INTERNATIONAL JOURNAL OF COMMUNICATION SYSTEMS
Int. J. Commun. Syst. **12**, 181–195 (1999)

Efficient multicast routing with delay constraints

Gang Feng* and Tak-Shing Peter Yum

Department of Information Engineering, The Chinese University of Hong Kong, N.T. Shatin, Hong Kong

SUMMARY

To support real-time multimedia applications in BISDN networks, QoS guaranteed multicast routing is essential. Traditional multicast routing algorithms used for solving the Steiner tree problem cannot be used in this scenario, because QoS constraints on links are not considered. In this paper, we present two efficient source-based multicast routing algorithms in directed networks. The objective of the routing algorithms is to minimize the multicast tree cost while maintaining a bound on delay. Simulation results show that these two heuristics can greatly improve the multicast tree cost measure in comparison with the shortest path routing schemes. Their performance is close to that of the known CST_c algorithm proposed by Kompell *et al.* in Reference 1, but requiring a much shorter computation time. Copyright © 1999 John Wiley & Sons, Ltd.

KEY WORDS: multicast routing; delay bound; QoS; time complexity; heuristic

1. Introduction

Multimedia applications such as videoconferencing and remote collaboration rely on the ability of the network to provide multicasting communication. Multimedia traffic consists of audio and video that consume large bandwidth and require a certain quality-of-service (QoS) when transferred through networks.[1] Hence efficient multicast routing algorithms which are capable of constructing low-cost multicast trees that satisfy the constraints imposed by the QoS requirements are essential for real-time multimedia services. Current multicast routing protocols, such as PIM,[2] DVMRP,[3] are based on simple algorithms: shortest path multicasting and reverse path multicasting. These multicast algorithms usually assume simply additive cost metric.

Algorithms for constructing multicast trees have been developed with two optimization goals. The first is the minimum average path delay, which is the average of the minimum path delay from the source to each of the destinations in the multicast group. This can be done in $O(n^2)$ time using Dijkstra's shortest path algorithm,[4] where n is the number of nodes in the graph. The second goal is to minimize the cost of the multicast tree, which is the sum of the cost on the edges in the multicast tree. The least cost tree is called a *Steiner tree*, and the problem of finding a Steiner tree is known to be NP-complete.[4] Many heuristics for low-cost multicast routes take $O(n^3)$ to $O(n^4)$ time[5,6] and can produce solutions that are within twice the cost of the optimal solution.

There are also publications on the Constrained Steiner Tree (CST) problem.[6–11] Kadaba and Jaffe[6] solved a problem that involves optimizing both the cost and delay of the multicast tree.

* Correspondence to: Gang Feng, Department of Information Engineering, The Chinese University of Hong Kong, N.T. Shatin, Hong Kong. E-mail: gfeng5@ie.cuhk.edu.hk

CCC 1074–5351/99/030181–15$17.50

Received 1 October 1998
Accepted 25 January 1999

Chung *et al.* proposed algorithms for the degree-constrained multicast trees. Rouskas and Baldine studied the multicast routing with end-to-end delay and delay variation constraints. Kompell *et al.*,[1] proposed a well-known Constrained Steiner Tree (CST_c) heuristic. This heuristic consists of three stages. First, a closure graph (complete graph) of the constrained cheapest paths between all pairs of members of the multicast group is found. Second, a constrained spanning tree of the closure graph is found using a greedy algorithm based on cost. An alternative selection mechanism is proposed based on a function of both cost and delay. The edges of the spanning tree are then mapped back onto their paths in the original graph. Finally, loops, if any, are removed by using a shortest path algorithm on the expanded constrained spanning tree. The overall time complexity is $O(\Delta n^3)$, where Δ is the delay bound and n is the number of nodes in the graph. Zhu *et al.*[1,2] presented a heuristic algorithm for constructing minimum-cost multicast trees with delay constraints. The algorithm can satisfy different delay bounds on different destinations and handles two variants of the network cost optimization goal: minimizing the total cost of the tree and minimizing the maximal link cost. Good solutions can usually be obtained by this algorithm. The expected time complexity of this algorithm is $O(kn^3 \log(n))$, where k is a parameter used for the delay bounded shortest path construction. The value of k depends on the delay bound and is usually small.

In real-time multimedia applications, the upper bound on end-to-end delay is an important QoS requirement for the transmission of video and audio streams. For example, in teleconferencing applications, such as speaker-video videoconferencing,[7,8] a change of speaker during a conference session may require a new multicast tree be computed. The new multicast connection should be set up within a very short time to guarantee the continuity of the conference procession. Efficient multicast routing algorithms are essential for this kind of applications. Many constrained Steiner tree algorithms can give near-optimal solutions. Unfortunately, the computation time is beyond the setup speed required in large size networks due to their high time complexities ranging from $O(\Delta n^3)$ to $O(n^4)$. Furthermore, in the dynamic multicast case exemplified by the joining and withdrawal of nodes, new trees have to be computed if a static routing algorithm is used. New dynamic multicast algorithms that allow the addition and removal of conference nodes without recomputing the entire tree need to be developed for this case. In addition, most existing algorithms are for undirected graphs. Practical communication networks are more suitably modelled as directed graphs.

In this paper, we present two new heuristics for constructing delay constrained multicast trees in directed networks. The first one called '*Delay-constrained Shortest Path Multicasting*' is a static minimum cost tree computation algorithm that satisfies the end-to-end delay requirement set by the application. It has good performance and a time complexity of only $O(n^2)$. So it can be used in the applications requiring fast multicast connection setup. The second one called '*Dynamic Delay-constrained Multicasting*' is an efficient dynamic multicast routing algorithm that can handle multicasting dynamics such as the joining of nodes during an existing connection session.

In the next section, we address the network model and problem definition. In Sections 3 and 4, we describe the two proposed heuristics in detail. Their performances are evaluated and compared to those in the literature in Section 5 and we conclude this paper in Section 6.

2. Network model and problem definition

The network model and the problem are similar to that in Reference 1. We paraphrase them as follows. A network is modelled as a connected, directed graph $G = (V, E)$, where V is the set of

network nodes and E the set of links. In addition, we have two real value functions associated with each link e ($e \in E$): delay $D(e)$ and cost $C(e)$. The link delay $D(e)$ is the delay a data packet experiences on link e and the link cost $C(e)$ is a measure of its bandwidth usage. Links can be asymmetrical, i.e. the cost and delay for the link $e = (i, j)$ and link $e' = (j, i)$ can be different.

The delay-constrained multicast routing problem can be stated as follows. Given a source node s ($s \in V$), a set of destination nodes, $Z (Z \subseteq V - s)$ and a delay tolerance or delay bound Δ, find a tree $T (T \subseteq G)$ rooted at s and spanning all of the nodes in Z such that

$$\sum_{e \in P(s, v)} D(e) < \Delta, \quad \forall v \in Z \tag{1}$$

Here $P(s, v)$ is the unique path in T from s to v, and that the tree cost $\sum_{e \in T} C(e)$ is minimized.

Throughout this paper, we let $n = |V|$ and $m = |Z|$. The nodes in Z are often called multicast group members and m is then the multicast group size. Obviously, when Δ is ∞, the delay-constrained multicast routing problem reduces to the NP-complete Steiner tree problem.

A tree rooted at s, spanning all the nodes in S and satisfying (1) is called a constrained multicast tree. A heuristic of the delay-constrained multicast routing problem should give a solution to the problem if a solution exists.

A centralized delay-constrained multicast routing algorithm can be easily extended to compute multicast trees that satisfy both a given end-to-end delay constraint and a given link bandwidth constraint. Its exposition, however, is beyond the scope of this paper.

3. Delay-constrained Shortest Path Multicasting Algorithm (Algorithm A)

The basic idea of the Delay-constrained Shortest Path Multicasting Algorithm is to construct a constrained minimum cost tree and a shortest delay path tree in parallel. Combining these two trees gives the solution tree. We use an algorithm called Delay-constrained Shortest Path (DCSP) algorithm to compute the minimum cost path tree. The details are shown in Figure 1. The computation of the shortest delay path tree can make use of Dijkstra's algorithm.[2] We now describe the algorithm as follows.

Algorithm A

Input: A directed graph $G = (V, E)$, delay matrix $\mathbf{D}_{n \times n}$ and cost matrix $\mathbf{C}_{n \times n}$, a source $s \in V$, a set of destinations $Z \subseteq V \backslash s$, a delay constraint Δ.

Output: A delay-constraint directed tree $T = (V_T, E_T)$ rooted at s and spanning all nodes in Z.

Step 1. Call DCSP function under delay matrix $\mathbf{D}_{n \times n}$, cost matrix $\mathbf{C}_{n \times n}$ and delay constraint Δ to compute a minimum cost spanning tree $T_1 = (V_1, E_1)$ rooted at s and spanning as many destination nodes in Z as possible.

Step 2. If $V_1 = s \cup Z$, $T = T_1$ is the multicast tree and return the tree T.

Step 3. Call Dijkstra's shortest path algorithm under delay matrix $\mathbf{D}_{n \times n}$ to compute a shortest delay path tree $T_2 = (V_2, E_2)$ that spans those nodes in $Z \backslash V_1$.

Step 4. Combine T_1 and T_2 to give a multicast routing tree T.

Step 5. Remove the loops in the combined tree T.

Algorithm Delay-Constrained Shortest Path (DCSP)

```
/* DCSP algorithm is to compute a constrained spanning tree by using the Dijkstra's algorithm with the additional
delay constraint for each path. */
Begin
/* S = source node
 * Z = multicast group
 * Δ = delay constraint
 * cost(u,v) = cost of link (u,v)
 * delay(u,v) =delay of link (u,v)
 */
INITIALIZATION:
 for all v ∈ Z do
   cost_label(v) ← ∞;
   delay_label(v) ← ∞;
   final(v) ← false;
 end for
 cost_label(s) ← 0;
 delay_label(s) ← 0;
 final(s) ← true;
 recent ← s;
 /* node S is permanently labeled with 0. All other nodes are temporarily labeled with ∞.
    Node S is the most recent node to be permanently labeled. */

ITERATION:
 while there are nodes in Z which have not been computed do
   for every immediate successor v of recent if not final(v) do
     if delay_label(recent) + delay(recent,v) < Δ then
       new_cost ← cost_label(recen) + cost(recent,v);
       if new_cost < cost_label(v) then
           cost_label(v) ← new_cost;
           delay_label(v) ← delay_label(recent) + delay(rcent,v);
       end if
     end if
   end for
   let y be the node with smallest temporary cost_label value, which is ≠ ∞
   final(y) ← true;
   recent ← y;
   /* y, the next closest node which does not violate the delay constraint from s
      gets permanently labeled */
 end while
end
```

Figure 1. Delay-constrained Shortest Path (DCSP) Algorithm

Note that loops may appear in combining T_1 and T_2. This can be simply detected by checking the number of inbound links for each node of the combined tree. If there are two inbound links at a node, a loop exists. That is, there are two paths (one in T_1 and another in T_2) from a common

node to this node. We need to remove segment in either path of T_1 or path of T_2 to break the loop for the node with two inbound links. If the path in T_1 can satisfy the delay constraint of all the downstream nodes (identified by the Dijkstra's algorithm in Step 3), then we keep the path of T_1 for minimum cost. To remove the segment of the path of T_2, we remove the links of the path of T_2 one by one along the upstream direction until a multicast destination is encountered. Otherwise, we need to keep the T_2 segment, or remove the T_1 segment in the same way. For example, Figure 2 illustrates a case of removing a loop (Step 5). The number pair next to an edge indicates the (*cost*, *delay*) of that edge. Assume that the delay constraint is 12. Figures 2a and b are the minimum cost tree and the shortest delay tree, respectively. Node C in combined tree in Figure 2c has two inbound links, say A to C and B to C. To remove the loop, we first consider keeping the path B to C and ignore the path A to C for minimizing the tree cost. However, the delay along the path (A, B, C, E, F) is 14. This violates the delay constraint. Therefore, we have to choose the path A to C in the shortest path tree and ignore the path B to C in the minimum cost path tree. We then obtain a loop free solution that does not violate the delay constraint shown in Figure 2d.

Figure 3 shows an example with source node $s = F$, destination node set $Z = \{A, C, D, E\}$ and delay constraint $\Delta = 11$. Figure 4 shows the procedure of the Algorithm A. In this example, Algorithm A cannot span node D using the minimum cost path tree. The path H to D as a segment of the shortest tree is used to connect node D.

The delay-constrained tree consists of T_1 supplemented by T_2. T_1 satisfies the delay constraints from step 1. T_2 is the delay constrained tree for the remaining destinations. The combined tree

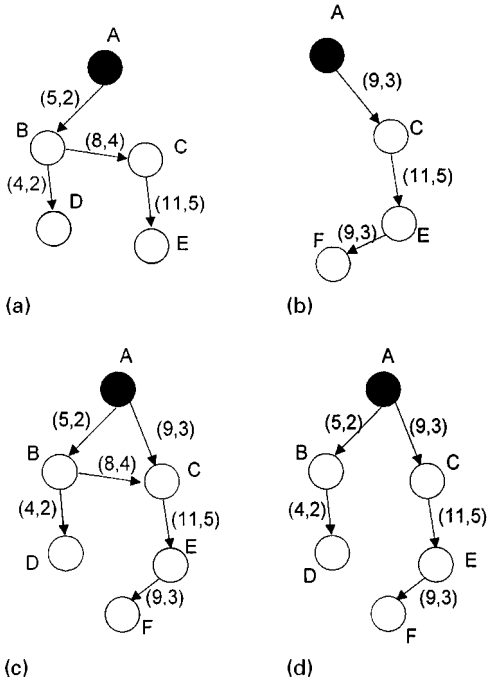

Figure 2. Loop removal in Algorithm A. (a) Minimum cost path tree, (b) shortest delay path tree, (c) combined tree and (d) solution tree

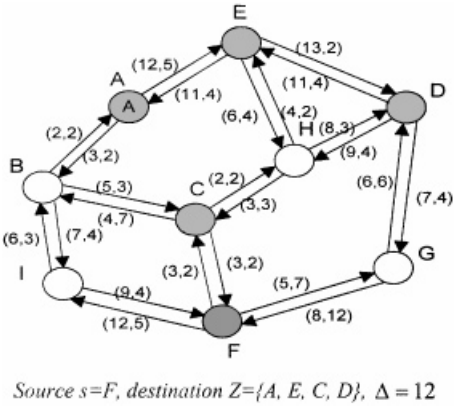

Figure 3. An example network

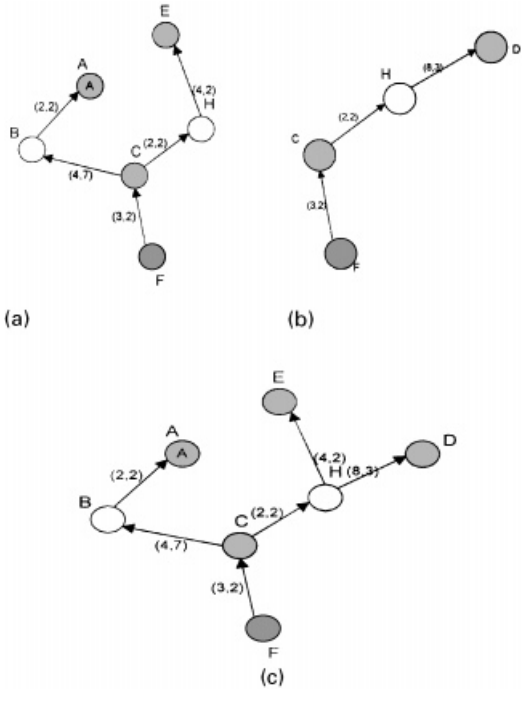

Figure 4. Trees generated in steps of Algorithm A. (a) Step 1: minimum cost path tree, (b) Step 2: shortest path tree and (c) Step 4 combined tree

therefore must be delay constrained. That is, Algorithm A can always find a delay-constrained tree if such a tree exists.

Steps 1 and 3 of Algorithm A have the same time complexity as that of Dijkstra's algorithm, which is at most $O(n^2)$. The last step has a time complexity of $O(n)$. Therefore the overall time complexity of Algorithm A is $O(n^2)$.

4. Dynamic Delay-constrained Multicasting Algorithm (Algorithm B)

We now present a new heuristic algorithm for the delay constrained multicast routing. We call this algorithm *Dynamic Delay-constrained Multicasting* or Algorithm B for short. Three notable features of this algorithm are: (1) capable of adding destinations without re-computing the existing tree, (2) provides trade-offs between performance (tree cost) and running time using a single user-specified parameter, and (3) low computational complexity and good performance in terms of tree cost.

4.1. Algorithm description

The basic structure of Algorithm B is similar to that of Prim's spanning tree algorithm.[4] A tree is grown starting from s. At every step, a branch, i.e. a directed path, is extended from the tree to one new destination not in the tree. This step is repeated until all of the destinations in Z are included in the tree.

In each step, an unconnected destination is chosen at random. Then nodes already in the tree are arranged in a priority queue,[4] denoted as Q, according to their delay values from the source along the computed path (i.e. in the existing tree). A knob k is used as a user-controlled parameter and the first k nodes in the priority queue are placed in a bin, denoted as B. The k minimum cost paths for each of the nodes in the bin to that destination are then computed. As a minimum cost path may pass through nodes already in the tree, we identify the portion of the path not in the existing tree and call it the subpath. We choose the path with the minimum cost subpath that satisfies the delay constraint for extending the tree to the destination. If no such subpath exists, a shortest delay path from the source node is used instead. This step is repeated until all destinations are linked. Obviously, a larger k value leads to lower cost trees at the expense of longer computational time.

4.2. Algorithm B

A formal description of Algorithm B is shown in Figure 6. The four functions used there are defined as follows:

(1) *Shortest_Path* $(s, t, \mathbf{w})$: a function that computes the shortest directed path from node s to node t under edge weight matrix $\mathbf{w}$, where $\mathbf{w}$ may be either delay matrix or cost matrix;
(2) $COST(P)$: a function that returns the cost of path P, i.e., $\sum_{e \in P} C(e)$;
(3) $DELAY(P)$: a function that returns the delay along directed path P, i.e. $\sum_{e \in P} D(e)$;
(4) *Sub_Path* $(P(u, v), T)$: a function that computes a subpath of path $P(u, v)$ that is outside the existing tree T. The output is path $P(w, v)$ where w is the linking node in T. Figure 5 shows the operation of function *Sub_Path*( ).

Algorithm B (Figure 6) can always find a delay-constrained tree if such a tree exists. Moreover, the resulting multicast tree is loop-free. To see this, note that in each iteration of the algorithm, either a minimum cost subpath or a shortest delay subpath is used for connecting a new destination. In both cases, the algorithm guarantees that no delay constraint is violated. Moreover, only *one* branch (subpath) is created in each outer '*for*' iteration in the algorithm for linking a new destination to the existing tree. The *Sub_Path*( ) function guarantees that the

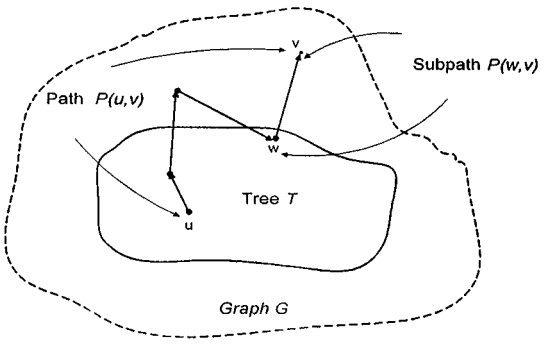

Figure 5. Illustration of function $Sub\_Path(P(u, v), T) = P(w, v)$

subpath has only one node (just the linking node) on the existing tree. Therefore, the adding of a destination does not cause a loop. Hence, the entire tree produced by Algorithm B is also loop-free.

4.3. Example

We again take the network shown in Figure 3 as an example to illustrate the operation of Algorithm B. Let the knob be $k = 2$. For comparison, we show the corresponding shortest delay path tree and the minimum cost path tree in Figures 7a and b, respectively. The cost of the shortest delay path tree is 28. Note that in the minimum cost path tree the path from F to D has delay 13, violating the delay constraint.

Figures 8a–d show the steps of connecting nodes in destinations set Z. Note that for connecting D in Figure 8c, Algorithm B first checks (in line 4 of the Algorithm B) the minimum cost path P_1 from F to D via G. Since $DLY(F) + DELAY(P_1) = 13 > \Delta$, another minimum cost path P_2 from C to D via H which satisfies the delay constraint used (line 7 of the Algorithm B). Here, $PATH = P_2$, $BRANCH = P(H, D)$ and $w = H$. In Figure 8d, both minimum cost paths P_1 and P_2 from F and C satisfy the delay constraint and $Sub\_Path(P_1) = P_2 = BRANCH$.

In this example, the tree generated by Algorithm B is the same as that found by Algorithm A and is also the optimal tree. The tree cost is 23, which is smaller than 28, the cost of shortest delay path tree.

4.4. Running time

Let n be the number of nodes in the graph, m be the number of destinations of a multicast group. The *Shortest_Path*( ) function in Algorithm B can be realized by using Dijkstra's algorithm.[4] The time complexity of Dijkstra's algorithm is of $O(n^2)$.

Line 4 in Algorithm B takes $O(n^2)$ time and the computation of subpath in line 5 takes $O(1)$ time. Thus the inner **for** loop from line 3 to 9 takes at most $O(kn^2) + O(k)$ time. Lines 11–18 are for computing a shortest delay path and its subpath if the computed minimum cost path does not satisfy the delay constraint. It takes at most $O(n^2) + O(1)$ time. Maintaining the priority queue in line 22 and adding of nodes and edges in lines 23 and 24 takes $O(1)$ time. Therefore the entire algorithm costs the running time of $O(kn^2)$.

Algorithm B

Input: A directed graph $G = (V, E)$, delay matrix $\mathbf{D}_{n\times n}$ and cost matrix $\mathbf{C}_{n\times n}$,
a source $s \in V$, a set of destinations $Z \subseteq V - s$, a delay constraint Δ, a knob k.

Output: A delay-constrained directed tree $T = (V_T, E_T)$ rooted at s and spanning all nodes in Z

```
Begin
INITIALIZATION:
Q ← {s}, V_T ← {s}, E_T ← { };
T = (V_T, E_T);
COST(BRANCH) ← INFINITY ;   /* Let DLY(u) be the delay along the computed path from s to u */
for each u in V do
  DLY(u) ← INFINITY ;
end for

ITERATION:
for each v in Z do                                                          (1)
  B ← first min(k, |Q|) nodes in Q;     /* |Q| is the length of the queue Q */   (2)
  for each u in B do                                                        (3)
    P ← Shortest_Path(u, v, C) ;                                            (4)
    PATH ← Sub_Path(P, T);                                                  (5)
    let W be the terminal ∈ T of the path PATH ;                            (6)
    if ( DLY(w) + DELAY(BRANCH) ≤ Δ ) and
       ( COST(PATH) < COST(BRANCH) ) then                                   (7)
      BRANCH ← PATH ;                                                       (8)
    end if                                                                  (9)
  end for
  if DLY(w) + DELAY(BRANCH) > Δ then /* minimum cost path does not          (10)
                                        satisfy delay constraint */
    P ← Shortest_Path(s, v, D) ;                                            (11)
    PATH ← Sub_Path(P, T);      /* find the shortest path to add to the tree */   (12)
    let W be the terminal ∈ T of the path PATH
    if DLY(w) + DELAY(PATH) > Δ then                                        (13)
      return T ;                                                            (14)
      break;
    else                                                                    (15)
      BRANCH ← PATH ;                                                       (16)
    end if                                                                  (17)
  end if                                                                    (18)
  for each u ∈ BRANCH \ w do                                                (19)
    DLY(u) ← DLY(w) + DELAY(P(w, u));                                       (20)
  /* where P(w, u) ⊆ PATH is the subpath of PATH */
  end for                                                                   (21)
  insert nodes in BRANCH into Q;                                            (22)
  V_T ← V_T ∪ {nodes in BRANCH };                                           (23)
  E_T ← E_T ∪ {edges in BRANCH};                                            (24)
  S ← S − {u};                                                              (25)
  T ← (V_T, E_T);                                                           (26)
  end for                                                                   (27)
return T = (V_T, E_T)                                                       (28)
end Algorithm                                                               (29)
```

Figure 6. Dynamic Delay-constrained Multicasting Algorithm

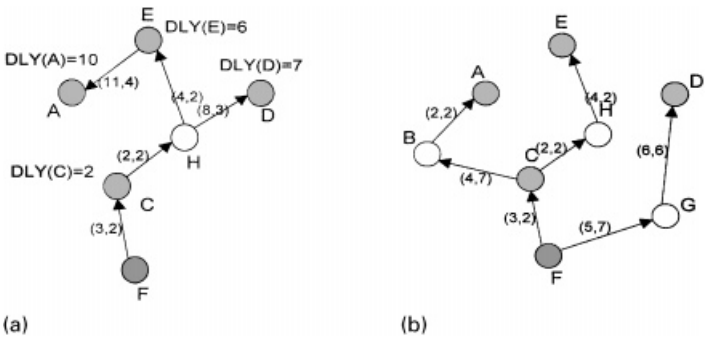

Figure 7. (a) The shortest delay path tree and (b) minimum cost path tree

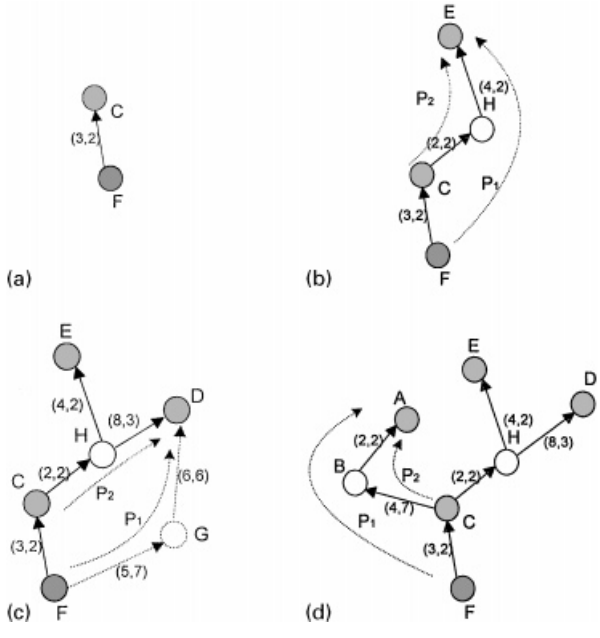

Figure 8. An example illustrating Algorithm B. (a) Connecting C, (b) connecting E, (c) connecting D and (d) connecting A

5. Performance studies

To evaluate the performance of these two algorithms, we run them on a large number of randomly generated graphs with various average degrees. This kind of random graph can represent the topologies of common point-to-point networks, e.g. the NSFNET.[1]

5.1. Random graph generation

We employ a similar method used in literature[1,5,6] to generate random graphs. Nodes are placed randomly in a rectangular co-ordinate grid by generating uniformly distributed random

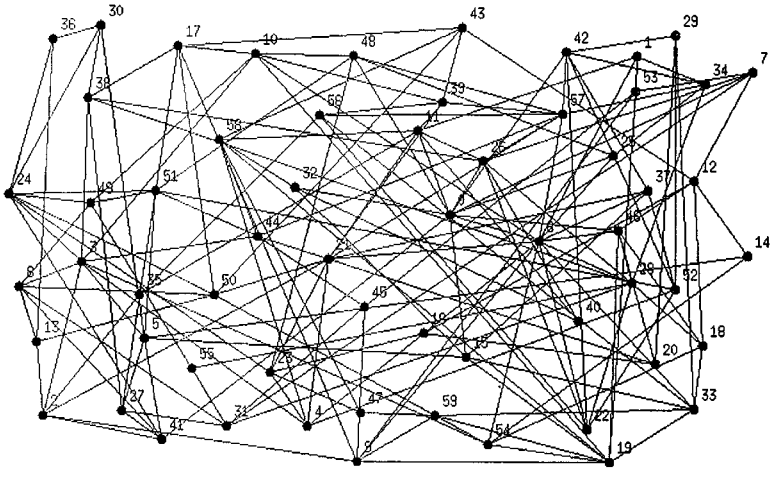

Figure 9. A random graph example

numbers for the x and y co-ordinates. A directed edge is placed from u to v with probability

$$P(u, v) = \beta \exp\left[\frac{-d(u, v)}{L\alpha}\right]$$

where $d(u, v)$ is the Euclidean distance between u and v, L is the maximum distance between two nodes and α and β are two parameters that control the characteristics of the graph produced. Increasing β increases the average vertex degree of the graph and increasing α increases the ratio of longer edges relative to shorter ones. These two parameters allow the generation of a wide variety of random graphs.

The edges' costs are randomly selected from the set $\{1, 2, \ldots, 10\}$. The random edge costs match typical values for costs used in the NSFNET backbone network.[1] The delays for edges are proportional to their lengths. The multicast groups are generated by randomly placing the source and destinations on the graphs. Figure 9 shows an example of a random graph so generated. The network has 60 nodes with parameters: $\alpha = 0{\cdot}2$, $\beta = 0{\cdot}65$. The average node is about 5.

5.2. *Performance comparison*

We compare the performance of Algorithms A and B, with CST_c[1] and the shortest delay tree created by Dijkstra's shortest path tree algorithm (SPT) with unnecessary branches pruned. We take the tree cost of CST_c algorithm as a comparison basis and define the relative cost G with respect to CST_c as follows:

$$G_A = \frac{\sum_{i=1}^{K}(H_A^{(i)} - H_{CST}^{(i)})}{\sum_{i=1}^{K} H_{CST}^{(i)}}$$

$$G_B = \frac{\sum_{i=1}^{K}(H_B^{(i)} - H_{CST}^{(i)})}{\sum_{i=1}^{K} H_{CST}^{(i)}}$$

$$G_{\mathrm{SPT}} = \frac{\sum_{i=1}^{\mathrm{K}} (H_{\mathrm{SPT}}^{(i)} - H_{\mathrm{CST}}^{(i)})}{\sum_{i=1}^{\mathrm{K}} H_{\mathrm{CST}}^{(i)}}$$

where subscripts A, B and SPT refer to Algorithms A, B, and SPT, superscript (i) refers to the ith run, H is the tree cost and K the number of runs.

In our simulations, the graph size ranges from 50 to 110 nodes. The average node degree is between 6 and 14 (by varying β). Each experiment generates a series of graphs with identical parameters. Each graph generated is first checked to ensure that a solution exists. This is done by checking if the delay values of the shortest delay paths from the source to individual destinations are all smaller than the delay constraint Δ. The 95 per cent confidence interval for the relative costs is within 2 per cent for all simulation results (in our simulations, $K \geqslant 300$ can satisfy this requirement).

Figure 10 shows the relative costs of SPT, Algorithms A and B relative to $\mathrm{CST_c}$ versus the number of network nodes. It is shown that the relative cost of Algorithm A is significantly smaller

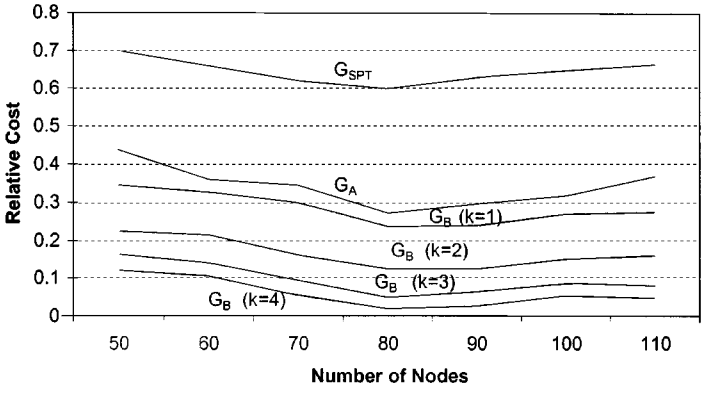

Figure 10. Relative costs versus the number of network nodes, comparing SPT, Algorithm A and Algorithm B ($k = 1, 2, 3, 4$), for group size = 15, $\Delta = 50$, maximum node degree = 9

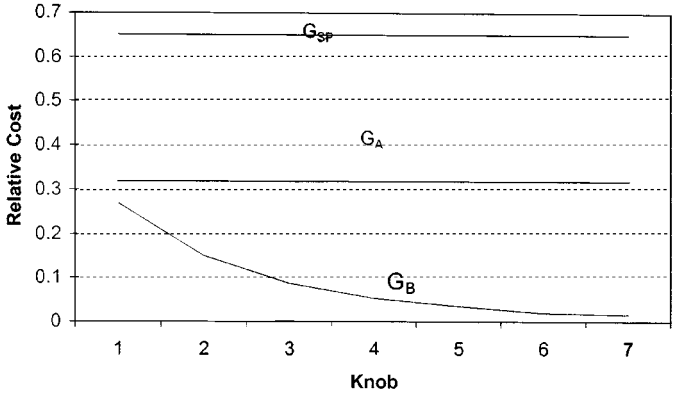

Figure 11. Relative costs versus knob, for group size = 15, $\Delta = 50$, $\alpha = 0{\cdot}25$, number of nodes = 100, average node degree = 7

than that of SPT, the relative cost of Algorithm B is significantly smaller than that of Algorithm A. Figure 11 shows the relative costs of Algorithm B for knob $k = 1, 2, \ldots, 7$. We can see that even for $k = 1$, the cost performance of Algorithm B is better than Algorithm A. When knob $k \geqslant 2$, the performance of Algorithm B is significantly better than that of Algorithm A. As knob $k \geqslant 4$, the difference between the relative costs of Algorithm B and CST_c is very small: below 6 per cent for $n = 100$. Another observation is that larger k results in smaller relative costs. However, as k increase, the improvement decreases.

Figure 12 shows that both G_{SPT} and G_A increase with the multicast group size. On the other hand, G_B maintains at the 5 per cent relative cost level ($k = 4$ and $n = 100$), which means the difference of cost performance between Algorithm B and CST_c does not vary much.

Figures 13 and 14 show the relative costs versus the delay constraint Δ and average node degree. We see that both G_{SPT} and G_A increase with Δ and average node degree. But G_B is generally not affected by the change of Δ and average node degree. We can see that the difference of cost performance between Algorithm B and CST_c is not affected by the characteristics of the network in which the multicast routing algorithms are examined.

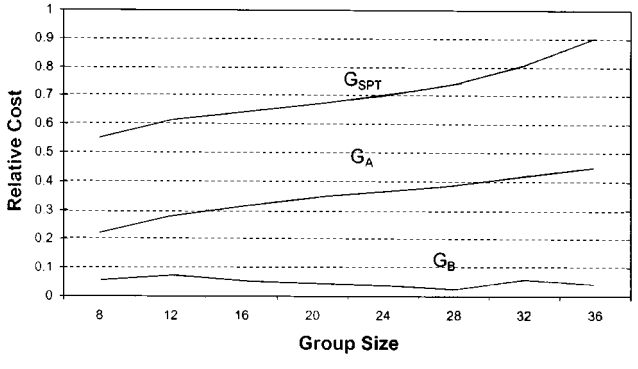

Figure 12. Relative costs versus group size, for $\Delta = 50$, number of nodes = 100, knob $k = 4$, average node degree = 7

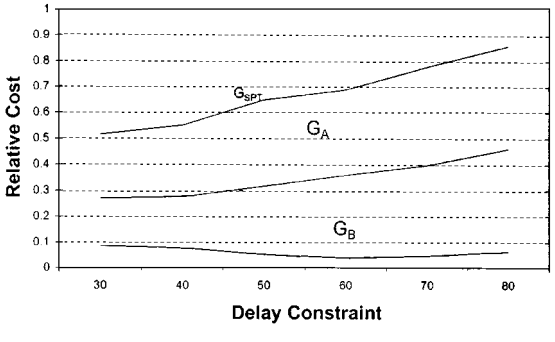

Figure 13. Relative costs versus delay constraint Δ, for group size = 15, number of nodes = 100, knob $k = 4$, average node degree = 7

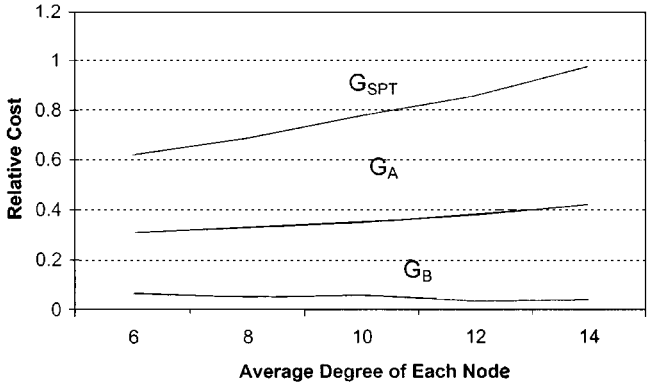

Figure 14. Relative costs versus average degree of each node, for group size = 15, number of nodes = 100, knob $k = 4$. Corresponding $\beta = 0{\cdot}20$–$0{\cdot}42$

6. Conclusion

The planning and running of multimedia applications in BISDN require an efficient multicast protocol. We have presented two efficient multicast routing algorithms that can produce good solutions and scale to large size networks. Algorithm A is very simple and is suitable for static multicast connection requests, while Algorithm B allows the tuning of the tree cost by the run time and can support multicasting dynamics. These two properties are important for multiparty conferencing applications where the setup speed of multicast connection is critical and the multicast group is dynamic. The performance of Algorithm B is found to be very close to that of CST_c but at a much lower time complexity.

References

1. V. P. Kompella, J. C. Pasquale and G. C. Polyzos, 'Multicast routing for multimedia communication', *IEEE/ACM Trans. Networking*, **1**(3), 286–292 (1993).
2. D. Waitzman *et al*., 'Distance vector multicast routing protocol', Internet RFC 1175, November 1988.
3. S. Ramanathan, 'Multicast tree generation in networks with asymmetric links', *IEEE/ACM Trans. Networking*, **4**(4), 558–568 (1996).
4 T.H. Cormen *et al*., *Introduction to Algorithms*, MIT Press, Cambridge, MA; 1990.
5. B. M. Waxman, 'Routing of multipoint connections', *IEEE J. Selected Areas Commun*., **6**(9), 1617–1622 (1988).
6. B. K. Kabada and J.M. Jaffe, 'Routing to multiple destinations in computer networks', *IEEE Trans. Commun*., **31**(3), 343–351 (1983).
7. T. S. Yum, M. S. Chen and Y. W. Leung, 'Video bandwidth allocation for multimedia teleconferences', *IEEE Trans. Commun*., **43**(2), 457–465 (1995).
8. Tat Keung Chan and Tak-Shing Peter Yum, 'Analysis of multipoint videoconferencing under basic reroutable route-configuration assignment', in *IEEE GLOBECOM'96,* pp. 899–906. Also to appear in *Int. J. Intelligent Networks.*
9. S.-J. Chung, S.-P. Hong and H.-S. Huh, 'A fast multicast routing algorithm for delay-sensitive applications', in *Proc. IEEE GLOBECOM'97*, pp. 1898–1902.
10. G. N. Rouskas and I. Baldine, 'Multicast routing with end-to-end delay and delay variation constraints', in *Proc. IEEE INFOCOM'96,* 1996, pp. 353–360.
11. S.-J. Chung, S.-P. Hong, S.-B. Kim and H.-S. Chung, 'Algorithms for the degree-constrained multicast trees in packet-switched networks', in *Proc. IEEE GLOBECOM'98*, 1998, pp. 2004–2009.
12. Qing Zhu, M. Parsa and J. J. Garcia-Luna-Aceves, 'A source-based algorithm for delay-constrained minimum-cost multicasting', *Proc. IEEE INFOCOM'95*, pp. 377–385.

13. L. Kou, G. Markowsky and L. Berman, 'A fast algorithm for Steiner trees', *Acta Inform.*, **15**, 141–145 (1981).
14. S. Deering *et al.*, 'Protocol independent multicast-dense mode(PIM-DM): protocol specification', Internet Draft, September 1995. Available at ftp://ftp.ietf.cnri.reston.va.us /internet-drafts/draft-ietf-idmr-pim-spec-02.txt.

Authors' biographies:

Gang Feng received the B. Eng degree and M. Eng degree in Electronics Engineering from the University of Electronic Science and Technology of China (UESTC), in 1986 and 1989, respectively, and the PhD degree in Information Engineering in 1998 from the Chinese University of Hong Kong. From 1989 to 1995, he was with the Research Institute of Information Systems, UESTC. There his research work included the modelling and equalization of non-linear digital channels. He is currently with the Department of Electronics Engineering, City University of Hong Kong, as a term staff. His current research interest includes routing and performance evaluation for ATM networks, Reliable Multicast in Internet and IP over ATM.

Peter T. S. Yum worked in Bell Telephone Laboratories, U.S.A. for two and a half years and taught in National Chiao Tung University, Taiwan, for two years before joining the Chinese University of Hong Kong in 1982. He has published original research on packet switched networks with contributions in routing algorithms, buffer management, deadlock detection algorithms, message resequencing analysis and multiaccess protocols. In recent years, he concentrated his work to the design and analysis of cellular network, lightwave networks and video distribution networks. He believes that the next challenge is designing an intelligent network that can accommodate the needs of individual customers. Professor Yum's research benefits a lot from his graduate students. Six of them are now professors at local institutions. His recent hobby is reading history books. He enjoys playing bridge and badminton.

