Topology Aggregation and Routing in Bandwidth-Delay Sensitive Networks

King-Shan Lui, Klara Nahrstedt
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
{kinglui, klara}@cs.uiuc.edu

Abstract - Large networks are often structured hierarchically by grouping nodes into different domains in order to deal with the scaling problem. The internal topologies of the domains are aggregated before broadcasted and this process is called topology aggregation. We propose a new method of aggregating networks that are delay-bandwidth sensitive. Traditional approaches represent each logical link as a delay-bandwidth pair which is basically a point on a delay-bandwidth plane. We introduce a new QoS parameter representation and present an aggregation algorithm with corresponding routing protocol. Our simulation results show that the algorithm has very good performance in terms of success ratio and crankback ratio.  

I. INTRODUCTION

As the network becomes larger and larger, it is impossible to broadcast the whole topology to every node in the network to do routing, because it takes enormous amount of bandwidth, time and space. One approach to reduce the bandwidth, time and space complexity is topology aggregation [5]. In the ATM Forum, PNNI [2] standard proposed a possible topology aggregation protocol. In this protocol, nodes are grouped hierarchically into several clusters called peer groups.

Although PNNI defines how the aggregated peer group looks like, it does not specify how to do the aggregation. Some aggregation schemes have been proposed. All aggregation algorithms suffer from distortion which means the resulted aggregation topology deviates from the original one. [1] describes how to aggregate a peer group of one metric with bounded distortion. 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 [6], [7], and [8]. Traditional approaches usually represent each logical link as a delay-bandwidth QoS pair using numerical values. However, if there are several paths between the source and the target, it is difficult to pick the best QoS pair. [7] describes some existing approaches in picking the “best” pair. Though, it is very obvious that using a single pair, which is a point on the delay-bandwidth plane, is not sufficient to reflect the parameters of various paths. [7] suggests 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 [7] is that the curve may be very far away from other parameters.

In this paper, we propose a new algorithm for topology aggregation that aggregates a peer group with two metrics - delay and bandwidth. The topology aggregation is accompanied by a corresponding QoS routing protocol. Our simulation results show that the algorithm has very good performance in terms of success ratio and crankback ratio.

The rest of the paper is organized as follows: Section II discusses the network model. Sections III and IV describe the topology aggregation and routing algorithm. Our simulation results are presented in Section V. Then, we conclude in Section VI.

II. NETWORK MODELS

The whole network is partitioned into disjoint peer groups. A peer group (PG) is a set of nodes that are connected by communication links. We model the network and the domains as directed graphs where links can be asymmetric in both directions to make it more flexible. Nodes within the same PG can see each other, but nodes outside the PG obtain only the aggregated topology. Within a PG, some nodes connect to nodes outside and they are called border nodes or simply borders. One of the representative topologies in PNNI is the star. The border nodes connect via links to a virtual nucleus generating the symmetric star. These links are called spokes. Each link is associated with a set of quality of service (QoS) parameters. A star is asymmetrical if at least one link
is different from the others. To make the representation more flexible, PNNI also allows a limited number of inter-border links called bypasses. We call the links in the star network logical links since they are not real links.

Formally, a PG 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. Each pair \((D, W)\) represents a single point on the delay-bandwidth plane. The delay of a path is the sum of the delays of the physical links along the path. The bandwidth of a path is the minimum of the bandwidths among the links. The parameter of a physical path is also a point on the delay-bandwidth plane. Hence, if there are \(m\) physical paths between two border nodes, there will be \(m\) points on the delay-bandwidth plane of the logical link between the two border nodes.

A network consists of a set of PG's and links that connect them. Fig. 1(a) is an example of a network of four domains. A network is denoted as \((G, L)\), where \(G = \{g_i|g_i = (V_i, B_i, E_i), 1 \leq i \leq |G|\}\). \(L\) is the set of the links between PG's. Each inter-PG link is denoted in the same way as the physical links in a PG.

After aggregation, a node in domain A sees all other nodes in the same PG but only aggregated topologies of other PG's. Fig. 1(b) shows the view of nodes in PG A. The topology of PG A is exactly the same as the original network but the topologies of other PG's are now represented by the border nodes and the star spokes (without bypasses in this example).

III. QoS-AWARE TOPOLOGY AGGREGATION

Our topology aggregation algorithm is based on the following basic principle: instead of using \(m\) points per logical link on the delay-bandwidth plane, we will use a line segment to approximate the \(m\) points and strongly decrease the storage space for QoS parameters. There are two phases in our topology aggregation algorithm: (1) find a line segment for each logical link in the mesh (complete graph) of the border nodes, (2) find the star with bypasses aggregation from the mesh of border nodes.

The mesh of border nodes is a complete graph with links between each pair of nodes. The parameter of a logical link is the best QoS parameter among all physical paths between two nodes of the logical link. It is clear that if we have two metrics per link, there exists no absolute order; however, 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 \(x < x'\) and \(y \geq y'\). Given a set \((S)\) of points in the delay-bandwidth plane, \((x, y) \in S\) is a representative of \(S\) if 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 representative is \((7, 9)\).

If we plot all points in \(S\) on a delay-bandwidth plane, it is easy to find out all representatives. In Fig. 2, the shaded area defines the region of supported services, that is, any request that falls in that region can be supported by a certain 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 algorithm for finding the representatives is presented in [10] and it takes \(O(|S| \cdot \log |S|)\) time. Due to storage limitation on the routers, it is too expensive to save every point for each logical link. In the worst case, there are \(|E|\) points on the staircase. Hence, it is desirable to reduce the storage space per logical link to the \(O(1)\) complexity. An obvious way to solve this problem is to keep only one point per logical link. However, no matter which point we pick, much information is lost. We solve this problem by using a line segment per logical link that approximates the staircase. Since every line segment is defined unambiguously using two endpoints, the total space we need for the mesh is \(O(b^2)\) points.

The line segment can be found using the least square method which takes linear time. After a line segment is found, we could accept all connection requests which fall under the line segment. However, not all of them can be supported. For example, in Fig. 2, if \(L_1\) is selected to approximate the staircase, then the unshaded areas below \(L_1\) represent connection requests that are accepted but not supported by any physical path. We have crankback in this case. On the other hand, we may reject supported QoS by using a line segment. For example, if the connection request is in the shaded region above the line in Fig. 2, it is rejected although it can be served. Therefore, the choice of line segment depends on the desired quality of the service. For instance, in Fig. 2, both \(L_1\) and \(L_2\) are possible line segments. Using \(L_2\) would probably reject more supported connection requests than \(L_1\) while using \(L_1\) would have more crankbacks than \(L_2\).

We call the endpoint with smaller bandwidth lower endpoint while the other one upper endpoint. We then denote a line segment as \([\text{lower endpoint}, \text{upper endpoint}]\). For example, \(L_2\) in Fig. 2 is \([2, 1.5], [9, 9]\). Furthermore, we use the notation \(l_{ij}\) to denote the line segment from node \(i\) to \(j\).

A mesh which consists of \(b \times (b - 1)\) logical links is still too expensive to be broadcasted. Our next step is to aggregate the mesh into a star network with bypasses.
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 → n → j in the star. Needless to say, our goal in the aggregation is to find out the QoS parameters of links i → n and n → j such that the delay and bandwidth of i → n → j in the star is the same as the delay and bandwidth of i → j in the mesh. Basically, we have to "split" a single link i → j in the mesh into two links i → n and n → j in the star. Before describing the algorithm for splitting a logical link, we first describe the operation join (+) to find the QoS parameters of i → n → j, given i → n and n → j. The problem is well-defined for numerical point parameters representation (D, W), but not obvious for a line segment parameter representation.

Definition 2 \[ [(a, b), (c, d)] + [(a', b'), (c', d')] = [(a + a', \min(b, b')), (c + c', \min(d, d'))] \]

We now proceed to discuss the process of finding the spokes and bypasses. The process consists of three steps: (1) Find the spokes from the border nodes to nucleus (Section A), (2) Find the spokes from the nucleus to the border nodes (Section B), (3) Find the bypasses between border nodes (Section C).

A. Spokes incoming to the nucleus

In order to distinguish line segments in the mesh and in the star representation, we denote the line segment from i to j in the mesh as \( l_{ij} \) and the star as \( l_{ij}^{*} \) and \( l_{ij}^{+} \) respectively.

In finding spokes, we have to "break" line segments in the mesh. From the definition of join, we have a general idea how the "broken" line segments look like. The endpoint delays of the spokes \( l_{in} \) and \( l_{nj} \) should be smaller than those of \( l_{ij}^{*} \), while the endpoint bandwidths of the spokes should not be smaller than those of \( l_{ij}^{*} \). Our algorithm of finding spokes from border node i to nucleus (n) is based on these observations. Denote the lower endpoint and upper endpoint of the line segment i to be \( l_{ip} \) and \( l_{up} \). Given a point p, let the delay and bandwidth be \( p.d \) and \( p.w \) respectively. \( l_{in}^{*} = [(\min ld, \max lw), (\min ud, \max uw)] \)

where \( \min ld = \min_{j \in B,i \neq j} \{l_{ij}^{+}.lp.d\}, \min ud = \min_{j \in B,i \neq j} \{l_{ij}^{+}.up.d\}, \min lw = \min_{j \in B,i \neq j} \{l_{ij}^{+}.lp.w\}, \) and \( \min uw = \min_{j \in B,i \neq j} \{l_{ij}^{+}.up.w\} \). 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 2 Suppose the line segments from node 0 to nodes 1 and 2 are \([9, 4], (19, 6)\) and \([3, 7], (3, 7)\) respectively. \( \min ld = \min ud = 3 \) and \( \max lw = \max uw = 7 \). Therefore, the line segment from 0 to nucleus is \([3, 7], (3, 7)\).

B. 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 (M) and we know the spokes from borders to nucleus (Let the set of those spokes be \( S_{b \rightarrow n} \)). More specifically, we know \( l_{in}^{*} \) as well as \( l_{nj}^{*} \), and we want to find \( l_{nj}^{+} \) such that the result of joining \( l_{nj}^{*} \) and \( l_{nj}^{+} \) is exactly \( l_{nj}^{*} \). We have \( l_{in}^{*} + l_{nj}^{*} = l_{nj}^{*} \) where the only unknown here is \( l_{nj}^{*} \). We now define a "-" function such that \( l_{nj}^{*} - l_{nj}^{*} = l_{nj}^{*} \).

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

Refer to Example 2, \( l_{in}^{*} = [(9, 4), (19, 6)] \) and \( l_{in}^{*} = [(3, 7), (3, 7)] \). Then, \( l_{in}^{*} - l_{in}^{*} \) would be \([(6, 4), (16, 6)] \). We say that \([(6, 4), (16, 6)] \) is the ideal \( l_{nj}^{+} \) for \( l_{in}^{*} \). It is not difficult to see we need another ideal \( l_{nj}^{+} \) for \( l_{in}^{*} \) since \( l_{in}^{*} - l_{in}^{*} \) may not be the same as \([(6, 4), (16, 6)] \). For example, if \( l_{in}^{*} = [(11, 4), (16, 6)] \), then \( l_{in}^{*} = [(11, 4), (16, 6)] \) and ideal \( l_{nj}^{+} = [(0, 4), (0, 6)] \). However, in the aggregation, we can have at most one \( l_{nj}^{+} \). We solve this problem by taking the averages of the delays and the bandwidths of the endpoints of all ideal \( l_{nj}^{+} \)'s to be the real \( l_{nj}^{*} \).

C. Finding bypasses

Obviously, due to the aggregation, \( l_{nj}^{*} + l_{nj}^{*} \) may no longer be the same as \( l_{nj}^{*} \) in the mesh. Some may deviate only a little.
bit while others may be quite different. In order to make the aggregation more precise, direct links between border pairs are introduced. Interested readers please refer to [10] for further reference.

IV. LINE-SEGMENT ROUTING ALGORITHM

Because our line segment representation is different from the traditional representation, no existing routing algorithm is applicable without modification. We present a routing algorithm that corresponds to our line segment parameter representation, and we call it the Line-Segment Routing Algorithm (LSRA). LSRA is a QoS-based source routing, integrating modified Dijkstra algorithm (DA) and the centralized bandwidth-delay routing algorithm (CBDRA) [12]. DA is an optimal algorithm for finding shortest-delay path when delay is the only metric in the network. CBDRA works in networks where there are two metrics: delay and bandwidth. It first prunes all links that do not satisfy the bandwidth requirement and then applies DA to find the minimum-delay path in the residue network. In LSRA, we further augment the idea to deal with the line segment representation. We first describe the information LSRA needs and then discuss the details of the routing algorithm.

Being a source routing protocol, LSRA requires that each node keeps the topology of its own peer group (PG) and the star aggregations of other PG’s. 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 space complexity of $O(b)$ instead of a mesh of $O(b^2)$ space. After each node obtains all the necessary information, routing can be performed. There are two levels of routing: inter-domain routing and intra-domain routing. An inter-domain routing path specifies border nodes in different PG’s to go through and intra-domain routing finds a path within a domain. As a result, LSRA has two phases accordingly.

A. Inter-domain Routing

After obtaining the star aggregations from other PG’s, each node can see all the nodes in its own PG and all border nodes of other PG’s as in Fig. 1(b). From the star aggregations, a node can compute the line segment between any border pair in an outside PG. We call the network that a node sees the node-view network (NVN). In NVN, nuclei of star aggregations are not considered as nodes since they are virtual. On the other hand, the complexity of NVN is a lot smaller than the real network as in Fig. 1(a) since only border nodes of outside PG’s are seen. A modified DA algorithm applying the technique of CBDRA is used to find the inter-domain path using NVN. Similar to DA, a path “grows” from the source to the target PG. The delay to each node in NVN is kept and each time a node is reached by the growing path, the delays of its neighbors are updated according to the links. There are two kinds of links in DVN: physical links and logical links. Physical links are points on the delay-bandwidth plane. They connect different PG’s and nodes in the source PG. Logical links are line segments connecting border nodes within the same PG obtained from the star aggregation. Suppose the routing request is $(req_d, req_u)$ and the accumulated delay from the source node to another node $i$ in NVN is $d[i]$. If there is a link, either physical or logical, from $i$ to $j$, LSRA updates the delay of node $j$ ($d[j]$) as follows:

Case I: A physical link with parameter $(D, W)$. If $W < req_u$, it means that no feasible path can go through this link, and so $d[j]$ is unchanged. It is the same as the pruning step in CBDRA. If $W >= req_u$, $d[j] = min(d[j], d[i] + D)$ as in DA.

Case II: A logical link $l = [(d_{ip}, w_{ip}), (d_{up}, w_{up})]$. If $w_{up} < req_u$, $d[j]$ is unchanged as in Case I; otherwise, $d[j] = min\{d[j], d[i] + d_{req}, d_{req\_lag} \}$, where $d_{req}$ is the delay-coordinate of the line segment $l$ with bandwidth equal to $req_u$.

The process stops when the path reaches one of the borders, say $t$, of the target PG. If $d[t] <= req_u$, the request is accepted and LSRA goes ahead to find the intra-domain routing. Otherwise, the request is rejected. The running time complexity of this phase is exactly the same as DA which is optimal.

B. Intra-domain Routing

After phase 1, the source knows how to traverse the nodes in its own PG to one of the border nodes and how to traverse the border nodes of other PG’s to get to the target PG. As the source node does not have the internal topology of an outside PG, it is impossible for it to compute an intra-domain route in the outside PG. As a result, in this phase, LSRA finds the route in a distributed fashion. A message or packet is sent from the source to travel along the inter-domain path found. When a border node $t$ of PG $g$ receives the message, and finds out the next hop in the inter-domain path is another border node $t^{'e}$ in $g$, it finds the path going from $t$ to $t^{'e}$ using CBDRA. The message keeps the accumulated delay along the path. If the accumulated delay exceeds $req_d$, cranking occurs and the message is forwarded back to the source. If the message successfully reaches the target PG, a feasible path is found. As the complexity of CBDRA is the same as DA, finding intra-domain routes is also optimal.

V. SIMULATION RESULTS

We compare LSRA with the shortest path (SP) algorithm. 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.

In Section IV, we describe how crankback occurs when LSRA accepts a request after finding an inter-domain path but unable to find the feasible intra-domain path. On the other hand, a feasible request may be rejected due to an inaccurate approximation of $d_{req}$.

Success ratio is used to measure quantitatively how well an algorithm finds feasible paths and it is defined as $\frac{\text{total number of feasible paths found}}{\text{total number of feasible requests}}$. A good algorithm should have a high success ratio but a small crankback ratio.

The simulated network topology tested consists of 10 domains, each has 15 to 35 nodes, having a total of 293 nodes. The number of borders varies from 3 to 4. 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. Over 3000 feasible routing requests are generated randomly. We measure the success ratios of both LSRA and SP. Since the parameters obtained by SP are real parameters, there is no crankback and so we study the crankback ratio of LSRA only. Fig. 3 shows the success ratios of the two algorithms w.r.t. end-to-end delay. Our LSRA achieves very good success ratio and performs much better than SP. In our simulation, LSRA has no crankback. Therefore, LSRA shows promising performance.

VI. CONCLUSION

In this paper, we present a novel algorithm for topology aggregation in delay-bandwidth sensitive networks fulfilling PNNI standard with a corresponding routing algorithm. We use line segments in the delay-bandwidth plane instead of points to represent the QoS parameters of logical links. We show an algorithm to find out the 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. Extensive simulations show that our algorithms achieve high success ratio with a very small crankback rate.

REFERENCES


