Hop-by-Hop Routing Algorithms For Premium-class Traffic In DiffServ Networks

Jun Wang, Klara Nahrstedt
Department of Computer Science
University of Illinois at Urbana-Champaign
{junwang3, klara}@cs.uiuc.edu

***Abstract*— Bear the provision of Quality of Service (QoS) in the Internet, Differentiated Service (DiffServ) model has been proposed as a cost-effective solution. Traffic is classified into several service classes with different priorities. The *premium* class traffic has the highest one. The routing algorithm used by the premium class service has significant effects not only on its own traffic, 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.**

Based on the hop-by-hop routing mechanism, an interesting problem is how to find an optimal routing algorithm for premium class traffic such that (1) it works correctly and efficiently for premium traffic; and meanwhile (2) it reduces negative influences to other classes of traffic (such as bandwidth starvation, excessive delay jitter, etc.). We call this problem the Optimal Premium-class Routing (OPR) problem which is NP-Complete.

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 apply to the OPR problem a novel heuristic algorithm, called the *Enhanced Bandwidth-inversion Shortest-Path (EBSP) algorithm*. We prove theoretically the correctness of the EBSP algorithm, i.e., it is a *consistent* and loop-free hop-by-hop routing algorithm.

Our extensive simulations in different network environments show clearly that the EBSP algorithm performs better in complex, heterogeneous networks than the other two hop-by-hop routing algorithms for the premium class traffic.

I. Introduction

With exploding volume of traffic and expanding Quality of Service (QoS) requirements from emerging multimedia applications, extensive and intensive researches have been carried out to address varieties of issues in QoS provision and routing in the Internet. In this paper, we raise an interesting problem of how to find optimal routing schemes under both Differential Service (DiffServ) and hop-by-hop IP routing assumptions. By combining service differentiation and QoS routing (we call it differentiated routing), the high-priority traffic, without compromise of their QoS guarantees, 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.

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.

A. Hop-by-hop IP routing

In today's Internet, the hop-count shortest-path (SP) algorithm is used for IP routing. Although extensive research has been done in QoS routing, most of them were focused on connection-oriented scenarios. In this paper, we investigate a QoS routing problem in the context of hop-by-hop routing. Hop-by-hop routing means that routing decisions are made at each node independently and locally, based only on packets destination addresses and their route computation using corresponding topology knowledge. The following figure shows one simple example of hop-by-hop SP routing decisions made by individual nodes independently.

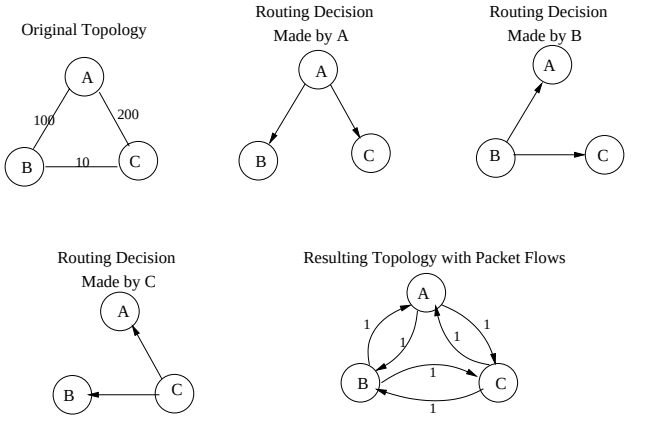


Fig. 1. An example of hop-by-hop SP routing: the numbers next to the links in the original topology denote link capacities, while the labels in the resulting topology denote "flow" numbers.

The resulting topology, shown in Figure 1, illustrates the real packet flows[1] between each pair of nodes. The numbers associated with the arcs are the numbers of flows going through that particular link in that direction.

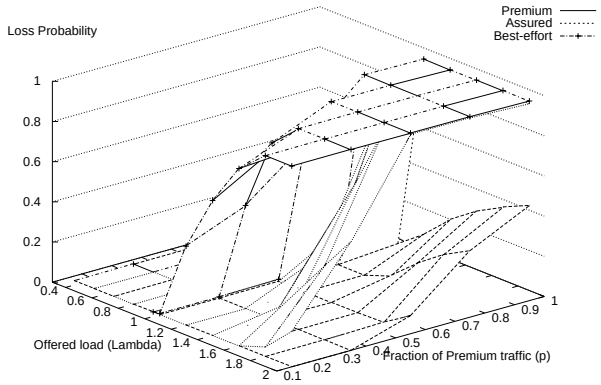
B. DiffServ model and inter-class effects

In order to provision better end-to-end QoS to applications, DiffServ scheme has been proposed as a cost-effective solution

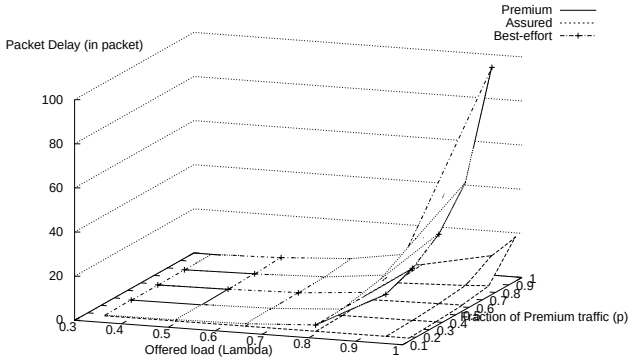
[1] The word "flow" here is different from the definitions in Integrated Service (IntServ) model or in RSVP. It refers to all packets moving between each pair of source and destination. For instance, Flow $\overrightarrow{AB}$ indicates all packets being transmitted from node A to node B.

[1], [2]. 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. Due to premium traffic's high priority, this could lead to some problems for the low-priority traffic when the volume of premium class traffic is high. That is to say, 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 [3]. In [4], the authors presented simple performance models and analysis for DiffServ schemes. However, to make a strong case for the negative impacts 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 QoS 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 QoS degradations (such as higher packet loss rates and larger queueing delays). Therefore, we must seriously take the inter-class effects into consideration when we choose routing mechanisms for the premium class traffic.



(a) Packet Loss Probability



(b) Packet Delay (in packet)

Fig. 2. Measures of the inter-class effects between all three classes in a DiffServ network

C. Routing and bandwidth reservation for premium-class traffic

In order to provide end-to-end QoS guarantees for the premium class traffic from node v_1 to v_n, assuming that the path $\langle v_1, v_2, \cdots, v_n \rangle$ is used from v_1 to v_n, certain amount of bandwidth must be successfully reserved on each link along the path between these two nodes. If a link (v_i, v_{i+1}) is shared by multiple paths between different pairs of nodes, it has to reserve certain amount of bandwidth not only for the premium traffic originated from v_i itself, but for all the transient premium traffic passing through all paths which are sharing the link as well.

Figure 3 depicts a scenario in which node A reserves a amount of bandwidth to C while B reserves b amount of bandwidth to C. Since the path $(A \rightarrow B \rightarrow C)$ is used for flow $\overrightarrow{AC}$ according to SP routing, therefore, the link (B, C) is shared by flow $\overrightarrow{AC}$ and $\overrightarrow{BC}$ so that it must reserve totally $(a+b)$ amount of bandwidth for both flows. As we can imagine, 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 (or more precisely, residual bandwidth) of each link along its path, but on the routing strategy as well. In this work, 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. This is reasonable when a network is in a setup phase: Every node tries to pre-reserve a virtual trunk with the same amount of bandwidth to all other nodes to accommodate its premium traffic. As we can see, given a network, for each routing scheme, the maximal value of B (written B_s) is fixed.[3] Thus we can measure routing schemes by comparing the values of B_s they can achieve. The larger B_s, the better the algorithm. We can understand this from two different perspectives: (1) The routing scheme with larger B_s is able to accommodate more premium traffic; (2) 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 available or residual bandwidth to other 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

[2]More simulation details and extensive results and measurements are presented in our technical report [3].

[3]We will define this maximal value of B, which each routing scheme can achieve, as the *Saturate Bandwidth (B_s)* later in this section.

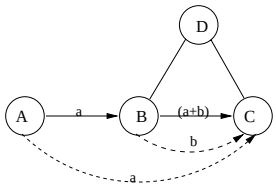


Fig. 3. An example of bandwidth reservations for premium traffic: Since link (B, C) is shared by flow $\vec{AC}$ and $\vec{BC}$, it must reserve $(a + b)$ amount of bandwidth for both flows.

bandwidth starvation for low-priority traffic may be reduced). Actually, if we look at the problem from the second perspective, B_s becomes a load-balancing or fairness index of a network, which is similar to the *max-min fairness* in [5], [6], [7].

Together with the assumption of hop-by-hop routing, an interesting problem is what is the optimal value of B_s (denoted as B_{max}) we can get. This problem is closely related to the routing algorithm the network is using. Given a specific routing algorithm R, there exists a maximum value of B which saturates some link in the network. The link is called the bottleneck link and the value is denoted as B_s^R. Therefore, $B_{max} = max\{B_s^{R_1}, B_s^{R_2}, \cdots, B_s^{R_n}\}$, where R_1, R_2, $\cdots$, R_n are all possible loop-free routing algorithms for the given network. To find the optimal routing algorithm and the value of B_{max} among all of these algorithms 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. It is not difficult to show that this problem is a NP-Complete problem by following the proof in [6]. Therefore, the SP routing algorithm may not always be the optimal one. By 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 the SP algorithm can achieve is $B_s^{R_{sp}} = 10$ 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.[4] Thus the problem that this paper will address turns out to be the finding a heuristic hop-by-hop routing algorithm R which can achieve a better B_s^R.

In order to solve the OPR problem in polynomial time, heuristic approximation algorithms are needed. In section IV, 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 based on the generalized Dijkstra's algorithm, therefore, it can run in $O(|V|^2)$ time. However, in terms of the saturate bandwidth, it yields much better results than the other two existing algorithms.

Extensive simulations in different network scenarios show that all three approximation algorithms outperform the hop-count shortest-path algorithm on average. The results also

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

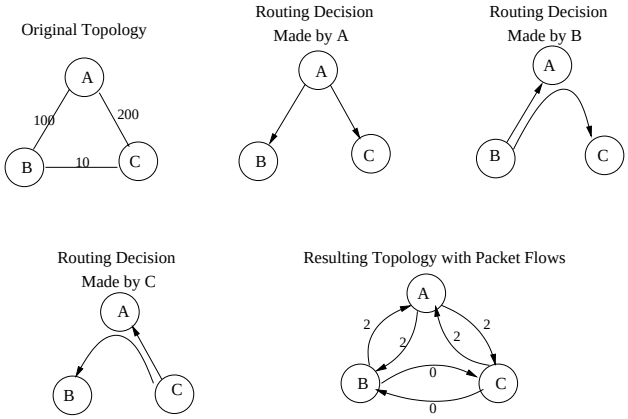


Fig. 4. An example of an optimal algorithm for the given topology, which can achieve better saturate bandwidth than the SP algorithm does.

confirm that, in a complicated and heterogeneous network environment, our EBSP algorithm outperforms the other two algorithms even more.

The rest of the paper is organized as follows. After the related work is covered in Section II, Section III presents the system model and our assumptions. Three heuristic routing algorithms are discussed and studied in Section IV. Simulations and results are illustrated in Section V. Finally, Section VI concludes this paper.

II. Related Work

Extensive researches have been conducted on QoS routing issues recently. In [8], S. Chen and K. Nahrstedt did a thorough survey on QoS routing algorithms. But they focused on network models in virtual circuit mode. Our work in this paper is based on the hop-by-hop routing scheme. Issues on hop-by-hop routing algorithms, such as isotonicity, search of optimal paths, were studied in [9]. The author provided an elegant algebra basis to study the QoS routing issues in the Internet. In our paper, although we will borrow some definitions and theorems from the work in [9], the problem we will address is totally different. In our work, we will address the Optimal Premium-class Routing (OPR) problem - to find an optimal routing algorithm to efficiently service premium class traffic and reduce negative inter-class effects simultaneously. In [6], J. Kleinberg addressed an NP-Complete problem which combined the selecting paths for routing and allocating bandwidth fairly among connections in the max-min sense. Following their proof of NP-Completeness, it is easy to show that our OPR problem is also NP-Complete. But their work was more connection-oriented with single source. Our work is based on the whole network topology and tries to address the OPR problem in a global view. In our work we show that an optimal local solution for a single source may not be the solution to the OPR problem. In [10], the authors addressed the QoS routing from the precomputation perspective and proposed a hierarchical algorithm to solve the All-Hops Problem. In [11], the authors discussed path selection algorithms to support QoS routes in the context of extensions to the OSPF protocol. Again, both of [10] and [11] focused on QoS routing algorithms for

connections or flows. They did not simultaneously take class-based and hop-by-hop routing into consideration. To our best knowledge, our paper is the very first work that raises the OPR problem and addresses the multi-class routing and inter-class effects in a global and hop-by-hop point of view.

III. System Model

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

A. 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 Internet) and E is the set of edges (links in the Internet), 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 any QoS metrics of the links, such as delay, bandwidth, hop count, etc. 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 $min(w_1, w_2)$.

For the premium class traffic, more specifically, a bandwidth reservation from v_1 to v_n through a simple path $p(v_1, v_n)$ is successful when and only when 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.

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). In this paper, we do not consider the inaccuracy incurred by delays, which means the topology information maintained at each router is accurate, consistent and up-to-date.
2) Hop-by-hop routing is used. Each node independently makes its own routing decisions based on the topology it maintains. A routing table is constructed at each node so that only the destination addresses of incoming packets are used to get the next-hop information from the table for those packets.
3) All nodes reserve the same amount of bandwidth to all other nodes for the premium traffic, denoted as B. For a given routing scheme R, there exists a maximal value of admissible B which saturates some link. This value is called the *Saturate Bandwidth* of R, and denoted as B_s^R. The link, where B_s^R is bounded, is called the bottleneck link.
4) Queueing delays at each node along a path account for the most significant part of the whole end-to-end delay for that path. With the highest priority, premium class traffic experiences almost no queueing delays [2], [3]. Therefore, choosing a fairly longer (in terms of hop-count) path from a source to a destination does not compromise its delay requirements.

B. Optimal Premium-class Routing (OPR) Problem

In Section I, we have introduced the problem we are going to address in this paper. 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 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 III-A, the OPR problem is to find the optimal routing scheme R_{opt} among all the possible loop-free hop-by-hop routing schemes on this network (denoted as $R_1, R_2, \cdots, R_n$) so that the optimal premium saturate bandwidth B_{max} is achieved, i.e., $B_s^{R_{opt}} = B_{max} = max\{B_s^{R_1}, B_s^{R_2}, \cdots, B_s^{R_n}\}$.

As stated in Section II, 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.

C. Some 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 IV, 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 R is said to be *consistent* if (1) R can find a simple path between every pair of nodes in the network, i.e., for any two distinct nodes $s, t \in V$, $R(s, t)$ is simple and non-empty, where $R(s, t)$ denotes the path found

by R between s and t; and (2) for any two distinct nodes $s, t \in V$, $R(s,t) = \langle s, v_1, v_2, \cdots, v_n, t\rangle$ implies that $R(v_i,t) = \langle v_i, v_{i+1}, \cdots, v_n, t\rangle$ (i.e., $R(v_i,t)$ is the sub-path of $R(s,t)$ 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 whole network. ■

Note that without consistency, loops may exist in a 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.

Formal definitions of *isotonicity* and *strict isotonicity* were given in [9], 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. Since both of them are too strong for finding a *consistent* routing scheme, here we give a weaker condition, called *left-isotonicity*, which is sufficient to verify a *consistent* routing scheme.

Definition 3: Left-isotonicity: Given a weight function, for any two paths p_1 and p_2 which are *prefixed* by a common, third path p (denoted as $p_1' = p \circ p_1$ and $p_2' = p \circ p_2$), if $w(p_1) \preceq w(p_2)$ implies $w(p_1') \preceq w(p_2')$, then the weight function is defined to be **left-isotonic**. Similarly, if $w(p_1) \prec w(p_2)$ implies $w(p_1') \prec w(p_2')$, then the weight function is defined to be **strictly left-isotonic**. If a network G uses a (strictly) left-isotonic weight function, 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.

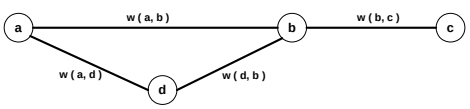


Fig. 5. Suppose $w(a,d) \oplus w(d,b) \prec w(a,b)$ and $w(a,d) \oplus w(d,b) \oplus w(b,c) \succ w(a,b) \oplus w(b,c)$. Then neither S-lightest path nor L-lightest path exists between a and c.

Being weaker than isotonicity, left-isotonicity can not guarantee the existence of S-lightest or even L-lightest paths between nodes. To see this, a simple example is shown in Figure 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 between a and c, but its sub-path $\langle a, b\rangle$ is not a lightest path. Therefore, neither S-lightest path nor L-lightest path exists between a and c in this scenario.

However, because we assume that weights are non-negative, 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 R, 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 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 R at s, a lightest path $p = \langle s, v_1, \cdots, v_n, t\rangle$ has been found. Assume it 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. Therefore, $w(p') \prec w(p_{i,t})$, 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 does not hold, the routing scheme based on generalized Dijkstra's algorithm can NOT guarantee to find the lightest path between every pair of source and destination. For example, in the case of Figure 5, by running a generalized Dijkstra's algorithm at node a, the path $\langle a, d, b, c\rangle$ is found , which is obviously NOT the lightest path from a to c. Therefore, generalized Dijkstra's algorithms are not sufficient to provide consistent routing any more. New algorithms are needed to find lightest paths between nodes in strictly left-isotonic networks.

On the other hand, 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 strict left-isotonic network $G(V,E)$, if a path p is the lightest path between two nodes $s, t \in V$, and suppose $p = \langle s, v_1, v_2, \cdots, v_n, t\rangle$, then for all sub-paths $p_{i,t} = \langle v_i, \cdots, v_n, t\rangle$ (where $1 \leq i \leq n$), $p_{i,t}$ is the lightest path from v_i to t.

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

Following Lemma 1, we have that the lightest path from s to t must be based on the lightest paths from some intermediate

nodes to t. Therefore, to find the lightest path between s and 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 in Algorithm WN-DIJKSTRA to find the lightest path between any two given nodes s and t in a strictly left-isotonic network.

WN-DIJKSTRA$(G(V,E), w, s, t)$
1 **for each** node $v \in V$
2 **do** $d[v] \leftarrow \infty$
3 $\varpi[v] \leftarrow$ NIL
4 $d[t] \leftarrow 0$
5 $Q \leftarrow V$
6
7 **while** $Q \neq \emptyset$
8 **do** $u \leftarrow$ EXTRACT-MIN(Q)
9 **if** $u = s$
10 **then** EXIT
11 **for each** $v \in Adj[u]$
12 **do if** $w(v,u) \oplus d[u] \prec d[v]$
13 **then** $\varpi[v] \leftarrow u$
14 $d[v] \leftarrow w(v,u) \oplus d[u]$

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

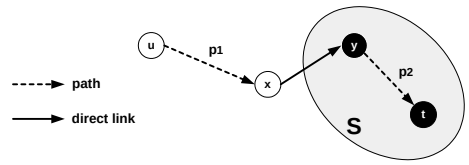


Fig. 6. The proof of Theorem 3: path p can be decomposed into two sub-paths p_1 and p_2.

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 6.

Now we show that $d[x] = \delta(x,t)$ when it is extracted from Q. Because u is the first node for which $d[u] \neq \delta(u,t)$ when it is extracted from Q, $d[y] = \delta(y,t)$ 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, we have that the sub-path from x to t is 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. Then, 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 the next section (Section IV), we will provide three heuristic algorithms to address the OPR problem. Their correctness is proven based on the definitions and theorems introduced above.

IV. ROUTING ALGORITHMS FOR PREMIUM CLASS TRAFFIC

The OPR problem we defined in Section III turns out to be very difficult to find an optimal solution in polynomial time. 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 III). All of them can run in $O(|V|^2)$ [12].

A. Widest-Shortest-Path Algorithm (WSP)

The WSP algorithm is the simplest heuristic algorithm which can achieve a better saturate bandwidth ($B_s^{R_{wsp}}$) 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 [9], [13]. Although it does not have a *strict* isotonicity, the isotonicity still holds. Therefore, by using the Dijkstra-old-touch-first (Dijkstra-OTF) algorithm proposed in [9], we can

guarantee that WSP finds a loop-free lexicographic-lightest (L-lightest) path from a source to a destination.

The simulation results (given in Section V) 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.

B. 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 [13]. 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 [9]. 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 ($B_s^{R_{bsp}}$) than SP or WSP (the simulation results will be shown in Section V), it is not very *stable*, meaning that the chance of producing a worse $B_s^{R_{bsp}}$ 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 $B_s^{R_{bsp}} = 50$ and $B_s^{R_{sp}} = 90$, that is, the BSP gets even worse saturate bandwidth than the SP does.

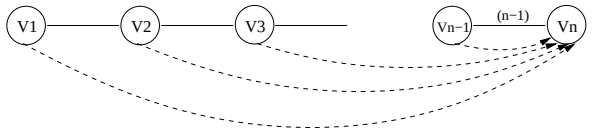


Fig. 7. Links in a longer path have to handle more flows, therefore, B_s decreases: for example, according to the consistency of hop-by-hop routing, link (v_{n-1}, v_n) has to hold $(n-1)$ flows: $v_1\vec{v}_n, v_2\vec{v}_n, \cdots, v_{n-1}\vec{v}_n$. The longer the path is, the smaller a B_s will be.

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 the path is taking, 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.[5] This phenomenon is illustrated in Figure 7. 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.

C. 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 the strict left-isotonicity property as well (Section III). The weight function for a path $p(v_1, v_n) = \langle v_1, v_2, \cdots, v_n\rangle$ is defined as follows.

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

For any two paths p_1 and p_2, $w(p_1) \prec w(p_2)$ if and only if $w(p_1) < w(p_2)$. A modified version of WN-Dijkstra's algorithm is used to find the lightest path between two nodes in terms of this new weight function.

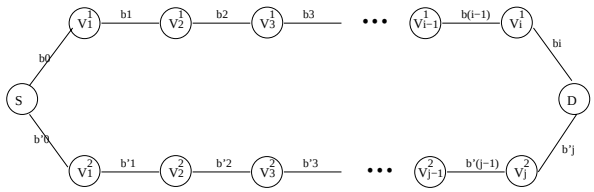


Fig. 8. The topology used to prove the left-isotonicity of the weight function in Equation 2. Originally, there are two paths between node S and D: $p_1 = \langle S, v_1^1, v_2^1, \cdots, v_i^1, D\rangle$ and $p_2 = \langle S, v_1^2, v_2^2, \cdots, v_j^2, D\rangle$. we suppose that p_1 is lighter than p_2.

Notice that the new weight function is no longer static. It dynamically changes with the hop count value. We can prove that this weight function is strictly left-isotonic, though it is not isotonic. 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 8. 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

$$w(p_1) = \sum_{m=0}^{i} \frac{2^m}{b_m}$$

and

$$w(p_2) = \sum_{m=0}^{j} \frac{2^m}{b'_m}$$

[5] 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 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.

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

$$\sum_{m=0}^{i} \frac{2^m}{b_m} < \sum_{m=0}^{j} \frac{2^m}{b'_m} \tag{3}$$

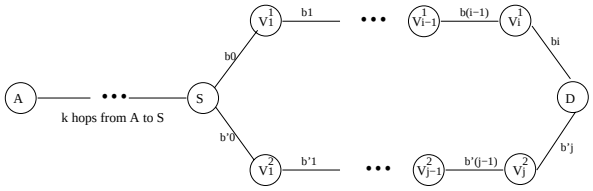


Fig. 9. Topology used to prove the left-isotonicity of the weight function in Equation 2. After a common prefix is added to the original topology (Figure 8) at node S, we can prove that $p'_1 = \langle A, \cdots, S, v_1^1, v_2^1, \cdots, v_i^1, D\rangle$ is still lighter than $p'_2 = \langle A, \cdots, S, v_1^2, v_2^2, \cdots, v_j^2, D\rangle$, that is, the strict left-isotonicity holds.

Now a common prefix $q = \langle A, \cdots, S\rangle$ is added to node S as shown in Figure 9. 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

$$w(p'_1) = w(A,S) + \sum_{m=0}^{i} \frac{2^{m+k}}{b_m} = w(A,S) + 2^k \sum_{m=0}^{i} \frac{2^m}{b_m}$$

and

$$w(p'_2) = w(A,S) + \sum_{m=0}^{j} \frac{2^{m+k}}{b'_m} = w(A,S) + 2^k \sum_{m=0}^{j} \frac{2^m}{b'_m}$$

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.

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 $v \in V$
2 **do** $d[v] \leftarrow \infty$
3 $\varpi[v] \leftarrow$ NIL
4 $Q \leftarrow V$
5 $d[t] \leftarrow 0$
6
7 **while** $Q \neq \emptyset$
8 **do** $u \leftarrow$ EXTRACT-MIN(Q)
9 **if** $u = s$
10 **then** EXIT
11 **for each** $v \in Adj[u]$
12 **do if** $2 \times d[u] + 1/b(v,u) < d[v]$
13 **then** $\varpi[v] \leftarrow u$
14 $d[v] \leftarrow 2 \times d[u] + 1/b(v,u)$

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 the SP with respect to the saturate bandwidth (i.e., $B_s^{R_{ebsp}}$ should be better than $B_s^{R_{sp}}$), meanwhile, it should be more *stable* than BSP. We will show in Section V that the simulation results confirm our expectations very well.

V. SIMULATIONS AND RESULTS

In the previous section, we introduced three heuristic algorithms to solve the OPR problem. Their performance depends on the network topology. To better understand the relation between them, and to show the advantage of our new EBSP algorithm, extensive simulations have been carried out. In this section, we present the simulations and results.

A. 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}. Given a specific value of D, we generate a random number within $[1, D]$ for each node as its degree. The destinations for each node are also randomly generated. 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 very easily.

Since it is very difficult to find the optimal saturate bandwidth B_{max} for each topology, therefore, comparisons among $B_s^{R_{wsp}}$, $B_s^{R_{bsp}}$, $B_s^{R_{ebsp}}$ and B_{max} become infeasible. In order to quantify the performance of the three algorithms and do comparisons, we use the SP algorithm (i.e., basic Dijkstra's algorithm on hop count) as our base. Given a randomly generated topology,[6] we first run the SP algorithm at each node of it so that the saturate bandwidth $B_s^{R_{sp}}$ is obtained. Then, the three heuristic algorithms, the WSP, BSP and EBSP, are executed on the same topology individually, and the saturate bandwidths, $B_s^{R_{wsp}}$, $B_s^{R_{bsp}}$, and $B_s^{R_{ebsp}}$ are obtained, respectively. We compare these three values with $B_s^{R_{sp}}$ to observe how much benefit we can gain by using different algorithms. For each configuration (N, D, C_{var}), we run the simulation 10000 times (i.e., we randomly generate 10000 different topologies with the same configurations in terms of N,

[6] Since the topology is randomly-generated, it may be un-routable at all. Before we run routing algorithms on it, its routability is checked. If it is not routable, we just simply drop it.

D and C_{var}, and run simulations on them). Finally, different configurations are tested and simulation results are collected.

We evaluate the performance of a particular algorithm R by using two metrics: (1) "*Speedup*" - defined as $B_s^R/B_s^{R_{sp}}$ - the larger, the better; and (2) "*Stability*" - measured as the total number of times when the particular algorithm yields a worse B_s^R than the SP does (i.e., when $B_s^R < B_s^{R_{sp}}$) - the smaller this number, the better stability. By analyzing the simulation results in the next subsection, we can observe some interesting relations between performance of the algorithms and configurations of topologies.

B. Simulation Results and Analysis

We list the simulation results in the following table (Table I), where "# of missing" means: how many times the particular algorithm produced worse saturate bandwidth values (B_s^R) than the SP algorithm did. The smaller this number is, the more stable the algorithm is.

Each "speedup" number in Table I is the *average* number on 10000 samples, while each "# of missing" number is the *total* number out of 10000 samples.

By observing the detailed results in Table I, we can analyze and find some interesting relations and trends between performance of algorithms and configurations of topologies.

General comparisons among WSP, BSP and EBSP: In general, the WSP is the most stable one among the three algorithms in terms of "number of missing" and the BSP is the least stable one. However, as far as the saturate bandwidth "speedup" is concerned, WSP does not gain much generally. For EBSP algorithm, it becomes more stable and yields much larger speedup values (in an order of magnitude) when N increases and/or D increases. The EBSP is also always much more stable than the BSP. For BSP, it has the similar trends as the EBSP in the sense of saturate bandwidth, but it always suffers the worst stability.

Performance trends of the algorithms w.r.t. C_{var}: For both BSP and EBSP algorithms, there exists a significant and consistent relation between their performance and the variance of link capacities C_{var}. When the variance of link capacities increases, whatever values N and D have, both algorithms yield better speedup and become more stable. Especially, when topology is complicated (large N and D), if C_{var} increases from 2 to 10, both algorithms get large speedup values. However, the performance of WSP is not that straightforward. Given a fixed N, for small D's, WSP gets larger speedup when C_{var} increases. (For example, when $N = 20, D = 4, 8$, its speedup increases with C_{var}'s increase.) But for large D's, we observe the reverse trends. For example, when $N = 20, D = 12, 16$, its speedup decreases with C_{var}'s increase!

Comparisons of the three algorithms w.r.t. topology configurations: If we compare the three algorithms when the capacity variance is small, surprisingly, we find that WSP performance is best in terms of both speedup and stability. But for large capacity variances, EBSP is the best algorithm among the three ones - it has large speedup values while it keeps fairly stable, especially in case of complicated topologies. Therefore, we can draw a conclusion that WSP is more suitable for small capacity variance networks (i.e., networks with similar links) and EBSP is more suitable for networks with large capacity variances (especially those large networks with a lot of heterogeneous links). There is no significant advantage of BSP. Although it yields slightly larger speedup values than EBSP when N and D is small, it suffers poor stability.

Performance trends w.r.t. average 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 speedup values, but after a certain point, all of them become decreasing. 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 - resulting in very small speedup values. As D increases, there are more and more optional routes between nodes for the routing algorithms to choose. Thus, it is more and more likely for all of them to find better routes. So the speedup values increase. But after a certain turning point (different algorithms may have different turning points), the topology becomes so connected (consider a fully-connected topology) that the hop-count shortest paths are preferred. Therefore, speedup values for all three algorithms start to decrease.

Brief conclusion: In summary, for a simple, homogeneous network, WSP should be used for the premium-class routing since it is more stable. While for a complicated, heterogeneous network, our EBSP is absolutely preferred since it yields much higher saturate bandwidth speedup as well as very strong stability property.

VI. 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. We have analyzed 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. Our simulation results showed that on average all three routing algorithms outperformed the basic hop-count shortest-path (SP) algorithm. Furthermore, for simple, homogeneous networks, the WSP algorithm outperformed all other algorithms. However, the EBSP algorithm significantly outperformed all other algorithms in complex, 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.

Topology Config.			WSP		BSP		EBSP	
			$B_s^{R_{wsp}}/B_s^{R_{sp}}$ (On Average)	# of Missing (Total)	$B_s^{R_{bsp}}/B_s^{R_{sp}}$ (On Average)	# of Missing (Total)	$B_s^{R_{ebsp}}/B_s^{R_{sp}}$ (On Average)	# of Missing (Total)
$N=3$	$D=1$	$C_{var}=2$	1.0	0	1.01089	372	1.005136	0
		$C_{var}=10$	1.0	0	1.5258	543	1.521771	143
	$D=2$	$C_{var}=2$	1.0	0	1.023388	808	1.009331	0
		$C_{var}=10$	1.0	0	2.06108	1121	2.052849	292
$N=12$	$D=2$	$C_{var}=2$	1.206707	502	1.252498	949	1.170699	837
		$C_{var}=10$	1.430197	113	3.221248	369	2.809879	323
	$D=4$	$C_{var}=2$	1.700889	124	1.584275	461	1.604896	275
		$C_{var}=10$	2.372344	23	6.612975	120	6.596185	80
	$D=6$	$C_{var}=2$	2.249493	45	1.904825	219	2.035591	121
		$C_{var}=10$	2.434147	7	8.552396	25	8.994029	17
	$D=8$	$C_{var}=2$	2.412555	9	1.827042	432	2.152137	61
		$C_{var}=10$	1.758308	0	8.876251	0	9.794656	0
	$D=10$	$C_{var}=2$	1.55669	17	1.109787	4546	1.527184	640
		$C_{var}=10$	1.12632	0	7.611648	0	8.771335	0
$N=20$	$D=4$	$C_{var}=2$	1.719031	66	1.63303	210	1.628082	137
		$C_{var}=10$	2.467735	7	7.733295	44	7.54983	27
	$D=8$	$C_{var}=2$	2.676883	7	2.272738	58	2.440101	26
		$C_{var}=10$	3.868552	1	12.54002	4	13.67351	1
	$D=12$	$C_{var}=2$	3.384082	0	2.632563	5	2.948973	2
		$C_{var}=10$	2.779093	0	14.22995	0	15.85364	0
	$D=16$	$C_{var}=2$	2.994321	0	1.774265	467	2.43386	18
		$C_{var}=10$	1.501094	0	10.73456	0	12.40077	0
$N=30$	$D=5$	$C_{var}=2$	1.960256	19	1.789388	120	1.832777	49
		$C_{var}=10$	2.917606	2	10.37922	18	10.46364	9
	$D=10$	$C_{var}=2$	3.034881	2	2.508176	23	2.764518	3
		$C_{var}=10$	5.63361	0	16.60106	1	18.72418	1
	$D=20$	$C_{var}=2$	4.455547	0	3.226183	0	3.69619	0
		$C_{var}=10$	3.363771	0	20.11695	0	22.00206	0
	$D=25$	$C_{var}=2$	3.762717	0	1.953244	148	2.777876	4
		$C_{var}=10$	1.672655	0	12.10766	0	13.97969	0

TABLE I

SIMULATION RESULTS WITH RESPECT TO DIFFERENT TOPOLOGY CONFIGURATIONS

ACKNOWLEDGMENTS

We would like to acknowledge the help of King-Shan Lui. We also thank the anonymous INFOCOM'02 reviewers for their helpful comments.

REFERENCES

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

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

[3] 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.

[4] 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.

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

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

[7] J. Jaffe, "Bottleneck flow control," *IEEE Transactions on Communication*, vol. 29, pp. 954–962, 1981.

[8] 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.

[9] J.L. Sobrinho, "Algebra and algorithms for QoS path computation and hop-by-hop routing in the Inernet," in *IEEE INFOCOM 2001, Anchorage, Alaska*, April 2001, pp. 727 – 735.

[10] Ariel Orda and Alexander Sprintson, "QoS Routing: the Precomputation Perspective," in *Proceedings of IEEE Infocom 2000*, March 2000.

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

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

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