

Routing with Topology Aggregation in Delay-Bandwidth Sensitive Networks

King-Shan Lui, Klara Nahrstedt, Shigang Chen

Abstract—Routing is a process of finding a network path from a source node to a destination node. The execution time and the memory requirement of a routing algorithm increase with the size of the network. In order to deal with the scalability problem, large networks are often structured hierarchically by grouping nodes into different domains. The internal topology of each domain is then aggregated into a simple topology that reflects the cost of routing across that domain. This process is called topology aggregation. For delay-bandwidth sensitive networks, traditional approaches represent the property of each link in the aggregated topology as a delay-bandwidth pair, which corresponds to a point on the delay-bandwidth plane. Since each link after aggregation may be the abstraction of many physical paths, a single delay-bandwidth pair results in significant information loss. The major contribution of this paper is a novel Quality-of-Service parameter representation with a new aggregation algorithm and a QoS-aware routing protocol. Our QoS representation captures the state information about the network with much greater accuracy than the existing algorithms. Our simulation results show that the new approach achieves very good performance in terms of delay deviation, success ratio, and crankback ratio.

1 Introduction

The goal of quality-of-service (QoS) routing is to find a network path from a source node to a destination node, which has sufficient resources to support the QoS requirements of a connection request. If such a path exists, the request is called a *feasible request* and the path is called a *feasible path*.

The execution time and the space requirement of a routing algorithm increase with the size of the network, which leads to the scalability problem. For very large networks, it is impractical to broadcast the whole topology to every node for the purpose of routing. In order to achieve scalable routing, large networks are structured hierarchically by grouping nodes into different domains [1, 2]. The internal topology of each domain is then aggregated to show only the cost of routing across the domain, that is, the cost of going from one *border node* (a node that connects to another domain) to another border node. This process is called *topology aggregation*. One typical way of storing

the aggregated topology is for every node to keep detailed information about the domain that it belongs to, and aggregated information about the other domains.

Since the network after aggregation is represented by a simpler topology, most aggregation algorithms suffer from distortion, that is, the cost of going through the aggregated network deviates from the original value. Consequently, routing has to be done based on inaccurate information, and it becomes NP-hard to find the paths that are most likely to satisfy delay requirements [3]. Nevertheless, reference [4] showed that topology aggregation reduces the routing overhead by orders of magnitude and does not always have a negative impact on routing performance.

Some aggregation approaches have been proposed. Reference [5] presented algorithms that find a minimum distortion-free representation for an *undirected* network with either a single additive or a single bottleneck parameter. Examples of additive metrics are delay and cost, while an example of bottleneck parameter is bandwidth. For an additive constraint, it may require $O(|B|^2)$ links to represent a domain in the distortion-free aggregation, where $|B|$ is the number of border nodes in the domain. Reference [6] proposed an algorithm that aggregates *directed* networks with a single additive parameter by using $O(|B|)$ links. The algorithm achieves bounded distortion with a worst-case distortion factor of $O(\sqrt{\rho} \log |B|)$, where ρ is the *network asymmetry constant*, defined as the maximum ratio between the QoS parameters of a pair of opposite directed links.

In this paper, we study networks with two QoS parameters, *delay* and *bandwidth*. Some related work can be found in [7], [8], and [9]. Reference [7] presented an aggregation method that aggregates an undirected delay-bandwidth sensitive domain into a spanning tree among border nodes. Therefore, there is a unique path between each pair of border nodes after aggregation and the space complexity is $O(|B|)$. The paper showed that a spanning tree can provide a distortion-free aggregation for bandwidth, but not for delay. Reference [8] studied the problem of topology aggregation in networks of six different QoS parameters. The aggregated topology follows the ATM PNNI standard [1]. The authors proposed to minimize the distortion by using a linear programming approach. Both [7] and [8] assumed certain precedence order among the parameters, so that among several paths that go between the same pair of border nodes, one path can be selected as the “best” path.

King-Shan Lui is with the Department of Electrical and Electronic Engineering, University of Hong Kong, Pokfulam Road, Hong Kong. Klara Nahrstedt is with the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA. Shigang Chen is with the Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611, USA. This work was supported by the Airforce grant under contract number F30602-97-2-0121 and the National Science Foundation Career grant under contract number NSF CCR 96-23867. The findings in this paper do not necessarily reflect the views of the awarding agency.

The state of a path in a delay-bandwidth sensitive network can be represented as a delay-bandwidth pair [9]. If there are several paths going across a domain, a single pair of values, which is a point on the delay-bandwidth plane, is not sufficient to capture the QoS parameters of all those paths [10]. Reference [9] was the first to use a curve on the delay-bandwidth plane to approximate the properties of multiple physical paths between two border nodes,¹ without assuming any precedence among the parameters. A curve is defined by three values: the minimum delay, the maximum bandwidth, and the smallest stretch factor among all paths between two border nodes. The stretch factor of a path measures how much the delay and the bandwidth of the path deviate from the best delay and the best bandwidth of all paths. The curve provides better approximation than a single point, but this approach has several shortcomings. First, the paper did not provide a routing algorithm with polynomial complexity to *find* a feasible path based on the aggregated topology. Instead, it provided an algorithm to *check* if a given path² is likely to be feasible. Essentially, the algorithm determined whether the point, defined by the delay/bandwidth requirement, is within the curve defined by the delay, bandwidth, and stretch factor of the path. Second, although the paper provided an aggressive heuristic to find the stretch factor of an inter-domain path, there are cases where only one QoS metric will contribute to the value, and the information about the other metric is lost.

In this paper, we propose a new way of representing the aggregated state in delay-bandwidth sensitive networks by using line segments. Our approach will solve some problems in [9] and other traditional approaches by introducing a new QoS parameter representation, a novel aggregation algorithm and the corresponding routing protocol. We compared our algorithm with other algorithms, and the simulation results show that our algorithm outperforms others due to smaller distortion.

The rest of the paper is organized as follows: Section 2 introduces the network and aggregation models. Section 3 describes our QoS parameter representation. Section 4 presents the aggregation algorithm, and the routing algorithm follows in Section 5. Our simulation results are discussed in Section 6. Finally, we conclude in Section 7.

2 Network and Aggregation Models

A large network consists of a set of domains and links that connect the domains. It is modeled as a directed graph, where link state can be asymmetric in two opposite directions. Figure 1(a) is an example of a network with four domains. There are two kinds of nodes in a domain. A node is called a *border node* if it connects to a node of

another domain. A node is an *internal node* if it is not a border node.

A domain is modeled as a tuple (V, B, E) , where V is the set of nodes in the domain, $B \subseteq V$ is the set of border nodes, and E is the set of directed links among the nodes in V . The entire network is modeled as (G, L) , where $G = \{g_i | g_i = (V_i, B_i, E_i), 1 \leq i \leq \eta\}$ is the set of domains, L is a set of links that connect border nodes of different domains, and η is the number of domains in G .

The links in L and $E_i, 1 \leq i \leq \eta$, are called *physical links*. The QoS parameter of a physical link is denoted as a pair (D, W) , where D is the delay of the link and W is the bandwidth of the link. Each pair (D, W) represents a single point on the delay-bandwidth plane. A *physical path* from node v_0 to node v_k , which is denoted as $v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_{k-1} \rightarrow v_k$, consists of a set of directed links $(v_i, v_{i+1}) \in E$, for $0 \leq i < k$. Let $(D_{i \rightarrow i+1}, W_{i \rightarrow i+1})$ be the QoS parameter of link (v_i, v_{i+1}) , where $D_{i \rightarrow i+1}$ is the delay and $W_{i \rightarrow i+1}$ is the bandwidth of the link. The delay of the path from v_0 to v_k is $\sum_{i=0}^{k-1} (D_{i \rightarrow i+1})$. The bandwidth is $\min_{i=0}^{k-1} \{W_{i \rightarrow i+1}\}$. For example, if $k = 3$ and the parameters of (v_0, v_1) , (v_1, v_2) , and (v_2, v_3) are $(3, 5)$, $(5, 4)$, and $(6, 4)$, respectively, then the delay of $v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow v_3$ is $3+5+6 = 14$ and the bandwidth is $\min\{5, 4, 4\} = 4$. Hence, the parameter of a physical path is also a point on the delay-bandwidth plane.

There are several aggregation models for large networks. In this paper, we shall use the topology aggregation model proposed by the Private Network-Network Interface (PNNI) [1, 11]. One of the representative topologies in PNNI is the *star* topology. Other popular ones are *simple-node* and *mesh*. In a simple-node topology, a domain is collapsed into one virtual node. This offers the greatest reduction of information as the space complexity after aggregation is $O(1)$, but the distortion is large. The mesh topology is a complete graph among the border nodes. The complexity of this topology is $O(|B|^2)$ and its distortion is much smaller. The star topology is a compromise between the above two. It has a space complexity of $O(|B|)$ and the distortion is between those of a simple node and a mesh. [12] compares the performance of the above three aggregation methods. It shows that the star topology outperforms the simple-node and performs slightly worse than the mesh in a uniform network.

Let us consider the domain F in Figure 2(a), where nodes a, b, c , and d are the border nodes. The mesh aggregation is shown in Figure 2(b), and the star aggregation is shown in Figure 2(c). In a star topology, the border nodes connect via links to a virtual *nucleus*. These links are called *spokes*. Each link is associated with some QoS parameters. To make the representation more flexible, PNNI also allows a limited number of links connected directly between border nodes. These links are called *bypasses*. Figure 2(d) is an example of a star with bypasses. We call the links in an aggregated topology as *logical links* since they are not real.

After aggregation, a node in a domain sees all other nodes in the same domain, but only aggregated topologies of the other domains. For example, for the network in

¹The approach proposed in [9] was designed for multiple additive/bottleneck parameters, while the discussions in this paper focus on only one additive parameter (delay) and one bottleneck parameter (bandwidth).

²The path may traverse several transit domains.

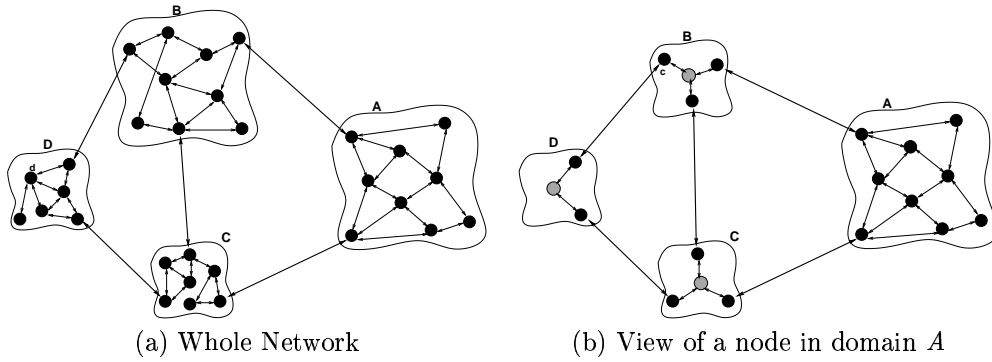


Figure 1: Network Model

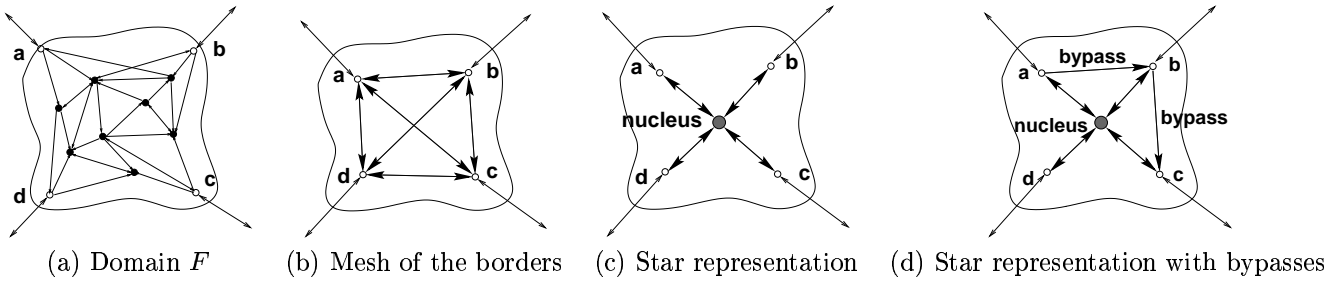


Figure 2: Topology Aggregation

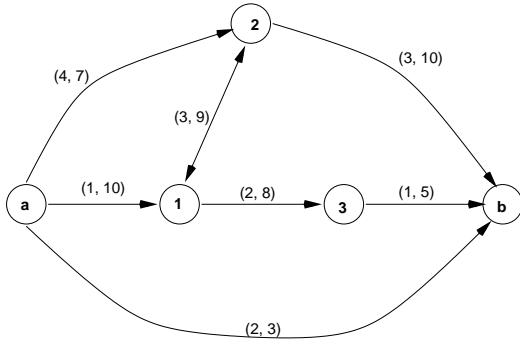


Figure 3: Multiple paths

Figure 1(a), the aggregated view of the network stored at a node in Domain A is shown in Figure 1(b). In such a view, the topology of Domain A is exactly the same as the original one but the topologies of the other domains are now represented by border nodes, nuclei, and spokes (without bypasses in this example). For a large network, this aggregated view is significantly smaller than the original topology and thus scalability is achieved. However, for the purpose of QoS routing, it is extremely important to develop solutions on how to represent the state information in this aggregated topology and how to control the information loss due to aggregation.

3 Line Segment Representation

In this section, we propose a line-segment representation for the aggregated state information. Given the original topology and a (D, W) pair for each link, we shall first transform every domain to a mesh among the border nodes as an intermediate computation step, and then transform the mesh to a star with bypasses. The state information of a logical link in either the mesh or the final star is represented by line segments, which will be discussed in depth shortly.

A mesh among the border nodes is a complete graph with logical links connecting each pair of border nodes (see Figure 2(b)). A logical link may represent multiple physical paths. For example, in Figure 3, there are five paths going from border node a to border node b . One possible path is $a \rightarrow 1 \rightarrow 3 \rightarrow b$. The end-to-end QoS parameter of this path is $(4, 5)$. We can find the parameters of all other paths and they are $(7, 9)$, $(10, 5)$, $(2, 3)$, and $(7, 7)$. When a logical link is used to represent all these paths, the QoS parameter of the link should be the “best” parameter among the paths. However, the “best” QoS parameter may not be defined since there does not exist an absolute order among those pairs. For example, parameter $(2, 3)$ is better than parameter $(7, 7)$ in terms of delay, but not in terms of bandwidth. Fortunately, a partial order can be developed.

Definition 1 A point (x, y) is more representative than a point (x', y') if

- they are not the same, i.e., either $x \neq x'$ or $y \neq y'$, and

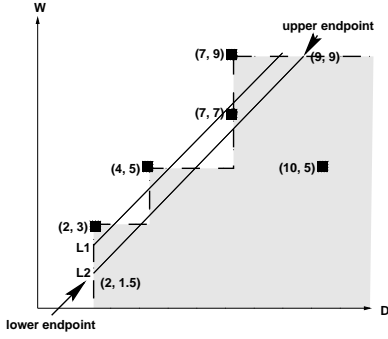


Figure 4: Representatives from Example 1

- $x \leq x'$ and $y \geq y'$.

In this paper, since the QoS parameter is a pair of values that represents a point on the delay-bandwidth plane, we often use *parameter*, *pair*, and *point* interchangeably. In the previous example, $(7, 9)$ is more representative than $(10, 5)$ since $7 \leq 10$ and $9 \geq 5$.

Definition 2 Given a set (S) of points on the delay-bandwidth plane, $(x, y) \in S$ is a representative of S if there does not exist any other point $(x', y') \in S$ which is more representative than (x, y) . It means that, $\forall (x', y') \in S$, $x < x'$ or $y > y'$.

Example 1 Let S be the set of the delay-bandwidth QoS pairs of the physical paths from a to b in Figure 3. $S = \{(4, 5), (7, 9), (10, 5), (2, 3), (7, 7)\}$. $(2, 3)$ is a representative of S , since its delay is less than all other points in S . The other representatives are $(4, 5)$ and $(7, 9)$.

We plot all QoS points in S on a delay-bandwidth plane as shown in Figure 4. The shaded area defines the region of *supported services*, that is, any QoS request that falls in that region can be supported by a physical path. The dotted line is a staircase rising from left to right. The representatives are points on the convex corners of the steps. The corresponding paths of the representatives are called nondominated paths or Pareto optimal paths in the literature. The size of S depends on how many physical paths there are between the pair of border nodes, which can be exponential. The number of representative points is $|E|$ in the worst case for a domain (V, B, E) . That is because there are at most $|E|$ possible bandwidth values for the paths in the domain and thus the staircase can have at most $|E|$ convex corners.

For the purpose of scalability, it is desirable to reduce the memory space for storing the QoS parameter of a logical link to $O(1)$. Hence, we shall neither store all QoS points in S nor store all representative points. Traditional way to solve this problem is to keep only one QoS point per logical link. However, no matter which point we pick, much information is lost. Our solution for this problem is to use a line segment that approximates the staircase, e.g., $L1$ or $L2$ in Figure 4. Since every line segment can be defined unambiguously by two endpoints, the space needed for a logical link is $O(1)$. The line segment representation strikes a tradeoff between the accuracy in approximating

the staircase and the overhead incurred.

After a line segment is chosen, all connection requests that fall under the line segment are accepted. However, it should be pointed out that not all requests under the line segment can be actually supported. For example, in Figure 4, if $L1$ is selected to approximate the staircase, then the unshaded areas below $L1$ represent connection requests that are accepted but not supported by any physical path. When a request is accepted but not supported, our routing process will detect it, and the request will eventually be rejected if a feasible path can not be found (see Section 5). On the other hand, when we reject the connection requests that are above the line segment, we may reject supported QoS. For example, if a connection request is in the shaded region above $L1$ in Figure 4, it is rejected although it can be served. Therefore, the choice of line segment depends on the strictness of the desired quality of the service. For instance, in Figure 4, both $L1$ and $L2$ are possible line segments. Using $L2$ would probably reject more supported connection requests than $L1$, while using $L1$ would accept more unsupported requests than $L2$. We use the least square method to find a line segment, which takes linear time with respect to the number of representative points. This line segment minimizes the least-square error, i.e., the summation of the squares of the distances from the points to the line.

We mentioned earlier that any line segment can be defined by two endpoints. Due to the nature of the staircase, the line always has a non-negative slope. We call the endpoint with smaller bandwidth the *lower endpoint*, while the other one is called the *upper endpoint*. We then denote a line segment as $[lower\ endpoint, upper\ endpoint]$. For example, $L2$ in Figure 4 is $[(2, 1.5), (9, 9)]$. We further denote the lower endpoint and the upper endpoint of a line segment l as $l.lp$ and $l.up$, respectively. The delay of a point p is $p.d$ and the bandwidth of a point p is $p.w$. Therefore, $L2.lp.d$ is 2 and $L2.up.w$ is 9.

We now define two operations for line segment parameters. The first operation is the *joint* operation, denoted as “+.” This operation defines the line segment parameter of a path $i \rightarrow n \rightarrow \dots \rightarrow j$, given the line segment of link $i \rightarrow n$ to be $[(a, b), (c, d)]$ and the line segment of subpath $n \rightarrow \dots \rightarrow j$ to be $[(a', b'), (c', d')]$.

Definition 3 $[(a, b), (c, d)] + [(a', b'), (c', d')] = [(a + a', \min(b, b')), (c + c', \min(d, d'))]$

The other operation is the *disjoint* operation, denoted as “-.” This operation is needed when we want to split a link (one line segment) into two links (two line segments).

Definition 4 $[(a, b), (c, d)] - [(a', b'), (c', d')] = [(a - a', \min(b, b')), (c - c', \min(d, d'))]$

The disjoint operation is defined in a way that serves as the algebraic inverse of the joint operation. The following lemma shows the relation between the joint and disjoint operations. The proof of this lemma can be found in Appendix B.1.

Lemma 1 If l_1, l_2 and l_3 are three line segments and they

satisfy the following conditions:

- $l_1.lp.d \geq l_2.lp.d$ and $l_1.up.d \geq l_2.up.d$, and
- $l_1.lp.w \leq l_2.lp.w$ and $l_1.up.w \leq l_2.up.w$, and
- $l_1 - l_2 = l_3$

then $l_1 = l_2 + l_3$.

Example 2 Let $l_1 = [(10, 4), (13, 7)]$ and $l_2 = [(6, 5), (11, 7)]$. $l_1.lp.d \geq l_2.lp.d$, $l_1.up.d \geq l_2.up.d$, $l_1.lp.w \leq l_2.lp.w$, and $l_1.up.w \leq l_2.up.w$. $l_1 - l_2 = [(10, 4), (13, 7)] - [(6, 5), (11, 7)] = [(10-6, \min(4, 5)), (13-11, \min(7, 7))] = [(4, 4), (2, 7)]$. $l_2 + [(4, 4), (2, 7)] = [(6, 5), (11, 7)] + [(4, 4), (2, 7)] = [(6+4, \min(5, 4)), (11+2, \min(7, 7))] = [(10, 4), (13, 7)] = l_1$.

4 QoS-aware Topology Aggregation

Our topology aggregation algorithm consists of two phases: (1) find a line segment for each logical link in the mesh (complete graph) of the border nodes, and (2) construct a star topology with bypasses from the mesh.

4.1 Mesh Formation

In this phase, representatives of the paths between each pair of border nodes are found. We obtain the representatives for each pair of border nodes by running the Dijkstra's algorithm for every link bandwidth value. We first sort the links in the domain by descending bandwidths. Let the largest bandwidth be w_{max} . We start by finding the smallest delay (d_{max-w}) in a graph that consists of only links of bandwidth w_{max} . The pair (d_{max-w}, w_{max}) is a representative that has the largest bandwidth. We then insert the links of the next largest bandwidth value (w') and find the smallest delay d' . If $d' < d_{max-w}$, (d', w') is another representative. The process ends after the smallest delays for all bandwidth values are identified. The detailed algorithm of finding the representatives can be found in Appendix A.1. Other related references are [13] and [14]. Sorting links takes $O(|E| \log |E|)$ time and Dijkstra's algorithm takes $O(|E| + |V| \log |V|)$ time [15]. Since there are at most $|E|$ different bandwidth values, Dijkstra's algorithm will be executed at most $|E|$ times. Then, the total running time of finding the representatives from a source border node to all other border nodes is $O(|E|^2 + |E||V| \log |V|)$. As there are $|B|$ border nodes, the total running time for finding all representatives is $O(|B||E|^2 + |B||E||V| \log |V|)$.

After the representatives are found, linear regression is used to find the line segment for each pair of border nodes. The time needed for linear regression is proportional to the number of representatives on the staircase. Since there are at most $|E|$ representatives, the time complexity for linear regression between one pair of nodes is $O(|E|)$. The complexity for all pairs of nodes is thus $O(|B|^2|E|)$. Therefore, the total time required for the mesh formation phase is $O(|B||E|^2 + |B||E||V| \log |V|) + O(|B|^2|E|) = O(|B||E|^2 + |B||E||V| \log |V|)$. Note that $|B|$ is typically small, and networks are typically sparse graphs, i.e., $|E| = O(|V|)$

instead of $O(|V|^2)$. Moreover, the computation is carried out on the graph of a domain instead of the entire network.

4.2 Star Formation

Our next step is to aggregate the mesh into a star topology with bypasses. According to the recommendation of PNNI, we can set a default parameter value to spokes and there can be one or more spokes of the default parameter value and at most $3|B| - 1$ links of other values. As it is possible that all links in our aggregation are of different values, we shall put at most $3|B|$ links in the domain topology after aggregation.

Let i and j be two border nodes and n be the nucleus in the star representation. If there is no bypass between i and j , the only path from i to j is $i \rightarrow n \rightarrow j$ in the star. Our goal in this phase is to find the QoS parameters of links $i \rightarrow n$ and $n \rightarrow j$, such that the line segment of $i \rightarrow n \rightarrow j$ in the star is the same as the line segment of $i \rightarrow j$ in the mesh. Basically, we have to "split" a single link $i \rightarrow j$ in the mesh into two links $i \rightarrow n$ and $n \rightarrow j$ in the star. There are three steps in this phase: (1) find the spokes from the border nodes to nucleus, (2) find the spokes from the nucleus to the border nodes, and (3) find the bypasses between border nodes.

4.2.1 Spokes incoming to the nucleus

In order to distinguish the line segments in the mesh from those in the star, we use superscript m for mesh and s for star. For instance, let us consider a line segment from i to j . If it is in the mesh, it is denoted as l_{ij}^m ; if it is in the star, it is denoted as l_{ij}^s .

In order to find spokes, we have to "break" the line segments in the mesh. From the definition of the joint operation, we have a general idea how the "broken" line segments look like. The endpoint delays of the spokes l_{in}^s and l_{nj}^s should be smaller than those of l_{ij}^m , while the endpoint bandwidths of the spokes should not be smaller than those of l_{ij}^m . Our algorithm of finding spokes from border node i to nucleus n is based on these observations. Recall that the lower endpoint and the upper endpoint of a line segment l are denoted as $l.lp$ and $l.up$, respectively. Given a point p , its delay and bandwidth are denoted as $p.d$ and $p.w$, respectively. We define $l_{in}^s = [(min_ld, max_lw), (min_ud, max_uw)]$, where $min_ld = \min_{j \in B, j \neq i} \{l_{ij}^m.lp.d\}$, $min_ud = \min_{j \in B, j \neq i} \{l_{ij}^m.up.d\}$, $max_lw = \max_{j \in B, j \neq i} \{l_{ij}^m.lp.w\}$, and $max_uw = \max_{j \in B, j \neq i} \{l_{ij}^m.up.w\}$. The pseudocode of the algorithm to compute l_{in}^s can be found in Appendix A.2. The total running time for finding one spoke is $O(|B|)$. There are $|B|$ spokes incoming to n . Therefore, it takes $O(|B|^2)$ time to find all spokes incoming to the nucleus.

Example 3 Suppose the line segments from node 0 to nodes 1 and 2 are $[(9, 4), (19, 6)]$ and $[(3, 7), (3, 7)]$, respectively. $min_ld = \min\{9, 3\} = 3$, $min_ud = \min\{19, 3\} = 3$, $max_lw = \max\{4, 7\} = 7$ and $max_uw = \max\{6, 7\} = 7$. Therefore, the line segment from 0 to nucleus is $[(3, 7), (3, 7)]$.

4.2.2 Spokes outgoing from the nucleus

We now proceed to find the spokes from the nucleus to the borders. Up to this point, we know the mesh, and we also know the set of spokes from borders to the nucleus, which is denoted as $S_{b \rightarrow n}$. More specifically, we know l_{ij}^m as well as l_{in}^s , and we want to find l_{nj}^s , such that the result of joining l_{in}^s and l_{nj}^s is l_{ij}^m . Since $l_{ij}^m.lp.d \geq l_{in}^s.lp.d$, $l_{ij}^m.up.d \geq l_{in}^s.up.d$, $l_{ij}^m.lp.w \leq l_{in}^s.lp.w$ and $l_{ij}^m.up.w \leq l_{in}^s.up.w$, we can obtain l_{nj}^s by evaluating $l_{ij}^m - l_{in}^s$, according to Lemma 1. However, for the same j , $l_{ij}^m - l_{in}^s$ may be different for different i , as illustrated in Example 4. Since we can have at most one l_{nj}^s , we solve this problem by assigning l_{nj}^s the average of $l_{ij}^m - l_{in}^s$, for all $i \in B$, $i \neq j$. It means to assign the average delay and the average bandwidth of the endpoints of $l_{ij}^m - l_{in}^s$, for all $i \in B$, $i \neq j$. The pseudocode of finding spokes outgoing from the nucleus is shown in Appendix A.3. The complexity of this step is $O(|B|^2)$.

Example 4 Refer to Example 3, $l_{01}^m = [(9, 4), (19, 6)]$ and $l_{0n}^s = [(3, 7), (3, 7)]$. Then $l_{01}^m - l_{0n}^s$ would be $[(6, 4), (16, 6)]$. On the other hand, if $l_{20}^m = [(3, 3), (4, 6)]$ and $l_{21}^m = [(8, 3), (20, 6)]$, then $l_{2n}^s = [(3, 3), (4, 6)]$ and $l_{21}^m - l_{2n}^s = [(5, 3), (16, 6)]$. Hence, l_{n1}^s is the average of $[(6, 4), (16, 6)]$ and $[(5, 3), (16, 6)]$, which is $[(5.5, 3.5), (16, 6)]$.

4.2.3 Finding bypasses

Due to the aggregation, $l_{ij}^s = l_{in}^s + l_{nj}^s$ may no longer be the same as l_{ij}^m in the mesh. Some may deviate only a little bit, while others may be quite different. In order to make the aggregation more precise, bypasses are introduced, which are direct links between border nodes (Figure 2(d)). When $l_{in}^s + l_{nj}^s$ deviates a lot from l_{ij}^m , we put a bypass between border nodes i and j , and the QoS parameters of the paths from i to j are defined by the bypass instead of $l_{in}^s + l_{nj}^s$. Since we have $2|B|$ spokes ($|B|$ outgoing from and $|B|$ incoming to the nucleus), we can put $|B|$ bypasses in the network according to the recommendation of the PNNI standard. We consider including bypasses between those $|B|$ pairs of border nodes that have the largest deviations.

In general, the deviation between two line segments is represented by an area. Figure 5 shows an example. In Figure 5(a), regions I and II are those services that are accepted by line $L1$, while regions I and III are accepted by $L2$. Hence, regions II and III represent deviations between $L1$ and $L2$. In Figure 5(b), although line segment $L1$ is a point, the deviation area is still II and III. Unfortunately, finding the deviation in the form of area is not always possible, because some regions are open areas with infinite size. An alternative approach for measuring deviation is to use the distance between the endpoints, which is easy to compute. We adopt this approach in our algorithm. We denote the Cartesian distance between two points p_1 and p_2 as $|p_1 p_2|$ and define the distance function as follows:

Definition 5 $|\cdot| : L \times L \rightarrow \mathfrak{R}$, where L is the set of line segments, is defined such that the distance ($|l_1 l_2|$) between two line segments l_1 and l_2 is:

| Case | line feature | $ l_1 l_2 $ |
|------|--------------------------|-------------------------------------|
| I | both are lines | $ l_1.lp.l_2.lp + l_1.up.l_2.up $ |
| II | both are points | $2^* l_1.lp.l_2.lp $ |
| III | only l_1 is a point | $ l_1.lp.l_2.lp +$ |
| IIIa | $l_2.up.d \geq l_1.up.d$ | $ l_1.up.w - l_2.up.w $ |
| IIIb | $l_2.up.d < l_1.up.d$ | $ l_1.up.l_2.up $ |

Figure 6 shows the different cases of calculating the distance between two line segments. The distance is the sum of the lengths of the arrows except for Case II, which is 2 times the length of the arrow. We define Case IIIa in this way, because the point l_1 covers the same region as the line segment from l_1 to X in Figure 6(c).

It takes constant time to compute the deviation between two line segments. The time complexity for computing deviations between l_{ij}^m and l_{ij}^m , for all $i, j \in B$, $i \neq j$, is $O(|B|^2)$. We can then select $|B|$ border pairs that have the largest deviations for putting bypasses. By using a heap, the selection takes $O(|B|^2 \log |B|)$ time. After finding the bypasses, we can recompute the spokes by excluding those border pairs that have bypasses.

4.3 Complexities

The total time complexity of mesh formation and star formation is $O(|B||E|^2 + |B||E||V| \log |V|) + O(|B|^2) + O(|B|^2) + O(|B|^2 \log |B|) = O(|B||E|^2 + |B||E||V| \log |V|)$. The computation for mesh formation dominates. The computation can be carried out by a designated node in the domain periodically, and the aggregation results (size of $O(|B|)$) are sent to the border nodes of the same domain. The border nodes then propagate the aggregation results to other domains. Multiple designated nodes (e.g., border nodes, internal nodes, or dedicated computers) may share the computation if necessary. For instance, each node computes the line segments from one border node to all other border nodes and then exchanges the results, so that the workload of mesh formation is spread among them. Multiple designated nodes also prevent the problem of single-point failure.

Recall that a network of η domains is denoted as (G, L) , where $G = \{g_i | g_i = (V_i, B_i, E_i), 1 \leq i \leq \eta\}$. Without aggregation, the storage complexity is $O(|L| + \sum_{i=1}^{\eta} (|V_i| + |E_i|))$, which increases with the total number of nodes and links in the entire network. With aggregation, the storage complexity for a node in domain g_k , $1 \leq k \leq \eta$, is $O(|L| + |V_k| + |E_k| + \sum_{i=1}^{\eta} |B_i|)$, which increases with the number of border nodes, the inter-domain links, and the size of domain g_k . The internal complexity of external domains is not a factor.

5 Line-Segment Routing Algorithm

In this section, we describe our routing algorithm. Although the routing problem with two additive metrics is known to be NP-complete [16], routing in a delay-bandwidth sensitive network can be solved in polynomial time [13]. Because our line segment representation is different from the traditional approaches, none of the existing routing

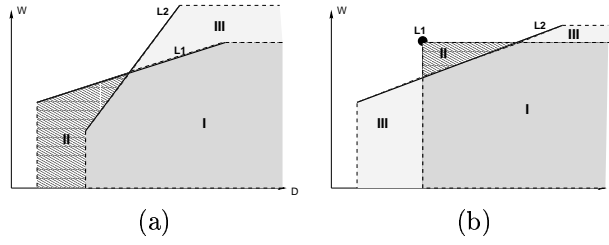


Figure 5: Deviation of line segment

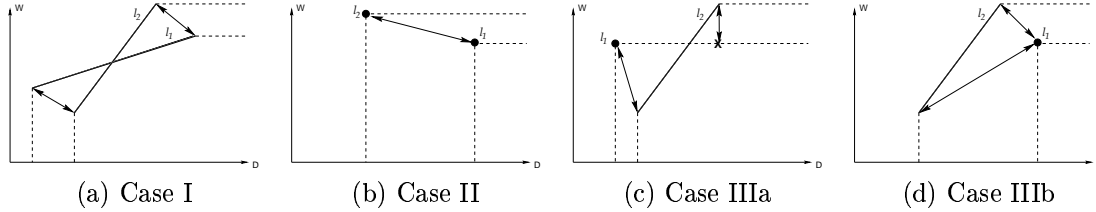


Figure 6: Different cases in Definition 5

algorithms can be applied. We present *Line-Segment Routing Algorithm* (LSRA), a QoS-based source routing algorithm, which integrates the line segment representation with Dijkstra’s algorithm (DA) and the centralized bandwidth-delay routing algorithm (CBDRA) [13, 17]. DA is the fastest known algorithm for finding the least-delay paths. CBDRA first prunes all links that do not satisfy the bandwidth requirement and then applies DA to find the least-delay path. LSRA extends the idea to capitalize the additional information provided by the line segment representation.

Being a centralized routing protocol, LSRA requires that each node keeps the topology of its own domain and the *star-with-bypasses* aggregation of the other domains. As broadcasting takes a lot of time and bandwidth, it is desirable to keep the amount of broadcasted information small. This justifies why our aggregation is a star with bypasses of $O(|B|)$ space instead of a mesh of $O(|B|^2)$ space. Routing is performed at two levels: *inter-domain routing* and *intra-domain routing*. An inter-domain routing path specifies the border nodes of the transit domains, and the intra-domain routing finds the subpath within each transit domain. Accordingly, LSRA has two routing phases.

5.1 Inter-domain Routing

After obtaining the star-with-bypasses aggregation from the external domains, each node can see all nodes in its own domain and all border nodes of the other domains. An example is shown in Figure 1(b). There are five steps in the LSRA inter-domain routing:

1. *Transform stars with bypasses to meshes;*
Since nuclei of stars are virtual, the actual routing paths should not include any nucleus.
2. *Prune logical links;*
This step prunes the logical links that do not satisfy

the bandwidth requirement.

3. *Determine the delays of logical links;*
The delay value, supported by a line segment, is a function of the bandwidth requirement. This step determines the delay values of all logical links under the bandwidth requirement.
4. *Prune physical links;*
This step prunes the physical links that do not satisfy the bandwidth requirement.
5. *Apply DA on the network;*
This step uses DA to find a shortest-delay path to the destination domain.

We now describe each step in detail and analyze the runtime complexity. The pseudocode of inter-domain LSRA can be found in Appendix A.4.

5.1.1 Transform stars with bypasses to meshes

For each external domain, the star-with-bypasses aggregation is transformed to a mesh among the border nodes. For border nodes i and j , if there is a bypass in the aggregation that goes from i to j , that bypass is the mesh link from i to j ; otherwise, the mesh link is the joint of the spoke from i to the nucleus and the spoke from the nucleus to j . The step takes $O(|B_i|^2)$ for domain g_i and $O(\sum_{i=1}^n |B_i|^2)$ for the whole network.

This step is important, because the bypass always carries more accurate information. Suppose that the bypass is inferior comparing with the joint of the two spokes. If this step was not performed and DA was directly applied on the star with bypasses, the shortest path would go through the joining of the two spokes instead of the bypass, which would result in inaccuracy of the path parameter.

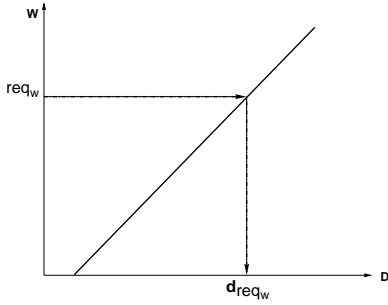


Figure 7: delay-coordinate

5.1.2 Prune logical links

Logical links are line segments connecting border nodes in the mesh calculated in the previous step. If the upper bound of the bandwidth supported by a logical link is less than the bandwidth requirement, the logical link can be pruned. Suppose that the routing request is (req_d, req_w) and a logical link is l . If $req_w > l.up.w$, l is pruned. The complexity of this step is also $O(\sum_{i=1}^{\eta} |B_i|^2)$.

5.1.3 Find the delays of logical links

A line segment shows the relation between the supported bandwidth and the supported delay. In other words, given certain bandwidth value, the line segment predicts the smallest delay value that can be supported by a physical path with the given amount of bandwidth. As shown in Figure 7, given the bandwidth requirement req_w , the best delay value supported by a line segment l is the corresponding delay coordinate (d_{req_w}). The complexity of this step is $O(\sum_{i=1}^{\eta} |B_i|^2)$.

5.1.4 Prune physical links

This step is the same as the pruning step in CBDRA. Let the parameter of physical link e be (D, W) . If $req_w > W$, e is pruned. This check is performed for each physical link. Therefore, the complexity of this step is $O(|L| + |E_a|)$, where L is the set of inter-domain physical links and E_a is the set of physical links in the source domain, $g_a = (V_a, B_a, E_a)$. Note that the physical links of the other domains have been replaced by logical links during aggregation, and thus do not appear in the graph.

5.1.5 Apply DA on the network

After the previous four steps, we have a network graph in which each link has a unique delay value. We then use the Dijkstra's algorithm (DA) to find the least-delay path from the source node to the destination domain. Among all border nodes of the destination domain, let t be the border node that is closest (in terms of delay) to the source node. Let the delay of the shortest path from the source node to t be $d[t]$, which can be calculated by DA. If $d[t] \leq req_d$, then the request is *accepted* and LSRA goes ahead with the intra-domain routing. Otherwise, the request is

rejected. The number of links in the network graph is $|E'| = |L| + |E_a| + \sum_{i=1, i \neq a}^{\eta} |B_i|^2$. The number of nodes is $|V'| = |V_a| + \sum_{i=1, i \neq a}^{\eta} |B_i|$. Their values depend only on the number of inter-domain links L , the size of the source domain g_a , and the number of border nodes $B_i, 1 \leq i \leq \eta$. The internal structures of all other domains have no impact. The complexity of this step is therefore $O(|E'| + |V'| \log |V'|)$, which is also the total complexity of the inter-domain routing, since the time complexities of the previous four steps are smaller.

5.2 Intra-domain Routing

After the inter-domain routing, an inter-domain path is determined. The source node knows how to traverse the nodes in its own domain to a border node, and how to traverse the border nodes of other domains to get to the destination domain. Because the source node does not have the detailed topology of any external domain, it does not know how to fill in the intra-domain path segments across the external domains. Therefore, LSRA does the intra-domain routing in a distributed fashion. A routing message is sent from the source to travel along the inter-domain path. When a border node t of domain g receives the message and the next hop on the inter-domain path is another border node t' of g , t locally computes the intra-domain path going from itself to t' by using CBDRA, since t has the complete knowledge about its own domain g . Node t inserts the intra-domain path into the inter-domain path that is carried by the message. Then the message is sent to t' along the intra-domain path. The message also keeps track of the accumulated delay of the path that has been traversed so far, including the intra-domain path segments. This accumulated delay is calculated from the actual link delays as the message travels. If the accumulated delay exceeds req_d ,³ the message is forwarded back to the previous node to find an alternate path. This is called *crankback* [1]. If the message successfully reaches the destination node, a feasible path is found. If no intra-domain path can be found without violating req_d , the message is sent back to the source to reject the request.

The complexity of CBDRA is identical to DA. That is, for each intermediate domain g_i that the routing message traverses, it takes $O(|V_i| \log |V_i| + |E_i|)$ time to compute an intra-domain path in g_i .

6 Simulation

6.1 Comparison Metrics

We compare the performance of LSRA and other approaches in terms of *delay deviation*, *success ratio*, and *crankback ratio*.

- *delay deviation* – Due to the distortion of aggregation, the delay between a source node and a destination do-

³The expected path delay may be different from the actual path delay due to the network dynamics or the inaccuracy introduced by aggregation.

main, obtained using the method described in Section 5.1, is not accurate. Delay deviation measures the difference between the real delay and the estimated delay, obtained by the aggregation. It is defined to be $| \text{estimated delay of the shortest path} - \text{actual delay of the shortest path} |$.

- *success ratio* – A feasible request may be rejected due to the imperfect approximation of the delay and bandwidth information during aggregation. Success ratio is used to measure quantitatively how well an algorithm finds feasible paths and it is defined as

$$\frac{\text{total number of accepted feasible requests}}{\text{total number of feasible requests}}.$$

The dividend represents all connection requests that are accepted by both inter-domain routing and intra-domain routing. The divider is the total number of *feasible* requests (not the total number of requests). Therefore, in our simulation, success ratio measures the relative performance with respect to the optimal performance (accepting all feasible requests).

- *crankback ratio* – When an algorithm finds an inter-domain path from the source node to the destination domain, it may overestimate the bandwidth or underestimate the delay and lead to crankbacks during intra-domain routing. Crankback ratio measures how often that happens and is defined as

$$\frac{\text{total number of requests crankbacked}}{\text{total number of requests accepted by inter-domain routing}}.$$

A good algorithm should have small crankback ratios, high success ratios, and small delay deviations for different bandwidth and delay requirements.

6.2 Comparing Schemes

We compare LSRA with other aggregation and routing schemes. They are the *Best Point algorithm* (BP), the *Worst Point algorithm* (WP), and the modified Korkmaz-Krunz algorithm (KK).

- *Best Point* – The best delay and the best bandwidth are used to represent a mesh link. For example, if the delay-bandwidth parameters between a border pair are (2, 3), (4, 4) and (5, 6), (2, 6) is used to represent the QoS of the mesh link. This optimistic approach is aggressive by choosing the best delay and the best bandwidth that may come from different paths. It suffers from large crankback ratios. We aggregate a BP mesh to a BP star by taking the maximum bandwidth among the mesh links as the spoke bandwidth and by taking half of the average delay among the mesh links as the spoke delay.⁴
- *Worst Point* – The worst delay and the worst bandwidth are used to represent a mesh link. That is, if the parameters are (2, 3), (4, 4) and (5, 6), then (5, 3) is used. As a result, this algorithm has low success ratios. The minimum bandwidth and average of half of the average delay are used for aggregating a WP mesh to a WP star.

⁴A mesh link from i to j splits to two spokes from i to n and from n to j .

- *modified Korkmaz-Krunz* – This is a modified version of the algorithm presented in [9]. Let us consider a set of delay-bandwidth parameters, which corresponds to a set of paths between two border nodes. The *stretch factor* of a parameter (d, w) is defined as $\frac{d}{d_{best}} + \frac{w_{best}}{w}$, where d_{best} and w_{best} are the best delay and the best bandwidth among all parameters, respectively. Refer to the examples used previously for BP and WP, $d_{best} = 2$ and $w_{best} = 6$. The stretch factor of (2, 3) is $\frac{2}{2} + \frac{6}{3} = 3$, and the stretch factors of (4, 4) and (5, 6) are both $3\frac{1}{2}$. The smallest stretch factor is denoted as *s_factor*, which is 3 in the above case. The mesh link between the two border nodes is represented by a tuple $(d_{best}, w_{best}, s_factor)$. The mesh link is likely to support a connection request (req_d, req_w) , if $d_{best} \leq req_d$, $w_{best} \geq req_w$, and $s_factor \leq \frac{req_d}{d_{best}} + \frac{w_{best}}{req_w}$. Readers are referred to the original paper [9] for detailed explanation.

The modification we made is on the transformation from a mesh to a star, in particular, on the transformation of delay. The formula used in [9] is as follows.

$$s_{in}.d = \frac{1}{b-1} \sum_{j=1, j \neq i}^b m_{ij}.d$$

$$s_{nj}.d = \frac{1}{b-1} \sum_{i=1, i \neq j}^b m_{ij}.d$$

where $m_{ij}.d$ is the delay of a mesh link from border node i to border node j , b is the number of border nodes, s_{in} is the delay of a spoke link from i to the nucleus, and s_{nj} is the delay of a spoke link from the nucleus to j . The problem of this averaging approach is that the delay after transformation deviates a lot from the original delay, because the delay of $s_{in}.d + s_{nj}.d$ may be twice the delay of the mesh link $(m_{ij}.d)$.⁵ Therefore, we apply the following modification

$$s_{in}.d = \frac{1}{2*(b-1)} \sum_{j=1, j \neq i}^b m_{ij}.d$$

$$s_{nj}.d = \frac{1}{2*(b-1)} \sum_{i=1, i \neq j}^b m_{ij}.d$$

The simulation results, using the setup described in Section 6.3, are shown in Figure 8. The horizontal axis is the bandwidth requirement. The results show that the modified algorithm has better success ratios than the original algorithm in [9]. For crankback ratios, the modified KK algorithm is worse than the original KK algorithm, because the spoke delays in the modified KK is smaller than the spoke delays in the original KK. More optimistic estimation leaves more room for overestimation, which may lead to more crankbacks. However, as the crankback ratios of the modified KK are less than 5% for all bandwidth values (Figure 11), we believe that the performance improvement in success ratios outweighs the degradation of crankback ratios. As a result, we use the modified KK algorithm for our further simulations.

It takes two values, bandwidth and delay, for BP or WP to store the aggregated state of a logical link. It takes

⁵Similar formula is used in [9] to transform bandwidth. However, for bandwidth QoS constraints, such a problem does not arise because bandwidth is not additive.

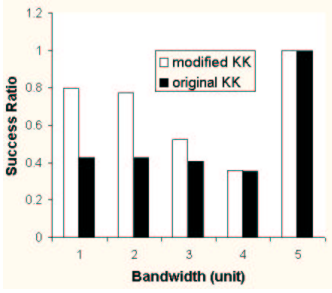


Figure 8: Success Ratios of Two Aggregation Schemes of the Korkmaz-Krunz Method

three values, bandwidth, delay, and stretch factor, for KK to store the state of a logical link. It takes four values, bandwidth and delay of two points, for LS (Line Segment) to store the state of a logical link. It takes two values, bandwidth and delay, for any algorithm to store the state of a physical link. Therefore, in the worst case, LS takes twice the space taken by BP or WP, or 1.5 times the space taken by KK, to store the aggregated topology.

6.3 Simulation Testbed

The inter-domain topology is generated based on the Power-Law model [18, 19], and the intra-domain topology is generated based on the Waxman model [20]. The degree of a domain is defined as the total number of inter-domain links adjacent to the border nodes of the domain. The simulation testbed and parameter configuration are set as follows: 10% of the domains has a degree of one, and the degrees of the other domains follow the power law, i.e., the frequency f_d of a domain degree is proportional to the degree $d(\geq 2)$ raised to the power of a constant $O = -2.2$.

$$f_d \propto d^O$$

After each domain is assigned a degree according to the power law, a spanning tree is formed among the domains to ensure a connected graph. Additional links are inserted to fulfill the remaining degrees of every domain with the neighbors selected according to probabilities proportional to their respective unfulfilled degrees. The Waxman topology for the internal nodes of each domain is formed as follows: the nodes are randomly placed in a one-by-one square, and the probability of creating a link between node u and node v is

$$p(u, v) \propto e^{-d(u, v)/\beta L}$$

where $d(u, v)$ is the distance between u and v , $\beta = 0.6$, and L is the maximum distance between any two nodes. The average node degree is 4. The other simulation parameters are: the number of domains in each topology is 200, the number of nodes in each domain is randomly selected between 10 and 40, the delay of a physical link is a random number between 2 to 10 units, and the bandwidth of a physical link is a random number between 1 to 5 units. For delay, a unit can be replaced by certain number

of millisecond; for bandwidth, a unit can be replaced by certain number of kilobytes (per second).

Each data point is the average of 1000 randomly generated requests. More specifically, given a bandwidth requirement and/or a delay requirement, we randomly generate five topologies. On each topology, 200 requests are generated with the source and the destination randomly selected from the topology. We run LS, BP, WP, and KK over these requests, respectively, and the average results give a set of data points in the figures.

6.4 Simulation Results

Figure 9 shows the delay deviations of LS, BP, WP, and KK. The horizontal axis presents the bandwidth requirement. Since there is no delay requirement, the least delay path with the required bandwidth is returned. The vertical axis is the average delay deviation of the paths returned by the algorithms. For fairness, we count the paths of a request only when all four algorithms accept that request. Figure 9 shows that LS has smaller delay deviations than the other algorithms. That is because a line segment approximates the parameter staircase better than the other approaches, and thus gives more accurate information about the delay. The deviations are small, when the bandwidth requirements are large (≥ 4), because the feasible paths are mostly short (within the same domain or between neighboring domains).

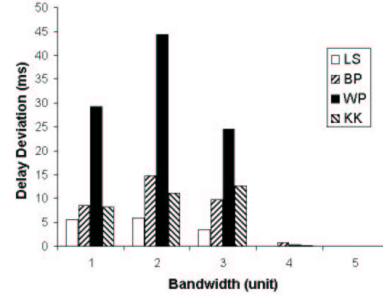


Figure 9: Delay Deviation of Different Aggregation Schemes

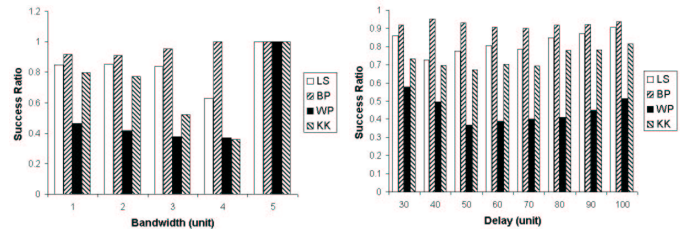


Figure 10: Success Ratio vs. Bandwidth and Delay

The success ratios are shown in Figure 10. It may be surprising that the success ratios are close to one when the bandwidth requirement is 5 units (maximum link capacity). Recall that our success ratio is defined relative to the number of feasible requests, not to the number of all

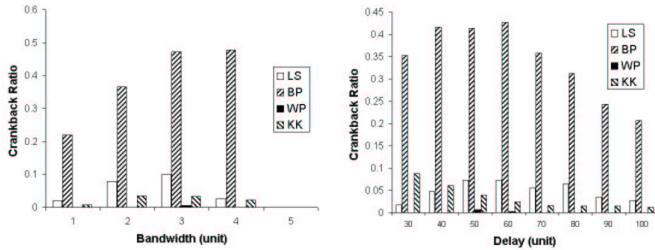


Figure 11: Crankback Ratio vs. Bandwidth and Delay

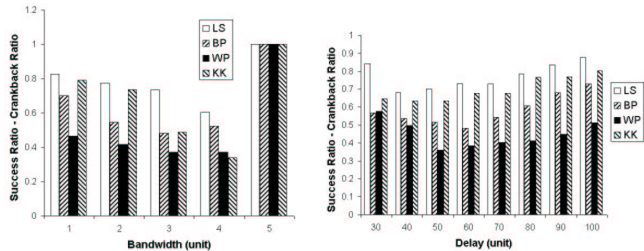


Figure 12: Success Ratio - Crankback Ratio

requests. When the bandwidth request is large, there are few feasible requests. These feasible requests are mostly between nodes that are close to each other (in the same domain or neighboring domains), which makes the delay deviation small as shown in Figure 9. That means it is less likely to reject a feasible request due to distortion. Therefore, the success ratio is high.

Generally speaking, the success ratios of BP are the best in Figure 10. It is because BP advertises the largest spoke bandwidths and small spoke delays. This over estimation reduces the chance of rejecting feasible requests during the inter-domain routing⁶. On the other hand, the same over estimation increases the chance of accepting infeasible requests by the inter-domain routing, which causes the intra-domain routing to have high crankback ratios. Figure 11 shows the crankback ratios of BP, WP, KK, and LS. It is clear that BP has very high crankback ratios compared to other methods. On the contrary, as an aggregated WP topology tends to overestimate the real delay and underestimate the real bandwidth of the paths, WP has very small crankback ratios⁷.

A good algorithm should have high success ratios and low crankback ratios. Therefore, we also compare *success ratio - crankback ratio*, which is shown in Figure 12. BP does not perform well in this comparison due to its high crankback ratios. In contrast, LS has higher success ratios than WP, and KK, and has reasonable crankback ratios. It outperforms all other schemes for different bandwidth and delay

⁶ It can be noted that the success ratios of BP are not always 1. This is because when we find spokes for BP, we take half of the average delay of the mesh links outgoing from i as the spoke delay from i to the nucleus, and take half the average delay of the mesh links incoming at j as the spoke delay from the nucleus to j . Therefore, it is possible that the real minimum delay from i to j is smaller than the delay represented by the aggregated topology. Then, the routing algorithm may reject a supported request.

⁷ The crankback ratios of WP are not always zero, because we take half of the average delay among the mesh links as the spoke delay.

requirements by making a better tradeoff between success ratio and crankback ratio.

7 Conclusion

In this paper, we present a novel QoS representation for topology aggregation in delay-bandwidth sensitive networks. We use line segments in the delay-bandwidth plane instead of points to represent the QoS parameters of logical links. We discuss algorithms to compute line segments for logical links of a mesh, to aggregate links into a star with bypasses representation, and to find QoS routes using line segments.

The presentation of our aggregation scheme follows the PNNI standard, where each node has a complete view of the domain it belongs to and an aggregated view of the rest of the network. Each node carries out its own on-demand inter-domain QoS routing, which spreads the workload of QoS routing across the network. On the other hand, the aggregation approach itself is new, general and independent of the standard. While there may be some low-end routers in the domain that do not have the CPU/memory capacity to handle the aggregation, our approach does not require every node to participate in the aggregation computation (Section 4.3) or to store the aggregated topology. These low-end routers may forward the routing requests to their default designated routers for QoS inter-domain routing. These designated routers for inter-domain routing do not have to be border nodes as is the case in BGP. This flexibility helps to prevent the performance degradation of the critical border nodes due to excessive number of QoS inter-domain routing requests.

Extensive simulations show that our algorithms achieve high success ratio with small crankback rate. Although this paper focuses on delay and bandwidth, the line segment aggregation is suitable for other additive/bottleneck metric pairs, such as cost and bandwidth.

Acknowledgment

The authors would like to thank the anonymous reviewers for their constructive comments and suggestions on improving the quality of the paper.

References

- [1] ATM Forum, “Private network-network interface specification version 1.0,” f-pnni-0055.000, Mar. 1996.
- [2] Y. Rekhter and T. Li, “A Border Gateway Protocol 4 (BGP-4),” RFC 1771, Mar. 1995.
- [3] R. Guerin and A. Orda, “QoS-based routing in networks with inaccurate information: Theory and algorithms,” *IEEE/ACM Transactions on Networking*, vol. 7, no. 3, pp. 350–364, June 1999.
- [4] Fang Hao and Ellen W. Zegura, “On Scalable QoS Routing: Performance Evaluation of Topology Aggregation,” in *INFOCOM '00*, 2000, pp. 147–156.

- [5] W. Lee, “Minimum equivalent subspanner algorithms for topology aggregation in ATM networks,” in *2nd International Conference on ATM (ICATM)*, June 1999, pp. 351–359.
- [6] Baruch Awerbuch and Yuval Shavitt, “Topology Aggregation for Directed Graphs,” *IEEE/ACM Transactions on Networking*, vol. 9, no. 1, pp. 82–90, Feb. 2001.
- [7] W. Lee, “Spanning tree method for link state aggregation in large communication networks,” in *INFOCOM ’95*, 1995, pp. 297–302.
- [8] A. Iwata, H. Suzuki, R. Izmailow, and B. Sengupta, “QoS Aggregation Algorithms in Hierarchical ATM Networks,” in *IEEE Proceedings of the ICC ’98*, 1998, pp. 243–248.
- [9] T. Korkmaz and M. Krunkz, “Source-oriented topology aggregation with multiple QoS parameters in hierarchical networks,” *ACM Transactions on Modeling and Computer Simulation*, vol. 10, no. 4, pp. 295–325, Oct. 2000.
- [10] 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*, Dec. 1998.
- [11] W. Lee, “Topology Aggregation for Hierarchical Routing in ATM Networks,” in *ACM SIGCOMM ’95, Computer Communication Review*, 1995, pp. 82–92.
- [12] L. Guo and I. Matta, “On state aggregation for scalable QoS routing,” in *IEEE Proceedings of the ATM Workshop ’98*, May 1998, pp. 306–314.
- [13] Z. Wang and J. Crowcroft, “Bandwidth-Delay Based Routing Algorithms,” in *IEEE Proceedings of GLOBECOM’95*, 1995, vol. 3, pp. 2129–2133.
- [14] D. Bauer, J.N. Daigle, I. Iliadis, and P. Scotton, “Efficient Frontier Formulation for Additive and Restrictive Metrics in Hierarchical Routing,” in *IEEE Proceedings of the ICC ’00*, 2000.
- [15] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, *Introduction to Algorithms*, M.I.T. Press, Cambridge, Massachusetts, U.S.A., 1990.
- [16] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York, USA, 1979.
- [17] Z. Wang and J. Crowcroft, “Quality-of-service routing for supporting multimedia applications,” *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 7, Sept. 1996.
- [18] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos, “On Power-Law Relationships of the Internet Topology,” in *ACM Proceedings of SIGCOMM ’99*, 1999.
- [19] C. Jin, Q. Chen, and S. Jamin, “Inet-3.0: Internet Topology Generator,” *University of Michigan Technical Report CSE-TR-456-02*, 2002.
- [20] B. M. Waxman, “Routing of Multipoint Connections,” *IEEE Journal of Selected Area in Communications*, pp. 1617–1622, Dec. 1988.

A Pseudocodes

A.1 Pseudocode for finding representatives

```

FIND-MESH( $V, B, E$ )
1  for each  $b \in B$ 
2    do FIND-REPRESENTATIVES( $V, B, E, b$ )

/* pseudocode for finding representatives of the paths
   from border node  $b$  to each other border node  $b' \in B$ 
*/

/*  $delay(b, b')$  is the minimum delay between  $b$  and  $b'$ 
   computed by the Dijkstra’s algorithm */

```

```

FIND-REPRESENTATIVES( $V, B, E, b$ )
1  sort the links in  $E$  in the order of descending bandwidths
2  add the sorted links in  $Q$ 
3   $E' \leftarrow \emptyset$ 
4   $prev\_delay(b, b') \leftarrow \infty$ , for each  $b' \in B$  and  $b' \neq b$ 
5  while  $Q \neq \emptyset$ 
6    do  $w \leftarrow$  next largest bandwidth in  $Q$ 
7      remove links of bandwidth  $w$  from  $Q$ 
8      add them in  $E'$ 
9      apply Dijkstra’s algorithm on  $(V, E')$  to find
10      $delay(b, b')$  for each  $b' \in B$  and  $b' \neq b$ 
11     for each  $b' \in B$  and  $b' \neq b$ 
12       do if  $delay(b, b') < prev\_delay(b, b')$ 
13         then  $prev\_delay(b, b') \leftarrow delay(b, b')$ 
14           add  $(delay(b, b'), w)$  to
15           the set of representative points

```

A.2 Pseudocode for finding spokes incoming to the nucleus

```

/* find the spokes from border node  $i$  to nucleus  $n$  given
   the mesh  $M = \{l_{ij}^m \mid i, j \in B \text{ and } i \neq j\}$  */

```

```

FIND-TO-NUCLEUS-SPOKE( $i, M$ )
1   $min\_ld \leftarrow \infty$ 
2   $min\_ud \leftarrow \infty$ 
3   $max\_lw \leftarrow 0$ 
4   $max\_uw \leftarrow 0$ 
5  for each  $j \in B$  and  $i \neq j$ 
6    do if  $l_{ij}^m.lp.d < min\_ld$ 
7      then  $min\_ld \leftarrow l_{ij}^m.lp.d$ 
8      if  $l_{ij}^m.ap.d < min\_ud$ 
9        then  $min\_ud \leftarrow l_{ij}^m.ap.d$ 
10     if  $l_{ij}^m.lp.w > max\_lw$ 
11       then  $max\_lw \leftarrow l_{ij}^m.lp.w$ 
12     if  $l_{ij}^m.ap.w > max\_uw$ 
13       then  $max\_uw \leftarrow l_{ij}^m.ap.w$ 
14
15   $l_{in}^s \leftarrow [(min\_ld, max\_lw), (min\_ud, max\_uw)]$ 

```

A.3 Pseudocode for finding spokes outgoing from the nucleus

/* find the spokes from nucleus n to border node j given the mesh $M = \{l_{ij}^m \mid i, j \in B \text{ and } i \neq j\}$ and $S_{b \rightarrow n} = \{l_{in}^s \mid i \in B\}$ */

FIND-FROM-NUCLEUS-SPOKE($j, M, S_{b \rightarrow n}$)

```

1  total_ld ← 0
2  total_ud ← 0
3  total_lw ← 0
4  total_uw ← 0
5  for each  $i \in B$  and  $i \neq j$ 
6    do  $l \leftarrow l_{ij}^m - l_{in}^s$ 
7       total_ld ← total_ld + l.lp.d
8       total_ud ← total_ud + l.up.d
9       total_lw ← total_lw + l.lp.w
10      total_uw ← total_ud + l.up.w
11
12   $l_{nj}^s \leftarrow [(\frac{total\_ld}{|B|-1}, \frac{total\_lw}{|B|-1}), (\frac{total\_ud}{|B|-1}, \frac{total\_uw}{|B|-1})]$ 

```

A.4 Pseudocode for inter-domain LSRA

/* find an inter-domain path from source node a of domain $D_a = (V_a, B_a, E_a)$ to destination domain $D_t = (V_t, B_t, E_t)$, given a request (req_d, req_w) and the star-with-bypasses aggregation, S_i , for each external domain g_i */

/* first step: transform a star to a mesh */

/* S is a star-with-bypasses aggregation. M is the mesh calculated from S . A bypass from border node i to border node j is represented as l_{ij}^s . The spoke from i to the nucleus is l_{in}^s , and the spoke from the nucleus to j is l_{nj}^s . A mesh link from i to j is l_{ij}^m . */

STAR2MESH(S)

```

1   $M \leftarrow \emptyset$ 
2  for each border node  $i$ 
3    do for each border node  $j$ 
4       do if  $i = j$ 
5          then continue
6       if there is a bypass from  $i$  to  $j$  in  $S$ 
7          then  $l_{ij}^m \leftarrow l_{ij}^s$ 
8          else  $l_{ij}^m \leftarrow l_{in}^s + l_{nj}^s$ 
9        $M \leftarrow M \cup l_{ij}^m$ 
10 return  $M$ 

```

/* second step: prune logical links */

/* M is a mesh of logical links with line segment representation. A mesh link from i to j is l_{ij}^m . When a link is pruned, it is marked. */

PRUNEMESH(M, req_w)

```

1  for each border node  $i$ 
2    do for each border node  $j$ 
3       do if  $i = j$ 
4          then continue

```

```

5          if  $l_{ij}^m.up.w < req_w$ 
6             then mark  $l_{ij}^m$ 

```

/* third step: find the delays of logical links */

FINDDELAY(M, req_w)

```

1  for each border node  $i$ 
2    do for each border node  $j$ 
3       do if  $i = j$ 
4          then continue
5       if  $l_{ij}^m$  is marked
6          then  $d_{ij} \leftarrow \infty$ 
7          else  $d_{ij} \leftarrow d_{req_w}$  of  $l_{ij}^m$ 

```

/* fourth step: prune physical links */

/* E is the set of physical links. A link is marked if it is pruned. */

PRUNELINK(E, req_w)

```

1  for each link  $(i, j) \in E$ 
2    do if the bandwidth of  $(i, j)$  is less than  $req_w$ 
3       then mark  $(i, j)$ 

```

/* pseudocode for inter-domain LSRA */

/* D_a the source domain

L set of links that connect domains

S set of star-with-bypasses aggregations for external domains. Formally, $S = \{S_i \mid S_i \text{ is the aggregation of external domain } g_i\}$

a the source node

B_t set of border nodes of the destination domain

req_d delay requirement of the routing request

req_w bandwidth requirement of the routing request

*/

LSRA($D_a, L, S, a, B_t, req_d, req_w$)

```

1
2  for each external domain  $g_i$ 
3    do  $M_i \leftarrow \text{STAR2MESH}(S_i)$ 
4
5  for each external domain  $g_i$ 
6    do PRUNEMESH( $M_i, req_w$ )
7
8  for each external domain  $g_i$ 
9    do FINDDELAY( $M_i, req_w$ )
10
11 PRUNELINK( $L \cup E_a, req_w$ )
12
13 /* construct a graph  $G = (V, E)$  as the input
14    to the Dijkstra's algorithm to find a path from
15     $a$  to the destination domain */
16  $V \leftarrow V_a \cup \{B_i \mid g_i = (V_i, B_i, E_i) \text{ is an external domain}\}$ 
17  $E \leftarrow \{e \mid e \in L \cup E_a \text{ and } e \text{ is unmarked}\} \cup$ 
18     $\{e \mid e \in M_i \text{ for some } i \text{ and } e \text{ is unmarked}\}$ 

```

19
20 apply the Dijkstra's algorithm to find the
21 least-delay path from source a to one of
22 the border nodes in B_t
23
24 **if** the delay of the path is not greater than req_d
25 **then**
26 accept the path as inter-domain routing path
27 **else**
28 reject the path and the request
29

Shigang Chen received his B.S. degree in computer science from the University of Science and Technology of China in 1993. He received M.S. and Ph.D. degree in computer science from University of Illinois at Urbana-Champaign in 1996 and 1999, respectively. After graduation, he had worked with Cisco Systems for three years before joining University of Florida in 2002. His research interests include network security, quality of service in IP networks, and distributed computing. His email address is sgchen@cise.ufl.edu.

B Proofs

B.1 Proof of Lemma 1

Let $l_1 = [(a_1, b_1), (c_1, d_1)]$ and $l_2 = [(a_2, b_2), (c_2, d_2)]$. If $l_1.lp.d \geq l_2.lp.d$, $l_1.up.d \geq l_2.up.d$, $l_1.lp.w \leq l_2.lp.w$ and $l_1.up.w \leq l_2.up.w$, according to the definition of disjoint, l_3 , which is $l_1 - l_2$ should equal to $[(a_1 - a_2, b_2), (c_1 - c_2, d_2)]$. Then, $l_2 + l_3$, according to the definition of joint, is equal to $[(a_2, b_2), (c_2, d_2)] + [(a_1 - a_2, b_2), (c_1 - c_2, d_2)] = [(a_2 + a_1 - a_2, b_2), (c_2 + c_1 - c_2, d_2)] = [(a_1, b_1), (c_1, d_1)] = l_1$. \square

King-Shan Lui (M '02) received her B.Eng. and M.Phil. degrees in computer science from the Hong Kong University of Science and Technology. She then received her PhD degree, also in Computer Science, from the University of Illinois at Urbana-Champaign. She is now an assistant professor in the Department of Electronic and Electrical Engineering in the University of Hong Kong. Her research interests are network protocol design and analysis. Her email address is kslui@eee.hku.hk.

Klara Nahrstedt is an associate professor at the University of Illinois at Urbana-Champaign, Computer Science Department. Her research interests are directed towards multimedia protocols, quality of service (QoS) provision, QoS routing, and QoS-aware resource management in distributed multimedia systems. She is the coauthor of the widely used multimedia book 'Multimedia: Computing, Communications and Applications' published by Prentice Hall, the recipient of the Early NSF Career Award, the Junior Xerox Award, IEEE Communication Society Leonard Abraham Award for Research Achievements, and the Ralph and Catherine Fisher Professorship Chair. Since June 2001 she serves as the editor-in-chief of the ACM/Springer Multimedia System Journal.

Klara Nahrstedt received her BA in mathematics from Humboldt University, Berlin, in 1984, and M.Sc. degree in numerical analysis from the same university in 1985. She was a research scientist in the Institute for Informatik in Berlin until 1990. In 1995 she received her PhD from the University of Pennsylvania in the department of Computer and Information Science. She is the member of IEEE, and ACM. Her email address is klara@cs.uiuc.edu.