# Algebra and Algorithms for QoS Path Computation and Hop-by-Hop Routing in the Internet

João Luís Sobrinho
Instituto de Telecomunicações, Instituto Superior Técnico
Torre Norte, Av. Rovisco Pais 1
1049-001 Lisboa, PORTUGAL
Tel: +351 21 841 8458
Fax: +351 21 841 8472
Email: joao.sobrinho@lx.it.pt

*Abstract*— **Prompted by the advent of QoS routing in the Internet, we investigate the properties that path weight functions must have so that hop-by-hop routing is possible and optimal paths can be computed with a generalized Dijsktra's algorithm. For this purpose we define an algebra of weights which contains a binary operation, for the composition of link weights into path weights, and an order relation. Isotonicity is the key property of the algebra. It states that the order relation between the weights of any two paths is preserved if both of them are either prefixed or appended by a common, third, path.**

**We show that isotonicity is both necessary and sufficient for a generalized Dijkstra's algorithm to yield optimal paths. Likewise, isotonicity is also both necessary and sufficient for hop-by-hop routing. However, without strict isotonicity, hop-by-hop routing based on optimal paths may produce routing loops. They are prevented if every node computes what we call lexicographic-optimal paths. These paths can be computed with an enhanced Dijkstra's algorithm that has the same complexity as the standard one. Our findings are extended to multipath routing as well.**

**As special cases of the general approach, we conclude that shortest-widest paths can neither be computed with a generalized Dijkstra's algorithm nor can packets be routed hop-by-hop over those paths. In addition, loop-free hop-by-hop routing over widest and widest-shortest paths requires that each node computes lexicographic-optimal paths, in general.**

## I. Introduction

Hop-by-hop and shortest-path routing are twin quintessences of Internet routing protocols [1]. Hop-by-hop routing means that forwarding decisions are made independently at each node based only on the destination addresses of incoming packets, and on path computations performed locally at the node. In shortest-path routing, the path computations performed locally at each node are such as to make packets travel over paths that minimize an additive weight function, often with delay-related semantics. Hop-by-hop and shortest-path routing are also key components in minimum delay routing [2], [3].

Metrics other than delay-related are of fundamental importance in the Internet, both for conventional datagram operation routing best-effort traffic, and for virtual circuit operation routing flows with strict Quality-of-Service (QoS) requirements. For datagram operation, link utilization is proposed in [4] as an adequate metric to deal with congestion, and, in [5], link utilization is used with advantage to route traffic belonging to classes with different QoS requirements. The popular Open Shortest Path First (OSPF) protocol also has provisions to route packets along different types of paths, including maximum throughput paths [6]. Although the semantics of link utilization and throughput would seem to call for a bottleneck weight function, where the weight of a path equals the weight of its bottleneck link, an additive weight function is used instead in the previous examples. As a matter of fact, our investigations show that routing loops may arise if the path computation algorithms used for additive weight functions are blindly transposed to bottleneck weight functions.

Most recent studies of QoS routing in the Internet [7], [8] presuppose a virtual circuit mode of operation, whereby flows can be pinned to paths using, for example, label-switching techniques [9]. In this framework, the relevance and performance of a variety of paths for routing flows have been addressed, including widest paths [10], [11], widest-shortest paths [12], [13], [14] and shortest-widest paths [10], [13], among others. A widest path is one of maximum available bandwidth, with bandwidth predicating a bottleneck weight function; a widest-shortest path is a widest path among the set of shortest paths between two nodes; and, conversely, a shortest-widest path is a shortest path among the set of widest paths between two nodes. In virtual circuit operation, hop-by-hop routing plays a role in setting-up paths for new flows [15]. For example, hop-by-hop routing allows for an expedient exploration of several paths at flow set-up time without crankback to the source every time resources cannot be reserved along the initial chosen path.

In this work, we consider intra-domain routing based on link-state protocols, such as OSPF, for which we can assume that all nodes have a topology database of the network. We investigate the properties that a path weight function must have so that hop-by-hop routing is possible and optimal paths can be computed locally at each node with simple variants of Dijkstra's algorithm [16]. To address this question, we present an algebra formed by a set of weights equipped with a binary operation and an order relation. The binary operation gives the weight of a path from the weights of its constituent links, and the order relation compares the weights of different paths. The binary operation and order relation are intertwined by the isotone property which essentially states that the order relation between the weights of any two paths is preserved if both of them are either

prefixed or appended by a common, third, path.

We show that, in the presence of isotonicity, a generalized Dijkstra's algorithm with "addition" substituted by the binary operation and "less than or equal" substituted by the order relation correctly determines optimal paths. Conversely, without isotonicity, the generalized Dijkstra's algorithm does not, in general, determine optimal paths. Interestingly, isotonicity is also both necessary and sufficient for hop-by-hop routing of packets over optimal paths. However, in the absence of strict isotonicity—a stronger form of isotonicity—hop-by-hop routing may not be loop-free. Moreover, inducing Dijkstra's algorithm to compute optimal paths with the minimum number of hops does not solve the problem either. We show that, in this case, loop-freedom is guaranteed if each node computes lexicographic-optimal paths, and we present a variant of Dijkstra's algorithm which computes those paths and has the same computational complexity as the standard algorithm. Our results are extended to multipath routing as well. A general loop-free condition is presented and used to produce alternative optimal paths to reach every destination.

As special cases of the general approach presented in this work, we conclude that shortest-widest paths cannot be computed with a generalized Dijkstra's algorithm and that hop-by-hop routing cannot be used over shortest-widest paths, whatever the algorithm used to compute those paths. In addition, nodes should compute lexicographic-optimal paths—rather than simply optimal paths—when routing packets hop-by-hop over widest or widest-shortest paths, to ensure loop-freedom.

Algebras for solving path problems have been known for quite some time [17], and ours is a special case in which an order relation can be defined among its elements. In contrast to [17], which follows a matrix approach to solving path problems with emphasis on finding the weights of optimal paths, rather than on finding the optimal paths themselves, we follow an algorithmic approach to finding optimal paths, and present both necessary and sufficient conditions for the correctness of the algorithms. More importantly, we construe on the application of the algebra to hop-by-hop routing and present a variant of Dijkstra's algorithm that guarantees loop-freedom. To the best of our knowledge, this is the first time that hop-by-hop routing is studied in the context of a generic algebra.

The paper is organized as follows. Section II presents the properties of the algebra and discusses a number of examples relevant to QoS routing in the Internet. Computation of optimal paths is addressed in Section III, and the implications of the algebra for hop-by-hop routing are discussed in Section IV. Section V extends the discussion to multipath routing.

## II. Algebra

### A. Properties

A network is modeled as a strongly connected directed graph $G = (V, E)$, where $V$ and $E$ denote the node and link sets, with cardinalities $|V|$ and $|E|$, respectively. If $(u, v) \in V$, we say that node $v$ is an out-neighbor of node $u$. The set of out-neighbors of node $u$ is denoted by $N(u)$. A walk is a sequence of nodes $\langle v_n, \ldots, v_2, v_1 \rangle$ such that $(v_i, v_{i-1}) \in E$ for every $i = 2, 3, \ldots, n$. Note that we choose to index the nodes of a walk backwards, from last to first. Walk $\langle v_n, \ldots, v_2, v_1 \rangle$ is a path if all nodes are distinct, and it is a cycle if $v_1 = v_n$ and nodes $v_n, \ldots, v_3, v_2$ are distinct. If the last node of walk $q$ coincides with the first node of walk $p$, then $q \circ p$ denotes the walk formed by the concatenation of $q$ and $p$. Given a walk $p$, the subwalk of $p$ that extends from the $i$th to the $j$th last nodes of $p$ is denoted by $p_{ij}$. For example, if $p = \langle v_n, \ldots, v_2, v_1 \rangle$, then $p_{ij} = \langle v_i, \ldots, v_{j+1}, v_j \rangle$, for $1 \leq j \leq i \leq n$. Clearly, $p = p_{n1}$, and $p_{ii}$, $1 \leq i \leq n$, is the empty walk.

The algebra we will be working with is defined as a set of weights, $S$, equipped with a binary operation, $\oplus$, and an order relation, $\preceq$, which have properties **A1**–**A4**, **O1**–**O4**, **AO1** and **AO2**, discussed in the sequel. The binary operation gives the weight of a walk from the weights of its constituent links. Formally, the following conditions are needed.

**A1** $S$ is *closed* under $\oplus$: $a \oplus b \in S$ for all $a, b \in S$
**A2** $\oplus$ is *associative*: $a \oplus (b \oplus c) = (a \oplus b) \oplus c$ for all $a, b, c \in S$
**A3** There is an *identity*, $\bar{0}$: $a \oplus \bar{0} = \bar{0} \oplus a = a$ for all $a \in S$
**A4** There is a *null element*, $\bar{\infty}$: $a \oplus \bar{\infty} = \bar{\infty} \oplus a = \bar{\infty}$ for all $a \in S$

The weight of the nonempty walk $p = \langle v_n, \ldots, v_2, v_1 \rangle$ is given by

$$w(p) = w(v_n, v_{n-1}) \oplus \cdots \oplus w(v_3, v_2) \oplus w(v_2, v_1),$$

where $w(x, y)$ is the weight of link $(x, y)$; the weight of the empty walk is $\bar{0}$. From these definitions, we easily conclude that $w(q \circ p) = w(q) \oplus w(p)$. In general, links can have weight $\bar{0}$. On the other hand, the null element, $\bar{\infty}$, serves only to indicate the absence of a walk between two nodes and is never used on links.

In order to compare the weights of different walks, a total order is defined on $S$, denoted by $\preceq$ ($a \preceq b$ is read as "$a$ is less than or equal to $b$").

**O1** $\preceq$ is *reflexive*: $a \preceq a$ for all $a \in S$
**O2** $\preceq$ is *anti-symmetric*: if $a \preceq b$ and $b \preceq a$ then $a = b$
**O3** $\preceq$ is *transitive*: if $a \preceq b$ and $b \preceq c$ then $a \preceq c$
**O4** For every $a, b \in S$ either $a \preceq b$ or $b \preceq a$

We let $a \prec b$ (read as "$a$ is less than $b$") mean $a \preceq b$ and $a \neq b$, and $a \succ b$ (read as "$a$ is greater than $b$") mean $b \prec a$. The next two properties intertwine the binary operation and order relation defined on $S$, and they are essential to the development of the theory.

**AO1** $\oplus$ is *isotone* for $\preceq$: $a \preceq b$ implies both $a \oplus c \preceq b \oplus c$ and $c \oplus a \preceq c \oplus b$ for all $a, b, c \in S$
**AO2** $\bar{0}$ is a *least element*: $\bar{0} \preceq a$ for all $a \in S$

A stronger form of isotonicity is defined by the following condition.

**AO1-S** $\oplus$ is *strictly isotone* for $\preceq$: $a \prec b$ implies both $a \oplus c \prec b \oplus c$ and $c \oplus a \prec c \oplus b$ for all $a, b \in S$ and $c \in S - \{\bar{\infty}\}$

TABLE I

EXAMPLES OF THE ALGEBRA. ROWS MARKED WITH ‡ INDICATE THAT ISOTONICITY IS NOT STRICT.

| $S$ | $\oplus$ | $\bar{0}$ | $\bar{\infty}$ | $\preceq$ | Problem |
|---|---|---|---|---|---|
| $\{0,1\}$ | $\min$ | 1 | 0 | $\geq$ | Connectivity |
| $R_0^+ \cup \{\infty\}$ | $+$ | 0 | $\infty$ | $\leq$ | Shortest path |
| $R_0^+ \cup \{\infty\}$ | $\min$ | $\infty$ | 0 | $\geq$ | Widest path‡ |
| $[0,1]$ | $\times$ | 1 | 0 | $\geq$ | Most-reliable path |
| $\{(d,b) \mid d \in R_0^+, b \in R_0^+ \cup \{\infty\}\} \cup \{(\infty,*)\}$ | $(d_1+d_2, \min(b_1,b_2))$ | $(0,\infty)$ | $(\infty,*)$ | $d_1 < d_2$ or $d_1 = d_2$ and $b_1 \geq b_2$ | Widest-shortest‡ path |
| $\{(d,p) \mid d \in R_0^+, p \in [0,1] \cup \{(\infty,*)\}$ | $(d_1+d_2, p_1 \times p_2)$ | $(0,1)$ | $(\infty,*)$ | $d_1 < d_2$ or $d_1 = d_2$ and $p_1 \geq p_2$ | Most-reliable-shortest path |

The lightest-path[1] weight from $s$ to $v$, denoted by $\delta(s,v)$, is defined as the least element of the set $\{w(p) \mid p$ is a path from $s$ to $v\}$. Any path $p$ from $s$ to $v$ with weight $w(p) = \delta(s,v)$ is called a lightest path, or, alternatively, an optimal path, from $s$ to $v$. Condition **AO2** implies that the weight of any cycle is greater than or equal to $\bar{0}$ and, thus, the lightest-path weight from $s$ to $v$ is also the lightest-walk weight from $s$ to $v$.

### B. Examples and counterexamples

Table I shows concrete examples of the algebra, relevant to QoS path computation and hop-by-hop routing in the Internet. The usual name of the lightest path is given in the last column. For example, a widest path is a path of maximum capacity; the capacity of a path is the minimum over the available bandwidths of all the links that comprise the path. A widest-shortest path is a widest path among the set of shortest paths between a source and a destination. In this case, the null element is denoted by $(\infty,*)$, where "$*$" stands for "don't care," so that $(d,b) \oplus (\infty,*) = (\infty,*) \oplus (d,b) = (\infty,*)$. A most-reliable-shortest path is a most-reliable path among the set of shortest paths between a source and a destination.

Isotonicity is the only nontrivial property of the algebra. In the table, we have marked with ‡ the examples for which isotonicity is not strict. For instance, in order to find widest-shortest paths, weights are pairs of the form $(d,b)$, with the binary operation and order relation as defined in the table. We have $(0,10) \prec (0,5)$, whereas $(0,10) \oplus (1,5) = (1,5) \oplus (0,10) = (1,5) = (0,5) \oplus (1,5) = (1,5) \oplus (0,5)$.

A noted absence from Table I is a row corresponding to shortest-widest paths. Suppose weights are pairs of the form $(b,d)$, with $(b_1,d_1) \oplus (b_2,d_2) = (\min(b_1,b_2), d_1+d_2)$, and $(b_1,d_1) \preceq (b_2,d_2)$ if $b_1 > b_2$, or $b_1 = b_2$ and $d_1 < d_2$. Then, a lightest path is a shortest-widest path. However, the isotone property fails: $(10,2) \prec (5,1)$, whereas $(10,2) \oplus (5,1) = (5,1) \oplus (10,2) = (5,3) \succ (5,2) = (5,1) \oplus (5,1)$.

## III. QOS PATH COMPUTATION

### A. Optimality of subpaths

The correctness of Dijkstra's algorithm for finding shortest paths in graphs relies on the following optimality principle, which is valid only under strict isotonicity: any subpath of an optimal path is an optimal path. Figure 1 shows that this principle may not hold in the absence of strict isotonicity. In the figure, links are labeled with available bandwidth. Path $\langle s,x,t \rangle$ is a widest path from $s$ to $t$, but its subpath $\langle s,x \rangle$ is not a widest path from $s$ to $x$. However, a weaker optimality principle holds even without strict isotonicity: we show in Theorem 1 that for every $s,v \in V$, there is at least a lightest path from $s$ to $v$ such that all of its subpaths with source $s$ are also lightest paths. As it turns out, the weaker optimality principle is sufficient to establish the correctness of Dijkstra's algorithm.

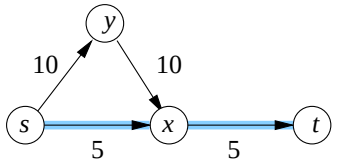


Fig. 1. The widest path $\langle s,x,t \rangle$ has a subpath, $\langle s,x \rangle$, which is not a widest path.

We define an S-lightest path from $s$ to $v$ to be a lightest path from $s$ to $v$ such that all of its subpaths with source $s$ are lightest paths on their own. An S-lightest path is always a lightest path but, as Figure 1 shows, the converse statement is not true in general.

*Lemma 1:* Suppose $p = \langle v_n, \ldots, v_2, v_1 \rangle$ is a path from $v_n$ to $v_1$ that is not an S-lightest path, and let $v_k$, $1 \leq k < n$, be the last node along $p$ for which $p_{nk}$ is not a lightest path. If $q$ is any lightest path from $v_n$ to $v_k$ then the walk $q \circ p_{k1}$ is a path, that is, all its nodes are distinct.

*Proof:* If the walk $q \circ p_{k1}$ is not a path, there is an index $j$, $1 \leq j < k$, such that $v_j$ is a node of $q$. Denoting by $q'$ the subpath of $q$ that extends from $v_n$ to $v_j$ we have, successively,

$$\begin{aligned} w(q') &\preceq w(q) = \delta(v_n, v_k) \\ &\prec w(p_{nk}) \preceq w(p_{nj}) = \delta(v_n, v_j). \end{aligned}$$

This is a contradiction, since the weight of path $q'$, from $v_n$ to $v_j$, cannot be less than the lightest-path weight from $v_n$ to $v_j$, $\delta(v_n, v_j)$. ∎

*Theorem 1:* For every $s,v \in V$, there is an S-lightest path from $s$ to $v$.

*Proof:* We construct a sequence of paths $p^i$ =

[1] We reserve the more conventional term shortest-path for the case when "$\oplus$" and "$\preceq$" are "$+$" and "$\leq$," respectively.

```
DIJKSTRA(G, w, s)
    for each node v ∈ V do
      l[v] ← ω̄
      π[v] ← NIL
    Q ← V
5:  l[s] ← 0̄
    while Q ≠ ∅ do
      u ← EXTRACT-LEAST-ELEM(Q)
      for each node v ∈ N(u) do
        if l[v] ≻ l[u] ⊕ w(u, v) then
10:       l[v] ← l[u] ⊕ w(u, v)
          π[v] ← u
```

Fig. 2. Dijkstra's algorithm, generalized for the binary operation $\oplus$ and order relation $\preceq$. The function EXTRACT-LEAST-ELEM($Q$) extracts a node of least lightest-path estimate from $Q$.

$\langle v^i_{n_i}, \ldots, v^i_2, v^i_1 \rangle$, $i \geq 1$, from $s = v^i_{n_i}$ to $v = v^i_1$, eventually leading to an S-lightest path from $s$ to $v$.

Paths $p^i$ are constructed according to the following procedure. Initially, choose any path from $s$ to $v$; this is path $p^1$. At step $i$, $i \geq 1$, if $p^i$ is an S-lightest path from $s$ to $v$ then stop. Otherwise, let $v_{k_i}$, $1 \leq k_i < n_i$, be the last node along $p^i$ for which $p^i_{n_i k_i}$ is not a lightest path. Choose $q^i$ to be any lightest path from $s$ to $v^i_{k_i}$ and let $p^{i+1} = q^i \circ p^i_{k_i 1}$. From Lemma 1, $p^{i+1}$ is also a path from $s$ to $v$. In addition, $k_{i+1} > k_i$. Since the number of nodes in any path is bounded by $|V|$ and the $k_i$'s grow monotonically, the procedure stops after a finite number of steps, and when it does we get an S-lightest path from $s$ to $v$. ∎

### B. Generalized Dijkstra's algorithm

Figure 2 gives the pseudo-code of a generalized Dijkstra's algorithm that determines lightest paths from source $s$ to every other node $v \in V$. The variable $l[v]$ is called the lightest-path estimate of node $v$, and it holds a running estimate of the lightest-path weight from $s$ to $v$. The function EXTRACT-LEAST-ELEM($Q$) extracts a node of least estimate from the nodes that belong to the data structure $Q$. The execution of lines 9, 10 and 11 is called the relaxation of link $(u, v)$. The variable $\pi[v]$ is called the predecessor of node $v$. Taken together, the variables $\pi[v]$, $v \in V$, define the predecessor subgraph, $G_\pi$, of paths found by the algorithm from source $s$ to the other nodes in the network. Formally, we write $G_\pi = (V_\pi, E_\pi)$ with

$$V_\pi = \{v \in V : \pi[v] \neq \text{NIL}\} \cup \{s\}$$

and

$$E_\pi = \{(\pi[v], v) \in E : v \in V_\pi - \{s\}\}.$$

The following three lemmas are easy extrapolations of their equivalents with "$\oplus$" and "$\preceq$" in place of "$+$" and "$\leq$," respectively [18]. The proofs are omitted.

*Lemma 2:* The lightest-path estimate $l[v]$ can only decrease during the execution of the algorithm. Moreover, once $l[v]$ achieves its lower bound $\delta(s, v)$, both $l[v]$ and $\pi[v]$ remain unchanged.

*Lemma 3:* Let $p \circ \langle u, v \rangle$ be an $S$-lightest path from $s$ to $v$, and assume that $l[u] = \delta(s, u)$ at any time prior to the extraction of $u$ from $Q$. Then, when $u$ is extracted from $Q$, link $(u, v)$ is relaxed and $l[v] = \delta(s, v)$ after the relaxation.

*Lemma 4:* At all times during the execution of the generalized Dijsktra's algorithm the predecessor subgraph, $G_\pi$, is a tree rooted at $s$.

We sketch the proof of correctness of the generalized Dijkstra's algorithm, adapted from [18].

*Theorem 2:* If the generalized Dijkstra's algorithm is run at source $s$ then, upon termination, the predecessor subgraph is a tree of S-lightest paths rooted at $s$.

*Proof:* We first prove that, for each $v \in V$, we have $l[v] = \delta(s, v)$ at the time when $v$ is extracted from $Q$. Suppose otherwise, and let $v$ be the first node for which $l[v] \neq \delta(s, v)$ when it is extracted from $Q$. Thus, from Lemma 2, $\delta(s, v) \prec l[v]$. Let $p$ be any S-lightest path from $s$ to $v$: the existence of such a path is assured from Theorem 1. Path $p$ can be decomposed into $p = p' \circ \langle x, y \rangle \circ p''$, where $y$ is the first node along $p$ that is still in $Q$ just before node $v$ is extracted, and $x$ is the predecessor of $y$.

When $x$ was extracted from $Q$, $l[x] = \delta(s, x)$ since $v$ is the first node to be extracted with $l[v] \neq \delta(s, v)$. From Lemma 3, we obtain $l[y] = \delta(s, y)$ after the extraction of $x$ and subsequent relaxation of link $(x, y)$. Hence,

$$l[y] = \delta(s, y) \preceq \delta(s, v) \prec l[v],$$

which is a contradiction, since when $v$ is extracted from $Q$ it is the node with the least lightest-path estimate of all nodes in $Q$.

To end the proof, we now show that, when the algorithm terminates, $G_\pi$ is a tree of $S$-lightest paths rooted at $s$. From Lemma 4, we know that $G_\pi$ is a tree rooted at $s$ at all times during the execution of the algorithm. It also follows easily from the code that if $u$ is the predecessor of $v$ in $G_\pi$ then $l[v] = l[u] \oplus w(u, v)$. As an induction hypothesis, suppose that the subgraph of $G_\pi$ induced by all nodes that are no longer in $Q$ when a new node is extracted from $Q$ is a tree of S-lightest paths, and that this tree remains unchanged afterwards. For the base case, we have $l[s] = \delta(s, s) = \bar{0}$ just after source $s$ is extracted from $Q$, and neither $l[s]$ nor $\pi[s] = \text{NIL}$ change afterwards. For the induction step, consider the time when node $v$, $v \neq s$, is extracted from $Q$. Let $u$ be the predecessor of $v$ at this time. The induction hypothesis tells us that the unique path in $G_\pi$ from $s$ to $u$ is an S-lightest path. Moreover, we saw earlier that $l[v] = \delta(s, v)$ when $v$ is extracted from $Q$. Therefore, $l[v] = \delta(s, v) = \delta(s, u) \oplus w(u, v)$, which means that the path in $G_\pi$ from $s$ to $v$ is also an S-lightest path. In addition, from Lemma 2 and the induction hypothesis, that path remains unchanged afterwards. ∎

Upon termination of the algorithm, the nodes of $G_\pi$ on the S-lightest path from $s$ to a particular destination $v \in V$ can be obtained in reverse order by recursively taking the predecessor

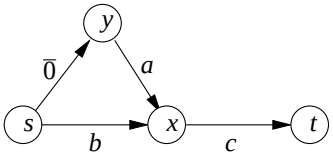


Fig. 3. Suppose that $a \prec b$ and $a \oplus c \succ b \oplus c$. Path $\langle s, x, t\rangle$ is the only lightest path from $s$ to $t$, but its subpath $\langle s, x\rangle$ is not a lightest path. In addition, the generalized Dijkstra's algorithm run at $s$ would compute path $\langle s, y, x, t\rangle$ to reach node $t$, which is not a lightest path.

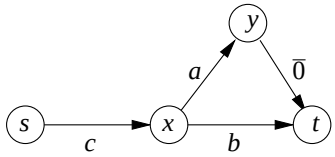


Fig. 4. Suppose that $a \prec b$ and $c \oplus a \succ c \oplus b$. The only lightest path from $s$ to $t$ is $\langle s, x, t\rangle$, and the only lightest path from $x$ to $t$ is $\langle x, y, t\rangle$. The optimal forwarding decision at node $x$ would depend on the source of the packet. Therefore, hop-by-hop routing over lightest paths is not possible.

of a node, starting at $v$ and stopping at $s$, for which the predecessor variable is NIL. The generalized Dijkstra's algorithm has the same asymptotic complexity as the standard one. For sparse graphs, a binary heap implementation of data structure $Q$ is normally used giving a complexity of $O(|E| \log |V|)$, provided that any elementary operation involving "$\oplus$" or "$\preceq$" can be considered a primitive computational step. On the other hand, a linear array implementation of $Q$ yields a complexity of $O(|V|^2)$ and is appropriate for dense graphs. For a discussion, and comparative evaluation, of advanced data structures that may improve the computational effort of Dijkstra's algorithm the reader is referred to [19].

### C. Converse statements when isotonicity fails

Theorems 1 and 2 were derived under the assumption of isotonicity. We can show converses of those theorems when isotonicity fails. Suppose the first condition in **AO1** does not hold. Thus, there are $a, b, c \in S$ such that $a \prec b$ and $a \oplus c \succ b \oplus c$. Consider the network of Figure 3. The only lightest path from $s$ to $t$ is $\langle s, x, t\rangle$, since $b \oplus c \prec a \oplus c = \bar{0} \oplus a \oplus c$. However, its subpath $\langle s, x\rangle$ is not a lightest path from $s$ to $x$, since $b \succ a = \bar{0} \oplus a$, and so, no S-lightest path exists from $s$ to $t$. In addition, if the generalized Dijkstra's algorithm is run at $s$ it would choose path $\langle s, y, x, t\rangle$ to reach node $t$, which is not a lightest path.

As a particular case, we conclude that the generalized Dijkstra's algorithm cannot be used to compute shortest-widest paths. This has previously been observed in [13], which also provides an alternative algorithm for finding those paths.

## IV. QoS ROUTING OF PACKETS

### A. Possible non-optimality of hop-by-hop routing

In hop-by-hop routing the forwarding decision made by a node upon receipt of a packet depends on the destination of the packet, but not on its source. Each packet contains the address of its destination, and every node maintains a forwarding table that maps each destination address into the out-neighbor to which packets addressed to the destination should be forwarded. Typically, the out-neighbor associated with a destination address is one that is along a lightest path towards the destination. With QoS routing, there is a distinct forwarding table for each type of QoS path, and each packet contains a QoS field that is used by the nodes to direct the packet to the appropriate forwarding table.

Hop-by-hop routing does not operate correctly for every type of QoS path. Without isotonicity, packets may not travel over lightest paths. To see this, assume that the second condition in **AO1** fails. Thus, there are $a, b, c \in S$ such that $a \prec b$ and $c \oplus a \succ c \oplus b$. Consider the network of Figure 4. The only lightest path from $s$ to $t$ is $\langle s, x, t\rangle$, since $c \oplus b \prec c \oplus a = c \oplus a \oplus \bar{0}$; the only lightest path from $x$ to $t$ is $\langle x, y, t\rangle$, since $a \oplus \bar{0} = a \prec b$.

With hop-by-hop routing, all packets with destination $t$ received at node $x$ are sent either to $t$ or to $y$, independently of their sources. Suppose that node $x$ forwards packets with destination $t$ directly to node $t$. Then, when node $x$ generates a packet with destination $t$ it forwards the packet directly to $t$. The packet travels over path $\langle x, t\rangle$ which is not a lightest path from $x$ to $t$. On the other hand, suppose that node $x$ forwards packets with destination $t$ to node $y$. Then, when node $x$ receives a packet from source $s$ with destination $t$, it forwards the packet to $y$. The packet ends up traveling over path $\langle s, x, y, t\rangle$ which is not a lightest path from $s$ to $t$. Therefore, no forwarding decision at node $x$ can guarantee that all packets with destination $t$ traverse lightest paths. In particular, if every node forwards packets along lightest paths then a packet with source $s$ and destination $t$ will travel over path $\langle s, x, y, t\rangle$, which is not a lightest path from $s$ to $t$.

As an example, we conclude that hop-by-hop routing does not ensure that packets travel over shortest-widest paths, whatever algorithms are used at the nodes to compute paths and build forwarding tables.

### B. Lexicographic-lightest paths and loop-free routing

If weights obey strict isotonicity and each node uses Dijkstra's algorithm to build its forwarding table, then packets will travel over lightest paths without looping. To avoid forwarding loops in networks having cycles of weight $\bar{0}$, an (additive) hop-count metric may be appended to the link weights, inducing Dijkstra's algorithm to compute lightest paths with the least number of hops. However, without strict isotonicity, not every implementation of Dijkstra's algorithm results in loop-free routing and, indeed, computing S-lightest paths with the least number of hops does not solve the problem. To see this, consider the network of Figure 5, where links are labeled with available bandwidth. There are three paths from $s_1$ to $t$. Path $\langle s_1, x, t\rangle$ is never found by Dijkstra's algorithm, since it is not an S-widest path. The S-widest path from $s_1$ to $t$ with the least number of hops is $\langle s_1, s_2, y, z, w, t\rangle$ (5 hops). On the other hand, the S-

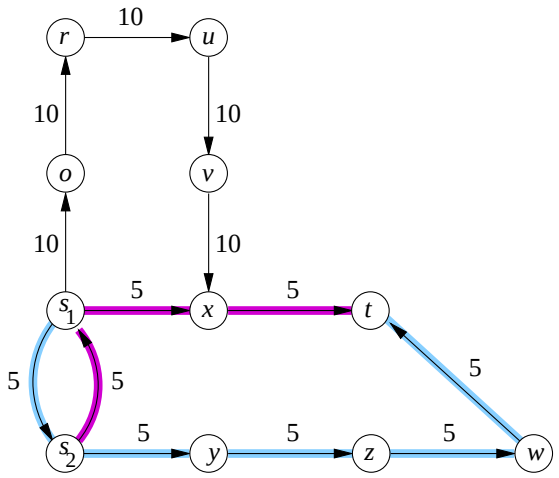


Fig. 5. The S-widest path from $s_1$ to $t$ with the least number of hops is $\langle s_1, s_2, y, z, w, t\rangle$ whereas the L-widest path from $s_1$ to $t$ is $\langle s_1, o, r, u, v, x, t\rangle$. Path $\langle s_2, s_1, x, t\rangle$ is both an S-widest path with the least number of hops and an L-widest path, from $s_2$ to $t$. If packets with destination $t$ are forwarded along S-widest paths with the least number of hops they will loop between $s_1$ and $s_2$. However, if they are forward along L-widest paths, loops will not occur.

widest path from $s_2$ to $t$ with the least number of hops is path $\langle s_2, s_1, x, t\rangle$ (3 hops). If a packet with destination $t$ appears at $s_1$, it is forwarded to $s_2$ which will send it back to $s_1$: the packet loops between these two nodes.

A possible solution to loop-free hop-by-hop routing can be found in what we call *lexicographic*-lightest paths, introduced next. Regard the set of weights $S$ as an alphabet and let $S^*$ be the set of all words over $S$. Given words $\alpha = \alpha_1\alpha_2\cdots\alpha_n$ and $\beta = \beta_1\beta_2\cdots\beta_m$ in $S^*$, we say that $\alpha$ is lexicographically less than $\beta$, written $\alpha \prec_L \beta$, if either (i) $n < m$ and $\alpha_i = \beta_i$ for $1 \leq i \leq n$, or (ii) there is an index $k$, $1 \leq k \leq \min(n, m)$, such that $\alpha_k \prec \beta_k$ and $\alpha_i = \beta_i$ for $1 \leq i < k$. We let $\alpha \preceq_L \beta$ (read as "$\alpha$ is lexicographically less than or equal to $\beta$") mean that either $\alpha \prec_L \beta$ or $\alpha = \beta$.

The word-weight of path $p = \langle v_n, \ldots, v_2, v_1\rangle$, denoted by $\omega(p)$, is defined as the word of $S^*$ the $i$th letter of which, $\omega_i(p)$, $1 \leq i < n$, is the weight of the subpath of $p$ that extends from the source to its $i$th last node. That is, $\omega_i(p) = w(p_{ni})$, and, thus, $\omega(p) = w(p_{n1})w(p_{n2})\cdots w(p_{n,n-1})$. Path $p$ is a lexicographic-lightest path, or L-lightest path for short, from $s$ to $v$ if its word-weight is lexicographic less than or equal to the word-weight of any other path from $s$ to $v$, that is, if $\omega(p) \preceq_L \omega(q)$ for every path $q$ from $s$ to $v$. It can easily be shown that an L-lightest path is an S-lightest path, although the converse statement does not hold in general.

In Figure 5, the only L-widest path from $s_1$ to $t$ is $p = \langle s_1, o, r, u, v, x, t\rangle$ with $\omega(p) = 5\ 10\ 10\ 10\ 10\ 10$, since $\omega(p) \prec_L \omega(\langle s_1, x, t\rangle) = 5\ 5$, as well as $\omega(p) \prec_L \omega(\langle s_1, s_2, y, z, w, t\rangle) = 5\ 5\ 5\ 5\ 5$. The only L-widest path from $s_2$ to $t$ is $\langle s_2, s_1, x, t\rangle$, with word-weight $5\ 5\ 5$. Hence, loops do not occur if packets are forwarded along L-widest paths, and this property is true in general as Theorem 3 shows.

The next lemma simplifies the proof of Theorem 3.

*Lemma 5:* Suppose that $p = \langle v_n, \ldots, v_2, v_1\rangle$ is a lexicographic-lightest path from $v_n$ to $v_1$. Then, $\omega(p_{j1}) \prec_L \omega(p)$, for $1 \leq j < n$.

*Proof:* For $1 \leq i < j$, the $i$th letter of the word-weight of $p_{j1}$ is $\omega_i(p_{j1}) = w(p_{ji})$ and the $i$th letter of the word-weight of $p$ is $\omega_i(p) = w(p_{ni}) = w(p_{nj}) \oplus w(p_{ji})$. Therefore, $\omega_i(p_{j1}) \preceq \omega_i(p)$, $1 \leq i < j$, and since, in addition, $j < n$, we have $\omega(p_{j1}) \prec_L \omega(p)$ as stated. ■

*Theorem 3:* If every node builds its forwarding table from lexicographic-lightest paths, then hop-by-hop routing is loop-free.

*Proof:* The proof is by contradiction. Under the conditions of the theorem, assume that loop $c = \langle s_l, \ldots, s_1, s_0\rangle$, with $s_0 = s_l$ and $l > 1$, is formed when nodes $s_i$ try to forward a packet to destination $t$. Let $p^i = \langle v^i_{n_i}, \ldots v^i_2, v^i_1\rangle$ be the L-lightest path from $s_i = v^i_{n_i}$ to $t = v^i_1$ computed at node $s_i$. Then, $v^i_{n_i-1} = s_{i-1}$, $1 \leq i \leq l$, and since $p^{i-1}$ is an L-lightest path from $s_{i-1}$ to $t$ we have $\omega(p^{i-1}) \preceq_L \omega(p^i_{n_i-1,1})$. From Lemma 5, we further obtain $\omega(p^i_{n_i-1,1}) \prec_L \omega(p^i)$. It follows that

$$\omega(p^{i-1}) \prec_L \omega(p^i),$$

for $1 \leq i \leq l$. This set of inequalities implies that $\omega(p^0) \prec_L \omega(p^l)$, and this is impossible since $s_0 = s_l$. ■

Although nodes forward packets over L-lightest paths, to avoid loops, packets may not travel over L-lightest paths. For example, in Figure 5, a packet dispatched by $s_2$ with destination $t$ ends up traveling over path $\langle s_2, s_1, o, r, u, v, x, t\rangle$ which is not an L-widest path from $s_2$ to $t$. However, as the next theorem shows, packets always traverse lightest paths.

*Theorem 4:* If every node builds its forwarding table from lexicographic-lightest paths, then hop-by-hop routing will have packets travel over lightest paths.

*Proof:* We know from Theorem 3 that packets are not forwarded in loops. Thus, for every source $s$ and destination $t$ there is a path $p = \langle s_l, \ldots, s_2, s_1\rangle$ which is traversed by all packets dispatched by $s = s_l$ with destination $t = s_1$. We have to show that $p$ is a lightest path. The proof is by induction. Define $p^i$ as in Theorem 3. For the base case, we have $w(p_{1,1}) = \bar{0} = \delta(t, t)$. Suppose that path $p_{i-1,1}$, with $1 < i \leq l$, is a lightest path from $s_{i-1}$ to $t$, that is, $w(p_{i-1,1}) = \delta(s_{i-1}, t)$. Path $p^i$ is an L-lightest path and, in particular, it is a lightest path. Hence, we may write

$$\begin{aligned} w(p_{i1}) &= w(s_i, s_{i-1}) \oplus w(p_{i-1,1}) \\ &= w(s_i, s_{i-1}) \oplus \delta(s_{i-1}, t) \\ &\preceq w(s_i, s_{i-1}) \oplus w(p^i_{n_i-1,1}) \\ &= w(p^i) = \delta(s_i, t), \end{aligned}$$

from which we conclude that $w(p_{i1}) = \delta(s_i, t)$. ■

### C. Dijkstra-old-touch-first algorithm

The goal now is to find a variant of Dijkstra's algorithm that computes lexicographic-lightest paths from any source $s$ to every other node in the network without explicitly computing or

```
DIJKSTRA-OLD-TOUCH-FIRST(G, w, s)
    for each node v ∈ V do
      l[v] ← ∞̄
      π[v] ← NIL        ▷ π[v] ← {}
      t[v] ← 0
5:  l ← 0̄
    t ← 0
    touch ← 1
    Q ← V
    l[s] ← 0̄
10: while Q ≠ ∅ do
      u ← EXTRACT-LEAST-ELEM-TOUCH(Q)
      if l[u] ≻ l or t[u] > t then
        touch ← touch + 1
        l ← l[u]
15:     t ← t[u]
      for each node v ∈ N(u) do
        if l[v] ≻ l[u] ⊕ w(u, v) then
          l[v] ← l[u] ⊕ w(u, v)
          π[v] ← u        ▷ π[v] ← {u}
20:       t[v] ← touch
  ▷   else if  l[v] = l[u] ⊕ w(u, v) and t[v] = touch
  ▷     π[v] ← π[v] ∪ {u}
```

Fig. 6. Dijkstra-old-touch-first algorithm computes L-lightest paths from source $s$ to every other node in the network. The variable $touch$ is incremented each time a node with a new lightest-path estimate or a new touch is extracted from $Q$. The function EXTRACT-LEAST-ELEM-TOUCH($Q$) extracts a node of least estimate from $Q$, and, in case of a tie, it extracts one of least touch. The last two lines allows for computation of all L-lightest paths from $s$.

comparing word-weights, as that would increase the computational complexity of the algorithm, as well as its storage requirements.

We claim that the Dijkstra-Old-Touch-First (Dijkstra-OTF) algorithm presented in Figure 6 computes lexicographic-lightest paths from source $s$ to every other node $v \in V$ (ignore, for the moment, the lines marked with ▷). This algorithm is more specific than the standard one in the choice of the node to be extracted from $Q$ at line 11. The function EXTRACT-LEAST-ELEM-TOUCH(Q) always extracts a node with least lightest-path estimate, but, in case of a tie, EXTRACT-LEAST-ELEM-TOUCH(Q) chooses a node of oldest touch. The touch of node $v$ is kept in variable $t[v]$ and marks the time when the lightest-path estimate of $v$ was last updated. The current touch value is kept in the variable $touch$, which is initialized to 1, and it is incremented with the extraction of a new node from $Q$, but only if the new node has a different (greater, $\succ$) lightest-path estimate or a different (greater, $>$) touch than the last node to have been withdrawn from $Q$. Note that Dijkstra-OTF determines L-lightest paths without explicitly keeping track of word-weights, and its complexity equals that of a standard Dijkstra's algorithm.

Let $h(x)$ denote the value of $touch$ immediately after node $x$ has been extracted from $Q$ and lines 12, 13, 14 and 15 have been executed. The following two lemmas make the proof of our main result, stated in Theorem 5, straightforward.

*Lemma 6:* Let $p = \langle v_n, \ldots, v_2, v_1 \rangle$ be an S-lightest path obtained by running Dijkstra-old-touch-first at source $s = v_n$. Then, $h(v_{j-1}) > h(v_j)$ for $1 < j \leq n$.

*Proof:* The proof is by induction. For the base case, note that the initial conditions imply $l = \bar{0}$, $t = 0$, and $h(v_n) = 1$ after source $s = v_n$ is extracted from $Q$. Link $(v_n, v_{n-1})$ is then relaxed setting $t[v_{n-1}] = h(v_n) = 1$; the touch of $v_{n-1}$ does not change thereafter. Since $t = 0 < 1 = t[v_{n-1}]$, sometime between the extractions of $v_n$ and $v_{n-1}$ from $Q$, a node—possibly $v_{n-1}$ itself—will be extracted with a touch greater than $t = 0$: the variable $touch$ is then incremented yielding $h(v_{n-1}) > 1$.

For the induction step, assume that $h(v_j) > h(v_{j+1})$, for $1 < j < n$. When $v_{j+1}$ is taken from $Q$, the relaxation of link $(v_{j+1}, v_j)$ sets $l[v_j] = \delta(s, v_j)$ and $t[v_j] = h(v_{j+1})$. Later, when $v_j$ is taken from $Q$, $l = \delta(s, v_j)$, $t = t[v_j]$, link $(v_j, v_{j-1})$ is relaxed setting $l[v_{j-1}] = \delta(s, v_{j-1})$ and $t[v_{j-1}] = h(v_j)$. As $t = t[v_j] = h(v_{j+1}) < h(v_j) = t[v_{j-1}]$, sometime between the extractions of $v_j$ and $v_{j-1}$ from $Q$, a node—possibly $v_{j-1}$ itself—will be extracted with a touch greater than $t = t[v_j]$. The variable $touch$ is then incremented. Therefore, $h(v_{j-1}) > h(v_j)$. ■

*Lemma 7:* Let $p = \langle v_n, \ldots, v_2, v_1 \rangle$ be an S-lightest path obtained by running Dijkstra-old-touch-first at source $s = v_n$, and let $q = \langle u_m, \ldots, u_2, u_1 \rangle$ be any other S-lightest path with source $s = v_n = u_m$. Assume there is $k$, $1 \leq k \leq \min(n, m)$, such that $h(v_k) > h(u_k)$ and $\delta(s, v_j) = \delta(s, u_j)$ for $1 \leq j < k$. Then, $h(v_j) > h(u_j)$ for $1 \leq j < k$.

*Proof:* Note, first, that since $touch$ is either incremented or remains constant when a node is extracted from $Q$, $h(v_j) > h(u_j)$ implies that $v_j \neq u_j$ and that $u_j$ is extracted from $Q$ before $v_j$.

By hypothesis, we have $h(v_k) > h(u_k)$. Assume that $h(v_j) > h(u_j)$, for $1 \leq j \leq k$. When $u_j$ is extracted from $Q$, link $(u_j, u_{j-1})$ is relaxed, and we have $l[u_{j-1}] = \delta(s, u_{j-1})$ and $t[u_{j-1}] \leq h(u_j)$ after the relaxation. Neither $l[u_{j-1}]$ nor $t[u_{j-1}]$ change afterwards. Similarly, when $v_j$ is extracted from $Q$, link $(v_j, v_{j-1})$ is relaxed leaving $l[v_{j-1}] = \delta(s, v_{j-1})$. Now, however, because $v_{j-1}$ is on the S-lightest path found by the algorithm, that is, $\pi[v_{j-1}] = v_j$, we have $t[v_{j-1}] = h(v_j)$ after the relaxation. Again, these values do not change afterwards. Node $u_{j-1}$ is extracted either before or after $v_j$. If $u_{j-1}$ is extracted before $v_j$, an appeal to Lemma 6 leads to the conclusion $h(v_{j-1}) > h(v_j) \geq h(u_{j-1})$. On the other hand, if $v_j$ is extracted before $u_{j-1}$, since $l[u_{j-1}] = l[v_{j-1}]$ and $t[u_{j-1}] \leq h(u_j) < h(v_j) = t[v_{j-1}]$, node $u_{j-1}$ is extracted from $Q$ before node $v_{j-1}$. When node $u_{j-1}$ is extracted from $Q$, the variables $l$ and $t$ are updated to $l[u_{j-1}]$ and $t[u_{j-1}]$, respectively, and $touch$ takes value $h(u_{j-1})$. Between the extractions of $u_{j-1}$ and $v_{j-1}$ from $Q$, a node—possibly $v_{j-1}$ itself—will be extracted with a touch greater than $t = t[u_{j-1}]$, and $touch$ gets incremented at that time. Therefore, $h(v_{j-1}) > h(u_{j-1})$. ■

*Theorem 5:* If Dijkstra-old-touch-first is run at source $s$ then, upon termination, the predecessor subgraph is a tree of

lexicographic-lightest paths rooted at $s$.

*Proof:* Suppose otherwise, and let $p = \langle v_n, \ldots, v_2, v_1 \rangle$ be the unique S-lightest path from $s = v_n$ to $v_1$ in the predecessor subgraph. Let $q = \langle u_m, \ldots, u_2, u_1 \rangle$ be any L-lightest path from $s = v_n = u_m$ to $u_1 = v_1$. Then, either (i) $m < n$ and $\delta(s, u_j) = \delta(s, v_j)$ for $1 \leq j < m$, or (ii) there is $k$, $1 \leq k < \min(m, n)$, such that $\delta(s, u_k) \prec \delta(s, v_k)$ and $\delta(s, u_j) = \delta(s, v_j)$ for $1 \leq j < k$. If Condition (i) is satisfied, $u_m \neq v_m$. Moreover, $u_m = v_n = s$ is the first node to be extracted from $Q$. From Lemma 6, we have $h(v_m) > h(u_m) = 1$, and, from Lemma 7, $h(v_1) > h(u_1)$. But this is impossible since $v_1 = u_1$.

On the other hand, if Condition (ii) is satisfied, when $u_k$ is extracted from $Q$, $v_k$ is still in $Q$. At this time, $l$ equals $\delta(s, u_k)$ and $touch$ equals $h(u_k)$. Because $\delta(s, u_k) \prec \delta(s, v_k)$, when $v_k$ is extracted from $Q$ we have $h(v_k) > h(u_k)$. Making use of Lemma 7, we once again arrive at the contradiction $h(v_1) > h(u_1)$. ■

## V. Multipath routing

Multipath routing offers the possibility to balance traffic from a source to a destination along multiple parallel paths. In datagram mode of operation, multipath routing is known to reduce the oscillatory behavior induced in the network when link metrics are dynamic and depend on the load that crosses the links [20]. In virtual circuit mode of operation, multipath routing allows for the expedient exploration of several paths at flow set-up time, improving the acceptance rate of new flows. Within the context of lightest paths, the aim is to find as many equal-weight lightest-paths as possible for every source-destination pair. We first discuss a simple extension to Dijkstra-OTF that determines all L-lightest paths from a given source to every other node in the network. Later, we present a general loop-free condition that leads to an increase in the number of paths that may be involved in multipath routing.

The extension of Dijkstra-OTF for multipath routing consists of the lines marked with ▷ in Figure 6: lines 21 and 22 were added and the variable $\pi[v]$ is now a set that collects the predecessors of node $v$. The algorithm finds all L-lightest paths from source $s$ to every other node in the network.

*Theorem 6:* If Dijkstra-old-touch-first with multipath extension is run at source $s$ then, upon termination, the predecessor subgraph is an acyclic graph of lexicographic-lightest paths with source $s$. Moreover, it is the largest subgraph of lexicographic-lightest paths having source $s$.

*Proof:* Given $v \in V$, let $p = \langle v_n, \ldots, v_2, v_1 \rangle$ be the unique L-lightest path from $s = v_n$ to $v = v_1$ that would be found by Dijkstra-OTF if multipath extension were omitted. If $q = \langle u_m, \ldots, u_2, u_1 \rangle$ is any S-lightest path from $s = u_m = v_n$ to $v = u_1 = v_1$ that is not an L-lightest path, then there is an index $k$, $1 \leq k \leq \min(n, m)$, such that $h(u_k) > h(v_k)$, and this is sufficient to show that path $q$ does not belong to the predecessor subgraph.

Conversely, if $q = \langle u_n, \ldots, u_2, u_1 \rangle$ is any L-lightest path from $s = u_n = v_n$ to $v = u_1 = v_1$ then $h(u_j) = h(v_j)$, for $1 \leq j \leq n$. Let $u_{i-1}$, $1 < i < n$, be any node of $q$ such that $v_{i-1} = u_{i-1}$ and $v_i \neq u_i$. When $v_i$ is extracted from $Q$, $t[v_{i-1}]$ is set to $h(v_i)$. Later, when $u_i$ is extracted from $Q$, $touch = h(u_i) = h(v_i) = t[v_{i-1}]$. Consequently, $u_i$ is added to the predecessor set of $v_{i-1} = u_{i-1}$. This argument can be applied across the nodes of $q$ leading to the conclusion that $q$ belongs to the predecessor subgraph. Since every L-lightest path with source $s$ belongs to the predecessor subgraph, the latter is the largest subgraph of L-lightest paths with source $s$. Finally, that the predecessor subgraph is acyclic follows from the intrinsic properties of L-lightest paths. ■

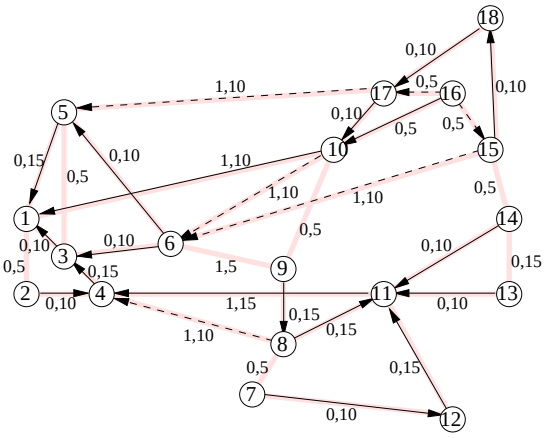


Fig. 7. Link weights are pairs with distance and available bandwidth, where distance is either 0 or 1. All links are bidirectional with the same weight in both directions. The solid arrows connect a node to its out-neighbors along L-widest-shortest paths to node 1. The dashed arrows are additional links that lead to widest-shortest paths to node 1, while preserving loop-freedom.

After executing Dijkstra-OTF with multipath extension, the source can find all of its out-neighbors along L-lightest paths towards a given destination by running a depth-first search on the predecessor subgraph starting at that destination.

An example is presented in Figure 7. Suppose that packets are to be forwarded along widest paths subject to the condition of crossing as few links as possible from the cut set $C = \{(1, 10), (4, 8), (4, 11), (5, 17), (6, 9), (6, 10), (6, 15)\}$. Thus, link weights are of the form $(d, b)$, where $d = 1$ if the link belongs to $C$ and $d = 0$ otherwise, and $b$ represents the bandwidth available at the link. A lightest path is, thus, a widest-shortest path. Bandwidth available at the links is quantized into the values 5, 10 or 15. All links in the network are bidirectional, with the same weight in both directions. The solid arrows mark the links that connect a node to its out-neighbors along L-widest-shortest paths to destination node 1, computed from Dijkstra-OTF with multipath extension. We observe that the possibilities of multipath routing are minor: only node 6 has two out-neighbors through which it can reach node 1. This is because lexicographic-lightness is a very strong property. The possibilities of multipath routing can be increased with the more general loop-free condition presented in Theorem 7, which is based on a similar condition given in [21] for the case of an additive weight function.

*Theorem 7:* Packets with any given destination get delivered without looping if every node forwards them to an out-neighbor whose L-lightest path word-weight to that destination is lexicographically less than its own L-lightest path word-weight to the same destination.

Let $v \in N(s)$, and $p$ and $q$ be L-lightest paths from $s$ to $t$ and from $v$ to $t$, respectively. The theorem states that if $\omega(q) \prec_L \omega(p)$, then packets with destination $t$ can be forwarded from $s$ to $v$ without incurring into loops. The proof follows the same steps as those of Theorem 3. In addition, if $w(p) = w(s,v) \oplus w(q)$, the proof of Theorem 4 implies that packets end up traveling over lightest paths. The dashed arrows in Figure 7 correspond to links that can be added using the loop-free condition of Theorem 7, while keeping equal-weight multipaths from every node to node 1, and they represent a significant improvement for multipath routing.

The following procedure can be used at node $s$ to determine whether the word-weight of an L-lightest path from its out-neighbor $v$ to destination $t$ is lexicographically less than the word-weight of an L-lightest path from $s$ to $t$: (i) run Dijkstra-OTF with multipath extension but with both the initial estimates of $s$ and $v$ set to $\bar{0}$ at line 9; (ii) run a depth-first search on the resulting predecessor subgraph starting at $t$. If the depth-first search does not find node $s$, then the word-weight of an L-lightest path from $v$ to $t$ is lexicographically less than the word-weight of an L-lightest path from $s$ to $t$. This procedure, however, increases the computational complexity at node $s$, since a new instance of Dijkstra-OTF has to be run for every one of its out-neighbors.

## VI. Conclusions

We presented an algebra and algorithms for QoS path computation and hop-by-hop routing in the Internet. Network links and paths are characterized by generic weights, themselves a function of one or more metrics. A binary operation and an order relation are defined on the set of weights, and they are intertwined by the isotone property. We have shown that, within this framework, a generalized Dijkstra's algorithm correctly computes lightest paths. On the other hand, without isotonicity, the generalized Dijkstra's algorithm does not determine lightest paths in general. Our approach unites in a common framework QoS path computation algorithms that were previously scattered in the literature and provides insight into why some QoS paths cannot be computed with variants of Dijkstra's algorithm.

Interestingly, isotonicity is also both necessary and sufficient for hop-by-hop routing. However, without strict isotonicity not every implementation of the generalized Dijkstra's algorithm results in loop-free hop-by-hop routing. In that case, nodes should compute lexicographic-lightest paths, which are a stronger form of lightest paths. We have presented the Dijsktra-old-touch-first algorithm which computes lexicographic-lightest paths, and, as such, is the first algorithm to guarantee loop-free hop-by-hop routing over general lightest paths. Moreover, the algorithm has the same complexity as a standard Dijkstra's algorithm.

Our results encompass multipath routing as well. Dijkstra-old-touch-first is extended to compute all lexicographic-lightest paths from a source to every other node in the network. Then, a general loop-free condition is presented, which increases the possibilities of multipath routing, albeit at the expense of additional computational effort.

## Acknowledgment

The author would like to thank José Brázio for his constructive comments and help in the organization of the paper, and the anonymous referees for their insightful remarks.

## References

[1] C. Huitema, *Routing in the Internet*, Prentice Hall PTR, Englewood Cliffs, NJ, 1995, ISBN 0-13-132192-7.

[2] R. G. Gallager, "A minimum delay routing algorithm using distributed computation," *IEEE Transactions on Communications*, vol. 25, no. 1, pp. 73–85, January 1977.

[3] F. P. Kelly, "Network routing," *Philosophical Transactions of the Royal Society*, vol. 337, no. 1647, pp. 343–367, December 1991.

[4] D. W. Glazer and C. Tropper, "A new metric for dynamic routing algorithms," *IEEE Transactions on Communications*, vol. 38, pp. 360–367, March 1990.

[5] I. Matta and A. U. Shankar, "Type-of-service routing in datagram delivery systems," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1411–1452, October 1995.

[6] J. T. Moy, *OSPF Anatomy of an Internet Routing Protocol*, Addison Wesley, Reading, MA, 1998, ISBN 0-201-63472-4.

[7] W. C. Lee, M. G. Hluchyj, and P. A. Humblet, "Routing subject to quality-of-service constraints in integrated communication networks," *IEEE Network*, pp. 46–55, July/August 1995.

[8] S. Chen and K. Nahrstedt, "An overview of quality-of-service routing for next-generation high-speed networks: Problems and solutions," *IEEE Network*, pp. 64–79, November 1998.

[9] T. Li, "MPLS and the evolving Internet arquitecture," *IEEE Communications Magazine*, pp. 38–41, December 1999.

[10] Z. Wang and J. Crowcroft, "Quality-of-service routing for supporting multimedia applications," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 7, pp. 1228–1234, September 1996.

[11] R. Guérin and A. Orda, "Networks with advance reservations: The routing perspective," in *Proc. INFOCOM'2000*, Tel-Aviv, Israel, 2000.

[12] G. Apostolopoulos, R. Guérin, S. Kamat, and S. K. Tripathi, "Quality-of-service based routing: A performance perspective," in *Proc. SIGCOMM'98*, Vancouver, BC, Canada, 1998, pp. 17–28.

[13] Q. Ma and P. Steenkiste, "On path selection for traffic with bandwidth guarantees," in *Proc. 5th IEEE International Conference on Network Protocols*, Atlanta, GA, October 1997, pp. 191–202.

[14] A. Shaikh, J. Rexford, and K. G. Shin, "Efficient precomputation of quality-of-service routes," in *Proc. Workshop on Network and Operating Systems Support for Digital Audio and Video*, July 1998, pp. 15–27.

[15] G. Apostolopoulos, R. Guérin, and S. Kamat, "Implementation and performance measurements of QoS routing extensions to OSPF," in *Proc. INFOCOM'99*, New York, NY, March 1999, pp. 680–688.

[16] E. Dijkstra, "A note on two problems in connection with graphs," *Numerical Mathematics*, vol. 1, pp. 269–271, 1959.

[17] B. Carré, *Graphs and Networks*, Clarendon Press, Oxford, 1979, ISBN 0-19-8596-22-7.

[18] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, MIT Press, 1990, ISBN 0-262-03141-8.

[19] B. Cherkassky, A. Goldberg, and T. Radzik, "Shortest paths algorithms: Theory and experimental evaluation," *Mathematical Programming*, pp. 129–174, 1996.

[20] S. Vutukury and J. J. Garcia-Luna-Aceves, "A simple approximation to minimum-delay routing," in *Proc. SIGCOMM'99*, August 1999, pp. 227–238.

[21] J. J. Garcia-Luna-Aceves, "Loop-free routing using diffusing computations," *IEEE/ACM Transaction on Networking*, vol. 1, no. 1, pp. 130–141, February 1993.