

Algebra and Algorithms for Efficient and Correct Multipath QoS Routing in Link State Networks

Haijun Geng^{*†‡}, Xingang Shi^{†‡}, Xia Yin^{*†}, Zhiliang Wang^{†‡}, Han Zhang^{*†}

^{*}Department of Computer Science & Technology, Tsinghua University

[†]Institute for Network Sciences and Cyberspace, Tsinghua University

[‡]Tsinghua National Laboratory for Information Science and Technology (TNList)

Abstract—The diversity of QoS (Quality-of-Service) requirements of Internet applications motivates various QoS routing algorithms that take different QoS metrics into consideration. Routing algebra has been proposed as a framework to study the fundamental properties of QoS routing algorithms, such as their optimality and loop-freeness. However, for multipath QoS routing, little has been done in these aspects. Existing multipath QoS routing algorithms often take a rather conservative approach to guarantee loop-freeness, at the cost of efficiency. On the other hand, simply adapting existing efficient multipath routing algorithms to support various QoS metrics cannot guarantee correctness. In face of that, we propose a routing metric algebra for multipath QoS routing in link state networks, where a key property of the routing metrics called isotonicity, which plays an important role. To let routers efficiently and correctly find multiple next-hops for each destination, we also develop two distributed multipath QoS routing algorithms. The algorithms are run locally and independently, without exchanging messages other than the basic link states. They are specifically tailored for algebras with strict or non strict isotonicity, and their correctness are formally proved.

Index Terms—Algebra, Multipath routing, QoS

I. INTRODUCTION

The objective of QoS routing is to meet the service quality requirement of different Internet applications. For this purpose, QoS routing protocol should take different routing metrics, such as hop count, delay, reliability, capacity and throughput [1], into consideration. For example, some routing metrics focus on the reliability of a path, some are concerned about link utilization, while some capture the bandwidth of a path. Routing algebra [2] has been proposed as a framework to study the fundamental properties of QoS routing algorithms, such as their optimality and loop-freeness. Existing multipath QoS routing schemes often take a rather conservative approach to guarantee loop-freeness, at the cost of efficiency. For example, explicit cooperation/signaling schemes are utilized in source routing like schemes [3], where both control and data plane overhead may become concerns. For hop-by-hop routing, a routing algebra based approach [2] computes lexicographic-lightest paths so that packets can always be routed to its final destination without being stuck in any loop. However, the significant overhead in the path computation prohibits its practical usage in real networks. To reduce protocol or algorithm complexity, extending existing efficient multipath routing schemes [4] to support different QoS metrics may be a viable way, but how to guarantee loop-freeness in the extended

schemes is still not clear.

In this work, we aim for efficient and correct hop-by-hop multipath routing algorithms that support various QoS metrics. To guarantee the correctness of the algorithms, we propose a routing metric algebra for link state networks, where a key property of the algebra called isotonicity plays an important role. Based on the routing metric algebra, we further propose two efficient distributed algorithms to help a router in a link state network find multiple next-hops for each destination. Both of the algorithms are run locally and independently on each router, without exchanging messages other than the basic link states. They are specifically tailored for algebras with strict or non strict isotonicity, and their loop-freeness are formally proved.

II. SYSTEM MODEL

A. Network Model

A network can be modeled as a graph $G = (V, E)$, where V and E denote the set of nodes and edges respectively. The link between node u and v is represented by an edge $l(u, v)$, which has a weight $w(u, v)$. A weight may correspond to different characteristics, such as delay, bandwidth, hop count, reliability, energy consumption, etc., or a combination of them. A path in the graph is a sequence of nodes $p = \langle v_n, v_{n-1}, \dots, v_2, v_1 \rangle$ such that $(v_i, v_{i-1}) \in E$ for $1 < i \leq n$. Note that we choose to index the nodes of a path backwards, from last to first, and use $n(p)$ to denote the number of nodes in path p . Let $p[i, j]$ ($1 \leq j \leq i \leq n$) be the subpath from the i -th node to the j -th node of path p , i.e., $p[i, j] = \langle v_i, v_{i-1}, \dots, v_j \rangle$. For example, a path $p = (a, b, c, d)$ can be expressed as $p = (v_4, v_3, v_2, v_1)$, where $v_4 = a, v_3 = b, v_2 = c, v_1 = d$; $p[3, 1] = (v_3, v_2, v_1) = (b, c, d)$. In particular, let $\lambda(p)$ be the subpath of p with only its first node excluded, and $\mu(p)$ the subpath of p with only its last node excluded.

B. Routing Metric Algebra

Now we define our routing metric algebra for multipath QoS routing, where the notations and semantics are heavily borrowed from some earlier works [2], [5]–[8]. The algebra is captured in structures of the form $(S, \circ, w, \oplus, \preceq)$. S is the set of paths; $\circ$ is the path concatenation operator, where if the last node of path p is the first node of path q , then $p \circ q$ denotes the concatenation of p and q ; w is a function that maps a path to its weight (i.e., routing metric priority), such that for

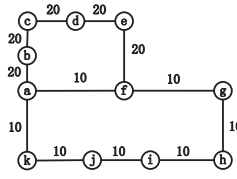


Fig. 1: Network Topology

a path $p = \langle v_n, v_{n-1}, \dots, v_2, v_1 \rangle \in S$, $w(p) = w(v_n, v_{n-1}) \oplus w(v_{n-1}, v_{n-2}) \oplus \dots \oplus w(v_2, v_1)$, where $\oplus$ is a binary operator for path weight; $\preceq$ is an order relation defined on the set of path weights, where $w(p) \preceq w(q)$ (i.e., $w(p)$ is less than or equal to $w(q)$) means p is better or equal to q , and $w(p) \prec w(q)$ (i.e., $w(p)$ is strictly less than $w(q)$) means p is strictly better than q .

The algebra has the following properties.

- 1) $w(p \circ q) = w(p) \oplus w(q)$.
- 2) $\bar{0}$ is the least and identity element in $\{w(p) | p \in S\}$, and $\bar{0} \oplus w(p) = w(p) \oplus \bar{0} = w(p)$. In particular, only when p is a trivial path with only one node, $w(p) = \bar{0}$.
- 3) ∞ is the maximal element in $\{w(p) | p \in S\}$, and $\infty \oplus w(p) = w(p) \oplus \infty = \infty$.

The routing metric algebra is isotonic if for any $p, q, r \in S$, $w(p) \preceq w(q)$ implies $w(p \circ r) \preceq w(q \circ r)$ and $w(r \circ p) \preceq w(r \circ q)$, and is strictly isotonic if $w(p) \prec w(q)$ implies $w(p \circ r) \prec w(q \circ r)$ and $w(r \circ p) \prec w(r \circ q)$. Our routing metric algebra extends earlier algebras [2], [7] by modeling both the routes (paths) and their metrics (path weights), which facilitates developing the QoS routing algorithms later.

C. Paths

There are three types of paths we are interested in, namely *lightest paths*, *S-lightest paths* and *L-lightest paths*. The lightest (i.e., smallest) path weight from s to v is defined as $w^*(s, v) = \min(w(p))$, where p is any path from s to v . A path p from s to v is called a *lightest path* from s to v , which is also denoted by $\rho(s, v)$, if $w(p) = w^*(s, v)$. A path from s to v is called an *S-lightest path* if all of its subpaths with source s are lightest paths on their own. Further, let $\bar{w}(p) = w(p[n, 1])w(p[n, 2]) \dots w(p[n, n-1])$ represent the word-weight of a path $p = \langle v_n, v_{n-1}, \dots, v_2, v_1 \rangle$. For a path p from s to v , we call it an *L-lightest path* from s to v if $\bar{w}(p)$ is lexicographically less than or equal to the word-weight of any other path from s to v .¹

An example is presented in Fig. 1, where the numbers represent link capacities. In particular, the capacity of a path is the minimal link capacity along that path, and for two paths p and q , $w(p) \prec w(q)$ means p has a larger capacity than q . Consider the three paths from a source node a to a destination node g , namely (a, b, c, d, e, f, g) , (a, f, g) , and

(a, k, j, i, h, g) . Their path capacities are all 10, so all of them are lightest paths. Among them, only (a, f, g) is not an S-lightest path, because its subpath (a, f) is not a lightest path from a to f . Their word-weights are 10 20 20 20 20 20, 10 10, and 10 10 10 10 10, respectively. Therefore, path (a, b, c, d, e, f, g) is the only L-lightest path from a to g .

III. WHEN ISOTONICITY IS STRICT

A. Basic Idea

In hop-by-hop routing, each node independently computes its next-hops for all destinations, so in the rest of the paper, our algorithm will be described with respect to a particular node c that performs such kind of computation. This section presents an algorithm that computes multiple next-hops for each destination when the routing metric/algebra is strictly isotonic, while the algorithm in section IV is for non-strictly isotonic metrics/algebras.

The node c builds a lightest path tree T_c rooted at itself, containing all the nodes in the network as potential destinations. We denote the parent of v in T_c by $\pi[v]$. The object of c is to compute a candidate set of next-hops $N_c(v)$ for each destination v ($v \neq c$), so that when a packet destined to v arrives at c , c can select a next-hop from $N_c(v)$ and forward this packet to. In particular, we use $B_c(v)$ to represent the best/default candidate, which lies along the lightest path from c to v , and save it as the first entry in $N_c(v)$ (i.e., $B_c(v) = N_c(v)[0]$). Since T_c is a lightest path tree, $w^*(c, v)$ is the lowest cost from c to v in the network, leading to the following lemma.

Lemma 1: The Best Next-Hop Rule

$$B_c(v) = \begin{cases} v & \pi[v] = c \\ B_c(\pi[v]) & \pi[v] \neq c \end{cases} \quad (1)$$

Equation (1) in Lemma 1 means the best next-hop $B_c(v)$ for a destination v is c 's direct child along the path from c to v in T_c .

To compute a set $N_c(v)$ of next-hops for v , we start with a simple loop-free rule called SDC. SDC rule is proposed in Lemma 2, which is based on a similar condition given in [9], for the case of an additive weight function, which is rephrased as follows:

Lemma 2: (SDC rule)

For packets destined to a destination v , node c ($c \neq v$) can forward them to its any neighboring node x as long as $w^*(x, v) \prec w^*(c, v)$, and there will be no forwarding loop in the induced forwarding path.

Theorem 1: Given a lightest path, the weight of any of its strict subpaths is less than its own weight when the routing metric is strictly isotonic.

Theorem 2: Forwarding packets to the best next-hop satisfies the SDC rule, that is to say $w^*(B_c(v), v) < w^*(c, v)$.

To run SDC rule, each node needs to obtain the lightest weight for its neighbors as well as itself. With a link state protocol, it requires multiple lightest path computations. Therefore the induced cost will be particularly high for high-degree nodes.

¹Given two strings $a = a_0a_1 \dots a_m$ and $b = b_0b_1 \dots b_n$, a is lexicographically less than b if either 1) there exists an integer i , where $0 \leq i \leq \min(m, n)$, such that $a_j = b_j$ for all $j = 0, 1, \dots, i-1$ and $a_i < b_i$, or 2) $m < n$ and $a_i = b_i$ for all $j = 0, 1, \dots, m$.

One of our contribution lies in the following rule, which is slightly more strict than the Theorem 2.

Definition 1: Given any two neighboring nodes u and v in the lightest path tree T_c . Let $p = \rho(c, u)$ be a lightest path in T_c , if

$$R(u, v) = w(\lambda(p) \circ (u, v)) \prec w^*(c, v) \quad (2)$$

, where $w(\lambda(p) \circ (u, v)) = d[u] \oplus w(u, v)$, $d[u] = w(\lambda(p))$, which is the weight of the subpath from the second to last node in a lightest path p , we say u can contribute (its best next-hop) to v .

B. Algorithm

Our MPA algorithm compute the lightest path tree in a way similar to the Generalized Dijkstra's algorithm [2]. We assign a *visited* attribute to each node, so that only when a node is already in the tree, its *visited* attribute is true, and false otherwise. The variable $l[v]$ is called the lightest-path estimate of node v , which is an upper bound on the weight of a lightest path from source c to v . MPA maintains a priority queue $Q(v, m, t)$ to select the proper node to be added to the tree, where m and t denote respectively the tentative parent of node v and the weight of path from c to v . The function $\text{Enqueue}(Q, \langle v, m, t \rangle)$ pushes a node into Q , if v already exists in Q , m and t will be updated only when the new tentative weight is smaller. The function $\text{ExtractMin}(Q)$ returns the node which has the smallest tentative weight in Q .

After the initiation step (lines 1-7), $\pi[v] = \text{NIL}$ for all $v \in V$, $l[c] = \bar{0}$, and $l[v] = \infty$ for $v \in V - \{c\}$, node c is added to the T_c . To build lightest path tree and compute multiple next-hop sets, the algorithm go through several iterations. In each iteration, a node v with the lightest path weight will be extracted from Q and added to the T_c . The best next-hop and $d[v]$ is updated (lines 9-19). The lines 20-24 performs a relaxation step on edge (v, u) . The relaxation on edge (v, u) consists of testing whether we can improve the lightest path to u found so far by going through v and, if so, updating $l[u]$ and $\pi[u]$. At last, we will update the next-hop set for u and v according to equation 2 (lines 26-29).

Theorem 3: The complexity of Algorithm 1 is $O(|V| \lg |V| + |E|)$ when the queue is implemented as a Fibonacci heap.

Lemma 3: At all times during the execution of the MPA the predecessor subgraph is a tree rooted at c .

The proof for lemma 3 is similar with Dijkstra's algorithm [6]. Therefore, the proof is omitted.

Theorem 4: If Algorithm 1 is run at source node c , the predecessor subgraph is a tree of lightest paths rooted at c .

Theorem 5: For packets destined to a destination v , if any node c ($c \neq v$) forwards them to any node in its next-hop set $N_c(v) = \{B_c(v)\} \cup \{B_c(u) | u \text{ can contribute to } v\}$, there will be no loop in the resulting forwarding path.

IV. WHEN ISOTONICITY IS NOT STRICT

A. Basic Idea

To compute a set $N_c(v)$ of next-hops for v when the isotonicity is not strict, we start with a loop-free rule called

Algorithm 1: MPA(G, c)

```

1: for  $v \in V$  do
2:    $l[v] \leftarrow \infty$ 
3:    $\pi[v] \leftarrow \text{NIL}$ 
4:    $v.\text{visited} \leftarrow \text{False}$ 
5:  $c.\text{visited} \leftarrow \text{True}$ 
6:  $l[c] \leftarrow \bar{0}$ 
7:  $\text{Enqueue}(Q, \langle c, \text{NIL}, \bar{0} \rangle)$ 
8: while  $Q$  is not empty do
9:    $\langle v, m, t \rangle \leftarrow \text{ExtractMin}(Q)$ 
10:  if  $v \neq c$  then
11:     $v.\text{visited} \leftarrow \text{True}$ 
12:     $\pi[v] \leftarrow m$ 
13:     $l[v] \leftarrow t$ 
14:    if  $\pi[v] = c$  then
15:       $d[v] \leftarrow \bar{0}$ 
16:       $B_c(v) \leftarrow v$ 
17:    else
18:       $B_c(v) \leftarrow B_c(m)$ 
19:       $d[v] \leftarrow d[m] \oplus w(m, v)$ 
20:  for each node  $u \in N(v)$  do
21:    if  $u.\text{visited} = \text{False}$  then
22:       $\text{newdist} \leftarrow l[v] \oplus w(v, u)$ 
23:      if  $\text{newdist} \prec l[u]$  then
24:         $\text{Enqueue}(Q, \langle u, v, \text{newdist} \rangle)$ 
25:    if  $u.\text{visited} = \text{True} \wedge B_c(u) \neq B_c(v)$  then
26:      if  $v$  contributes to  $u$  then
27:        Add  $B_c(v)$  to  $N_c(u)$ 
28:         $u$  contributes to  $v$ 
29:        Add  $B_c(u)$  to  $N_c(v)$ 

```

LDC, which is expressed as follows:

Theorem 6: (LDC rule)

For packets destined to a destination v , node c ($c \neq v$) can forward them to any of its neighboring nodes x as long as $\bar{w}(\ell(x, v)) \prec \bar{w}(\ell(c, v))$ and there will be no forwarding loop in the induced forwarding path.

Theorem 7: Forwarding packets to the best next-hop satisfies the LDC rule, that is to say $\bar{w}(\ell(B_c(v), v)) \prec \bar{w}(\ell(c, v))$. The Theorem 7 has been proved in [2].

Definition 2: Given any two neighboring nodes u and v in the L-lightest path tree L_c , for two L-lightest paths $p = \ell(c, u)$ and $q = \ell(c, v)$ in L_c , if

$$D(u, v) = \bar{w}(\lambda(p) \circ (u, v)) \prec \bar{w}(q) \quad (3)$$

, where $\bar{w}(\lambda(p) \circ (u, v)) = t[u] \oplus w(u, v)\bar{w}(\lambda(p))$, $t[u] = w(\lambda(p))$, which is the weight of the subpath from the second to last node in a L-lightest path p , we say u can contribute (its best next-hop) to v .

Theorem 8: (The Extended Next-Hop Contribution Rule)

If u can contribute to v , then $\bar{w}(\ell(B_c(u), v)) \prec \bar{w}(\ell(c, v))$, i.e., the best next-hop for u is a viable next-hop for v as suggested by the DC rule.

B. Algorithm

We propose Algorithm 2 to compute a L-lightest path tree and next-hop set from a source node c to every other nodes. Node c compute a L-lightest path tree L_c rooted at itself, containing all the nodes in the network as potential destinations. We assign a sequence number attribute to each node denoted by seq , which records the order of the node being added to the tree. LMPA maintains a priority queue $Q(v, m, t)$ to select the proper node to be added to the tree, where m and t denote respectively the tentative parent of node v and the weight of path from s to v . The function $Enqueue(Q, < v, m, t >)$ pushes an node into Q , if v already exists in Q , p and d will be updated only when the new tentative weight is smaller, or when the weight is the same but the new tentative parent has a smaller sequence number. The function $ExtractMin(Q)$ returns the node which has the

Algorithm 2: LMPA(G, c)

```

1: for  $v \in V$  do
2:    $l[v] \leftarrow \infty$ 
3:    $\pi[v] \leftarrow NIL$ 
4:    $v.visited \leftarrow False$ 
5:  $c.visited \leftarrow True$ 
6:  $l[c] \leftarrow \bar{0}$ 
7:  $sequence \leftarrow 1$ 
8:  $Enqueue(Q, < c, NIL, \bar{0} >)$ 
9: while  $Q$  is not empty do
10:   $< v, m, t > \leftarrow ExtractMin(Q)$ 
11:   $v.seq \leftarrow sequence$ 
12:   $sequence \leftarrow sequence + 1$ 
13:  if  $v \neq c$  then
14:     $v.visited \leftarrow True$ 
15:     $\pi[v] \leftarrow m$ 
16:     $l[v] \leftarrow t$ 
17:    if  $\pi[v] = c$  then
18:       $B_c(v) \leftarrow v$ 
19:       $\kappa[v] \leftarrow l[v]$ 
20:       $t[v] \leftarrow \bar{0}$ 
21:       $\varphi[v] \leftarrow \bar{0}$ 
22:    else
23:       $B_c(v) \leftarrow B_c(m)$ 
24:       $t[v] \leftarrow r[m] \oplus w(m, v)$ 
25:       $\kappa[v] \leftarrow l[v] \kappa[m]$ 
26:       $\varphi[v] \leftarrow t[v] \varphi[m]$ 
27:  for each neighbor  $u \in N(v)$  do
28:    if  $u.visited = False$  then
29:       $newdist \leftarrow l[v] \oplus w(v, u)$ 
30:      if  $newdist < l[u]$  then
31:         $Enqueue(Q, < u, v, newdist >)$ 
32:      if  $u.visited = True \wedge B_c(u) \neq B_c(v)$  then
33:        if  $v$  contributes to  $u$  then
34:          Add  $B_c(v)$  to  $N_c(u)$ 
35:        if  $u$  contributes to  $v$  then
36:          Add  $B_c(u)$  to  $N_c(v)$ 

```

smallest tentative weight in Q , where its parent's sequence number is used as a tie breaker in case of ties.

Theorem 9: The complexity of Algorithm 2 is $O(|V| * \lg(|V|) + |E| + D * |V|^2)$ when the queue is implemented as a Fibonacci heap.

Theorem 10: For packets destined to a destination v , if any node c ($c \neq v$) forwards them to any node in its next-hop set $N_c(v) = \{B_c(v)\} \cup \{B_c(u) | u \text{ can contribute to } v\}$, there will be no loop in the resulting forwarding path.

V. THE OPTIMAL INCREMENTAL DEPLOYMENT PROBLEM

In this section, we will address the partial deployment problem. A node-by-node incremental deployment scheme is highly preferred. The main objective of MPA and LMPA is to achieve high network availability. We formally define the network availability $A(G, M)$ as follows (similar to that in [10]), and use it as a main metric to evaluate the capability of our scheme.

The end-to-end availability of a source-destination (s - d) pair is defined as the probability that the packets can be correctly forwarded from s to d . Assume that there exist n different forwarding paths from s to d , the i -th which is denoted by $p_i(s, d)$. Further, let the event that $p_i(s, d)$ works be denoted by $A_i(s, d)$, whose probability can be expressed as:

$$P(A_i) = \prod_{\forall (m,n) \in P_i(s,d)} r(m,n) \quad (4)$$

where $r(m, n)$ is failure probability of edge (m, n) .

According to the Inclusion-Exclusion principle [11], the end-to-end availability of a source-destination pair can be expressed as:

$$A(s, d, M) = \sum_{k=1}^n (-1)^{(k-1)} S_k \quad (5)$$

where S_k denote the sum of the probabilities that a unique set of k paths from s to d are simultaneously working, which is further expressed as:

$$\begin{aligned} S_k &= \sum_{i < j < \dots < k} P(A_i \cap A_j \dots \cap A_k) \\ &= \sum_{i < j < \dots < k} \left(\prod_{(m,n) \in P_i(s,d) \cup P_j(s,d) \cup \dots \cup P_k(s,d)} r(m,n) \right) \end{aligned}$$

Then, the network availability can be computed as

$$A(G, M) = \frac{\sum_{s,d \in V, s \neq d} A(s, d)}{|V| * (|V| - 1)} \quad (6)$$

The optimal deployment can be defined as:

Optimal Deployment: Given a graph $G(V, E)$ and a positive integer k , our objective is to find a deployment M where, $M \subseteq V$ and $|M|=k$ such that network availability is maximized.

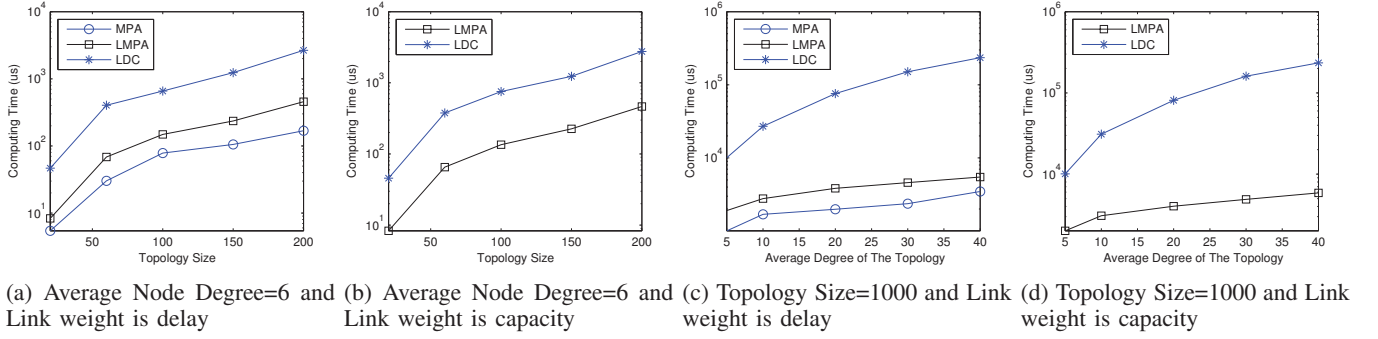


Fig. 2: Computing Time on Different Topologies

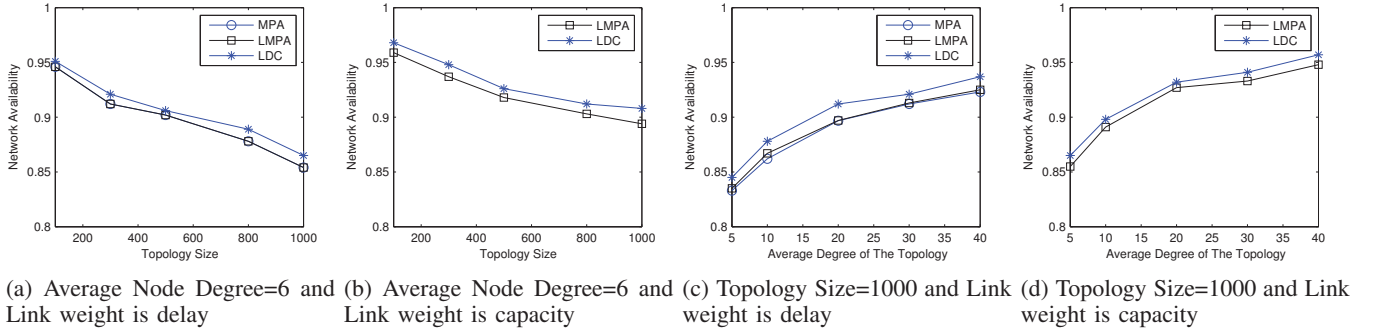


Fig. 3: Network Availability on Different Topologies

Algorithm 3: OptimalDeploy(G, m)

```

1:  $M \leftarrow \emptyset$ 
2: while  $|M| \leq k$  do
3:    $a \leftarrow \max(B(a, M))$ 
4:    $M \leftarrow M \cup \{a\}$ 
5: return  $S$ 

```

The benefit of node a with respect to M , which is the nodes have been already deployed, is denoted by $B(a, M)$, which can be expressed as:

$$B(a, M) = A(G, M \cup \{a\}) - A(G, M) \quad (7)$$

It has been proved in [12], [13] that finding the optimal deployment is NP-Complete. We propose a simple greedy heuristic algorithm *OptimalDeploy()* for selecting nodes to be deployed. In each iteration, we select a node which has the maximum benefit to be deployed (line 3).

VI. PERFORMANCE EVALUATION

In this section, we present our evaluation methods as well as results on both real and synthetic topologies. We compare the results achieved by MPA, LMPA against LDC, which requires L-lightest path tree computation for each neighbor. To evaluate our algorithm, we use the real Abilene network, and four other ISP topologies inferred by Rocketfuel [14], including Exodus, Telstra, Tiscali and Sprint. We also generate synthetic topologies with BRITE [15], using parameters as listed in

TABLE I: Parameters for BRITE Topology Generation

Model	N	HS	LS
Waxman	20-1000	1000	100
m	NodePlacement	GrowthTypem	α
2-40	Random	Incremental	0.15
β	BWDist	BwMin	BwMax
0.2	Constant	10.0	1024.0

Table I. In order to show the impact of the routing metric for these algorithms, we employ delay and capacity as link weight respectively on synthetic topologies. For similarity, we use a simple model to characterize link failure events. The fail probability of each link e is randomly generated in the range from 0 to 0.02. All the simulations are conducted on a PC with Intel i5 CPU at 1.7GHz and 1.5G Memory.

A. Computation Time

First, we compare the computation efficiency of these algorithms. For each topology, the final result for each algorithm is its computation time averaged on all nodes.

The computation time of MPA, LMPA and LDC on different ISP topologies is listed in Table II, it is obvious that MNP has a higher performance than that of LMPA and LDC, especially when the network is larger. For example, in the Sprint topology, MPA uses 165.33 μ s, while LMPA and LDC's complexity are two and ten times higher, respectively. This is due to the single tree construction in MPA and LMPA, whose

complexity is in the same order of that in OSPF, while LDC need to compute multiple trees.

TABLE II: Computation time for Real Topologies

	Network	Computation time (μ s)		
		MPA	LMPA	LDC
Real	Abilene	6.75	7.04	8.98
	Exodus	52.13	78.35	178.84
	Telstra	56.78	121.34	256.32
	Tiscali	84.35	235.34	878.45
	Sprint	165.33	435.63	1576.34

The impact of the network topology on the computing time is further illustrated in Fig. 2 using synthesized topologies generated by BRITE. Fig. 2(a), 2(c) and Fig. 2(b), 2(d) depict the corresponding computation time of each scheme using delay and capacity as routing metric, respectively. From Fig. 2, we can see that when the topology size or the average node degree increases, the average computation time increases accordingly. From Fig. 2, we can see that the average node degree has a remarkable influence on LDC but has little influence on LMPA. The reason is that on each node LDC have to compute a L-lightest path tree for each neighbor, while LMPA only need to compute a L-lightest path. In a large network with dense connections, i.e., when $|V| = 1000$ with an average node degree of 40, LDC may be several orders slower than LMPA.

B. Network Availability

We use network availability as our main evaluation criterion. We first study the network availability with different topology sizes generated by BRITE. From Fig. 3(a) and Fig. 3(b), we can come to the conclusion that as the topology size increases, network availability decreases too. MPA, LMPA and LDC can provide comparable performance.

We also investigate how the network availability will be affected when the average node degree of a topology increases, as shown in Fig. 3(c) and Fig. 3(d). It is obvious that when nodes have more neighbors, the network availability increase too. MPA, LMPA and LDC can also provide comparable performance.

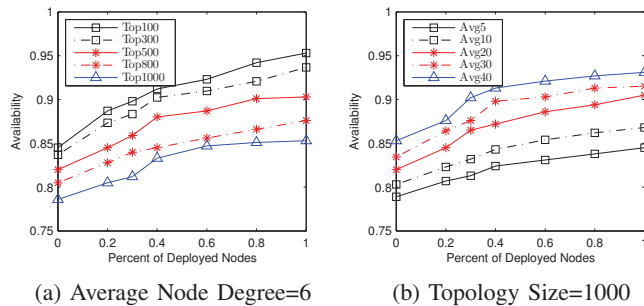


Fig. 4: Partial Deployment Results

C. Partial Deployment

Due to space limitations, we only present the results for MPA on different topology size and different average node

degree, while the results for LMPA are similar. Fig. 4(a) and Fig. 4(b) shows the impact of deployed ratios on different topology size and different average node degree using MPA respectively. From Fig. 4 we can see that as the node deployed ratios increases, the network availability increase very fast. We can see that when a deployment on only 40% nodes already increase the network availability clearly.

VII. CONCLUSION

We propose a routing metric algebra for Multipath QoS routing in the network. Based on the routing metric algebra, we propose two distributed multipath QoS routing algorithms to help a router in a link state network find multiple next-hops for each destination. They are specifically tailored for algebras with strict or non strict isotonicity, and their correctness are formally proved.

VIII. ACKNOWLEDGE

This work is partially supported by the National High Technology Research and Development Program of China (863 Program) No. 2015AA015603, the National Natural Science Foundation of China (Grant No. 61402253, 61373161), and the Project for 2012 Next Generation Internet technology research and development, industrialization, and large scale commercial application of China (No. 2012 1763).

REFERENCES

- [1] J. T. Moy, *OSPF: anatomy of an Internet routing protocol*. Addison-Wesley Professional, 1998.
- [2] J. a. L. Sobrinho, "Algebra and algorithms for qos path computation and hop-by-hop routing in the internet," *IEEE/ACM Trans. Netw.*, vol. 10, no. 4, pp. 541–550, Aug. 2002.
- [3] P. K. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, D. Wetherall *et al.*, "Improving the reliability of internet paths with one-hop source routing," in *OSDI*, vol. 4, 2004, pp. 13–13.
- [4] S. Gjessing, "Implementation of two resilience mechanisms using multi topology routing and stub routers," pp. 29–29, 2006.
- [5] H. Geng, X. Shi, X. Yin, and Z. Wang, "Dynamic distributed algorithm for computing multiple next-hops on a tree," in *ICNP*, 2013, pp. 1–10.
- [6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.
- [7] Y. Yang and J. Wang, "Design guidelines for routing metrics in multi-hop wireless networks," in *INFOCOM 2008. The 27th conference on computer communications. IEEE*. IEEE, 2008.
- [8] S. Dwyerowicz and T. G. Griffin, "On the forwarding paths produced by internet routing algorithms," in *ICNP*, 2013, pp. 1–10.
- [9] X. Yang and D. Wetherall, "Source selectable path diversity via routing deflections," in *SIGCOMM*. ACM, 2006, pp. 159–170.
- [10] R. Terruggia, "Reliability analysis of probabilistic networks," *PhD Thesis*, 2010.
- [11] W. Feller, "An introduction to probability theory and its applications," *Wiley series in probability and mathematical statistics.*, 1971.
- [12] D. S. Hochbaum *et al.*, *Approximation algorithms for NP-hard problems*. PWS publishing company Boston, 1997, vol. 20.
- [13] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 7–12.
- [14] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring isp topologies with rocketfuel," *IEEE/ACM Transactions on Networking*, vol. 12, no. 1, pp. 2 – 16, 2004.
- [15] A. Medina, A. Lakhina, I. Matta, and J. Byers, "Brite: An approach to universal topology generation," in *IEEE International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems - Proceedings*, 2001, pp. 346–353.