Hierarchical QoS Routing in Delay-Bandwidth Sensitive Networks

King-Shan Lui
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
kinglui@uiuc.edu

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

Shigang Chen
Internet Services Management Group
Cisco Systems
San Jose, CA 95051
sgchen@cisco.com

Abstract

Large networks are often structured hierarchically by grouping nodes into different domains in order to deal with the scaling problem. In such networks, it is infeasible to maintain the detailed network information at every router. Therefore, topology information of domains are summarized before broadcasted. This process is called topology aggregation. Hierarchical routing protocols are then used to find a route among the domains. We study several basic problems associated with hierarchical QoS routing, including (1) how to make QoS-aware topology aggregation, (2) how to represent the aggregated network state, and (3) how to find an end-to-end route based on aggregated information. The novelty in this research is our new network QoS representation which is line segments on the delay-bandwidth plane. We also present a distributed routing mechanism that works with our representation. Our theoretical and simulation results show that the protocol achieves scalability and improved routing performance.[1]

1. Introduction

Nowadays, internetworks are global and consist of enormous number of routers and networks. In order to do routing more efficiently, the Internet is partitioned into different *autonomous systems* [14]. Each autonomous system (AS) is a *domain* that consists of routers, networks, and hosts. The router, that connects to an outside AS, is called a *gateway*. Computers or hosts are identified by IP addresses that consist of two parts: the first part identifies the AS and the second part identifies the host inside the particular AS. This kind of addressing is called *hierarchical addressing* and the corresponding routing algorithm is called *hierarchical routing*. The routing algorithm first finds how to reach the gateway of the AS where the destination resides (*inter-domain routing*). Then, a path is found from that gateway to the target node (*intra-domain routing*). As the QoS traffic (voice and video) grows exponentially, there is a need to extend this hierarchical structure to support *QoS routing*, which means to find a network path that has sufficient resources to meet the throughput/real-time requirement of a traffic flow between two end hosts.

QoS routing has been studied separately at the inter-domain level and the intra-domain level. At the inter-domain level, *topology aggregation* is introduced to reduce the amount of information a router needs to maintain so that the scalability can be achieved. The idea is to aggregate each domain into a simple topology, e.g. a star, which captures the cost of going from one end of the domain to another [12], [8]. This simplified topology is broadcasted and is used in finding an inter-domain path. At the intra-domain level, many QoS routing protocols have been proposed recently, and they can be grouped into two major classes: *source routing* and *distributed routing* [3]. In source routing, each node maintains the complete global network state, based on which a routing path is computed locally at the source node. In distributed routing, a path is determined in a distributed manner. Routing messages are exchanged among the nodes so that the network state information maintained at different nodes can

[1]The first two authors were supported by the Airforce grant under contract number F30602-97-2-0121, the National Science Foundation Career grant under contract number NSF CCR 96-23867, and the National Science Foundation PACI grant under contract number PACI E/E QLTY SVC 1-1-13006.

0-7695-0912-6/00 $10.00 © 2000 IEEE

be collectively utilized.

In this paper, we study (1) how to make QoS-aware topology/state aggregation, (2) how to represent the aggregated network state, and (3) how to find an end-to-end route based on such aggregated information. We introduce a new framework for hierarchical QoS routing in delay-bandwidth sensitive networks. In this framework, we use a new representation of link information and discuss the corresponding topology aggregation algorithm and routing algorithm. Theoretical and simulation results show that our algorithm has satisfiable performance in terms of storage, running time and success ratio.

The rest of the paper is organized as follows: Section 2 discusses the related work, Section 3 presents our model, Section 4 describes the topology aggregation algorithm for domains, Section 5 describes the routing algorithm (both inter-domain and intra-domain) that works with our framework, Section 6 presents the performance analysis, and finally, we draw conclusions in Section 7.

2. Related Work

Topology aggregation and QoS routing, have been discussed, mostly independently, by various studies.

A topology aggregation algorithm, that achieves bounded distortion in domains with only one parameter, is studied in [1]. However, the theoretical bound of distortion of networks having two or more parameters is still unknown. There are not many studies addressing topology aggregation of multiple parameters. Some studies can be found in [9], [10], and [11]. Traditional approaches usually represent each logical link as a delay-bandwidth QoS pair using numerical values representing the parameter of the "best" path. However, this approach is obviously not sufficient since if there are several paths between the source and the target, it is difficult to pick the best QoS pair. Moreover, no matter which single pair is picked, parameters of some path are not reflected [10]. In [10], the authors describe some existing approaches in picking the best pair and also suggest instead of having only the numerical values of bandwidth and delay, each logical link keeps an extra parameter which implicitly defines a curve passing through a particular bandwidth-delay pair on a delay-bandwidth plane. A drawback of the curve proposed in [10] is that the curve may be very far away from other parameters.

Although routing of only one parameter can be solved easily by the Dijkstra algorithm, routing of multiple parameters is not easy. The problem of finding a path satisfying more than one *additive* constraint (e.g. delay and cost) is NP-complete [6], [16]. In a bandwidth-delay sensitive network, the problem can be solved by a modified Dijkstra algorithm proposed in [15] and [16]. Most of the existing routing algorithms represent parameters using numerical values alone. In our framework, we use a different kind of representation, which is a line segment on the delay-bandwidth plane. According to our knowledge, this representation has never been studied before and therefore no existing routing algorithm can be used directly.

3. System Models

The whole network is partitioned into disjoint domains. A domain is a set of nodes that are connected by communication links. Some nodes are connected to another domain and these nodes are called *borders* or *gateways*. We model the network and the domains as directed graphs where links can be asymmetric in both directions to make it more flexible. We first discuss the domain model and then the network model.

3.1. Domain Model

A domain is modeled as a tuple (V, B, E), where V is the set of nodes, $B \subset V$ is the set of borders, and E is the set of directed links among the nodes in V. We call those links in E *physical links*. We denote the QoS parameter of each physical link as (D, W), which means the delay of the link is D units and the bandwidth is W units.

A *physical path* from node v_0 to node v_k is denoted as $(v_0 \rightarrow v_1 \rightarrow v_2 ... \rightarrow v_{k-1} \rightarrow v_k)$, where the directed link $(v_i, v_{i+1}) \in E$ for $0 \leq i < k$. Let $(D, W)_{i \rightarrow j}$ be the parameter of physical link (v_i, v_j). Also, let $D_{i \rightarrow j}$ and $W_{i \rightarrow j}$ be the delay and bandwidth of (v_i, v_j) respectively. The delay of the physical path from v_0 to v_k is the sum of the delays of all physical links along the path which is $\sum_{i=0}^{k-1}(D_{i \rightarrow i+1})$. The bandwidth of it is the minimum bandwidth among all bandwidths along the physical path which is $min_{i=0}^{k-1}\{W_{i \rightarrow i+1}\}$.

Let 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). Then, the delay of $v_1 \rightarrow v_2 \rightarrow v_3$ is 3+5+6 = 14 and the bandwidth is $min\{5, 4, 4\}$=4.

3.2. Network Model

A network consists of a set of domains and links that connect them. A network is denoted as (G, L) where $G = \{g_i | g_i = (V_i, B_i, E_i), 1 \leq i \leq |G|\}$. In order to identify a node in the network unambiguously, we use the notation $g_i.v_j$ to refer to the node v_j of domain g_i. L is the set of the links between domains. Each inter-domain link is denoted in the same way as the physical links in a domain.

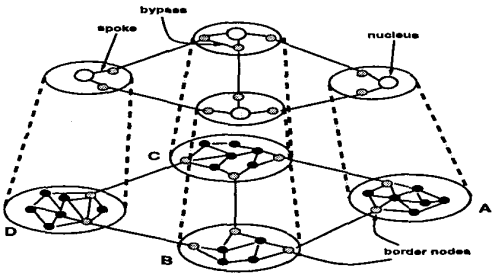


Figure 1. Topology Aggregation

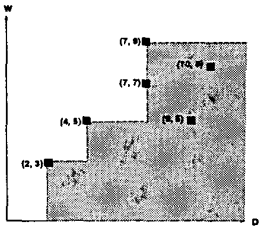


Figure 2. Representatives from Example 1

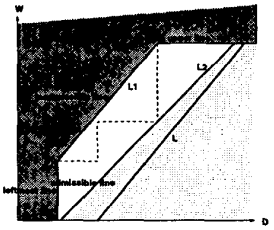


Figure 3. Areas of different admissibility

4. QoS-aware Topology Aggregation

In order to do inter-domain routing more efficiently, each domain is aggregated into a simpler topology. We adopt the *star with bypasses* topology described in PNNI [2]. The topology consists of two kinds of nodes - *border nodes*, those connect to outside domains, and a single virtual *nucleus*. Each border node connects to the nucleus by a link called *spoke*. This is called a *star* topology. Each link has a set of parameters associated with it which shows the QoS of the link. If there are b border nodes, and the spokes are different, the complexity of this representation is $O(b)$ which is usually a lot smaller than the original domain. To make the representation more flexible, we add inter-border links called *bypasses* into the star. We call the links, both spokes and bypasses, in the star network *logical links* since they are not real physical links. The number of bypasses should not be too large. In our algorithm we limit the number of bypasses to be b as suggested in PNNI. Figure 1 presents an example of aggregation.

Ideally, given a pair of border nodes b_1 and b_2, the QoS parameter of going from b_1 to b_2 in the aggregation should be the same as the QoS parameter of the best path in the original domain. Even for a single metric, it is difficult to achieve [1]. There is another issue for networks with multiple metrics: how to pick the best parameter. One path may be the best in terms of delay but the other may be the best if bandwidth is considered. For example, a delay-bandwidth QoS pair $(5, 8)$ is better than another pair $(10, 10)$ in terms of delay but worse than the same pair if bandwidth is considered. Fortunately, partial order does exist among the QoS parameters. $(7, 9)$ is better than $(10, 8)$ in both delay and bandwidth. In order to solve this problem, we use a *line segment* on the delay-bandwidth plane instead of a single delay-bandwidth pair to represent each logical link in the aggregation. There are two phases in our aggregation scheme: (1) find the mesh (complete graph) of the border nodes, (2) find the star with bypasses representation from the mesh. Most existing aggregation mechanisms adopt the same steps. We first discuss an algorithm for finding the mesh and then we describe an algorithm that aggregates the mesh into a star network with bypasses.

4.1. Mesh Formation

The mesh is a complete graph of the border nodes. The parameter of each logical link should ideally be the best parameter among all physical paths going between the two nodes. As mentioned above, there is no absolute order for parameters having two metrics. But, 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 and $x \leq x'$ and $y \geq y'$.*

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

Example 1 *Let S be a set of the delay-bandwidth QoS pairs and S = {(4, 5), (7, 9), (10, 8), (9, 5), (2, 3), (7, 7)}. $(2, 3)$ is a representative of S since its delay is less than all other points in S. Another representatives are $(4, 5)$ and $(7, 9)$.*

If we plot all these ordered pairs on a bandwidth vs. delay Cartesian plane, it is easy to find out all representatives. Interested readers can refer to [13] for the detailed algorithm. Note that the number of representatives are bounded by $|E|$ in the domain (V, B, E) because each path must be constrained by a certain link and there are at most $|E|$ links. On the other hand, the number of representatives is also bounded by the number of different bandwidth values. Therefore, if we divide the bandwidth requirements into several groups, the number of representatives can be further reduced.

In Figure 2, the shaded area defines the region of acceptable services. The dotted line is a staircase rising from

left to right. The representatives are points on the convex corners of the steps. There may be many points on the staircase. Due to storage limitation on the routers, it is favorable to keep only a constant number of points. We suggest using line segments to approximate the staircase.

4.1.1 Partitioning of Delay-Bandwidth Plane

When a QoS request arrives, the routing algorithm must be able to identify whether it can be supported or not. In the case where parameters are numerical values, it can be done by simply comparing the requested parameter with the "best" parameter. However, in a network with multiple metrics, no matter which path parameter is picked to be the best, some supported requests will be rejected. In our representation, in order to identify unsupported and supported requests quickly, we partition the whole delay-bandwidth plane into three areas using two line segments and each logical link in the mesh is represented by these two lines. These three areas reflect the difference in admissibility of the QoS requests:

1. *admissible region* – the region of QoS parameters that are guaranteed to be served
2. *inadmissible region* – the region of QoS parameters that cannot be served for sure
3. *uncertain region* – the region where it is not certain whether QoS parameter can be served or not

Figure 3 shows the division of the delay-bandwidth plane into three non-overlapping areas using two line segments $L1$ and $L2$ of the same staircase as in Figure 2. The lower lighter shaded region is the *admissible region*. It is a subset of the shaded staircase region in Figure 2 which means that any QoS parameter that falls in this region can be served by a particular path. The darker shaded region is the *inadmissible region*. It does not intersect with the staircase and so none of the QoS requirements in that region can be fulfilled by any path. Finally, the white area between the admissible and the inadmissible regions is the *uncertain region*. This region contains the staircase together with the areas that are inside the staircase and outside the staircase. As a result, some of the QoS pairs can be accepted while some others are not. It implies that given a QoS parameter that lies in this region, without knowing the staircase, we are not certain whether it can be served or not.

In the following discussion, we call $L1$, which defines the inadmissible region, the *inadmissible line*, and $L2$ the *admissible line*. Furthermore, we call the highest horizontal line in the staircase *upper line* and the vertical line with the smallest delay *leftmost line*. The staircase can be used to find $L1$ and $L2$. The main difference between the two lines is that the admissible line is below the staircase while the inadmissible line is above the staircase. Many different line segments can represent admissible lines and inadmissible lines. For example, line L can be an admissible line too. However, since we want to minimize the size of the uncertain region, we would like to find the lines that are closest to the staircase. Therefore, for both $L1$ and $L2$, the bandwidth of the upper endpoint should have the same bandwidth as the upper line. On the other hand, the delay of the lower endpoint should not be smaller than the delay of the leftmost line. In our simulations, we used linear regression to find the slopes of the lines. It takes $O(m)$ time to find each line, where m is the number of points on the staircase.

If there is only one representative, the line becomes a single point. Since each line segment can be defined unambiguously by two endpoints, we denote a line segment as *[lower endpoint, upper endpoint]* where the endpoints are denoted using traditional Cartesian coordinates. A single point is denoted as a line where both endpoints are the same.

4.2. Star Formation

For a domain having b border nodes, there are $b * (b - 1)$, which is $O(b^2)$, logical links in the mesh. This is too expensive to be broadcasted. Our next step is to aggregate the mesh into a star network with at most b bypasses so that the total number of links is $O(b)$.

Let i and j be two border nodes in the star representation. If there is no bypass between i and j, the only path going from i to j is $i \rightarrow n \rightarrow j$ in the star, where n is nucleus. Our goal in aggregation is to find out the parameters of links $i \rightarrow n$ and $n \rightarrow j$ such that the delay and bandwidth of $i \rightarrow n \rightarrow j$ in the star is the same as the delay and bandwidth of $i \rightarrow j$ in the mesh. Basically, we have to "split" a logical link $i \rightarrow j$ in the mesh into two logical links $i \rightarrow n$ and $n \rightarrow j$, which are spokes in the star.

Before describing the algorithm for splitting a logical link, we first describe how we find the line segments of $i \rightarrow n \rightarrow j$, given $i \rightarrow n$ and $n \rightarrow j$. The problem is well-defined for numerical parameters like the ones we used in Section 3.1 (D, W), but not obvious for a line segment parameter representation. The *join* operation is denoted as "+" and defined as:

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

We now proceed to discuss the process of finding the spokes and bypasses. The process consists of three steps: (1) Find the spokes from border node i to nucleus (Section 4.2.1); (2) Find the spokes from nucleus to i (Section 4.2.2); (3) Find the bypasses between border nodes (Section 4.2.3).

4.2.1 Finding the spokes incoming to the nucleus

As we discussed in the previous section, we have to "break" line segments to form spokes. Since the join operation adds up the delays and takes the minimum of the bandwidths, each spoke in the star should have a smaller delay and maybe a higher bandwidth than the line segment in the mesh. Our algorithm of finding spokes from border node i to n is based on these observations. Denote ${}^{m}l_{ij}^{ad}$ to be the admissible line segment from i to j in the mesh m and ${}^{s}l_{in}^{ad}$ to be the admissible spoke from i to the nucleus.

Algorithm 1: Admissible spoke from i to n

(1) Given $M^{ad} = \{{}^{m}l_{ij}^{ad} \mid i, j \in B \text{ and } i \neq j\}$, find the set $M^{ad}{}_{i} = \{{}^{m}l_{ij}^{ad} \mid j \in B \text{ and } i \neq j\}$ where ${}^{m}l_{ij}^{ad}$ is the admissible line segment from i to j in the mesh.

(2) For the line segments in $M^{ad}{}_{i}$,

  (a) find out the smallest delay (min_ld) of the lower endpoints

  (b) find out the smallest delay (min_ud) of the higher endpoints

  (c) find out the highest bandwidth (max_lw) of the lower endpoints

  (d) find out the highest bandwidth (max_uw) of the higher endpoints

(3) ${}^{s}l_{in}^{ad} = [(min\_ld, max\_lw), (min\_ud, max\_uw)]$

The algorithm for finding inadmissible spokes is similar except we use the set of inadmissible lines. $O(b)$ time is needed to find one spoke and it takes $O(b^2)$ time to find all spokes incoming to the nucleus.

Example 2 (Illustration of Algorithm 1) *Let b = 3 and the admissible line segments from node 0 to border nodes 1 and 2 are [(9, 4), (19, 6)] and [(6, 4), (6, 4)](Figure 4). The endpoints of the line segment from node 0 to node 2 are the same. It means that there is only one representative among all paths that goes from node 0 to node 2. min_ld = min_ud = 6, and max_lw = 4 and max_uw = 6. Therefore, the admissible line segment from 0 to nucleus is [(6, 4), (6, 6)].*

4.2.2 Finding the spokes outgoing from the nucleus

We now proceed to find the spokes from the nucleus to the borders. For admissible lines, we know M^{ad} as well as ${}^{s}l_{in}^{ad}$, and we want to find ${}^{s}l_{nj}^{ad}$ such that the result of joining ${}^{s}l_{in}^{ad}$ and ${}^{s}l_{nj}^{ad}$ is exactly ${}^{m}l_{ij}^{ad}$. We now define a *disjoint* ("$-$") function such that ${}^{m}l_{ij}^{ad}$ - ${}^{s}l_{in}^{ad}$ would be the desirable ${}^{s}l_{nj}^{ad}$. (The same holds for inadmissible lines too.)

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

If we apply the formula to different j for the same i, we may have several different ${}^{s}l_{nj}^{ad}$. How we pick ${}^{s}l_{nj}^{ad}$ depends on whether it is an admissible spoke or an inadmissible one. We first look at the situation for admissible lines. After the aggregation, we want ${}^{s}l_{in}^{ad}$ + ${}^{s}l_{nj}^{ad}$ to be an admissible line too. Since ${}^{s}l_{in}^{ad}$ + ${}^{s}l_{nj}^{ad}$ may not be ${}^{m}l_{ij}^{ad}$, we have to find an alternative line l such as ${}^{s}l_{in}^{ad}$ + ${}^{s}l_{nj}^{ad}$ = l and l is defining a correct admissible region for the paths from i to j. Lemma 1 gives the properties of l. Let $l.up$ and $l.lp$ be the upper endpoint and lower endpoint of l respectively. Denote the delay and bandwidth of a point p as $p.d$ and $p.w$. Thus, the delay of the upper endpoint of a line l is denoted as $l.up.d$.

Lemma 1 *Let l^{ad} be an admissible line defining an admissible region R on the delay-bandwidth plane. Any line l such that $l.lp.d \geq l^{ad}.lp.d$, $l.up.d \geq l^{ad}.up.d$, $l.lp.w \leq l^{ad}.lp.w$, and $l.up.w \leq l^{ad}.up.w$ defines an admissible region R' where $R' \subseteq R$ (Figure 5).*

Due to space limitation, the proof of Lemma 1 is skipped. The following algorithm shows how to find the admissible spokes from the nucleus to the border nodes. It is devised from Lemma 1. The total time for finding all spokes outgoing from n is also $O(b^2)$ due to the same reason as Algorithm 1.

Algorithm 2: Admissible spoke from n to i

(1) Given M^{ad} and $\{{}^{s}l_{jn}^{ad} \mid j \in B \text{ and } i \neq j\}$, find the set $M^{ad}{}_{i}^{-} = \{{}^{m}l_{ji}^{ad} - {}^{s}l_{jn}^{ad} \mid j \in B \text{ and } i \neq j\}$.

(2) For the line segments in $M^{ad}{}_{i}^{-}$,

  (a) find out the largest delay (max_ld) of the lower endpoints

  (b) find out the largest delay (max_ud) of the upper endpoints

  (c) find out the smallest bandwidth (min_lw) of the lower endpoints

  (d) find out the smallest bandwidth (min_uw) of the higher endpoints

(3) ${}^{s}l_{ni}^{ad} = [(max\_ld, min\_lw), (max\_ud, min\_uw)]$.

Example 3 (Illustration of Algorithm 2) *Refer to the line segments in Example 2, ${}^{m}l_{02}^{ad}$ - ${}^{s}l_{0n}^{ad}$ = [(0, 4), (0, 4)]. Suppose ${}^{m}l_{12}^{ad}$ - ${}^{s}l_{1n}^{ad}$ = [(0, 5), (3, 7)]. max_ld = 0, max_ud = 3, min_lw = 4, and min_uw = 4. Therefore, the admissible line segment from nucleus to 2 is [(0, 4), (3, 4)]. The admissible line segments from nucleus to 0 is [(1, 4), (0, 6)] and nucleus to 1 is [(3, 4), (13, 6)].*

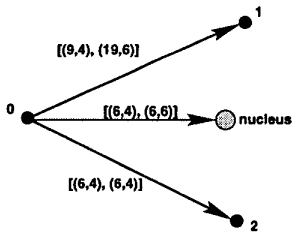


Figure 4. Example 2

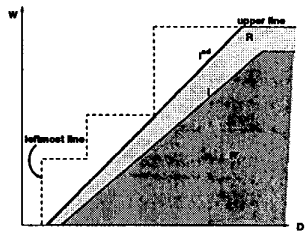


Figure 5. Lemma 1

For the inadmissible spokes, instead of finding largest delays and smallest bandwidths as in Algorithm 2, smallest delays and largest bandwidths are used to form the spokes according to the following lemma.

Lemma 2 *Let l^{inad} be an inadmissible line defining an inadmissible region R on the delay-bandwidth plane. Any line l such that $l.lp.d \leq l^{inad}.lp.d$, $l.up.d \leq l^{inad}.up.d$, $l.lp.w \geq l^{inad}.lp.w$, and $l.up.w \geq l^{inad}.up.w$ defines an inadmissible region R' where $R' \subseteq R$.*

4.2.3 Finding Bypasses

Obviously, due to the aggregation, ${}^{s}l_{in}^{ad} + {}^{s}l_{nj}^{ad}$ may be different from ${}^{m}l_{ij}^{ad}$. In order to make the aggregation more precise, bypasses are introduced. Intuitively, bypasses should be put in the border nodes where ${}^{s}l_{in}^{ad} + {}^{s}l_{nj}^{ad}$ deviates a lot from ${}^{m}l_{ij}^{ad}$. Therefore, we need a quantitative measure of *deviations*. The actual deviation of ${}^{s}l_{in}^{ad} + {}^{s}l_{nj}^{ad}$ from ${}^{m}l_{ij}^{ad}$ is the difference in the areas covered by the two line segments. However, since the delay-bandwidth plane is unbounded, we cannot find the actual areas. Since the distance between endpoints is directly related to the area bounded by line segments, and is easy to compute, we use this as the mean of measurement. After finding the deviation of each border pair, the algorithm puts bypasses between those border pairs that have b largest deviations. A more detailed explanation can be found in [13].

5. Routing

IP networks are represented by a two-level hierarchical structure. Therefore, there are two levels of routing: *inter-domain routing* and *intra-domain routing* as mentioned in Section 1. For example, a connection request from node $A.1$ (node 1 of domain A) to $D.2$ (node 2 of domain D) requires an inter-domain path from A to D and an intra-domain path from the border node of D to node 2. There may be some other domains which lie in the inter-domain path. In passing a domain, we go from one border node to another. This is also an intra-domain path. In the following, we first describe the general idea of our hierarchical routing algorithm and then discuss how to find the inter- and intra-domain paths.

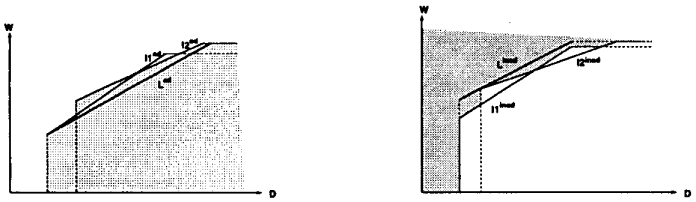
5.1. Overview

The hierarchical algorithm is based on the *ticket-based probing* in [4]. It is a distributed routing process. Different from many existing routing algorithms for multiple metrics, users do not have to specify precedence in QoS parameters. A *ticket* represents the permission of searching one path. Only the source node can issue tickets based on its own state information and more tickets are generated for requests of tighter requirements. *Probes* carrying one or more tickets are sent from the source towards the destination to search for a feasible path. When an intermediate node receives a probe, it decides how to forward the tickets in the probe received to its own neighbors. Since each probe carries at least one ticket, number of probes sent to neighbors is restricted by the number of tickets. Finally, if a probe successfully arrives at the destination, then a path is found.

5.2. State Information

Most distributed routing algorithms require each node in the network to keep a *distance table* showing the least cost or least delay from that node to every destination. In our algorithm, the entry of the distance table is a pair of admissible line and inadmissible line to that destination. There are two tables: *intra-domain table* and *inter-domain table*. As the names imply, the intra-domain table stores the line segments to every node within the same domain and inter-domain table keeps the line segments to each outside domain.

The intra-domain table can be obtained by applying the distance-vector or link-state protocols. In these protocols, comparisons of two parameters are necessary. As our parameters are line segments, not numerical values, we need another mechanism in comparing. Figure 6 illustrates the idea. In (a), suppose $l1^{ad}$ and $l2^{ad}$ are the admissible lines of two different paths. The shaded area is the "combined" admissible region which is the union of the two regions defined by $l1^{ad}$ and $l2^{ad}$. Line L^{ad} that falls in the union



(a) Admissible Line (b) Inadmissible Line

Figure 6. Illustration of line picking

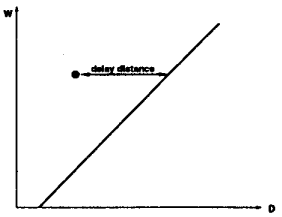


Figure 7. Delay Distance

region can be used to be the admissible line. The situation of inadmissible lines is shown in Figure 6(b). The shaded area is the intersection of the inadmissible regions defined by $l1^{inad}$ and $l2^{inad}$ in the figure and L^{inad} can be used to represent the region. Both lines should be able to be obtained in $O(1)$ time since there are at most 4 points to consider. Therefore, the total running time of obtaining the line segments should be the same as when only one metric is considered.

Unlike the intra-domain table, the computation of inter-domain table is not straight-forward. Since the star aggregations are broadcasted to border nodes only, an internal node alone does not have enough information to compute the line segments to an outside domain. On the contrary, as the border nodes collect the star aggregations of all outside domains, they can fill out their own inter-domain tables easily. The inter-domain tables of border nodes are then broadcasted to the nodes inside the same domain. Let the admissible line from an internal node $g.s$ to a border node $g.bd$ be $l^{ad}_{g.s \to g.bd}$. Also, let the admissible line from $g.bd$ to an outside domain g' be $l^{ad}_{g.bd \to g'}$. The admissible line of the path going from $g.s$ to g' through $g.bd$ is $l^{ad}_{g.s \to g.bd} + l^{ad}_{g.bd \to g'}$. By comparing and evaluating the formula for each border bd, $g.s$ can fill out the entry of the inter-domain table for domain g'.

5.3. Ticket-based Probing

We now describe the idea of our routing algorithm. We will explain how to find the inter- and intra- domain routes in the next section. Suppose there is a QoS request of parameter (D_{req}, W_{req}) going from source node s to target node t. Let the admissible line and the inadmissible line from s to t be $l^{ad}_{s \to t}$ and $l^{inad}_{s \to t}$ respectively.

5.3.1 Ticket Generation

A ticket represents the permission of searching one path. Tickets can be generated by the source node only. Since tickets can never be created in the process of forwarding, if a node receives more tickets, it can forward them to more neighbors, and more paths are searched. Let the number of tickets at source s be N_0. N_0 can be determined by (D_{req}, W_{req}), $l^{ad}_{s \to t}$ and $l^{inad}_{s \to t}$ in $O(1)$ time.

1. If (D_{req}, W_{req}) falls in the inadmissible region defined by $l^{inad}_{s \to t}$, the request should be rejected since no path can support the QoS service. $N_0 = 0$ then.

2. If (D_{req}, W_{req}) falls in the admissible region defined by $l^{ad}_{s \to t}$, there should be a feasible path and one ticket is sufficient. Therefore, $N_0 = 1$.

3. If (D_{req}, W_{req}) falls in the uncertain region, N_0 depends on the distance between (D_{req}, W_{req}) and the lines. Intuitively, the closer the point is to the admissible line, the easier it is to find a path and so less tickets are needed. On the contrary, if the point lies very close to the inadmissible line, we need more tickets so that more paths are searched. We use the *delay distance*, as shown in Figure 7 as the mean of measurement. Let d_{ad} be the delay distance between (D_{req}, W_{req}) and $l^{ad}_{s \to t}$, and d_{inad} be the delay distance between (D_{req}, W_{req}) and $l^{inad}_{s \to t}$. Let Φ be the maximum allowable number of tickets in the system. Then $N_0 = \lceil \frac{d_{ad}}{d_{ad}+d_{inad}} \times \Phi \rceil$. N_0 is larger when (D_{req}, W_{req}) is closer to $l^{inad}_{s \to t}$.

5.3.2 Ticket Forwarding Protocol

If $N_0 > 0$, s generates one or more probes. These probes will carry tickets and will be sent to t. To each neighbor, a node can send at most one probe. Each probe stores the delay and the bandwidth of the path that it has gone through. Each probe p carries the information (D_p, W_p), where D_p is the sum of the delays of all links it traverses so far and W_p is the minimum bandwidth. When a node receives a probe p, it uses (D_p, W_p), the admissible and the inadmissible lines of its neighbor to determine how to forward the tickets to its neighbor. There are two issues: which neighbors to forward and how many tickets to forward.

A node i forwards a probe p to a neighbor j only if it is possible to find a feasible path from j to the destination. We call such a neighbor *candidate neighbor*. The bandwidth of a feasible path must be at least W_{req} and

the delay of a feasible path should not be greater than D_{req}. The delay of the path from $s \rightarrow i \rightarrow j \rightarrow t$ is $D_p + D_{i\rightarrow j} + D_{j\rightarrow t}$. $D_p + D_{i\rightarrow j} + D_{j\rightarrow t} \leq D_{req}$ implies $D_{j\rightarrow t} \leq D_{req} - D_p - D_{i\rightarrow j}$. On the other hand, the bandwidth of the path is $min\{W_p, W_{i\rightarrow j}, W_{j\rightarrow t}\}$. $min\{W_p, W_{i\rightarrow j}, W_{j\rightarrow t}\} \geq W_{req}$ implies $W_{i\rightarrow j} \geq W_{req}$ and $W_{j\rightarrow t} \geq W_{req}$. Therefore, a neighbor j is a candidate neighbor if (1) $W_{i\rightarrow j} \geq W_{req}$, and (2) $(D_{j\rightarrow t}, W_{j\rightarrow t})$ is better than or the same as $(D_{req} - D_p - D_{i\rightarrow j}, W_{req})$, i.e., the point does not lie in the inadmissible region defined by $l^{inad}_{j\rightarrow t}$.

If there is no candidate neighbor, it means that it is impossible to go to the target from the current node. All the tickets should be dropped. If some candidate neighbors are found, let $(D_{req} - D_p - D_{i\rightarrow j}, W_{req})$ be (D_{cond}, W_{cond}), there are two cases: (1) (D_{cond}, W_{cond}) lies in the admissible region of one of the neighbors, and (2) (D_{cond}, W_{cond}) lies in the uncertain regions of all neighbors. Case (1) means that a feasible path can be found definitely. Therefore, one ticket is needed to be forwarded to that neighbor and all other tickets can be dropped without forwarding to others. For case (2), in order to increase the probability of finding a feasible path, the closer (D_{cond}, W_{cond}) to $l^{ad}_{j\rightarrow t}$, the more tickets j should receive. Let d^j_{ad} and d^j_{inad} be the delay distances from (D_{cond}, W_{cond}) to $l^{ad}_{j\rightarrow t}$ and $l^{inad}_{j\rightarrow t}$ respectively. Let the number of tickets that i receives be N_i. The number of tickets that should be forwarded to a node j in candidate neighbor set R is $\frac{(d^j_{ad}+d^j_{inad})/d^j_{ad}}{\sum_{j'\in R}(d^{j'}_{ad}+d^{j'}_{inad})/d^{j'}_{ad}} \times N_i$. The function is inversely proportional to d^j_{ad}. The total running time of calculating tickets is linear to the number of neighbors.

5.4. Inter- and Intra- Domain Routings

An inter-domain route specifies which domains to traverse in order to go from the source domain to the target domain. In other words, an inter-domain route identifies which border nodes to go through. When a border node b in domain g receives a probe p from other domain, the probe must leave g through another border node b' in g if the destination of the probe is $t \neq g$. b' can be found by applying the technique of finding candidate neighbor in the ticket forwarding protocol. The admissible line from $g.b$ to t through $g.b'$ is $l^{ad}_{g.b\rightarrow g.b'} + l^{ad}_{g.b'\rightarrow t}$. $l^{ad}_{g.b\rightarrow g.b'}$ can be obtained from the intra-domain table of $g.b$ while $l^{ad}_{g.b'\rightarrow t}$ can be obtained from the inter-domain table of $g.b'$ (Section 5.2). By checking whether $(D_{req} - D_p, W_{req})$ falls in the inadmissible region defined by $l^{inad}_{g.b\rightarrow g.b'} + l^{inad}_{g.b'\rightarrow t}$, we can identify whether b' is a *candidate border*. The number of tickets to be forwarded to each candidate border can be evaluated in a similar fashion as described in Section 5.3.2. After a candidate neighbor, say b', is found, the routing problem from $g.b$ to $g.b'$ is an intra-domain one and the mechanism described in Section 5.3.2 can be applied directly.

6. Performance Analysis

In this section, we are going to analyse the storage and the running time our algorithm needs and present our simulation results. In the following, we assume the network has n domains. Domain i, g_i, is a graph (V_i, B_i, E_i) where V_i is the set of nodes, B_i is the set of borders and E_i is the set of edges.

The main information in each node is stored in two tables. Number of entries in the intra-domain table is $|V_i| - 1$ and number of entries in the inter-domain table is $n - 1$. The storage for the whole domain g_i is $O(|V_i| * (|V_i| + n))$. The total storage needed for the whole network is $O(\sum_{i=1}^{n} |V_i| * (|V_i| + n))$. If n is in the order of $|V_i|$, it is reduced to $O(\sum_{i=1}^{n} |V_i|^2)$. If hierarchical structure was not used, storage in each node would be $O(\sum_{i=1}^{n} |V_i|)$ and the total storage would be $O((\sum_{i=1}^{n} |V_i|)^2)$, which is a lot more expensive than $O(\sum_{i=1}^{n} |V_i|^2)$.

Table 1. Running Time of Aggregation

finding all the representatives between two borders	$O(\|V_i\|\|E_i\|^2)$
number of representatives on a staircase	$O(\|E_i\|)$
finding the line segments for a staircase	$O(\|E_i\|)$
total time for finding a logical link in the mesh	$O(\|V_i\|\|E_i\|^2)$ + $O(\|E_i\|) + O(\|E_i\|)$ = $O(\|V_i\|\|E_i\|^2)$
time for finding all the logical links in the mesh	$O(\|B_i\|^2\|V_i\|\|E_i\|^2)$
time to find all the spokes	$O(\|B_i\|^2)$
time to find all the bypasses	$O(\|B_i\|^3)$
Total Time in Aggregating g_i	$O(\|B_i\|^2\|V_i\|\|E_i\|^2)$ + $O(\|B_i\|^2) + O(\|B_i\|^3)$ = $O(\|B_i\|^2\|V_i\|\|E_i\|^2)$

Table 1 summarizes the worst case running time of aggregating domain g_i in each step. It is obvious from the table that the most expensive step is finding the representatives by running the modified Bellman-Ford algorithm. Therefore, our aggregation mechanism takes at most the same time as the algorithm proposed in [10].

In our simulation, we compare our algorithm (TBP) with the flooding (FD) and the shortest path (SP) algorithms. In the following discussion, we call a path that supports a request a *feasible path* and a request that can be supported by a certain path a *feasible request*. Furthermore, we call a request *accepted request* of a certain algorithm if that algorithm is able to find a feasible path for the request. Obviously, an accepted request is always a feasible request but

a feasible request may not be an accepted request since the algorithm may be unable to find a feasible path for the request. In the flooding algorithm, every node sends out routing messages to all neighbors as long as the accumulated delay is less than the required one and the bandwidth requirement is satisfied. Therefore, the flooding algorithm always finds a feasible path if one exists. However, it generates enormous amounts of messages. The shortest path algorithm, that we use in the simulation, is a centralized modified Dijkstra algorithm. The source, which has the centralized topology, first finds out an inter-domain path that traverses the least number of domains. Then, within each domain, a path that goes through the least number of physical nodes is found using the centralized information. If the parameter of this shortest path can satisfy the request, a feasible path is found. Due to the centralized nature of SP, no routing message is generated, but, feasible requests may be rejected. Our algorithm is a compromise between the two: it generates reasonable number of routing messages and finds more feasible paths than SP can do.

The simulated network topology testbed consists of 10 domains, each has 10 to 30 nodes, having a total of over 180 nodes. The number of borders varies from 3 to 5. All the nodes are connected by directed links and each node is connected to at least 3 other nodes in the same domain. The domains are connected by 40 inter-domain directed links. The delay of each link is between $2ms$ to $10ms$ and the bandwidth is in the range of 5 $kByte/s$ to 10 $kByte/s$. Routing requests are generated randomly and only inter-domain requests are studied. We measure the *sucess ratios* and *numbers of routing messages* generated of the algorithms. We measure the success ratio using the formula $\frac{\text{number of accepted requests}}{\text{total number of requests}}$. For routing messages, we count the messages that each node sends to its neighbors in order to establish an end-to-end delay-bandwidth constrained route. In TBP, each message is a probe that carries one or more tickets and other information mentioned in Section 5.

Figures 8 and 9 show the success ratios of the three algorithms w.r.t. delay and bandwidth respectively. The curves of all three algorithms in Figure 8 are rising because a request is easier to be accepted/feasible when the required delay is less restricted. It can be seen that TBP performs better than SP, especially when the delay requirement is tight. In fact, the success ratios of TBP and FD are the same when the acceptable delay is very small. Although SP performs much better when requested delay becomes larger, TBP still has a higher success ratio than SP. When we consider bandwidth as in Figure 9, TBP again performs better than SP. The performance of TBP in tight bandwidth requirement seems to be worse than when the delay requirement is tight. It is because we use delay distance in calculating how many tickets to generate and forward (Section 5.3.1). Hence, by

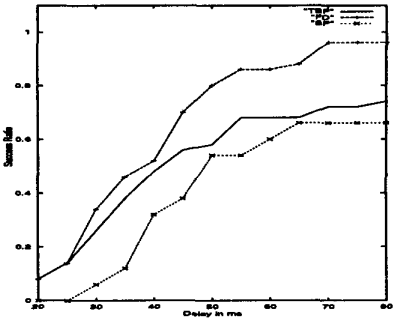


Figure 8. Success Ratio w.r.t. delay

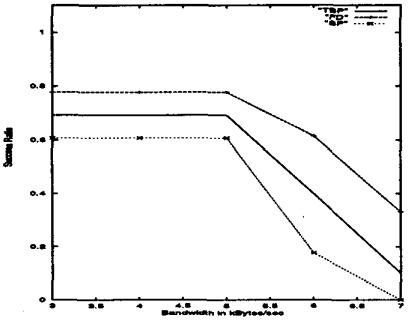


Figure 9. Success Ratio w.r.t. bandwidth

using delay distance, TBP becomes more sensitive to delay requirement.

Figures 10 and 11 show the numbers of probes (messages) TBP generates for the requests. Intuitively, the tighter the requirement, the more probes we need. However, when the requests are tighter, a larger portion of them become infeasible. Since TBP would reject some of the infeasible requests in Case (1) of ticket generation, *no* probe is generated for those requests. It reduces the average numbers of probes needed then. In fact, the rejection mechanism of TBP effectively reduces the number of routing messages. In the simulation, at least 60% of infeasible requests are rejected without generating any probe for all values of delay and bandwidth. For FD, since there is no rejection mechanism as in TBP, routing messages are generated for all requests, no matter it is feasible or not. In our simulation, due to the size of our network, FD generates more than 100 messages for each routing request while the average number of probes TBP needs is only between 2 to 12.

Overall, our theoretical and simulation results both show that our hierarchical QoS routing framework has improved performance in terms of storage, running time, and routing performance when compared with algorithms such as QoS-aware flooding and Dijkstra algorithms.

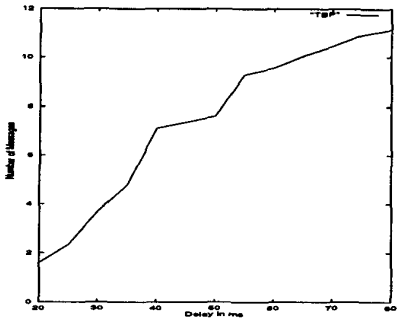


Figure 10. Number of Messages w.r.t. delay

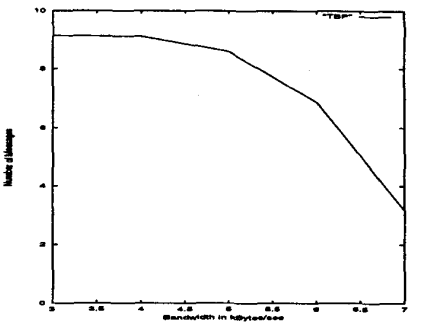


Figure 11. Number of Messages w.r.t. bandwidth

7. Conclusion

In this paper, we present a new framework for QoS hierarchical routing in delay-bandwidth sensitive network. We introduced a new parameter representation that can represent parameters with both additive and attributive metrics. Based on the new parameter representation, we device a new topology aggregation algorithm. We show in detail how to obtain the aggregation and present a distributed hierarchical routing algorithm that works with our representation. Both theoretical and simulation results show that the new algorithm yields improved performance when compared with the flooding and the shortest path algorithms. Our future direction is to investigate the performance of the routing algorithm when imprecision exists and compare the performance with other existing algorithms. Furthermore, we will explore the possibility of applying the technique to the NP-complete routing problem of two additive metrics.

References

[1] B. Awerbuch and Y. Shavitt. Topology aggregation for directed graphs. Technical Report 98-14, DIMACS, Feb. 1998. Also, in In *IEEE Proceedings of the ISCC' 98*, p. 47-52, 1998.

[2] The ATM Forum. Private network-to-network interface specification version 1.0 (pnni 1.0), March 1996. f-pnni-0055.000.

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

[4] S. Chen and K. Nahrstedt. Distributed QoS Routing with Imprecise State Information. *International Conference of on Computer, Communications and Networks, ICCCN'98*, Oct. 1998.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest. *Introduction to Algorithms, the MIT Press.*

[6] M. R. Garey and D.S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-completeness*, W.H. Freeman and Co., San Francisco, CA, 1979.

[7] L. Guo and I. Matta. On state aggregation for scalable QoS routing. In *IEEE Proceedings of the ATM Workshop'98*, p. 306-314, May 1998.

[8] Fang Hao, Ellen Zegura. On Scalable QoS Routing: Performance Evaluation of Topology Aggregation. In *IEEE Proceedings of the INFOCOM'00*, Mar. 2000.

[9] A. Iwata, H. Suzuki, R. Izmailow, and B. Sengupta. QoS Aggregation Algorithms in Hierarchical ATM Networks. In *IEEE Proceedings of the ICC'98*, p.243-148, 1998.

[10] T. Korkmaz and M. Krunz. Source-oriented topology aggregation with multiple QoS parameters in hierarchical ATM networks. In *IEEE Proceedings of the IWQoS'99*, p. 137-146, 1999.

[11] W. C. Lee. Spanning tree method for link state aggregation in large communication networks. In *IEEE Proceedings of the INFOCOM'95*, p. 297-302, 1995.

[12] W. C. Lee. Topology aggregation for hierarchical routing in ATM networks. In *ACM SIGCOMM'95, Computer Communications Review*, p. 82-92, 1995.

[13] K. Lui and K. Nahrstedt. Topology aggregation of Bandwidth-Delay sensitive Networks. Technical Report, Department of Computer Science, UIUC, Sept. 1999.

[14] E.Rosen. *Exterior gateway protocol (EGP).* RFC 896, Network Information Center, SRI Int., Menlo Park CA, Oct. 1982.

[15] Z. Wang and J. Crowcroft. Bandwidth-Delay Based Routing Algorithms. In *IEEE Proceedings of GLOBECOM'95*, vol. 3, p.2129-33, 1995.

[16] Z. Wang and J. Crowcroft. Quality-of-service routing for supporting multimedia applications. In *IEEE Journal on Selected Areas in Communications* , vol.14, no.7, Sept. 1996.