Hop-by-Hop Routing Algorithms For Premium Traffic *

Jun Wang†
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, U.S.A.
junwang3@cs.uiuc.edu

Klara Nahrstedt
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, U.S.A.
klara@cs.uiuc.edu

ABSTRACT

In Differentiated Service (DiffServ) networks, the routing algorithms used by the *premium* class traffic, due to the high priority afforded to that traffic, may have a significant impact not only on the premium class traffic itself, but on all other classes of traffic as well. The shortest hop-count routing scheme, used in current Internet, turns out to be no longer sufficient in DiffServ networks. This paper studies the problem of finding optimal routes for the premium-class traffic in a DiffServ domain, such that (1) no forwarding loop exists in the entire network in the context of hop-by-hop routing; and (2) the residual bandwidth on bottleneck links is maximized. This problem is called the Optimal Premium-class Routing (OPR) problem. We prove in this paper that the OPR problem is NP-hard.

To handle the OPR problem, first, we analyze the strength and weaknesses of two existing algorithms (Widest-Shortest-Path algorithm and Bandwidth-inversion Shortest-Path algorithm). Second, we propose a novel heuristic algorithm, called the *Enhanced Bandwidth-inversion Shortest-Path (EBSP) algorithm*. We prove theoretically the correctness of the EBSP algorithm, i.e., we show that it is *consistent* and loop-free. Our extensive simulations in different network environments show clearly that the EBSP algorithm performs better when routing the premium traffic in complex, heterogeneous networks.

Keywords

Hop-by-hop Routing, Differentiated Service, Premium Class, Saturate Bandwidth.

1. INTRODUCTION

Exploding traffic and expanding Quality-of-Service (QoS) requirements, coming from emerging multimedia applications, initiated intensive research in QoS provision and routing within the Internet. In this paper, we raise an interesting problem of how to find optimal routing schemes under both Differentiated Service (DiffServ) and hop-by-hop IP routing assumptions. By combining service differentiation and QoS routing together, the high-priority premium traffic should be transmitted in an efficient manner with low negative influences to other low-priority traffic. Before presenting our work in detail, some background knowledge is introduced in the following subsections.

1.1 Hop-by-Hop Shortest-path Routing

Hop-by-hop routing forms the basis of today's IP networks. Hop-by-hop routing means that routing decisions are made at each router independently and locally. For each incoming packet at a router, its destination address (maybe some other fields in its IP header, too) is used to get the next hop by consulting the router's routing table. Therefore, hop-by-hop routing is also referred to as destination-based table-driven routing. The hop-count shortest-path routing (SP) is the most commonly used method for IP routing in today's Internet. Algorithms of finding the shortest-paths between nodes, such as the Dijkstra's algorithm, can guarantee that no forwarding loop exists in a network. Figure 1 shows how the SP method works in a hop-by-hop scenario.

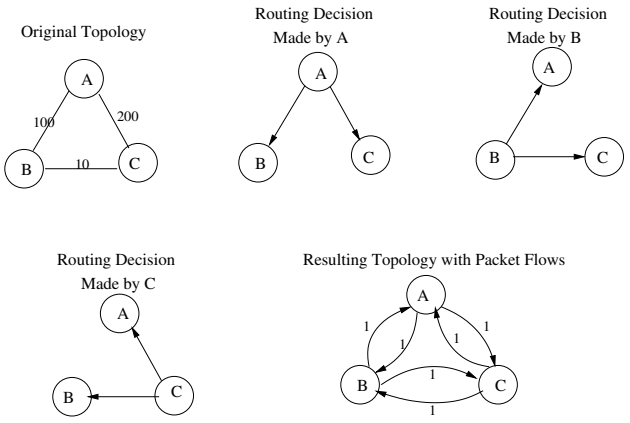


Figure 1: An example of hop-by-hop SP routing: the numbers next to the links in the original topology denote link capacities, while the labels in the resulting topology mean how many "flows" go through the links.

The resulting topology, shown in Figure 1, illustrates the real packet flows[1] between each pair of nodes. The num-

*This work was supported by NSF Grant under contract number NSF ANI 00-73802 and NSF CISE Grant under contract number NSF EIA 99-72884. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

†Please address all correspondences to Jun Wang and Klara Nahrstedt at Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, phone: (217) 244-5841, fax: (217) 244-6869.

[1]The word "flow" here is different from the definitions in

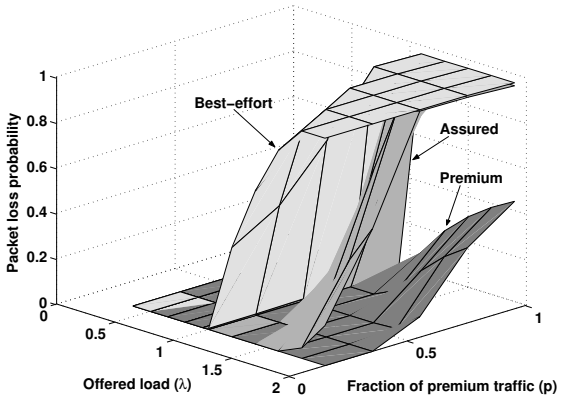
bers associated with the arcs are the numbers of flows going through that particular link in that direction.

In this paper, we investigate a QoS routing problem in the context of hop-by-hop routing, where bandwidth, rather than hop-count, is sensitive and the SP method turns out to be insufficient.

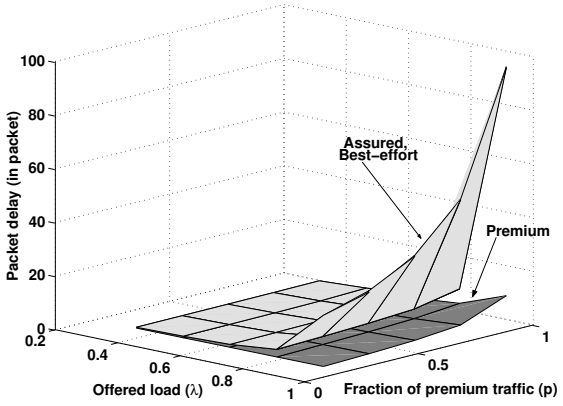
1.2 DiffServ Model and Inter-class Effects

In order to provision better end-to-end QoS, DiffServ scheme has been proposed as a cost-effective solution [6, 11]. In DiffServ networks, traffic is classified into three service classes: premium, assured and best-effort. The premium class traffic has the highest priority in comparison to other classes of traffic. Originally, the DiffServ scheme is decoupled from IP routing intentionally, meaning that all traffic between each source-destination pair follows the same path no matter which service class it belongs to and DiffServ itself has no effect on IP routing decisions. However, DiffServ does affect the queueing and drop behavior in routers. More specifically, two separate queues are used in DiffServ. One of them is used by premium traffic and the other is shared by non-premium traffic. The premium queue has absolutely higher priority than the other queue, therefore, as long as there are packets in the premium queue, these packets will be scheduled first. Due to premium traffic's high priority, a bad routing decision could lead to some problems for the low-priority traffic when the volume of premium class traffic is high. This means, without taking routing into consideration, the premium class traffic imposes very negative influences on other classes of traffic, especially when the network is highly loaded. We call this the inter-class effects [27]. In [21], the authors presented simple performance models and analysis of DiffServ schemes. However, to make a strong case of the negative impacts that the premium class traffic may impose on all other service classes in DiffServ networks, we have run simulations to measure the inter-class effects among *all* three service classes. Our results are shown in Figure 2.[2]

As we can see in Figure 2, the premium class traffic has significant inter-class effects on the assured class and best-effort class traffic with respect to some important metrics, such as the *packet loss probability* (Figure 2(a)) and the *packet delay* (Figure 2(b)). When the network is highly loaded (large offered load λ) or the fraction of premium traffic is high (large p), the traffic with low-priorities experiences severe performance degradations (such as higher packet loss rates and larger queueing delays). Therefore, we must take the inter-class effects into consideration when we choose routing algorithms for networks which support premium class traffic.



(a) Packet Loss Probability



(b) Packet Delay (in packet)

Figure 2: Measurements of the inter-class effects between all three classes in a DiffServ network

1.3 Routing and Bandwidth Reservation for Premium Traffic

During a certain time interval, presuming the full knowledge of the network topology, we can regard the topology metrics as static. Hence, we concentrate only on algorithmic aspects of static QoS routing schemes for the premium-class traffic in a DiffServ network.

In order to provide end-to-end QoS guarantees for the premium-class traffic between two nodes, certain amount of bandwidth must be successfully reserved on each link along the path between these two nodes. (How to implement such bandwidth reservation is out of the scope of this paper. We assume that on top of our routing scheme, some class-based resource reservation protocol, similar to the RSVP, is used for premium bandwidth reservation in the network.) If a link is shared by multiple paths between different pairs of nodes, the bandwidth it has to reserve should accommodate the summation of the premium traffic on all the paths sharing this link.

Integrated Service (IntServ) model or in RSVP. A "flow" here refers to a channel between two nodes, through which all premium traffic between the two nodes will follow. So each pair of source and destination in the network identifies a "flow". For instance, Flow $\overrightarrow{AB}$ indicates the channel from A to B such that all premium packets from A to B are transmitted through $\overrightarrow{AB}$.

[2]More simulation details and extensive results and measurements are presented in our technical report [27].

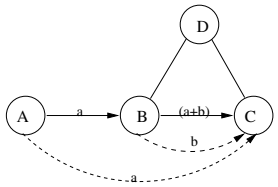


Figure 3: An example of bandwidth reservations for premium traffic: since link (B, C) is shared by flow $\vec{AC}$ and $\vec{BC}$, it must reserve $(a + b)$ amount of bandwidth.

Figure 3 depicts a scenario in which node A requires a amount of bandwidth to C while B requires b amount of bandwidth to C. Since the path $(A \rightarrow B \rightarrow C)$ is used for the flow $\overrightarrow{AC}$ according to SP routing, the link (B, C) is shared by flows $\overrightarrow{AC}$ and $\overrightarrow{BC}$, so it must reserve totally $(a + b)$ amount of bandwidth for both flows. As one can expect, with the variance of link capacities, the success of a reservation depends not only on how much bandwidth it tries to reserve and the capacity (more precisely, residual bandwidth) of each link along its path, but on the routing strategy as well.

In this paper, for both simplicity and expediency of problem definition, we assume each node in the topology tries to reserve same amount of bandwidth, say B, to all other nodes for its premium traffic. As we can see, given a network, for each routing scheme $\mathcal{R}$, there exists a maximum value of B, called the *saturated bandwidth* for $\mathcal{R}$ and denoted as $B_s(\mathcal{R})$, which saturates some link in the network. The saturated link is called the bottleneck link. Notice that the *saturate bandwidth* B_s is a virtual metric which is used only for finding better paths and for evaluating performance of different algorithms. (The larger B_s an algorithm can achieve, the better algorithm it will be.) In reality, we do not have to reserve that large amount of bandwidth for the premium traffic to always saturate some link(s) in the network. We can understand this from two different perspectives. Firstly, The routing scheme with maximum B_s is able to accommodate maximum amount of premium traffic in the network, thus maximizing the potential for future premium traffic growth. Secondly, in a real network, premium traffic may only reserve small portion of B_s bandwidth. Therefore, a routing scheme with larger B_s will leave more residual bandwidth to the best-effort traffic on those stringent links, thus reducing inter-class effects and bringing the whole network into a more load-balanced mode (for example, the chance of bandwidth starvation for low-priority traffic may be reduced). In fact, under the homogeneous bandwidth demand assumption, it is not difficult to verify that the optimality of the maximum saturate bandwidth is equivalent to the optimality of the minimum *relative congestion* among all links [9, 13, 15] (i.e., the ratio of the total amount of reserved bandwidth for the premium traffic, divided by the link capacity). As we can see from Figure 2, minimizing relative congestion of the premium traffic can minimize the negative inter-class effects on the low-priority traffic.

Therefore, an interesting problem is how to achieve the optimal value of B_s (denoted as B_{max}) by choosing optimal paths between nodes. More specifically,

$$B_{max} = max\{B_s(\mathcal{R}_1), B_s(\mathcal{R}_2), \cdots, B_s(\mathcal{R}_n)\},$$

where $\mathcal{R}_1$, $\mathcal{R}_2$, $\cdots$, $\mathcal{R}_n$ are all possible loop-free routing solutions for the given network. To find the optimal routing solution and the value of B_{max} is called the Optimal Premium-class Routing (OPR) problem and it is not a trivial task. An optimal local solution for a single source may not be the solution to the whole OPR problem. We will prove in Section 3 that this problem indeed is NP-hard. Since the SP routing algorithm is polynomial, it can not be optimal. Actually, as we will see later, it may be far away from the optimal solution. Using the same topology in Figure 1, but applying a different routing algorithm, Figure 4 illustrates a simple case where SP routing algorithm can NOT achieve B_{max}. From Figure 1 we know that, the saturate bandwidth that the SP algorithm can achieve is $B_s(\mathcal{R}_{sp}) = 10$ with link (B, C) as the bottleneck link. Figure 4 shows an optimal routing algorithm, which can achieve $B_{max} = 50$ with (A, B) as the bottleneck link.[3] Thus the problem that this paper will address turns out to be finding a heuristic hop-by-hop routing algorithm $\mathcal{R}$ which can achieve a better $B_s(\mathcal{R})$.

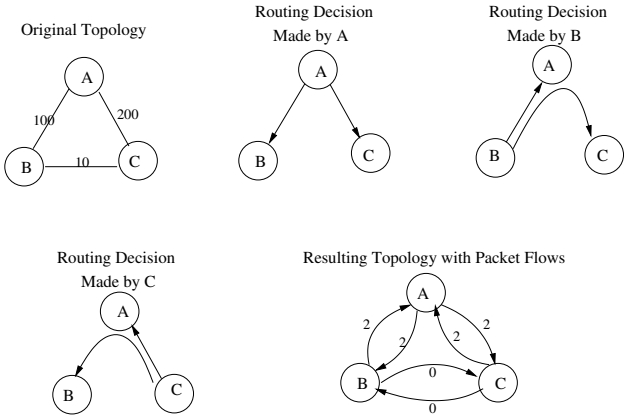


Figure 4: An example of an optimal algorithm for the given topology, which can achieve better saturate bandwidth allocation than the SP algorithm does.

In order to solve the OPR problem in polynomial time, heuristic approximation algorithms are needed. In section 4, we first apply two existing algorithms, the Widest-Shortest-Path (WSP) algorithm and the Bandwidth-inversion Short-Path (BSP) algorithm. Both of them are based on the generalized Dijkstra's algorithm. After showing both strength and weaknesses of them, we propose a novel approximation algorithm, called the Enhanced Bandwidth-inversion Shortest-Path (EBSP) algorithm. It is also a Dijkstra-based algorithm, therefore, it can run in $O(|V|^2)$ time, where $|V|$ is the total number of nodes in a given network. Extensive simulations in different network scenarios show that all three approximation algorithms outperform the hop-count

[3]Since link (A, B), which has capacity of 100, is shared by two flows $\overrightarrow{AB}$ and $\overrightarrow{CB}$, the maximum bandwidth for each flow is 50.

shortest-path algorithm on average. The results also confirm that, in a large-scale and heterogeneous network environment, our novel EBSP algorithm outperforms the other two existing algorithms in terms of much better saturate bandwidth.

We should emphasize that our contribution is in designing and evaluating routing algorithms which can find better paths for the premium traffic within a DiffServ domain, and in proving the correctness of the new EBSP algorithm, rather than in changing the DiffServ scheme itself. Moreover, we concentrate only on intra-domain IP layer routing.

The rest of the paper is organized as follows. After the presentation of our system model and assumptions in Section 2, we give rigorous NP-hardness proof of the OPR problem in Section 3. Three heuristic routing algorithms are then discussed and studied in Section 4. Simulations and results are illustrated in Section 5. Related work is covered in Section 6. Finally, Section 7 concludes this paper.

2. SYSTEM MODEL

We formally define a network model in this section. Based on this model, we give a more formal description of our assumptions and the problem we will address.

2.1 Network Model and Assumptions

Formally, a network is defined as a strongly connected directed graph $G(V, E)$, where V is the set of nodes (routers in the network) and E is the set of edges (links in the network), with cardinalities $|V|$ and $|E|$, respectively. Links are bidirectional with the same capacity in each direction. An edge from node x to y is represented as (x, y) and there is a positive bandwidth $b(x, y)$ associated with that edge. There is also a positive flow number $h(x, y)$, representing the total number of premium flows moving from x to y. Recall that all premium packets with the same source and destination addresses compose a premium flow. More generally, a weight function is associated with links, denoted as $w(x, y)$ if $(x, y) \in E$. The weight function may use metrics of a link, such as delay, bandwidth, hop count, etc., and map them into a real value. By default, we assume that the weight function is non-negative. A path p from v_1 to v_n is denoted as $p(v_1, v_n)$ (p_{v_1,v_n} for short) and $p(v_1, v_n) = \langle v_1, v_2, \cdots, v_{n-1}, v_n \rangle$, and it is *simple* if all nodes from v_1 to v_n are distinct. If v_1 and v_n are the same node, $p(v_1, v_n)$ forms a *loop*. We use $p \circ q$ to denote the concatenation of p and q. The same way as it is done for links, a weight function may be applied to paths too: $w(p(v_1, v_n)) = w(v_1, v_2) \oplus w(v_2, v_3) \oplus \cdots \oplus w(v_{n-1}, v_n)$, with $p(v_1, v_n)$ being a path from v_1 to v_n and "$\oplus$" being a binary operation. If the path p is empty, we have $w(p) = \bar{0}$. Weight values have a total order, denoted by "$\prec$". $w_1 \prec w_2$ means w_1 is *lighter* (better) than w_2, or w_2 is *greater* (worse) than w_1. For example, the capacity of a path is the minimum link capacity of all the links that comprise the path. Here $w_1 \oplus w_2$ means $min(w_1, w_2)$. Given a path $p(v_1, v_n) = \langle v_1, v_2, \cdots, v_{n-1}, v_n \rangle$ and a weight function w, specially, we define the *word-weight* of p, written $w_L(p)$, to be the sequence: $w_L(p) = w(p_{v_1,v_n})w(p_{v_1,v_{n-1}}) \cdots w(p_{v_1,v_2})$. Suppose we have two paths, p and p', with the word-weights $w_L(p) = \alpha_1\alpha_2 \cdots \alpha_n$ and $w(p') = \beta_1\beta_2 \cdots \beta_m$, respectively. We say p is *lexicographically lighter* than p' if either (i) $n < m$ and $\alpha_i = \beta_i$ for $1 \le i \le n$, or (ii) $\exists k$, $1 \le k \le \min(n, m)$, such that $\alpha_k \prec \beta_k$ and $\alpha_i = \beta_i$ for $1 \le i < k$. [25]

Given two nodes s and t in a network, we say that a path p^* is the:

- *lightest* path from s to t if $w(p^*) \preceq w(p)$, $\forall p$ from s to t;
- *S-lightest* path from s to t if p^* is a *lightest* path from s to t such that all of its sub-paths with source s are also *lightest* paths on their own;
- *L-lightest* path from s to t if its word-weight is lexicographically lighter than or equal to the word-weight of any other path from s to t.

Note that: (1) An S-lightest path is always a lightest path but not vice versa; and (2) An L-lightest path is always an S-lightest path but not vice versa.

For the premium class traffic, a bandwidth reservation from v_1 to v_n through a simple path $p(v_1, v_n)$ is successful if and only if the reservation with the same amount of bandwidth is successfully allocated on every link along the path. Therefore, each link (v_i, v_{i+1}) has to reserve certain amount of bandwidth not only for the premium traffic originated from v_i itself, but for all the transient traffic passing through all paths which are sharing the link as well.

Since premium class traffic experiences almost no queueing delays [11, 27], in this paper, we only consider bandwidth constraints.

The following are our assumptions:

1. Topology information can be obtained at each router by using some link-state routing protocol such as Open Shortest-Path First Protocol (OSPF).
2. Hop-by-hop routing is used in the given network.
3. Bandwidth demand for the premium traffic is homogeneous among all nodes, meaning that all nodes require the same amount of bandwidth to all other nodes for the premium traffic (denoted as B).

2.2 Optimal Premium-class Routing Problem

In Section 1, we have briefly introduced the Optimal Premium-class Routing (OPR) problem. Based on the system model and assumptions stated above, we give a more formal definition to the OPR problem as follows.

Given a network $G(V, E)$ with the bandwidth assignment $b(v_i, v_j)$ for any $v_i, v_j \in V$ and $(v_i, v_j) \in E$, and the assumptions outlined in 2.1, the OPR problem is to find the optimal routing scheme $\mathcal{R}_{opt}$ among all the possible loop-free hop-by-hop routing schemes on the network G (denoted as $\mathcal{R}_1, \mathcal{R}_2, \cdots, \mathcal{R}_n$), so that the optimal premium saturate bandwidth B_{max} is achieved, i.e., $B_s(\mathcal{R}_{opt}) = B_{max} = max\{B_s(\mathcal{R}_1), B_s(\mathcal{R}_2), \cdots, B_s(\mathcal{R}_n)\}$.

As stated in Section 6, a very interesting point of the OPR problem is that, an optimal local solution for a single source may not be the solution to the whole OPR problem. To address this problem, we need to consider both routing and bandwidth reservation in a global view.

2.3 Theoretical Definitions and Theorems

To solve the OPR problem, heuristic approximation algorithms are needed. Before introducing the heuristic algorithms, which we design and validate in Section 4, some fundamental definitions and theorems are presented in this section. We shall use them later to prove the correctness of those algorithms.

Since hop-by-hop routing scheme is used, in order to make the whole network work correctly, the routing algorithm should be able to guarantee the *consistency*, defined as follows.

DEFINITION 1. Routability: *A network $G(V, E)$ is said to be* routable *if every node $v \in V$ is able to find a simple path to all the other nodes.*

DEFINITION 2. Consistent routing: *Given a* routable *network $G(V, E)$, a routing scheme $\mathcal{R}$ is said to be* consistent *if (1) $\mathcal{R}$ can find a simple path between every pair of nodes in the network, i.e., for any two distinct nodes $s, t \in V$, $\mathcal{R}(s,t)$ is simple and non-empty, where $\mathcal{R}(s,t)$ denotes the path found by $\mathcal{R}$ between s and t; and (2) for any two distinct nodes $s, t \in V$, $\mathcal{R}(s,t) = \langle s, v_1, v_2, \cdots, v_n, t\rangle$ implies that $\mathcal{R}(v_i, t) = \langle v_i, v_{i+1}, \cdots, v_n, t\rangle$ (i.e., $\mathcal{R}(v_i, t)$ is the sub-path of $\mathcal{R}(s,t)$ between v_i and t) for all $i = 1, 2, \cdots, n$. That is, if node s makes a decision that the traffic to t will follow a certain path p, then all the on-path nodes along p should make the same decisions as s.*

THEOREM 1. *Given a routable network, a consistent hop-by-hop routing algorithm is loop-free.*

PROOF: The proof is straightforward. Given a routable network $G(V, E)$, every node $v \in V$ can find a simple path to every other node. Suppose a source node s chooses the path $\langle s, v_1, \cdots, v_n, t\rangle$ to the destination t, then according to the definition of *consistency*, $\langle v_i, v_{i+1}, \cdots, v_n, t\rangle$ is the sub-path from node v_i to t, for any $i = 1, \cdots, n$. By the definition of a *simple* path, there is no loop in the entire network. ■

Note that without consistency, loops may exist in the network, even if the path between every pair of nodes is simple. For example, if node a chooses the simple path $\langle a, b, c\rangle$ to node c, while node b chooses the simple path $\langle b, a, c\rangle$ to c, then there will be a forwarding loop between a and b in the network, although the two paths are loop-free individually.

Two topological properties, *isotonicity* and *strict isotonicity*, provide guarantees for the generalized Dijkstra's algorithms[4] to be consistent. Formal definitions of *isotonicity* and *strict isotonicity* were given in [25], and they state that the order relation between the weights of any two simple paths is preserved if both of them are either *prefixed* or *appended* by a common, third, simple path. That is, given two simple paths from s to t, say p and p', assuming $w(p) \preceq w(p')$ for some weight function w, then the *isotonicity* means: (i) if we prefix a common simple path q to both p and p', denoted as $q \circ p$ and $q \circ p'$ respectively, then $w(q \circ p) \preceq w(q \circ p')$; and (ii) if we append a common simple path q' to p and p', then $w(p \circ q') \preceq w(p' \circ q')$. In *strict isotonicity*, "$\preceq$" is replaced by "$\prec$". In [25], the authors also proved that if the strict isotonicity holds in a network for a weight function w, then the Dijkstra's algorithm can always find S-lightest paths (in terms of w) between nodes and it is sufficient to guarantee the routing consistency. However, if only the weaker condition, the isotonicity, holds, then a modified version of Dijkstra's algorithm, called the Dijkstra-Old-Touch-First (Dijkstra-OTF)[5], is needed to find L-lightest paths between nodes and guarantee the consistency.

As we will see next, an even weaker condition, called *left-isotonicity*, is sufficient to verify a *consistent* routing algorithm.

DEFINITION 3. Left-isotonicity: *Given a network topology G and a weight function w, let us consider any two paths p_1 and p_2, which are* prefixed *by a common path p, resulting in $p'_1 = p \circ p_1$ and $p'_2 = p \circ p_2$. Then the* **left-isotonicity** *holds, if $w(p_1) \preceq w(p_2)$ implies $w(p'_1) \preceq w(p'_2)$. Similarly, if $w(p_1) \prec w(p_2)$ implies $w(p'_1) \prec w(p'_2)$, then the* **strict left-isotonicity** *holds. If (strict) left-isotonicity holds for G, then G is said to be (strictly) left-isotonic.*

As we can see, the left-isotonicity states that the order relation between the weights of any two paths is preserved if both of them are *prefixed* by a common, third path. However, the order relation is NOT necessarily preserved if both of them are *appended* by a common, third, simple path.

Similarly, we define the *right-isotonicity* (R-isotonicity for short) by the other half of the isotonicity, that is, if two paths are appended by a common, third path, then their previous order relation holds.

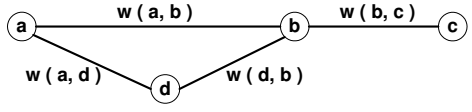


Figure 5: Suppose $w(a,d) \oplus w(d,b) \prec w(a,b)$ and $w(a,d) \oplus w(d,b) \oplus w(b,c) \succ w(a,b) \oplus w(b,c)$. Then neither S-lightest path nor L-lightest path exists between a and c.

Being weaker than isotonicity, left-isotonicity can not guarantee the existence of S-lightest or L-lightest paths between nodes. To see this, a simple example is shown in Figure

[4]In this paper, we use the term *generalized Dijkstra's algorithm* to refer to a Dijkstra-based algorithm which computes the "lightest", rather than the "shortest", path between two nodes. That is, the algorithm can handle some non-additive metrics.

[5]The Dijkstra-OTF computes L-lightest paths between nodes without explicitly computing or comparing word-weights.

5. Since the right half of isotonicity does not hold, it is valid that $w(a,d) \oplus w(d,b) \prec w(a,b)$ and $w(a,d) \oplus w(d,b) \oplus w(b,c) \succ w(a,b) \oplus w(b,c)$. Thus, path $\langle a,b,c \rangle$ is the only lightest path from a to c, and its sub-path $\langle a,b \rangle$ is not a lightest path. Therefore, neither S-lightest path nor L-lightest path exists from a to c in this scenario.

However, because we assume that weights are non-negative values, the lightest path always exists between each pair of nodes. In fact, we will argue later in Theorem 2 that given a strictly left-isotonic network G and a routing algorithm $\mathcal{R}$, $\mathcal{R}$ is consistent if it can guarantee to find the lightest path between every pair of nodes in G.

THEOREM 2. *Given a routable network $G(V,E)$, if the strict left-isotonicity property holds in G, then a routing algorithm $\mathcal{R}$ is consistent, as long as it can guarantee to always find the lightest path between any pair of nodes in G.*

PROOF: We prove this theorem by contradiction. Suppose there are multiple paths between a source node $s \in V$ and a destination node $t \in V$, and by running the algorithm $\mathcal{R}$ at s, a lightest path $p = \langle s, v_1, \cdots, v_n, t \rangle$ has been found. Let us assume $\mathcal{R}$ is not consistent, i.e., $\exists i, 1 \leq i \leq n$, such that v_i is the first node along the path p which picks up another different path $p' = \langle v_i, v'_{i+1}, \cdots, v'_j, t \rangle$ as the lightest path from v_i to t. Then, $w(p') \prec w(p_{i,t})$, where $p_{i,t} = \langle v_i, v_{i+1}, \cdots, v_n, t \rangle$ is the sub-path of p between v_i and t. Following the strict left-isotonicity, if we append a common prefix $q = \langle s, v_1, \cdots, v_i \rangle$ onto both p' and $p_{i,t}$, the order relation between two paths should be preserved, i.e., $w((q \circ p')) \prec w((q \circ p_{i,t}))$, where "$\circ$" stands for path concatenation. As we know, $q \circ p_{i,t} = p$, that is, $w((q \circ p')) \prec w(p)$. Therefore, p is NOT the lightest path between s and t, which contradicts our assumption and completes the proof. ■

Since another half of the isotonicity property (*Right-isotonicity*) does not hold, no existing Dijkstra-based algorithm can guarantee to find the lightest path between every pair of source and destination, thus incurring inconsistency. The following figure shows a counterexample.

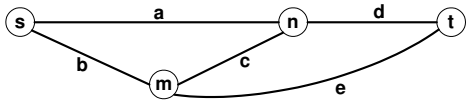


Figure 6: Suppose $a \prec (b \oplus c), (a \oplus d) \succ (b \oplus c \oplus d), (c \oplus d) \prec e, (b \oplus c \oplus d) \prec (b \oplus e), (b \oplus e) \prec (a \oplus d)$. Then inconsistency occurs.

In the case of Figure 6, we assume that:

(1) $a \prec (b \oplus c)$

(2) $(a \oplus d) \succ (b \oplus c \oplus d)$

(3) $(c \oplus d) \prec e$

(4) $(b \oplus c \oplus d) \prec (b \oplus e)$

(5) $(b \oplus e) \prec (a \oplus d)$

where (1) and (2) both hold because *R-isotonicity* is NOT preserved. On the other hand, both (3) and (4) hold due to *L-isotonicity* property. The relationship given in (5) leads to an inconsistency.

By running Dijkstra's algorithm at node s, the path from s to t found by node s will be $\langle s,m,t \rangle$. Obviously, it is NOT the lightest path $\langle s,m,n,t \rangle$.[6] However, if we run the same Dijkstra's algorithm at the on-path node m, the path from m to t found by m will be $\langle m,n,t \rangle$. Inconsistency occurs. Therefore, any existing generalized Dijkstra's algorithms are not sufficient to provide consistent routing any more. The fundamental reason behind this is that the existing generalized Dijkstra's algorithms can NOT find the lightest paths if such paths contain any non-optimal sub-paths (i.e., when the *isotonicity* does not hold). So, a new Dijkstra-based algorithm is needed to find lightest paths between nodes in strictly left-isotonic networks.

However, we observe that the strict left-isotonicity property does have a very nice feature which is shown in the following lemma.

LEMMA 1. *Given a strictly left-isotonic network $G(V,E)$, if a path $p = \langle s, v_1, \cdots, v_n, t \rangle$ is the lightest path between two nodes $s, t \in V$, then every sub-path $p_{i,t} = \langle v_i, \cdots, v_n, t \rangle$ (where $1 \leq i \leq n$) is the lightest path from v_i to t.*

PROOF: We can prove this lemma by contradiction, which is similar to the proof of Theorem 2. Due to space limitation, we omit the detailed proof in this paper. ■

Following Lemma 1, the lightest path from s to t must be based on the lightest paths from any intermediate nodes to t. If we consider a destination node t, and the lightest paths from all other nodes to t, then we can see that the lightest paths form exactly a *spanning tree* for the given network G rooted at t. Therefore, to find the lightest path from any s to t, we can start from the destination node t and perform relaxation step-by-step backward to s. Based on the above observation, we present a new algorithm WN-DIJKSTRA to find the lightest path from any s to t in a strictly left-isotonic network G, as shown in Algorithm 1.

In WN-DIJKSTRA, $\varpi[u]$ denotes the successor (or next hop node) of u, and $d[v]$ denotes the weight of the current path from v to t. The function call EXTRACT-MIN(Q) is the same as in the original Dijkstra's algorithm, which extracts a node from set Q with the lightest value of $d[v]$.

It is clear that WN-DIJKSTRA takes $O(|V|^2)$ execution time, the same as the original Dijkstra's algorithm.

THEOREM 3. **(Correctness of WN-Dijkstra's algorithm)** *If we run WN-Dijkstra's algorithm on a strictly*

[6]The direct link (s,n) is chosen by s as the lightest path to n, since $a \prec (b \oplus c)$ (Condition (1) above). Thereafter, (s,n) and its weight a, rather than the non-optimal sub-path $\langle s,m,n \rangle$ and its weight $(b \oplus c)$, are used to continue the relaxation from node n. This eliminates the chance of finding the real lightest path, which passes exactly through $\langle s,m,n \rangle$, since Condition (4) and (5) hold above.

```
WN-DIJKSTRA(G(V,E), w, s, t)
 1  for each node v ∈ V
 2  do d[v] ← ∞
 3     ϖ[v] ← NIL
 4  d[t] ← 0
 5  Q ← V
 6
 7  while Q ≠ ∅
 8  do u ← EXTRACT-MIN(Q)
 9     if u = s
10       then EXIT
11     for each v ∈ Adj[u]
12     do if w(v,u) ⊕ d[u] ≺ d[v]
13          then ϖ[v] ← u
14               d[v] ← w(v,u) ⊕ d[u]
```

Algorithm 1: WN-Dijkstra algorithm which finds the lightest path from s to t.

left-isotonic network $G(V,E)$ with non-negative weight function w, source s and destination t, then the lightest path is found from s to t with $d[s] = (weight\ of\ the\ lightest\ path)$ at termination.

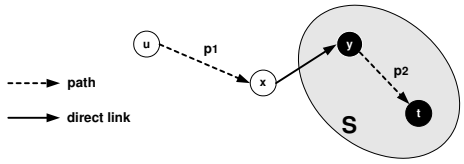


Figure 7: The proof of Theorem 3: path p can be decomposed into two sub-paths p_1 and p_2.

PROOF: Let $\delta(a,b)$ denote the weight of the lightest path from a to b. We claim that for each node $u \in V$, we have $d[u] = \delta(u,t)$ at the time when u is extracted from Q and this equality is maintained thereafter. Since $s \in V$, this claim holds for s, too. For convenience, we let S be the set of nodes that are extracted from Q before u.

We shall show this claim by contradiction. Let u be the first node for which $d[u] \neq \delta(u,t)$ when it is extracted from Q. Clearly, $u \neq t$ because t is the first node extracted from Q and $d[t] = \delta(t,t) = \bar{0}$ when it is extracted from Q. Because $u \neq t$, we have $S \neq \emptyset$ just before u is extracted from Q. Since the network is routable, there must be some path from u to t. Thus there must be a lightest path p from u to t, for the weight function is non-negative. Without loss of generality, we assume that x is the first node along p such that $x \in (V - S)$ (i.e., x is the first node along p which has not been extracted from Q yet), and let $y \in S$ be x's direct successor, i.e., $y = \varpi[x]$. Then p can be decomposed into two sub-paths p_1 and p_2 at x, as shown in Figure 7.

Now we show that $d[x] = \delta(x,t)$ when it is extracted from Q. Because u is chosen as the first node for which $d[u] \neq \delta(u,t)$ when it is extracted from Q, and because y is extracted from Q before u, we have $d[y] = \delta(y,t)$ when y is extracted from Q. Since p is the lightest path from u to t, and G is strictly left-isotonic, by Lemma 1, the sub-path from x to t is also a lightest path and thus $d[x] = w(x,y) \oplus d[y] = w(x,y) \oplus \delta(y,t) = \delta(x,t)$.

Because the weight function is non-negative, clearly we have $\delta(x,t) \preceq \delta(u,t)$, and thus

$$d[x] = \delta(x,t) \preceq \delta(u,t) \preceq d[u] \quad (1)$$

However, because x is still in Q when u is extracted from Q, we have $d[u] \preceq d[x]$. By Equation 1, the following equations hold:

$$d[x] = \delta(x,t) = \delta(u,t) = d[u]$$

Therefore, $d[u] = \delta(u,t)$, which contradicts our choice of u. We conclude that for any node $s \in V$, when it is extracted from Q, $d[s] = \delta(s,t)$, and the path found and stored in $\varpi[s]$ is the lightest path from s to t. ■

By Theorem 2 and Theorem 3, the WN-DIJKSTRA algorithm provides a *consistent* routing.

In Section 4, we will provide three heuristic algorithms to address the OPR problem. Their correctness is proven based on the definitions and theorems introduced above.

3. NP-HARDNESS OF OPR PROBLEM

In the hop-by-hop routing context, the routing consistency must be guaranteed (Section 2.3). Therefore, in a feasible solution to the OPR problem, if we consider any destination node t, and paths that all other nodes are taking to t, then the paths must form a spanning tree for the entire network rooted at t itself.

For the purpose of showing that the OPR problem is NP-Hard, it can be replaced by the following simplified problem P1.

P1 - *The simplified OPR problem*
INSTANCE: A network $G(V,E)$ and a destination node $t \in V$, where all edges in E have the same constant capacity C, and C is a non-negative integer.
QUESTION: Is there a spanning tree T in G, rooted at t, that satisfies the following constraint: for $\forall e \in T$, if $U(e) = \{u \in V |$ the path from u to t contains $e\}$, then $|U(e)| = (\sum_{u \in U(e)} 1) \leq C$?

As we can see, the problem P1 simplifies the original OPR problem in the following two senses: (1) The link capacities are unified to a constant integer C; (2) Instead of considering all nodes in V, only one node $t \in V$ is considered in P1. It is apparent that the original OPR problem is harder than P1. Therefore, our objective is to show that even Problem P1 is hard to solve.

Actually, the problem P1 is also a simplified variant of the *Capacitated Minimum Spanning Tree* problem in [7], where the length constraint is lifted and the edge capacities are unified to the same constant integer C.

At the first look, problem P1 seems very simple. If there is not the tree constraint (i.e., if T is not necessary to be a

tree), the problem becomes a special case of the *Maximum-Flow problem*, which is polynomially solvable by using the Ford-Fulkerson method [5]. However, problem P1 is surprisingly difficult to solve. Indeed, it is NP-hard. The NP-hardness is due to the combination of the tree constraint (due to the consistency requirement from hop-by-hop routing) and its essence of the *Bin Packing problem* [7]. It is clear that, the tree constraint deprives the nodes of their independence to choose paths to the destination t, thereby making it hard to solve. In fact, the NP-hardness holds even for a *Directed Acyclic Graph* (DAG).

Before we give the formal NP-hardness proof of P1, we briefly introduce one existing NP-hard problem, the *Non-uniform Load Balancing problem* [14, 10], from which we shall construct our reduction to P1. For ease of notation, we denote the *Non-uniform Load Balancing problem* as P0.

P0 - *The Non-uniform Load Balancing problem*
INSTANCE: A set of jobs $J = j_1, j_2, \cdots, j_k$, and a set of machines $M = m_1, m_2, \cdots, m_n$; for each job $j_i \in J$, there is a set $S_i \subset M$ on which j_i can be run; each job j_i has a *requirement* r_i which is equal to either 1 or 2.
QUESTION: Is there an assignment from J to M such that each job $j \in J$ is assigned to a machine $m \in M$ so that the sum of the requirements assigned to each machine is at most 2?

Problem P0 is NP-hard [14, 10]. Now we prove that the problem P1 is NP-hard, too.

THEOREM 4. *The simplified OPR problem,* P1*, is NP-hard.*

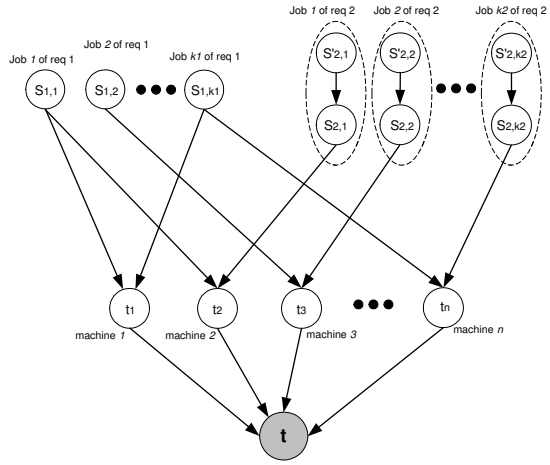


Figure 8: The NP-hardness reduction

PROOF: The reduction algorithm begins with an instance of P0. We construct a graph $G(V, E)$ to encode the instance of P0 step by step as follows. The construction is also shown in Figure 8.

(1) Without loss of generality, we suppose that, among k jobs in J, there are k_1 jobs with requirements of 1 and k_2 jobs with requirements of 2. For simplicity of notation, we denote by J_1 the set of jobs with requirements of 1 and J_2 the set of jobs with requirements of 2, respectively. Then, we have $|J_1| = k_1$ and $|J_2| = k_2$. It is apparent that $J = J_1 \bigcup J_2$ and $k = k_1 + k_2$.

(2) For each job j_i in J_1, we add one single node $s_{1,i}$ into V. For each job j_l in J_2, firstly, we add two nodes $s_{2,l}$ and $s'_{2,l}$ into V; secondly, a link between these two nodes, $(s'_{2,l}, s_{2,l})$ is added into E, as illustrated in Figure 8.

(3) For each machine m_i in M, we place a node t_i into V as well.

(4) A termination node, t, is added into V. For each node t_i, a link (t_i, t) is added into E, which connects t_i to t. So far, the set of nodes, V, is completed. We have

$$V = \{s_{1,1}, \cdots, s_{1,k_1}\} \cup \{s_{2,1}, \cdots, s_{2,k_2}\} \cup \{s'_{2,1}, \cdots, s'_{2,k_2}\} \cup \{t_1, \cdots, t_n\} \cup \{t\}$$

and $|V| = k_1 + 2k_2 + n + 1 = k + k_2 + n + 1$.

(5) For each job j_i in J, since it has a machine set $S_i \subset M$ on which it can run, for each machine $m_u \in S_i$, if j_i is in J_1, then we add one link $(s_{1,i}, t_u)$ into E; or if j_i is in J_2, then we add a link $(s_{2,i}, t_u)$ into E instead. Through these links, we encode the condition that job j_i can only be assigned to a machine in S_i, that is, node t_u is reachable from $s_{x,i}$ if and only if $m_u \in S_i$, where $x = 1$ or 2. Till now, the link set E is completed, too.

(6) Finally, we assign all links in E with the same capacity $C = 3$.

Figure 8 illustrates the construction of G. It is clear that such construction of G can be completed within polynomial time. Now the Problem P0 has been transformed into an instance of Problem P1 where the constructed G is the graph in P1, $C = 3$ and the question is to look for a spanning tree T for G that satisfies the capacity constraints.

We show that this transformation of the instance of P0 into G is a reduction. First, if there is a feasible allocation of jobs to machines in Problem P0, then it is not difficult to come up with a spanning tree T for G accordingly. Specifically, for any job $j_i \in J_1$, if it is assigned to machine m_u, then we take the corresponding link $(s_{1,i}, t_u)$; for any job $j_i \in J_2$, if it is assigned to machine m_u, then we take both links $(s'_{2,i}, s_{2,i})$ and $(s_{2,i}, t_u)$. Furthermore, we take all links (t_u, t). As such, we show that the resulting sub-graph, G', is a spanning tree for G: (1) since all jobs are assigned to some machines, all s_i nodes and s'_j nodes are covered in the resulting sub-graph $G'(V', E')$; since all links (t_u, t) are taken, all t_u nodes and t are covered. That is, all nodes in V are covered in G' $(V = V')$. (2) Since each job can be assigned to at most one machine, there are exactly k $(s_{x,i}, t_u)$ links in E', where $x = 1$ or 2; plus k_2 links $(s'_{2,i}, s_{2,i})$ and n links (t_u, t), we have totally $k + k_2 + n = k_1 + 2k_2 + n$ links in E'. By the construction of G, $|V| = k_1 + 2k_2 + n + 1$. So we have $|E'| = |V| - 1$, thereby making G' a spanning tree T rooted at t. (3) Since each machine has capacity 2, it is apparent that $|U(e)| = (\sum_{u \in U(e)} 1) \leq 3$ for any $e \in E'$, that is, the capacity constraints are simply satisfied. Therefore, T is a solution to Problem P1 on G.

Conversely, suppose that there is a solution T satisfying P1, i.e., T is a spanning tree for G rooted at t such that the capacity constraint of each link is preserved. Since $C = 3$, for each node t_u, at most 2 nodes can be contained in the sub-tree rooted at t_u. (Node t_u itself consumes 1 unit from the link (t_u, t). Therefore, at most two other paths can pass through this link) Since for each node $s'_{2,i}$, there is no direct link between it and t_u or t (there is only one link connecting it to corresponding $s_{2,i}$), its path to t has to pass through the corresponding node $s_{2,i}$. Furthermore, the path has to follow the same path from $s_{2,i}$ to t (otherwise, T will no longer be a tree). So the nodes $s'_{2,i}$ and $s_{2,i}$ are bound to each other in T. Hence, each sub-tree rooted at t_u can only have the following four options: contains 0 node, or any 1 node $s_{1,i}$, or any 2 nodes $s_{1,i}$ and $s_{1,j}$, or any one pair of bound nodes $s'_{2,i}$ and $s_{2,i}$. Following the spanning tree T, we can easily find an allocation of jobs accordingly: for any link $(s_{x,i}, t_u)$ in T, we just simply assign the corresponding job j_i to the machine m_u. As we discussed above, the link capacity constraint $C = 3$ guarantees that the workload at each machine does not exceed 2. Moreover, since all nodes in V are covered by T, all jobs are assigned. Therefore, the corresponding allocation is a solution to the instance of Problem P0. ■

As we can see from the proof and Figure 8, the NP-hardness holds even for a *Directed Acyclic Graph* (DAG).

4. ROUTING ALGORITHMS FOR PREMIUM TRAFFIC

The OPR problem, as we have defined in Section 2 and studied in Section 3, turns out to be very difficult to find an optimal solution in polynomial time since it is NP-hard. Therefore, in this section, we introduce two existing heuristic hop-by-hop routing algorithms that are based on the generalized Dijkstra's algorithm, as well as one novel algorithm based on the WN-Dijkstra's algorithm (Section 2). All of them can run in $O(|V|^2)$ [5].

4.1 Widest-Shortest-Path Algorithm (WSP)

The WSP algorithm is the simplest heuristic algorithm which can achieve a better saturate bandwidth ($B_s(\mathcal{R}_{wsp})$) than the basic hop-count shortest-path (SP) algorithm. When a tie occurs, the basic SP algorithm simply chooses the node with the smallest identifier to break the tie. However, the WSP always chooses the widest path among the set of shortest paths between any pair of source and destination. The WSP has been well-studied in [25, 15]. Although it does not have a *strict* isotonicity, the isotonicity still holds. Therefore, by using the Dijkstra-OTF algorithm proposed in [25], we can guarantee that WSP finds a loop-free L-lightest path from any source to any destination.

The simulation results (given in Section 5) show that, on average, the WSP algorithm outperforms the SP algorithm. However, since the WSP only chooses a path from those "shortest" paths (in the sense of hop-count) between two nodes, the gain is limited.

4.2 Bandwidth-inversion Shortest-Path Algorithm (BSP)

In order to overcome the WSP's limitation of performance gain, we need to choose a "best" path from a broader range. Therefore, the BSP algorithm is introduced. The BSP algorithm was studied and called the "Shortest-distance path algorithm" in [15]. It was used in a call-based or connection-oriented network (such as the ATM network).

The BSP is basically a shortest-path algorithm with the distance or weight function defined as

$$w(v_i, v_j) = \frac{1}{b_{i,j}}$$

and

$$w(p\langle v_1, v_2, \cdots, v_n\rangle) = \sum_{i=1}^{n-1} \frac{1}{b_{i,i+1}}$$

where $b_{i,j} = b(v_i, v_j)$ is the bandwidth of link (v_i, v_j). Since the weight function is additive and the strict isotonicity property holds, the BSP algorithm with this weight function can guarantee that packets are transmitted through the lightest path between any pair of source and destination without loop [25]. Therefore, it can be used as a hop-by-hop routing algorithm as well. If there are more than one lightest paths between the source and destination, the path with the least hop count is selected.

Although generally BSP can achieve much better saturate bandwidth ($B_s(\mathcal{R}_{bsp})$) than SP or WSP (the simulation results will be shown in Section 5), its performance varies drastically if topology changes, meaning that the chance of producing a worse $B_s(\mathcal{R}_{bsp})$ than the SP is high. For example, in Figure 1, if we change the $b(A, C)$ to 1000 and $b(B, C)$ to 90, then $B_s(\mathcal{R}_{bsp}) = 50$ and $B_s(\mathcal{R}_{sp}) = 90$, that is, the BSP gets even worse saturate bandwidth than the SP does.

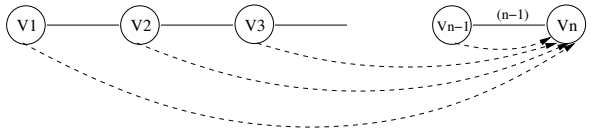


Figure 9: Links in a longer path have to handle more flows, therefore, B_s decreases: for example, according to the consistency of hop-by-hop routing, link (v_{n-1}, v_n) has to hold $(n-1)$ flows: $v_1\vec{v}_n, v_2\vec{v}_n, \cdots, v_{n-1}\vec{v}_n$. The longer the path is, the smaller a B_s will be.

The reason behind the failure of BSP is that it prefers a wider path too much. In fact, being related to both the link capacities along the path and the number of flows which are taking this path, there are two constraints behind the OPR problem: (1) On the one hand, a wider path may achieve higher saturate bandwidth; (2) On the other hand, when the wider path grows longer, according to the consistent routing policy, more nodes, hence more flows will have to share the same path, resulting in a decrease of the saturate bandwidth.[7] This phenomenon is illustrated in Figure 9.

[7]According to the hop-by-hop routing assumption, every node makes routing decisions independently without any coordination between each other. Therefore, taking only the bandwidth into consideration, we may experience the following behavior: if one node chooses a wider link, then other

Therefore, in order to prevent putting too many flows onto a wide path, the length of the chosen path should be carefully limited. Intuitively, a "penalty" associated with the hop count of a path can be used to prevent that it becomes too long. Hence, a novel algorithm based on the BSP is introduced as follows.

4.3 Enhanced Bandwidth-inversion Shortest-Path Algorithm (EBSP)

Intuitively, the introduction of a "penalty" helps to prevent a path becoming too long. The value of such "penalty" is related to hop count value. In order to guarantee the routing consistency, the new weight function with "penalty" should hold at least the left-isotonicity property (Section 2). One direct option is to add a constant term to the link cost, e.g. for a link of bandwidth b, its weight function can be (say) $1/b + c$, where c is a non-negative constant [22]. (The weight function of BSP then becomes a special case when $c = 0$.) However, the "penalty" may not necessarily be linear with the hop count. We claim that this penalty is likely to be exponential with respect to the hop count. The reason is briefly presented as follows. If we consider only one destination t, the routes from all the other nodes to t forms a spanning tree (also see Section 3). Hence, given a link e, the totally number of routes going through e to t could increase exponentially with respect to the depth of the sub-tree rooted at one endpoint of e. Therefore, if we observe conversely from a node v in the downstream of e, when we reach e during searching the route from v to t, the number of routes on e could potentially be an exponential function of the hop count of e along the current searching path. Therefore, we should use an exponential penalty in the new weight function.

The new weight function for a path $p(v_1, v_n) = \langle v_1, v_2, \cdots, v_n \rangle$ is then defined as follows.

$$w(p) = \sum_{i=1}^{n-1} \frac{\theta^{i-1}}{b_{i,i+1}} \tag{2}$$

where θ is a constant discount factor (in the first part of our simulation, we let $\theta = 2$). We can see that: (1) θ^i serves as an exponential penalty; and (2) for any two paths p_1 and p_2, $w(p_1) \prec w(p_2)$ if and only if $w(p_1) < w(p_2)$. Another interesting observation is that the BSP algorithm now becomes one special case of EBSP in which $\theta = 1$. On the other hand, when θ is set to be large enough, the EBSP tends to be another variant of the SP algorithm, just like the WSP. This observation leads us to a hint that we could find an optimal θ^* such that the EBSP algorithm yields even larger saturate bandwidth. We will address this issue later in our simulations.

Notice that the new weight function is no longer static. It dynamically changes with the hop count value. Before we use a generalized Dijkstra algorithm to find the lightest path between two nodes in terms of this new weight function, we should show that the strict left-isotonicity holds

nodes are very likely to choose the same link too, resulting in the decrease of B_s. This confirms our earlier statement that a local optimal solution for a single source node may not necessarily be the global optimal solution to the OPR problem.

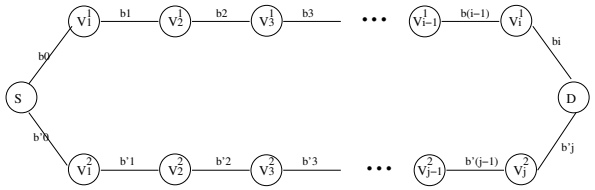


Figure 10: The topology used to prove the left-isotonicity of the weight function in Equation 2. Originally, there are two paths between node S and D: $p_1 = \langle S, v_1^1, v_2^1, \cdots, v_i^1, D \rangle$ and $p_2 = \langle S, v_1^2, v_2^2, \cdots, v_j^2, D \rangle$. We assume that p_1 is lighter than p_2.

for this new weight function. Suppose we have two paths from node S to D: $p_1 = \langle S, v_1^1, v_2^1, \cdots, v_i^1, D \rangle$ and $p_2 = \langle S, v_1^2, v_2^2, \cdots, v_j^2, D \rangle$, as shown in Figure 10. For convenience, let $S = v_0^1 = v_0^2$ and $D = v_{i+1}^1 = v_{j+1}^2$. According to the definition in Equation 2, we have

$$w(p_1) = \sum_{m=0}^{i} \frac{\theta^m}{b_m}$$

and

$$w(p_2) = \sum_{m=0}^{j} \frac{\theta^m}{b'_m}$$

where $b_m = b(v_m^1, v_{m+1}^1)$ and $b'_m = b(v_m^2, v_{m+1}^2)$. Without loss of generality, we suppose that p_1 is lighter than p_2, i.e., $w(p_1) \prec w(p_2)$. Then we have

$$\sum_{m=0}^{i} \frac{\theta^m}{b_m} < \sum_{m=0}^{j} \frac{\theta^m}{b'_m} \tag{3}$$

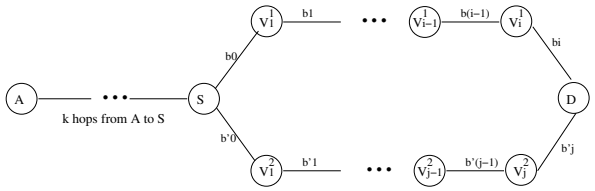


Figure 11: Topology used to prove the left-isotonicity of the weight function in Equation 2. After a common prefix is added to the original topology (Figure 10) at node S, we can prove that $p'_1 = \langle A, \cdots, S, v_1^1, v_2^1, \cdots, v_i^1, D \rangle$ is still lighter than $p'_2 = \langle A, \cdots, S, v_1^2, v_2^2, \cdots, v_j^2, D \rangle$. It means, the strict left-isotonicity holds.

Now a common prefix $q = \langle A, \cdots, S \rangle$ is added to the node S as shown in Figure 11. Suppose there are k hops from node A to S, then the weights for the two augmented paths from A to D, $p'_1 = q \circ p_1 = \langle A, \cdots, S, v_1^1, v_2^1, \cdots, v_i^1, D \rangle$ and $p'_2 = q \circ p_2 = \langle A, \cdots, S, v_1^2, v_2^2, \cdots, v_j^2, D \rangle$, are

$$w(p'_1) = w(A, S) + \sum_{m=0}^{i} \frac{\theta^{m+k}}{b_m} = w(A, S) + \theta^k \sum_{m=0}^{i} \frac{\theta^m}{b_m}$$

and

$$w(p_2') = w(A,S) + \sum_{m=0}^{j} \frac{\theta^{m+k}}{b'_m} = w(A,S) + \theta^k \sum_{m=0}^{j} \frac{\theta^m}{b'_m}$$

respectively. Following the inequality in Equation 3, we can easily draw the conclusion that $w(p_1') \prec w(p_2')$. Therefore, by Definition 3, the strict left-isotonicity holds. We can also show easily that the new weight function is NOT isotonic.

Following Theorem 2 and Theorem 3, an enhanced WN-Dijkstra's algorithm can be used to produce a consistent routing scheme for a network based on the new weight function given in Equation 2. We give the detailed description of this enhanced WN-Dijkstra's algorithm as follows, which is called *WN-Dijkstra-EBSP*.

```
WN-DIJKSTRA-EBSP(G(V,E), b, s, t)
 1  for each node v ∈ V
 2  do d[v] ← ∞
 3     ϖ[v] ← NIL
 4  Q ← V
 5  d[t] ← 0
 6
 7  while Q ≠ ∅
 8  do u ← EXTRACT-MIN(Q)
 9     if u = s
10        then EXIT
11     for each v ∈ Adj[u]
12     do if θ × d[u] + 1/b(v,u) < d[v]
13           then ϖ[v] ← u
14                d[v] ← θ × d[u] + 1/b(v,u)
```

Algorithm 2: WN-Dijkstra-EBSP algorithm which handles the OPR problem.

The asymptotic execution time of WN-DIJKSTRA-EBSP is $O(|V|^2)$, which is the same as the original Dijkstra's algorithm. Notice that we use the new weight function in Equation 2 to compute the weight for a path in line 12 and 14, which depends on the current node's hop count value and the bandwidth of the associated link.

By taking both the hop count and bandwidth into consideration, we expect that the EBSP algorithm can outperform both the SP and BSP algorithms with respect to the saturate bandwidth (i.e., $B_s(\mathcal{R}_{ebsp})$ should be better than $B_s(\mathcal{R}_{sp})$ and $B_s(\mathcal{R}_{bsp})$). We will show in Section 5 that the simulation results confirm our expectations very well.

5. SIMULATIONS AND RESULTS

In the previous section, we introduced three heuristic algorithms to solve the OPR problem. Their performance varies in different network topologies. To reveal the relationship between the performance and underlying network topologies, as well as to show the advantage of our new EBSP algorithm, extensive simulations have been carried out. In this section, we present these simulations and their results. Note that, although the algorithms can be used in a dynamic manner (for example, they can be plugged into a link state protocol, such as the OSPF), our simulations assume static routing. That is, given a topology, we perform routing statically.

5.1 Simulation Model

We design a topology generator to automatically generate topologies. The parameters we use for modeling topologies are: (1) Number of nodes N; (2) Maximum degree of each node D; and (3) Variance index of link capacities C_{var}. Specifically, we first generate N different nodes. Then, given a specific value of D, we generate a random number within $[1, D]$ for each node as its degree. The link destinations are also randomly selected among all nodes except the node itself. We normalize the base link capacity to 100 units and quantize C_{var} into 10 levels, from 1 to 10. For each link, we assign its capacity based on the value of C_{var}. For example, if $C_{var} = c, c \in [1, 10]$, we generate a random number between $[100, 100c]$ and assign it to a link as its capacity. The larger C_{var} is, the greater variance of link capacities. In this way, we can adjust the variance of link capacities easily by simply changing the value of C_{var}.

Since the OPR problem is NP-hard, it is impossible to find the optimal saturate bandwidth B_{max} for a given topology within polynomial time. Hence, comparisons between B_{max} and $B_s(\mathcal{R}_{wsp})$, $B_s(\mathcal{R}_{bsp})$, and $B_s(\mathcal{R}_{ebsp})$ are therefore infeasible. In order to quantify the performance and compare the three algorithms, we use the SP algorithm (the basic shortest-path algorithm) as our benchmark. Given a randomly generated topology G,[8] we first run the SP algorithm at each node of G so that the saturate bandwidth $B_s(\mathcal{R}_{sp})$ is obtained. Then, the three heuristic algorithms, WSP, BSP and EBSP, are executed on the same topology one by one. Their saturate bandwidths, $B_s(\mathcal{R}_{wsp})$, $B_s(\mathcal{R}_{bsp})$, and $B_s(\mathcal{R}_{ebsp})$ are obtained, respectively. We compare these three values with the benchmark, $B_s(\mathcal{R}_{sp})$, to observe how much benefit we can gain by using these three algorithms.

More specifically, we evaluate the performance of a particular algorithm $\mathcal{R}$ by using two metrics defined as follows.

(1) *Bandwidth Gain (β)* and *Average Bandwidth Gain ($\bar{\beta}$)*: β is defined as

$$\beta = \frac{B_s(\mathcal{R})}{B_s(\mathcal{R}_{sp})} \quad ,$$

where $B_s(\mathcal{R})$ can be $B_s(\mathcal{R}_{wsp})$, $B_s(\mathcal{R}_{bsp})$, or $B_s(\mathcal{R}_{ebsp})$. For each configuration (N, D, C_{var}), totally 2000 topologies are randomly generated and simulated. The *Average Bandwidth Gain* ($\bar{\beta}$) is then computed by

$$\bar{\beta} = \frac{\sum_1^{2000} \beta}{2000} \quad .$$

The larger $\bar{\beta}$ the algorithm $\mathcal{R}$ can achieve, the better.

(2) *Miss Rate (γ)*: For each given configuration (N, D, C_{var}), γ is simply measured as the fraction of total 2000 simulated topologies in which the algorithm $\mathcal{R}$ yields worse saturate bandwidth than $\mathcal{R}_{sp}$ does (i.e., $B_s(\mathcal{R}) < B_s(\mathcal{R}_{sp})$). That is,

$$\gamma = \frac{\text{Number of topologies in which } B_s(\mathcal{R}) < B_s(\mathcal{R}_{sp})}{\text{Total number of simulated topologies (2000)}}.$$

[8] If G is not *routable*, we simply drop it and generate another one.

The smaller γ, the better.

The following subsection demonstrates the detailed simulation results and analysis.

5.2 Simulation Results and Analysis

We present our simulation results in three parts. The first part (Section 5.2.1) focuses on the *Average Bandwidth Gain ($\bar{\beta}$)* and the *Miss Rate (γ)* with respect to different topology configurations, where we fix $\theta = 2$ in the EBSP algorithm. In the second part (Section 5.2.2) we study how θ affects the performance of the EBSP algorithm. Finally, we make a stronger and more realistic case in the third part (Section 5.2.3), applying some larger and more realistic network topologies generated by the famous topology generator BRITE. Note that in the first two parts, we do not intend to simulate realistic networks or large-scale ones. We focus only on the algorithms' average performance and its relationship with different topology configurations. Thus, a large number of topologies (2000 for each topology configuration) are randomly generated and simulated. While in the third part, we focus on evaluating the algorithms' performance in more realistic networks with much more nodes (up to 500).

5.2.1 Performance In Various Topologies

We illustrate the detailed simulation results in Figures 12, 13, 14 and 15. Each value of *bandwidth gain* or *miss rate* in these figures is the *average* value on 2000 randomly generated topology samples for a given topology configuration. So the results in this part reflect the overall performance of the algorithms.

We fix $\theta = 2.0$ for the EBSP algorithm in this part. How different θ values may affect the algorithm and the optimal θ values will be addressed next in Section 5.2.2.

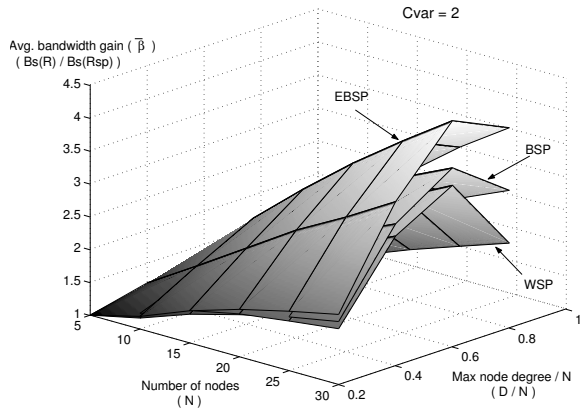


Figure 12: Average bandwidth gain ($\bar{\beta} = B_s(\mathcal{R})/B_s(\mathcal{R}_{sp})$) w.r.t. topology configurations ($C_{var} = 2$)

The figures show us some interesting and useful relations between performance of algorithms and different topology configurations.

General comparisons among WSP, BSP and EBSP: In general, the WSP is the best one among the three algorithms

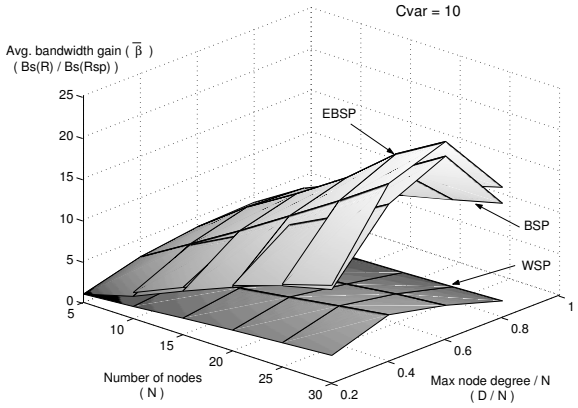


Figure 13: Average bandwidth gain ($\bar{\beta} = B_s(\mathcal{R})/B_s(\mathcal{R}_{sp})$) w.r.t. topology configurations ($C_{var} = 10$)

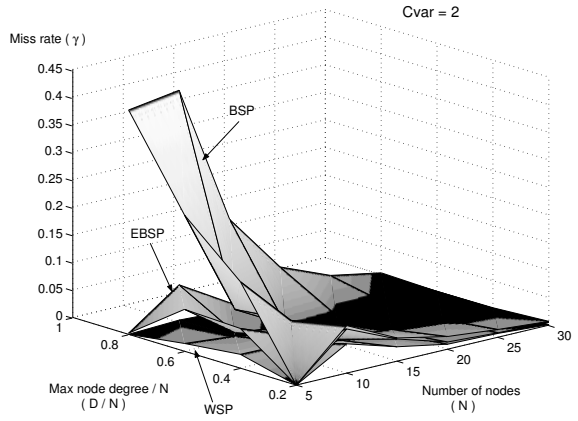


Figure 14: Miss rate (γ) w.r.t. different topology configurations ($C_{var} = 2$)

in terms of the miss rate and the BSP is the worst one (Figures 14 and 15), no matter what kind of network topologies are used. However, as far as the saturate bandwidth gain is concerned, WSP does not gain much generally (Figures 12 and 13). This is intuitively reasonable, because WSP is only a simple variant of the SP algorithm. For EBSP algorithm, it yields much larger values of bandwidth gain (in an order of magnitude better if compared with the results of WSP, especially when N and C_{var} are large) with small miss rates. The EBSP is always much better than the BSP in terms of the miss rate. For BSP, it yields fairly large bandwidth gains when N and C_{var} are large, but not as large as the EBSP does. Nevertheless, it always suffers from the highest miss rates among the three. We can observe in Figure 14 that, when $N = 10$, $D/N = 0.8$, the BSP has a miss rate as high as almost 0.45, meaning that in nearly half of the simulated topologies, it performs worse than the basic SP algorithm.

Comparison of the three algorithms in different topology configurations: If we compare the three algorithms when the capacity variance C_{var} is small, surprisingly, we find that WSP is the best in terms of both bandwidth gain and miss rate. However, the margin over EBSP is small. For large capacity

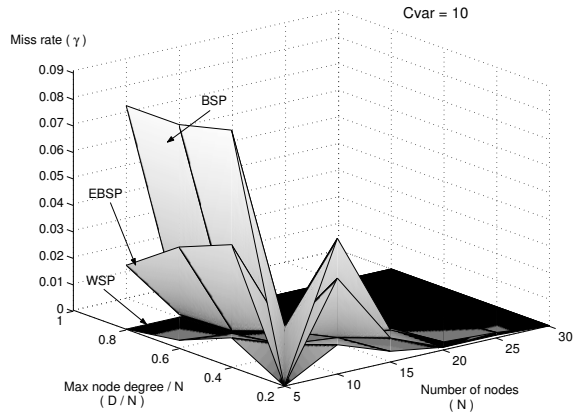


Figure 15: Miss rate (γ) w.r.t. different topology configurations ($C_{var} = 10$)

variances C_{var}, EBSP is definitely the best algorithm among the three - it not only achieves large saturate bandwidth values, but keeps fairly low miss rates as well, especially in case of large-scale, complex topologies. Therefore, we can draw a conclusion that both the WSP and EBSP are favorable for those networks with homogeneous links, meanwhile, the EBSP is more suitable for large-scale, complex networks with heterogeneous links. There is no significant advantage of BSP. Although it can yield comparable bandwidth gains as EBSP when N and D are small, it suffers from very high miss rates.

Performance trends w.r.t. C_{var}: Comparing Figures 12 and 13, as well as Figure 14 and 15, we find that when C_{var} increases from 2 to 10, the bandwidth gains of both BSP and EBSP increase significantly. In the meantime, their miss rates decrease with the increase of C_{var}. However, the performance of WSP does not change significantly with C_{var}.

Performance trends w.r.t. the maximum degree D: If we fix the value of N and C_{var}, by changing the values of D, we can observe the same trends for all three algorithms: when D increases, all three algorithms first yield larger saturate bandwidth values, but after a certain point, all of them start decreasing. Intuitively, the reason behind this phenomenon might be: at the beginning, when D is very small, the topology has very low connectivity, meaning that there are very few optional routes between nodes for the routing algorithms to choose. Therefore, all three algorithms yield almost the same results as SP does, hence small bandwidth gains. As D increases, there are more and more optional routes among nodes for the routing algorithms to choose. Thus, it is more and more likely for the algorithms to find better routes. As a result, the saturate bandwidth values increase. But after a certain turning point (different algorithms may have different turning points), the topology is getting so connected that the hop-count shortest paths become more preferable. Therefore, bandwidth gains start to decrease.

Brief conclusions: In summary, for a simple, homogeneous network, especially when the link capacity variance is small, EBSP or WSP should be used for the premium-class routing since they yield larger saturate bandwidth meanwhile keep miss rates low. However, for a complex, heterogeneous network, the EBSP algorithm is absolutely preferred since it achieves much higher saturate bandwidth gains as well as relatively low miss rates in such networks.

5.2.2 *Impacts of θ*

As we can see from the description of the EBSP algorithm, θ is an important constant. In the last subsection (Section 5.2.1), we fixed $\theta = 2.0$. In this part, we investigate its impacts on the performance in terms of the average bandwidth gain ($\bar{\beta}$).

Let $\bar{\beta}_{1.0}$, $\bar{\beta}_{2.0}$ and $\bar{\beta}_{\infty}$ denote the average bandwidth gains of EBSP when $\theta = 1.0, 2.0, \infty$, respectively. Based on the fact that $\bar{\beta}_{2.0} > \bar{\beta}_{1.0}$ and $\bar{\beta}_{2.0} > \bar{\beta}_{\infty}$,[9] for a given topology, intuitively, we project that there may exist an optimal θ, written θ^*, with which the EBSP algorithm achieves its maximal average bandwidth gain $\bar{\beta}^*$. To confirm this projection, we conduct another set of simulations and the results are shown in Figures 16 and 17.

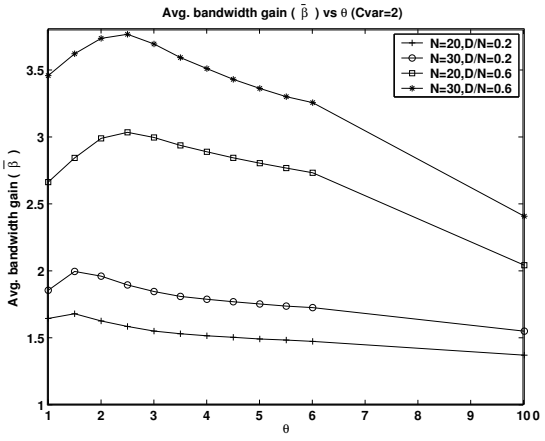


Figure 16: Average bandwidth gain w.r.t. θ ($C_{var} = 2$)

The figures show clearly that all curves of $\bar{\beta}$ versus θ have absolute maximum values, therefore, each topology configuration has an optimal θ^*. Notice that both figures are distorted a little bit. The reason is that the actual last point of each curve is for $\theta = 100$ (to mimic a very large θ). However, we put all last points in the figures at the position for $\theta = 10$ to make the details of curves clearer.

Furthermore, we conduct more simulations to obtain the optimal values of θ for different topology configurations. The results are demonstrated in Figure 18. Although the optimal values are scattered between 1.0 and 5.0, given topology configuration (N, D, C_{var}) ranges from $(5, 0.2, 2)$ to $(30, 0.8, 10)$, most of them are gathered between 1.5 and 3.0. This is exactly why we choose $\theta = 2.0$ in Section 5.2.1 when we evaluate the overall performance of EBSP. Figure 18 also reveals an overall trend between topology configuration and

[9] As we discussed before, $\theta = 1.0$ degrades EBSP to BSP and $\theta = \infty$ degrades EBSP to a variant of SP algorithm, both of which yield smaller $\bar{\beta}$ than when $\theta = 2.0$.

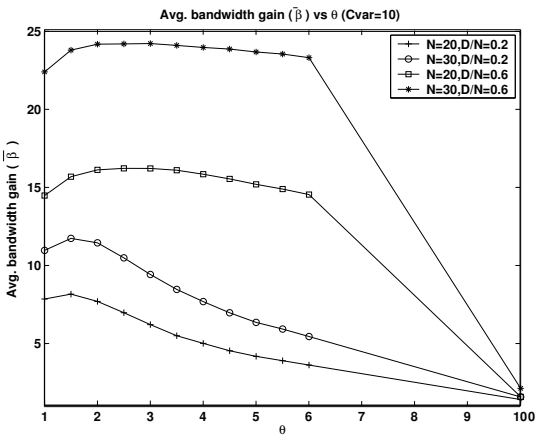


Figure 17: Average bandwidth gain w.r.t. θ ($C_{var} = 10$)

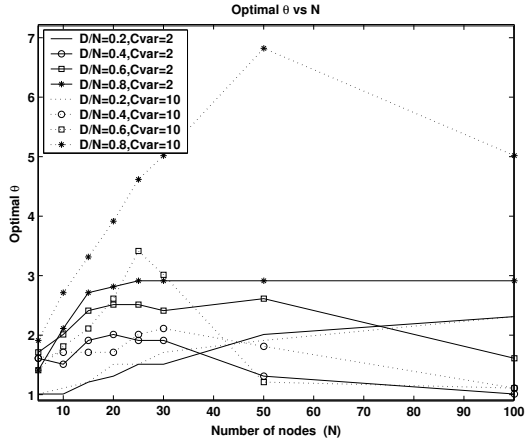


Figure 18: Optimal θ w.r.t. topology configurations

θ^*: topologies with larger number of nodes (N) or larger link capacity variance (C_{var}) tend to have larger optimal values of θ^*.

Brief conclusions: In summary, our simulation results observe an overall trend between topology configuration and θ^*: topologies with larger number of nodes (N) or larger link capacity variance (C_{var}) tend to have larger optimal values of θ^*. However, for medium-sized networks (e.g., $N = 20$ or 30), the values of θ^* are mostly gathered between 1.5 and 3.0, thus we can use $\theta = 2.0$ to evaluate the overall performance of EBSP in such networks. The performance of EBSP in large scale networks will be studied later in Section 5.2.3.

5.2.3 Special Cases with Larger Number of Nodes

In the previous parts, since we focused on average performance which was based on a large number of topologies (2000 for each topology configuration) to eliminate potential biases, we could not simulate very large scale networks because of high computation complexity (the maximum number of nodes we simulated was 100). Moreover, for each given topology configuration, the topologies were totally randomly generated. Some of them might not be realistic.

Topology Type		1 level router (IP) only
Topology Parameters	**Model Type**	Waxman: $\alpha_w = 0.15, \beta_w = 0.2$
	Network Size (HS)	1000 (number of squares)
	# of Nodes (N)	200, 300, 400, 500, respectively
	Node Degree (m)	20
	Node Placement	random
	Bandwidth Distr.	uniform
	Min. bandwidth	10
	Max. bandwidth	1024

Table 1: BRITE topology generator configuration

# of Nodes (N)	**Bandwidth Gain (β)**				
	WSP	**BSP**	**EBSP** ($\theta = 1.5$)	**EBSP** ($\theta = 2.0$)	**EBSP** ($\theta = 3.0$)
200	13.41	33.94	37.86	42.79	43.75
300	7.65	29.13	33.38	38.77	40.37
400	6.58	28.26	31.16	33.39	36.39
500	10.42	45.79	50.01	55.23	58.75

Table 2: Bandwidth gains (β) in large realistic networks

In this part, in order to further investigate the algorithms in larger and more realistic networks, we use a famous topology generator, BRITE[10], to generate more realistic topologies with larger number of nodes (up to 500 nodes), in which we compare the performance of the three algorithms.

We configure the BRITE as shown in Table 1 to generate our simulating topologies, where HS means "size of the main plane" [19] and it simulates the geometrical size of a network. Waxman model is used. Most configurations in the table (e.g., for the Waxman model, $\alpha_w = 0.15$ and $\beta_w = 0.2$) are default values. Totally, 4 topologies are generated and they consist of 200, 300, 400, and 500 nodes, respectively.

Table 2 shows the detailed simulation results. It is clearly demonstrated that all three algorithms, WSP, BSP, and EBSP, perform better than the SP algorithm in all 4 cases (i.e., the bandwidth gain, β, is always larger than 1). It is also shown that the EBSP algorithm outperforms WSP and BSP in all cases, no matter what value is used for θ (1.5, 2.0, or 3.0). Furthermore, for the EBSP algorithm itself, the results show that a larger scale network may favor a larger θ. That is, if the node degree is fixed, the more nodes a network consists of, the larger the optimal value θ^* will be. This conforms to the results in the previous part (Figure 18).

6. RELATED WORK

Extensive research has been conducted on QoS routing issues recently. In [3], S. Chen and K. Nahrstedt did a thorough survey on QoS routing algorithms, where routing problems were classified into four categories in terms of types of constraints and optimization targets. However, if we use a different criterion, routing mechanisms can also be categorized into flow-based routing and hop-by-hop routing. Most of existing QoS routing solutions focus on virtual circuit network models which are connection-based or flow-based. In [15, 16], the authors proposed and evaluated some new

[10]BRITE is developed by Alberto Medina et al at Boston University.[19]

Dijkstra-based QoS routing algorithms, such as the *shortest-distance path* algorithm which uses the reciprocal of link bandwidth as weight function, similar to the BSP algorithm in [26] and in this paper. In [12], the authors proposed the Minimum Interference Routing algorithm (MIRA) which routes a newly connection onto a path that does not interfere too much with those *critical links*. The algorithm shows good performance compared to other algorithms. In [23], the authors addressed the QoS routing from the precomputation perspective and proposed a hierarchical algorithm to solve the *All-Hops Problem*. In [1], the authors discussed path selection algorithms to support QoS routes in the context of extensions to the OSPF protocol. In [4, 28], the authors studied the *ticket-based routing algorithms* in an environment where state information was imprecise. All research work above was flow-based or connection-based and did not consider hop-by-hop routing constraints, where routing consistency and forwarding loop-freedom should be guaranteed.

In the hop-by-hop routing category, Van Mieghem, De Neve and F. Kuipers proposed the Tunable Accuracy Multiple Constraints Routing Algorithm (TAMCRA) to address QoS routing problems with multiple constraints in a hop-by-hop manner [20]. The authors proved that the TAMCRA is an exact QoS routing algorithm which can guarantee loop-freeness. The major differences between [20] and our paper include: (1) TAMCRA is not a consistent routing algorithm; and (2) TAMCRA solves the routing problem for one source-destination pair, i.e., given a source and destination, TAMCRA finds a sub-optimal route from the source to the destination; while our algorithms try to approach an optimal solution for the entire network, given multiple sources and destinations. Some basic issues on hop-by-hop routing algorithms, such as consistency, isotonicity, and search of optimal paths, were studied in [25]. The author provided an elegant algebra basis to study the correctness of Dijkstra-based QoS routing algorithms in the context of hop-by-hop routing in the Internet. We borrowed some definitions and results from his work.

Another related research area is the Type-of-Service (TOS) routing. Matta and Shankar proposed the TOS2 routing scheme in [18], where two TOS's were considered: *low delay* and *high throughput*. Two additive link cost functions were given accordingly. Dijkstra's algorithm was used. But the main focus of the paper was not routing algorithms, instead, the authors thoroughly analyzed the network stability and convergence features by using the Liapunov function method. In [24], the authors classified flows into *long-lived* and *short-lived* two types and proposed a hybrid routing scheme for them. Briefly, the paper suggested that dynamic routing should be used for the long-lived flows, while static preprovisioned paths should be used for the short-lived ones. However, their approach was basically in a flow-based manner, thus each long-lived flow could be routed individually and dynamically in the residual network to achieve high stability and load balance.

In [26], we raised the OPR problem for the first time. The idea of finding a solution to the OPR problem is somewhat similar to the idea of *Max-min routing*, which is also NP-hard and has been studied in [2, 8, 10, 17]. Although both problems are dealing with unsplittable "flows", a solution to the Max-min routing problem does not have to maintain routing consistency, i.e., each flow can be routed independently. Therefore, their results can not simply be applied to the OPR problem.

In this paper, we introduced two existing hop-by-hop QoS routing algorithms and proposed a new one (EBSP - Enhanced Bandwidth-inversion Shortest Path). All of them are related to the generalized Dijkstra's algorithm. We also provide rigorous proofs for the NP-hardness of the OPR problem and for the correctness of the novelly-proposed EBSP algorithm. The performance of EBSP, as well as the impact of EBSP's discount factor θ, was studied through simulations.

7. CONCLUSION

In order to service premium traffic efficiently while keeping its negative inter-class effects low, the premium class routing algorithm must be carefully chosen. In this paper, we argued that the choice of the optimal premium class routing algorithm leads to the Optimal Premium-class Routing (OPR) problem, which is NP-hard. We have briefly examined two existing routing algorithms (the Widest-Shortest-Path (WSP) algorithm and the Bandwidth-inversion Shortest-Path (BSP) algorithm) and designed the novel Enhanced Bandwidth-inversion Shortest-Path (EBSP) algorithm to compare and study the tradeoffs of these different solutions for the OPR problem. We also provided a rigorous proof for the correctness of the EBSP algorithm. Our simulation results showed that on average all three routing algorithms outperformed the basic hop-count shortest-path (SP) algorithm. Furthermore, our new EBSP algorithm significantly outperformed all other existing algorithms in large-scale, heterogeneous networks. As the DiffServ networks are becoming more and more heterogeneous and complex, our results strongly indicate that the EBSP routing algorithm is a strong candidate for routing of premium class traffic in DiffServ networks.

In this paper, we assume that each node requires the same premium bandwidth to other nodes. This is a very strong assumption. In our future work, we will address a more general OPR problem where the premium bandwidth demand can be heterogeneous. The existing EBSP algorithm's performance will be examined in the new environment. Possible enhancements will also be studied.

8. ACKNOWLEDGMENTS

The authors would like to acknowledge the help of Li Xiao and King-Shan Lui for their invaluable discussions and suggestions. The authors would also like to thank the anonymous reviewers for their detailed and insightful comments.

9. REFERENCES

[1] G. Apostolopoulos, D. Williams, S. Kamat, R. Guerin, A. Orda, and T. Przygienda. QoS Routing Mechanisms and OSPF Extensions. RFC 2676, Aug. 1999.

[2] D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall, 1990.

[3] S. Chen and K. Nahrstedt. An Overview of Quality-of-Service Routing for the Next Generation High-Speed Networks: Problems and Solutions. *IEEE Network, Special Issue on Transmission and Distribution of Digital Video*, Nov./Dec. 1998.

[4] S. Chen and K. Nahrstedt. Distributed QoS Routing with Imprecise State Information. In *Proceedings of 7th IEEE International Conference on Computer, Communications and Networks (ICCCN'98)*, pages 614–621, Lafayette, LA, October 1998.

[5] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 1990.

[6] S. et. al. An Architecture for Differentiated Services. *RFC 2475*, December 1998.

[7] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, San Francisco, CA, 1979.

[8] J. Jaffe. Bottleneck flow control. *IEEE Transactions on Communication*, 29:954–962, 1981.

[9] J. Kleinberg. Single-source Unsplittable Flow. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 68–77, October 1996.

[10] J. Kleinberg, Y. Rabani, and E. Tardos. Fairness in routing and load balancing. In *40th Annual Symposium on Foundations of Computer Science*, pages 568 – 578, 1999.

[11] K.Nichols, V.Jacobson, and L.Zhang. A Two-bit Differentiated Services Architecture for the Internet. *RFC 2638*, July 1999.

[12] M. Kodialam and T. Lakshman. Minimum Interference Routing with Applications to MPLS Traffic Engineering. In *Proceedings of IEEE Infocom 2000*, March 2000.

[13] S. Kolliopoulos and C. Stein. Improved Approximation Algorithms for Unsplittable Flow Problems. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, 1997.

[14] J. Lenstra, D. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. In *28th IEEE FOCS*, 1987.

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

[16] Q. Ma and P. Steenkiste. Supporting Dynamic Inter-Class Resource Sharing: A Multi-class QoS Routing Algorithm. In *IEEE Proceedings of INFOCOM '99*, pages 649–660, 1999.

[17] Q. Ma, P. Steenkiste, and H. Zhang. Routing high-bandwidth traffic in max-min fair share networks. In *Proceedings of ACM SIGCOMM'96*, pages 206–217, Palo Alto, CA, Oct 1996.

[18] I. Matta and A. U. Shankar. Type-of-Service Routing in Datagram Delivery Systems. *IEEE Journal of Selected Areas in Communications*, 13(8), October 1995.

[19] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: An Approach to Universal Topology Generation. In *Proceedings of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems-MASCOTS '01*, Cincinnati, Ohio, August 2001.

[20] P. V. Mieghem, H. D. Neve, and F. Kuipers. Hop-by-hop Quality of Service Routing. *Computer Networks*, 37(3-4):407–423, 2001.

[21] M.May, J.C.Bolot, C.Diot, and A.Jean-Marie. Simple Performance Models of Differentiated Services Schemes for the Internet. In *Proceedings of IEEE Infocom 1999*, March 1999.

[22] A. Orda. Routing with end-to-end QoS guarantees in broadband networks. *IEEE/ACM Transactions on Networking*, 7(3), June 1999.

[23] A. Orda and A. Sprintson. QoS Routing: the Precomputation Perspective. In *Proceedings of IEEE Infocom 2000*, March 2000.

[24] A. Shaikh, J. Rexford, and K. Shin. Load-Sensitive Routing of Long-Lived IP Flows. In *Proceedings of ACM SIGCOMM'99*, 1999.

[25] J. Sobrinho. Algebra and algorithms for QoS path computation and hop-by-hop routing in the Internet. In *IEEE INFOCOM 2001, Anchorage, Alaska*, pages 727 – 735, April 2001.

[26] J. Wang and K. Nahrstedt. Hop-by-Hop Routing Algorithms For Premium-class Traffic In DiffServ Networks. In *Proceedings of IEEE Infocom 2002*, June 2002.

[27] J. Wang, Y. Wang, and K. Nahrstedt. Quantitative Study of Differentiated Service Model Using UltraSAN. *Tech. Report UIUCDCS-R-2001-2237, Department of Computer Science, University of Illinois at Urbana-Champaign*, July 2001.

[28] L. Xiao, J. Wang, and K. Nahrstedt. The Enhanced Ticket-based Routing Algorithm. In *Proceedings of IEEE ICC 2002*, New York, NY USA, April 28 - May 2 2002.