;
7: **else**
8: $\mathcal{X}(v) := \mathcal{X}(v) - 1, \forall v \in p$;
9: **end if**
10: **end for**
11: **return** $< \mathcal{P}, \mathbf{b} >$;

We propose algorithm UPS (Algorithm 2) to update the path set $\mathcal{P}$ by checking available forwarding table sizes on nodes. By sorting paths in descending order of their corresponding bandwidth allocation (Line 3 in Algorithm 2), we prefer to route paths with higher bandwidth allocation first. Next, if flow path p cannot pass the checking, it will be removed from path set $\mathcal{P}$, together with its corresponding bandwidth allocation on p.

Let us use the example in Fig. 1(a) for further illustration. 3 paths in $\mathcal{P}$ from A to D with the corresponding bandwidth allocation are provided by our solution in previous section; $3(green)$, $30(blue)$ and $30(red)$ marked by the purple number on each path, respectively, in Fig. 1(c). We sort three paths in descending order of their bandwidth allocation and get $\mathcal{P} = \{red, blue, green\}$. First, the red path will be installed given switches *A* and *C* both have enough space for the flow entrance. Next, the blue path will be implemented since switches *A* and *B* have available flow table spaces. Note that at this moment, node *B* does not have any more free forwarding table size. Therefore, the green path, which goes through switch *B*, cannot be installed, and will be dropped. Eventually, entries for red and $blue$ paths are installed in switches with total allocated bandwidth 60. Correspondingly, we update $\mathcal{X}$ for all the switches on red and $blue$ paths to $\{\mathcal{X}(A) = 3, \mathcal{X}(B) = 0, \mathcal{X}(C) = 1, \mathcal{X}(D) = 1\}$. Since the bandwidth request (60 in the example) can be satisfied by the path set $\mathcal{P}$, this demand can be satisfied, and the solution will be sent to the flow configuration agent.

V. Approximation Scheme for ODEMUR Problem

For an arbitrary instance of DEMUR, the edge delay on each link could be real value. To tackle the DEMUR-SD problem in Section III, we first introduce an *optimization version* of the DEMUR-SD problem, denoted by ODEMUR.

Definition 3 (ODEMUR Problem)**.** Let $G = (V, E, c, d, \mathcal{X})$ be a weighted directed graph with node set V and link set E, where each link $e \in E$ is associated with a bandwidth capacity $c(e) > 0$ and a delay $d(e) \geq 0$, and *each node* $v \in V$ *is associated with an available forwarding table size* $\mathcal{X}(v)$. Let f be a demand with {source, destination, bandwidth request B, reliability requirement $r\%$, delay requirements d_{min}}. The *optimization version* of the DEMUR problem seeks a set of $s-t$ flow paths, $\mathcal{P}$, such that:
Objective: **Minimize** D
Subject to:

$$(1)\ b(\mathcal{P}) \geq B \quad (2)\ d_{min} \leq d(p) \leq D, \quad \forall p \in \mathcal{P}$$
$$(3)\ b(e) \leq r\% \cdot B, \quad \forall e \in E \quad (4) \sum_{\{p|v \in p\}} x(p) \leq \mathcal{X}(v), \forall v \in V$$

A. Approximation Scheme for ODEMUR

For a given positive real number θ and an instance ODEMUR(G, s, t, r, B), we construct an auxiliary graph $G^\theta = (V, E, d^\theta)$ of $G = (V, E, d)$. The edge delay is changed to $d^\theta = \lfloor d(e) \cdot \theta \rfloor + 1$ for each edge e. This is the *scaling and rounding* technique used in [9], [16].

Algorithm 3 ODEMUR$(G, s, t, r, d_{min}, d_{max}, B, \mathcal{X})$

1: **Input:** G, s, t, r, B, d_{min}, d_{max}, $\mathcal{X} =< \mathcal{X}(v) >$
2: **Output:** $\mathcal{P}$
3: Construct $G^\theta := (V, E, s, t, d^\theta)$ by changing the edge delay $d^\theta := \lfloor d(e) \cdot \theta \rfloor + 1$ on each link e;
4: Find the *minimum integer* D, say D_L, such that IDEMUR$(G^\theta, s, t, r, B, 0, D, \mathcal{X})$ is feasible;
5: Solve IDEMUR$(G^\theta, s, t, r, B, 0, D_L, \mathcal{X})$ and find the corresponding path set $\mathcal{P}$;
6: **if** (**NO** feasible solution is found) **then**
7: Drop the connection request.
8: **end if**
9: **for** each path $p \in \mathcal{P}^\theta$ **do**
10: **if** $(d(p) < d_{min})$ **then**
11: Remove p from $\mathcal{P}^\theta$; $b(\mathcal{P}^\theta) = b(\mathcal{P}^\theta) - b(p)$;
12: **end if**
13: **end for**
14: **if** $b(\mathcal{P}^\theta) \geq B$ **then**
15: Output $\mathcal{P}^\theta$ as the solution.
16: **else**
17: Drop the request.
18: **end if**

For a given positive real number θ and an instance DEMUR, we construct an auxiliary graph $G^\theta = (V, E, c, d^\theta)$ by updating delays on each edge e $d^\theta = \lfloor d(e) \cdot \theta \rfloor + 1$. After change, each link in G^θ has an integer delay. Then, we try to find the minimum integer D which can return feasible solution for an instance of IDEMUR with $d_{min} = 0, d_{max} = D$ (Line 4). For the returned paths $\mathcal{P}$, we have the following theorem.

Theorem 1. Algorithm 3 is an $(1+\phi)$-approximation solution for ODEMUR$(G, s, t, x, B, d_{min}, d_{max}, \mathcal{X})$. □

Proof: Let $\mathcal{Q}$ and d_{opt} denote the *optimal set of paths* and the *optimal value* of ODEMUR, respectively. Let y_L and y_U be given positive constants such that $y_L \leq d_{opt} \leq y_U$. For any given $\phi > 0$, we set $\theta = \frac{n-1}{y_L \cdot \phi}$, where n is the number of nodes in the network.

For any path p in the optimal solution $\mathcal{Q}$, its delay value in G^θ is denoted as $d^\theta(p)$. We have

$$\begin{aligned} d^\theta(p) &= \sum_{e \in p} d^\theta(e) = \sum_{e \in p} (\lfloor \theta \cdot d(e) \rfloor + 1) \\ &\leq \sum_{e \in p} (\theta \cdot d(e) + 1) \leq \theta \cdot d(p) + (n-1) \end{aligned} \tag{5.1}$$

Meanwhile, according to ODEMUR definition, we know that $0 \leq d(p) \leq d_{opt}$, and $d^\theta(p)$ is *integer*. From (5.1) we know that

$$\begin{aligned} d^\theta(p) &\leq \theta \cdot d(p) + (n-1) \leq \theta \cdot d_{opt} + (n-1) \\ &= \frac{n-1}{y_L \cdot \phi} \cdot d_{opt} + (n-1) \leq \lfloor \frac{n-1}{y_L \cdot \phi} \cdot d_{opt} \rfloor + (n-1) \end{aligned} \tag{5.2}$$

(5.2) shows that *the optimal solution $\mathcal{Q}$ is a feasible solution* to IDEMUR$(G^\theta, s, t, r, B, 0, \lfloor \frac{n-1}{y_L \cdot \phi} \cdot d_{opt} \rfloor + (n-1), \mathcal{X})$. In the meantime, D_L is the minimum integer value of D to guarantee IDEMUR$(G, s, t, r, B, 0, D, \mathcal{X})$ being feasible (Line 4 in Algorithm 3). Therefore, we have

$$\begin{aligned} D_L &\leq \lfloor \frac{n-1}{y_L \cdot \phi} \cdot d_{opt} \rfloor + (n-1) \\ &\leq \theta \cdot d_{opt} + y_L \cdot \phi \cdot \theta \end{aligned} \tag{5.3}$$

In Line 5, when we get a feasible solution $\mathcal{P}^\theta$ for IDEMUR$(G^\theta, s, t, r, B, 0, D_L, \mathcal{X})$, for each $p_\theta \in \mathcal{P}^\theta$, we know that

$$d^\theta(p_\theta) \leq D_L \leq \theta \cdot d_{opt} + y_L \cdot \phi \cdot \theta \tag{5.4}$$

Meanwhile,

$$\begin{aligned} d^\theta(p_\theta) &= \sum_{e \in p_\theta} (\lfloor \theta \cdot d(e) \rfloor + 1) \\ &\geq \sum_{e \in p_\theta} \theta \cdot d(e) = \theta \cdot d(p_\theta) \geq 0 \end{aligned} \tag{5.5}$$

Combining (5.4) and (5.5), we have

$$0 \leq \theta \cdot d(p_\theta) \leq d^\theta(p_\theta) \leq \theta \cdot d_{opt} + y_L \cdot \phi \cdot \theta \tag{5.6}$$

$$\begin{aligned} &\Rightarrow 0 \leq d(p_\theta) \leq d_{opt} + y_L \cdot \phi \leq d_{opt} + d_{opt} \cdot \phi \\ &\Rightarrow 0 \leq d(p_\theta) \leq (1+\phi) \cdot d_{opt} \qquad \forall p_\theta \in \mathcal{P}^\theta \end{aligned} \tag{5.7}$$

Furthermore, it is obvious that reliability and flow table size constraints will be satisfied for an ODEMUR instance if they are satisfied for the corresponding IDEMUR instance. In addition, Lines 10-12 make sure that delay lower bound d_{min} will be held.

Therefore, for each flow path p in $\mathcal{P}^\theta$, it satisfies the reliability and flow table size constraints, meets delay lower bound d_{min} and has delay bounded by no more than $(1+\phi)$-times the optimal value. This proves that a solution $\mathcal{P}$ returned by Algorithm 3 is an $(1+\phi)$-approximation to the ODEMUR problem. ■

The most important part of our approximation to ODEMUR is *finding the minimum integer* D_L such that IDEMUR$(G^\theta, s, t, r, B, 0, D_L, \mathcal{X})$ is feasible (Line 4 in Algorithm 3). In (5.3), we have shown that $D_L \leq \lfloor \frac{(n-1) \cdot d_{opt}}{y_L \cdot \phi} \rfloor + (n-1)$. Thus, we know:

$$0 \leq D_L \leq \lfloor \frac{(n-1) \cdot y_U}{y_L \cdot \phi} \rfloor + (n-1)$$

To calculate D_L, we first need to find a pair of lower and upper bounds of d_{opt}, denoted by y_L and y_U. *Scaling and*

testing technique [9], [16] is adopted to calculate y_L and y_U. Then, using binary search, we can find D_L by testing $\log(\frac{n \cdot y_U}{\phi \cdot y_L})$ instances of IDEMUR in pseudo-polynomial time. Solving each instance takes $O(m^3(\frac{n}{\phi})^4 \mathcal{M} log(\frac{n}{\phi}))$ [9], where n, m, $\mathcal{M}$ are the number of nodes, the number of edges, the input size of the instance of IDEMUR, respectively.

VI. Proposed solutions for DEMUR-MD Problem

In this section, we study the DEMUR-MD problem, which aims to accommodate multiple demands, each is associated with various utility. We are trying to seek a set of paths, $\mathcal{P}$, for each demand such that the total utility from all the satisfied demands is maximized while all the constraints are satisfied.

Algorithm 4 DEMUR **M**aximum **U**tility (DEMUR-MU)

Input: G, $\mathcal{F} =< f_i >$, $\mathcal{X} =< \mathcal{X}(v) >$
Output: $\mathcal{P} =< \mathcal{P}_i >$, W_{total}
$W_{total} := 0$;
Sort demands in the decreasing order of $\frac{W_i}{B_i}$;
for each demand f_i in $\mathcal{F}$ **do**
  $\mathcal{P}_i :=$ ODEMUR$(G, s_i, t_i, r_i, B_i, d^i_{min}, d^i_{max}, \mathcal{X})$;
  Update residual link capacity and node flow table size;
  if $\mathcal{P}_i$ found **then**
    $W_{total} := W_{total} + W_i$;
  end if
end for
return $< \mathcal{P}, W_{total} >$;

1) ODEMUR-based heuristic: In order to solve DEMUR-MD problem, we first sort all demands in $\mathcal{F}$ based on demand request size B_i, demand associated utility W_i and accommodate them in this order. A flow with higher utility/bandwidth ratio will be given higher priority to be processed.

2) Fast heuristic: One concern for Algorithm 4 is the running time. In this section, we propose a fast heuristic for DEMUR-MD problem, named by DEMUR-Fast. In this heuristic, we process every flow according to their arrival order, not by their demand sizes and associated utilities. Also, we do not bound our solution space to the paths returned from ODEMUR. Instead, we use *conventional maximum flow* to search paths.

For each flow, we first update link capacities by taking reliability constraint into account. Then we compute a maximum flow from the source to the destination. If the maximum flow is no less than the demand size, we filter the paths in $\mathcal{P}_i$ according to differential delay constraint and forwarding table size constraint. After the process, if the aggregated bandwidth of return paths in $\mathcal{P}_i$ can satisfy the demand, we modify network status and add demand's utility W_i into W_{total}.

VII. Performance Evaluation

In this section, we show evaluation results of the performance and effectiveness of our solutions. Mininet [10] for large-scale emulation tests were used to evaluate *suitability* of DEMUR.

A. Large-scale emulation using Mininet

To better evaluating the performances of our scheme, we implemented four solutions using Mininet-based emulations.

- Baseline, which is the de facto protection/restoration scheme by using disjoint working/backup paths.
- DEMUR-MU, Algorithm 4,
- DEMUR-MN, which is the special case of DEMUR-MU with the same utility associated with each flow (both based on ODEMUR approximation),
- DEMUR-Fast, which is the heuristic described in VI-2.

All the emulations were run in Mininet which was installed on a 2.6GHz Intel Core i7 Linux PC with 16GB of memory.

Both *DARPA CORONET* [4] and randomly generated topologies were used to verify the *suitability* of our algorithms. Nodes were randomly generated in a square field of size 10000×10000 square kilometers. The link capacity was set to $40Mbps$ due to the fact that Mininet can support $10Gbps$ at most and it needs to be shared among all the links. Since path latency is highly related to the propagation delay, in Mininet, we set link delay proportional to link distances which can be easily calculated based upon node topology. In order to measure path latency, we *embedded timestamp bytes in the payload of stream*. By parsing timestamp bytes at the destination, path latency can be calculated in the granularity of milliseconds. We set $d_{min} = 0$ and $d_{max} = 160ms$ for the demands, which is compatible with the test data from Amazon EC2. Their utilities are in $[0.5, 1.5]$*unit/Mbps*, bandwidth requests in $[b_{min}, b_{min} + 80]$ *Mbps*, and the reliability requirements are randomly distributed in $[r_{min}\%, (r_{min} + 10)\%]$.

We simulated *multiple flows as a Poisson process* with 5 minutes as mean inter-arrival delay. The lifetime of each flow is randomly distributed in $[50, 120]$ minutes. We tested multiple scenarios by varying network sizes, demands, bandwidth requests, reliability requirements, and forwarding table size.

B. Suitability evaluation

1) Evaluation on CORONET*:* The performance of ODEMUR is listed in Table I compared to DEMUR-Fast. We observed that for all five random demands, the differential delay from ODEMUR is smaller than that from DEMUR-Fast. This supports that ODEMUR has better performance than DEMUR-Fast.

Node A	Node Z	ODEMUR	DEMUR-Fast
Syracuse	New York City	3.38ms	8.68ms
Louisville	Detroit	3.39ms	10.29ms
Albany	Chicago	5.57ms	11.26ms
Boston	Buffalo	5.88ms	18.72ms
Dallas	Las Vegas	13.68ms	36.21ms

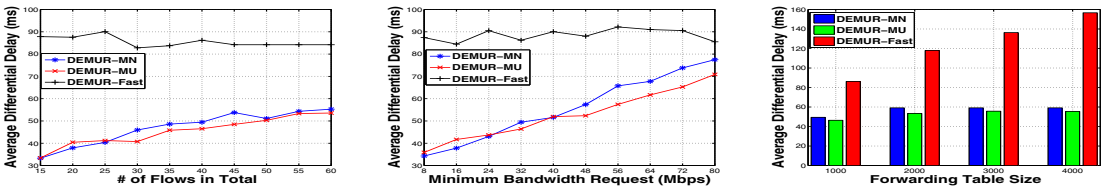
TABLE I: Differential delay with *b=50Mbps, r=50%, ϕ = 0.5, d_min=0, d_max=40ms*

Next, we tested the performance of DEMUR-MU with 120 demands on CORONET topology, listed in Table II. DEMUR-MU generated 28.4% more utility than DEMUR-Fast did and 67.3 more utility than Baseline did.

nodes	Total utility	DEMUR-MU	DEMUR-Fast	Baseline	Outperform
75	12144	10025	6578	1857	28.4%

TABLE II: Demands utility with *n_flow=120, b_min=32Mbps, r=50%, s=1000, d_min=0, d_max=40ms*

2) Evaluation on random topologies: We also evaluated our solutions on large random topologies based on two performance metrics: **average differential delay** of all demands, and **total generated utility**.

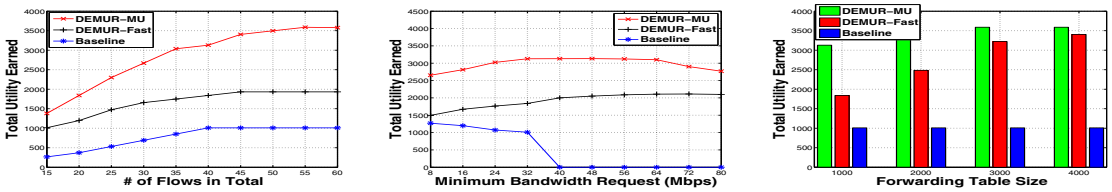


(a) $b_{min} = 32Mbps$, $r = 50\%$, $\phi = 0.5$, $s = 1000$

(b) $n_{flow} = 40$, $r = 50\%$, $\phi = 0.5$, $s = 1000$

(c) $n_{flow} = 40$, $b_{min} = 32Mbps$, $r = 50\%$, $\phi = 0.5$

Fig. 2: Average differential delay of demands



(a) $b_{min} = 32Mbps$, $r = 50\%$, $\phi = 0.5$, $s = 1000$

(b) $n_{flow} = 40$, $r = 50\%$, $\phi = 0.5$, $s = 1000$

(c) $n_{flow} = 40$, $b_{min} = 32Mbps$, $r = 50\%$, $\phi = 0.5$

Fig. 3: Total utility generated

Evaluations of average differential delay among DEMUR-MN, DEMUR-MU and DEMUR-Fast are shown in Fig. 2. The average differential delay results from DEMUR-MN and DEMUR-MU are much lower than those from DEMUR-Fast. In most cases, the average differential delay from DEMUR-MN and DEMUR-MU is less than half of that from DEMUR-Fast. This is because ODEMUR, which is used in DEMUR-MN and DEMUR-MU, is a $(1+\phi)$-approximation, and provides guarantee on maximum path delays. While DEMUR-Fast, applying traditional maximum flow scheme, cannot guarantee the maximum delay.

From Fig. 2(b) we observed that the average differential delay increases with higher demands requests. This is because more (maybe longer) paths would be needed for routing larger traffic demands. Fig. 2(c) evaluated the impact of forwarding table size. One observation is that when forwarding table size increases to a certain level (2000 entries in this case), link capacity constraint becomes the bottleneck, and dictates the outcome. Even with larger forwarding table $(3000, 4000)$, no more demands can be accommodated with better performances due to the link capacity bottleneck.

Evaluations of total generated utility are shown in Fig. 3. As expected, more utilities were generated by DEMUR-MU than DEMUR-Fast and Baseline. In DEMUR-MU, demands with higher utility/request ratio have higher priority. Meanwhile, in DEMUR-Fast and Baseline, the processing order is the arriving order of the demands. So it is possible that demands with large request but low utility can be processed first. Hence, the revenue obtained from DEMUR-Fast and Baseline is limited. From Fig. 3(b), we can see that with higher bandwidth request for each demand, utility earned from DEMUR-MU increased at the beginning, reached a maximum and then decreased. This is because when demands get larger, less number of demands can be accommodated. Consequently, less total utility can be generated even though each demand is large. To make maximum utility, there should be a delicate balance between the size of demands and the number of requests.

VIII. Conclusion

In this work, we studied the problem of multipath routing in a software-defined ISP backbone, considering requirements of reliability, differential delay, and bounded SDN switch forwarding table size. We formulated the Dependable Multipath routing (DEMUR) problem for both a single demand, and multiple demands. We presented a pseudo-polynomial $(1+\phi)$-approximation algorithm for the DEMUR-SD problem, and effective heuristics for the DEMUR-MD problem. Extensive evaluations have been conducted to support our analysis and to confirm the performance of our solutions.

References

[1] S. Acharya, B. Gupta, P. Risbood, and A. Srivastava, PESO: Low Overhead Protection for Ethernet over SONET Transport, *IEEE INFOCOM'2004*, pp. 165-175.

[2] S. Ahuja, M. Krunz, T. Korkmaz, Optimal path selection for minimizing the differential delay in ethernet-over-SONET, *Computer Networks*, Vol. 50, Issue 13, 2006, pp. 2349-2363.

[3] R. Cohen, L. Lewin-Eytan, J. Naor, and D. Raz, On the effect of forwarding table size on SDN network utilization, *Proceedings of IEEE INFOCOM'2014* , pp. 1734-1742.

[4] CONUS Topology; http://monarchna.com/topology.html.

[5] P. Djukic, S. Valaee, Reliable packet transmissions in multipath routed wireless networks *IEEE Transactions on Mobile Computing*, Vol. 5, Issue 5, 2006, pp. 548-559.

[6] Project Flightlight, http://www.projectfloodlight.org.

[7] C.Y. Hong, et al., Achieving High Utilization with Software-Driven WAN, *Proceedings of ACM SIGCOMM'2013*, pp. 15-26.

[8] S. Jain, et al. , B4: experience with a globally-deployed software defined WAN, *Proceedings of ACM SIGCOMM'2013*, pp. 3-14.

[9] D. Lorenz and D. Raz, A Simple Efficient Approximation Scheme for the Restricted Shortest Path Problem, *Operations Research Letters*, Vol. 28, Issue. 5, 2001, pp. 213-219.

[10] Mininet; http://mininet.org.

[11] Open Networking Foundation, OpenFlow Switch Specification, *Version 1.3.4*, March 27, 2014.

[12] Open Switching; www.pica8.com/open-switching.

[13] Christoph Paasch, Olivier Bonaventure. Multipath TCP. *Communications of the ACM*, 57(4):51-57, April 2014.

[14] A. Srivastuva, S. Acharyu, M. Alicherry, B. Gupta and P. Risbood, Differential Delay Aware Routing for Ethernet over SONET/SDH, *IEEE Infocom'2005*, pp.1117-1127.

[15] M. Suchara, D. Xu, R. D. Doverspike, D. Johnson, J. Rexford, Network architecture for joint failure recovery and traffic engineering, *SIGMETRICS 2011*, pp 97-108.

[16] G. Xue, W. Zhang, J. Tang and K. Thulasiraman, Polynomial Time Approximation Algorithms for Multi-Constrained QoS Routing, *IEEE/ACM Transactions on Networking*, Vol. 16, 2008, pp. 656-669.