Comments

Comments and An Improvement on "A Distributed Algorithm of Delay-Bounded Multicast Routing for Multimedia Applications in Wide Area Networks"

Tzu-Lun Huang and D. T. Lee, *Fellow, IEEE*

***Abstract*—In this correspondence, we first point out an error in Jia's algorithm (1998) by a counterexample. Second, we provide a fix and make an improvement to the part of Dynamic-Join. Finally, we give an analysis and a correctness proof of our algorithm.**

I. A Counterexample to Jia's Work

In [1], Jia gave an algorithm for finding a Multicast Routing Tree (MRT) with a bounded delay. We showed that the resulting network as produced by the algorithm may contain cycles and therefore is not a tree. Fig. 1 is a simple network in which the cost and delay for each link are assumed identical. The destination nodes are shown in solid circles and the delay constraint for each destination is 6. In Fig. 1, because node b is the node which is closest to the tree (initially, there is only one source node s in the tree), the source node will first set up a path $s \rightarrow a \rightarrow b$ to destination b by sending a *Setup* state message. When the *Setup* state message arrives at the destination b, node b according to the algorithm in [1] will choose node a as the next fork node to set up the second path to destination h, that is, $a \rightarrow d \rightarrow e \rightarrow h$. When the *Setup* state message arrives at destination h, node h will send a *Fork* state message to the source node s for setting up the final connection $s \rightarrow d \rightarrow e \rightarrow h \rightarrow i$. And at this time, a cycle occurs. Fig. 2 is the result after executing Jia's algorithm and a cycle $s \rightarrow a \rightarrow d \rightarrow s$ will be produced. Hence, the result of Jia's algorithm is not guaranteed to be a tree.

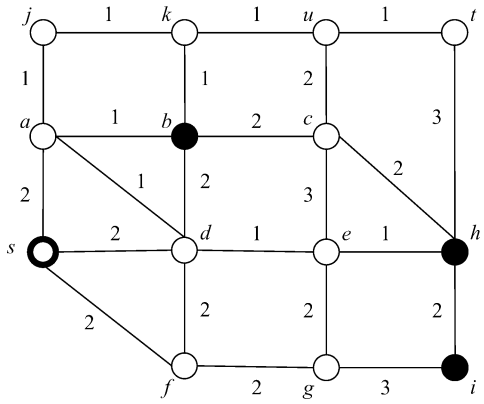

Fig. 1. Example of a network graph, $\Delta = 6$.

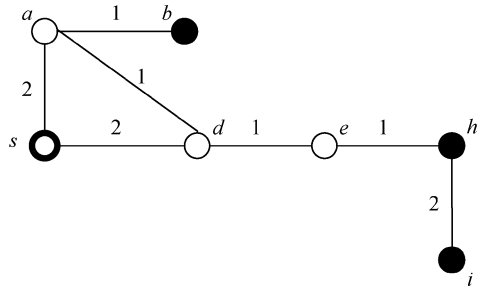

Fig. 2. Counterexample of cycle occurrence, $\Delta = 6$.

II. Corrections and Improvement

A. Corrections

Other than the cycle processing and tree pruning given below, the other processes of constructing an MRT are the same as [1].

1) Cycle Processing: In this part, we will describe how to deal with the problem of cycles during the construction of the MRT. A cycle can be easily detected by checking whether the node is already in the current MRT. In other words, when a node v, which had a parent (old), gets another parent (new), a cycle has resulted. Cycle processing is composed of two parts: upward adjustment and downward modification. These two parts of actions can be done in a concurrent way.

Upward adjustment is to adjust the parent of the cycle node v from old to new. Node v sends a *Break* state message to its old parent node v_p and resets its parent node to the new one. Upon receiving the *Break* state message, node v_p removes its link to node v. Lemma 2 shows that all destination nodes in the sub-tree rooted at node v are guaranteed to satisfy their delay constraints after changing the parent node from old to new.

Downward modification is to modify the contents in the *T2D* table in Jia's algorithm and to modify the accumulated delay associated with each node in the sub-tree T_v rooted at the cycle node v, due to the change of the tree topology and the new delay D_v, which is smaller than the old delay. Node v broadcasts *Modify* state message to every node in the sub-tree rooted at node v to modify the accumulated delay layer by layer from top (the cycle node v) to bottom (the leaf nodes). However, the process of modifying the *T2D* table begins from bottom to top layer by layer. Every leaf node in the sub-tree will use the heuristic function, equation (8) described in [1], for changing the contents in the *T2D* table by using the new accumulated delay, and send the updated *T2D* table to its parent node. Each nonleaf node will modify the entries in the *T2D* table according to the contents in the *T2D* table sent from all its child nodes and itself. When node v gets the new *T2D* table sent from all its child nodes, it will create a new routing message (i.e., *Setup* state message) and forward the new routing message to the next hop node to the destination.

2) Tree Pruning: After v_p breaks the connection to the cycle node v, v_p may be a *dangling* node, i.e., it is neither a destination node nor a relay node. Hence, it is necessary to prune the unnecessary links in the MRT. After the source receives the *Complete* state message, the source will check if the cycle variable C_i in the message is true, reflecting if a cycle occurred during the construction of MRT. If there is no occurrence of a cycle (C_i is "False"), the process finishes and the current tree is the final MRT. However, if the variable C_i is "True", the source will broadcast a *Probe* state message to each node in the MRT for pruning

Manuscript received July 23, 2004; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor E. Zegura. This work was supported in part by the National Science Council under Grants NSC-92-3112-B-001-018-Y, NSC-92-3112-B-001-021-Y, NSC-93-2213-E-001-013, NSC-93-2422-H-001-0001, and NSC-93-2752-E-002-005-PAE.

T. L. Huang is with the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, R.O.C. (e-mail: d88002@csie.ntu.edu.tw; jaran@iis.sinica.edu.tw).

D. T. Lee is with the Department of Computer Science and Information Engineering, National Taiwan University, and also with the Institute of Information Science, Academia Sinica, Nankang, Nankang 115, Taipei, Taiwan, R.O.C. (e-mail: dtlee@iis.sinica.edu.tw; dtlee@ieee.org).

Digital Object Identifier 10.1109/TNET.2005.860091

the unnecessary connections. The process of pruning the MRT begins from bottom to top layer by layer. Upon receiving the *Probe* state message, each leaf node in the MRT will check whether it is a destination or not. If it is not a destination, it would send *Prune* state message to its parent node. After receiving the *Prune* state message, the parent node will delete the connection and continue to send the *Prune* state message if it is neither a relay node nor a destination node after deletion. If the node in the MRT is either a destination or a relay node, then it will send an *OK* state message to its parent node. When the source node receives replies (either *OK* state message or *Prune* state message each) sent from all its child nodes, the MRT is finished and the algorithm terminates.

B. Improvement in the Part of Dynamic-Join

In the part of Dynamic-Join in [1], the source node has to maintain the information of how many leaf nodes there are in the current MRT (because Jia's algorithm requires all leaf nodes sending a reply message to the source for deciding the closest path from the tree to the new destination node and all destination nodes are not necessarily located at the leaves of the tree, i.e., the number of leaf nodes are not fixed). In our improvement, we require only that every destination node in the tree send a reply message back to the source. After the source node receives replies equal to the size of the multicast group, the source would know which node in the tree is closest to the new destination and its cost. Thus, the source node need not maintain the information regarding the number of leaf nodes in the MRT.

III. Analysis and Correctness Proof of the Algorithm

A. Analysis of the Algorithm

Theorem 1: The time complexity and message complexity of constructing a MRT (i.e., algorithm *Tree-Construction*) are $O((|M|+|C|)\cdot |V|)$ where $|M|$ is the size of the multicast group M, $|C|$ is the number of cycles created, and $|V|$ is the number of vertices in the network.

Proof: For algorithm *Tree-Construction*, all destinations are joined in the multicast tree sequentially. For each destination node we need to initiate a *Setup* state message (there are $|M|$ destinations in total, so it needs $|M|$ *Setup* state messages), which is passed from node to node until the destination is reached. For each destination the total path length is at most $|V|-1$. Meanwhile we need to send $|M|-1$ *Fork* state messages for each subsequent destination node. Since a cycle may be created occasionally, for each cycle detected we need to send a *Break* state message to eliminate the cycle and to adjust the accumulated delay and the contents in the *T2D* table by broadcasting *Modify* state messages to the sub-tree rooted at the cycle node. We need $O(|C|\cdot|V|)$ time for cycle processing. Finally, we need to prune the unnecessary connections (redundant edges) in the MRT and to send a *Complete* state message to finish the multicast group connection. Thus, we conclude that the total time complexity of algorithm *Tree-Construction* is $O((|M|+|C|)\cdot|V|)$.

The message complexity of the algorithm *Tree-Construction* is the same as the time complexity because the construction of the MRT is sequential. Therefore, the message complexity of the algorithm *Tree-Construction* is also $O((|M|+|C|)\cdot|V|)$. ■

B. Correctness Proof

In this part, we will give a correctness proof of the cycle processing. It includes upward adjustment and downward modification two parts.

Theorem 2: After finishing downward modification, the *T2D* table in Jia's algorithm can correctly reflect the optimal path information from the new tree to every remaining destination node.

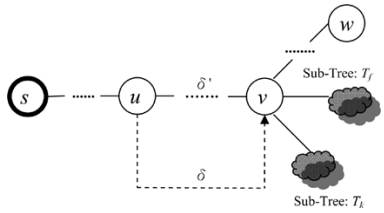

Fig. 3. Illustration for the proof of *Theorem* 3.

Proof: Due to the change of the tree topology, downward modification relatively needs to do two tasks. One is to modify the accumulated delay of every node in sub-tree rooted at the cycle node v, and the other is to modify the content in *T2D* table. The process of modifying the accumulated delay is from top (the cycle node v) to bottom (leaf nodes). However, the process of modifying the *T2D* table is from bottom to top layer by layer. Every node in the sub-tree T_v will look up its routing table and choose the best one candidate path from the results of the returns (*T2D* table) of its child nodes and itself. After deciding the result for every destination nodes in the *T2D* table, it will return the modified *T2D* table to its parent node for further decision. Upon the root (cycle node v) of the sub-tree T_v receiving the modified *T2D* table sent from all its child nodes, it also does the same actions described above and modifies the content in the *T2D* table. Thus, after the change of the tree topology, the final *T2D* table modified by the cycle node v can correctly reflect the optimal path information from the new tree to every remaining destination node. ■

Theorem 3: After finishing upward adjustment, destination nodes in sub-tree rooted at the cycle node v still satisfy their individual QoS constraint.

Proof: From Fig. 3, suppose a cycle was created when destination node w ($s \rightarrow u \rightarrow v \rightarrow w$) was included in the MRT. It means that $delay(s,u)+delay(\delta)+delay(v,w) \leqq \Delta_w$, but $delay(s,u)+delay(\delta')+delay(v,w) > \Delta_w$. Hence, $delay(\delta) < delay(\delta')$. For each destination node d_k in the sub-tree T_v rooted at node v, we have

$$delay(s,u)+delay(\delta)+delay(v,d_k) < delay(s,u)+delay(\delta')+delay(v,d_k).$$

That is, the delay for every destination node in the original MRT is actually reduced and hence no violation to any delay constraints is created. ■

IV. Conclusion

We have pointed out an error in Jia's algorithm [1] that the multicast routing network produced may contain cycles, i.e., it is not necessarily a routing tree, and shown that the error can be fixed with a cycle processing mechanism, including cycle detection and cycle elimination procedure. With this mechanism, the final result of our algorithm is guaranteed to be a tree (i.e., cycle free) and every destination node in the MRT must satisfy their individual QoS constraint.

Acknowledgment

The authors would like to thank X. Jia for his openness, when the error was discovered. They also would like to thank Editor-in-Chief E. Zegura for her help.

References

[1] X. Jia, "A distributed algorithm of delay-bounded multicast routing for multimedia applications in wide area networks," *IEEE/ACM Trans. Netw.*, vol. 6, no. 6, pp. 828–837, Dec. 1998.