Fault Recovery for a Distributed SP-based Delay Constrained Multicast Routing Algorithm

Hasan Ural and Keqin Zhu
School of Information Technology and Engineering, University of Ottawa
Ottawa, Ontario, K1N 6N5 Canada
Email: {ural, kzhu}@site.uottawa.ca

Abstract

This paper proposes a new distributed shortest path (SP) based delay constrained multicast routing algorithm which is capable of constructing a delay constrained multicast tree when node failures occur during the tree construction period and recovering from any node failure in a multicast tree during the on-going multicast session without interrupting the running traffic on the unaffected portion of the tree. The proposed algorithm performs the failure recovery efficiently, which gives better performance in terms of the number of exchanged messages and the convergence time than the existing distributed SP-based delay constrained multicast routing algorithms in a network where node failures occur.

1. Introduction

Multicast is the simultaneous transmission of data from a source to a group of destinations. There are many multimedia applications (e.g., video conferencing) which require multicast communication. Moreover, multimedia applications are often high bandwidth and delay sensitive, which require real-time communication with low end-to-end delays [19][16].

To meet the requirements, distributed delay constrained multicast routing algorithms have been proposed, which generates a multicast tree that satisfies the end-to-end delay constraint along the individual paths from source to each destination, and meanwhile tries to minimize the cost of the multicast tree. The cost of a multicast tree can be in terms of network bandwidth utilization.

One of the main problems with the distributed delay constrained multicast routing algorithms is that they do not take into account the changes in the topology of the network (e.g., node failures) [3]. Whenever the topology of the network changes, these algorithms will fail to complete the construction of a multicast tree and it is up to the application to re-start the algorithm. As a result, these algorithms will have a low success rate for the construction of a multicast tree (which is the number of trials to build a multicast tree successfully divided by the number of total trials). Similarly, as these algorithms do not respond to the changes in the topology of the network after the multicast tree is built, it is up to the application to re-start the algorithm to re-build the tree for the changed network topology, which causes the interruption of the running traffic for all existing members in a multicast session. The re-starting approach can be broadly considered as a kind of fault recovery approach which will be called *naïve fault recovery approach*.

In this paper, we propose a new distributed delay constrained multicast routing algorithm that takes into consideration the node failures in a network and recovers from these failures. Though we only consider the node failures in the proposed algorithm, the algorithm can be easily extended to cover other types of changes in the topology of the network such as link failures. The algorithm is shortest path (SP) based which means that it mainly uses the available shortest path information for constructing the multicast tree. The analysis and simulation results show that the proposed algorithm not only does the fault recovery correctly, but also performs the fault recovery efficiently, which gives better performance in terms of the number of exchanged messages and the convergence time than the existing distributed SP-based delay constrained multicast routing algorithms [9] in a network where node failures occur.

There are two major benefits for having such a fault tolerant algorithm: First, it will make the fault recovery actions transparent to applications during the construction of a multicast tree and during an on-going multicast session. Second, it will re-connect the disconnected sub-tree of a multicast tree without interrupting the running traffic on the rest of the multicast tree.

The rest of this paper is organized as follows. Section 2 gives more details on the background information and related work; Section 3 defines the problem formally; Section 4 describes the proposed algorithm; Section 5 discusses the performance of the proposed algorithm; and finally Section 6 gives the conclusions.

2. Background and related work

The routing function is defined as consisting of two parts [4]. The first part selects a route for the session during the connection establishment phase, and the second part ensures that each packet of that session is forwarded along the assigned route. In this paper, only the route selection algorithms are considered. During the connection establishment phase, the system selects a route along which sufficient resources ought to be reserved to meet the quality of service (QoS) requirements specified by applications, such as network bandwidth and maximum message delay. This is an important step to guarantee real-time data to be delivered to destinations with an acceptable delay.

Multicast routing is to find a routing tree (called *multicast tree*) which is rooted from a source node *s* and contains all nodes in a set *S* of destination nodes. There are two reasons for basing efficient multicast routes on trees: i) the data can be transmitted simultaneously to various destinations along the branches of the tree; and ii) a minimum number of copies of the data are transmitted, with replication of data being necessary only at forks in the tree.

There are two important requirements on algorithms for constructing multicast trees. The first is the bounded end-to-end delay along the individual paths from source to each destination where the delay bound is specified by the application performing the multicast. The second is the minimum cost of the multicast tree. The cost of the multicast tree is the sum of the costs on the edges in the multicast tree. For example, the costs on the edges in the multicast tree can be in terms of network bandwidth utilization. The minimum cost tree is called a *Steiner tree*, and the problem of finding a Steiner tree is known to be NP-complete [10][11]. The delay-bounded minimum cost tree is called a *constrained Steiner tree* (or *delay constrained multicast tree*) [12]. The problem of finding a constrained Steiner tree is also NP-complete [12].

Algorithms based on heuristics for constructing constrained Steiner trees have been developed in recent years. Some algorithms are classified as *source-based* (or *centralized*) multicast routing algorithms if the source is assumed to have all the information necessary to construct the multicast tree, such as KPP [13], CDKS [20], CKMB [21], CAO [24], and BSMA [25]; while others are classified as *distributed* multicast routing algorithms, if it is not required to maintain the entire network status information in each node, and multiple nodes are participating in constructing the multicast tree, such as DKPP [14] and DSPH [9]. Distributed routing algorithms are usually slow and complex. On the other hand, source-based routing algorithms are not practical for very large networks, since they assume complete knowledge of the network on the source node.

In addition to the algorithms mentioned above, there are also many QoS-based protocols that have been proposed in the literature, such as YAM (or spanning join) [5], QoSMIC [8] and QMRP [7]. They are receiver(destination)-initiated and based on the multiple-path approach; that is, the protocols will generate multiple paths between the existing multicast tree and a receiver, and then the receiver will pick the best one based on certain QoS criteria to connect itself into the tree. In the following, we will focus on discussing the algorithms for constructing constrained Steiner trees, which are source-initiated and do not use the multiple-path approach.

Among distributed delay constrained multicast routing algorithms, DSPH has distinguished itself from the others with its good performance in terms of tree cost, message and time complexity. DSPH can be considered as an SP-based algorithm. In an SP-based multicast routing algorithm, the multicast tree is expanded by a delay constrained shortest path from the tree to one destination node at a time until all destination nodes are covered. A *delay constrained shortest path* is the path with the minimum cost among all paths whose delays are under the delay constraint. Like other distributed delay constrained multicast tree algorithms, DSPH does not take into account the changes in the topology of the network (e.g., node failures).

Several fault recovery approaches for unconstrained multicast routing problem have been proposed in the literature. The one specified in [1][2] uses the approach in which the disconnected sub-tree is flushed and all members in the sub-tree attempt to rejoin the tree individually, which may cause a substantial increase in network traffic as the control messages are propagated through the network. The one described in [18] uses a "reversing tree edges" method to re-connect the disconnected sub-tree with the multicast tree in order to reduce the traffic of control messages, which may not always generate loop free multicast trees. Both approaches described above are for the receiver-initiated multicast routing algorithms, specifically the core based tree (CBT) protocol.

The algorithm that we study is for the source-initiated, delay constrained multicast routing problem. It should be not only fault-tolerant to node failures in the network, but also very efficient in terms of the number of exchanged messages and the convergence time. It should always generate a loop-free multicast tree. Therefore, the existing fault recovery approaches for multicast routing algorithms are neither applicable to our problem nor comparable to our fault recovery approach included in our algorithm.

3. Network model and problem definition

A network is modeled as a connected, directed graph $G=(V, E)$ where nodes in node set V represent network nodes and edges in edge set E represent links. In addition, there are two values associated with each link e ($e \in E$): delay $D(e)$ and cost $C(e)$. It is assumed that both values do not change during a multicast session. The link delay $D(e)$ is the delay a packet experiences on the corresponding link, the link cost $C(e)$ is a measure of the utilization of the corresponding link's resources. Links are asymmetrical, namely the cost and delay for the link $e = (i, j)$ and the link $e' = (j, i)$ may not be the same.

The constrained Steiner tree (CST) problem (or delay-constrained multicast tree problem) can be stated as follows. Given a network $G = (V, E)$ with n nodes, a source node s ($s \in V$), a set of m destination nodes S ($S \subseteq V - \{s\}$) called *multicast group*, and a delay constraint Δ, find a tree T ($T \subseteq G$) rooted at s that spans the nodes in S such that
i) for each node v in S, the delay on the path from s to v is bounded above by Δ; that is,
$\sum_{e \in P(s, v)} D(e) < \Delta$ where $P(s, v)$ is the unique path in T from s to v, and
ii) the tree cost $\sum_{e \in T} C(e)$ is minimized.
This tree is called a *constrained Steiner tree* (or *delay constrained multicast tree*).

Some definitions that will be used later are given below. *A shortest path from a tree T to a non-tree node v* is a shortest path (in terms of cost) from a tree node u to v, denoted by $SP(u, v)$, and u satisfies
$\sum_{e \in SP(u, v)} C(e) \le \sum_{e \in SP(k, v)} C(e)$ for any tree node k.

The cost of the shortest path from a tree T to a non-tree node v is called *the cost from a tree T to a non-tree node v*, and denoted by $C(T, v)$. A non-tree node v is said to be *the closest to a tree T* if it satisfies $C(T, v) \le C(T, w)$ for any non-tree node w.

4. A distributed multicast routing algorithm

4.1. Overview of the proposed algorithm

The proposed algorithm is built on top of the existing distributed SP-based delay constrained multicast algorithm that was proposed in [9]. Briefly, that algorithm works as follows: the multicast tree starts with a tree containing only the source node s; then the tree is expanded by a delay constrained shortest path from the tree to one destination node v in S at a time until all destination nodes in S are covered. Each time, the destination v in S which is selected to be covered next should not be in the tree yet and be the closest to the tree under the delay constraint. We call it as *Distributed Shortest Path Heuristic* (DSPH).

The ideas behind DSPH can be attributed to the TM heuristic [22] which mimics Prim's minimum spanning tree (MST) algorithm [17][6] but expands the tree path by path instead of edge by edge. However, TM heuristic does not work for the constrained Steiner tree problem and is a centralized algorithm. So, DSPH added the delay constraint checking into the algorithm and designed the algorithm as a distributed one. DSPH assumed that each node has the information about the shortest path (in terms of cost) and the delay of the shortest path to every other node in the graph through running a distributed shortest path algorithm.

DSPH is characterized by the following features. On one hand, DSPH generates the best quality of delay constrained multicast tree in terms of tree cost [9]. It is very efficient in terms of O(mn) message complexity. On the other hand, DSPH builds the multicast tree sequentially destination by destination. At each moment, there is only one node that is actively doing calculation. Let us call the node in active state as *active node*. The active node "moves" along the delay constrained shortest paths between the tree and each destination. It can be seen that this procedure is very fragile and takes a long time to complete. During this long period, network topology could be changed and DSPH will fail to complete the delay constrained multicast tree. DSPH depends on a naive fault recovery approach, which simply waits for applications to re-start the algorithm from scratch. This makes DSPH take even longer time to complete when node failures occur.

We can see that without an efficient fault recovery approach, DSPH does not work properly in practice. The motivation of our work is to design an efficient fault recovery approach which will make the recovery transparent to the applications, not interrupt the running traffic on the rest of the multicast tree, and shorten the time to recover from node failures. The proposed algorithm will recover by itself and adaptively construct a constrained multicast tree when node failures occur. So, the proposed algorithm is called *Adaptive distributed Shortest Path Heuristic* (ASPH).

In addition, we want ASPH to generate the delay constrained multicast trees whose quality is as good as that of DSPH in terms of tree cost, and perform as well as DSPH in terms of message complexity. These goals are in general conflicting with each other. For example, to be able to do fault recovery, extra messages will be needed. The key approach used in ASPH is to localize the recovery action to the failed portion of the multicast tree without re-building the whole tree again. Then, the problem is how the information about the failed sub-tree can be communicated to the active node, so that the new information can be taken into consideration in the rest of the tree construction process. One straightforward approach is to flood the information in the network. However, this approach requires O(n^2) messages just for the notification of the node failure alone. We do not take this approach. In ASPH, the failed sub-tree is refrained from flooding the network with fault information messages, and the fault information is propagated through the regular tree setup messages and scenarios as much as possible.

With respect to the main control steps of the algorithm, ASPH progresses in the same way as DSPH. Between the steps, ASPH checks if any tree node fails. When a failure is detected, it removes the sub-tree rooted at the failed node and notifies the source node *s* of the destination nodes that were covered by the removed sub-tree. Thus, the source node *s* is enabled to add these removed destination nodes later as ASPH will report back to the source node *s* when all the remaining destination nodes have been added to the multicast tree.

Furthermore, loops may be introduced in the multicast tree due to the network topology changes. ASPH will detect these loops by checking if a node (say *v*) to be added is already in the tree. A loop is removed by choosing the path from the source node *s* to node *v* which has the minimum delay. This will ensure that all existing paths between the source node *s* and destinations that go through node *v* still satisfy the delay constraint. If the new parent of *v* has the minimum delay from the source node, node *v* will accept the new parent and break itself from its previous parent. Otherwise, node *v* will reject the new parent.

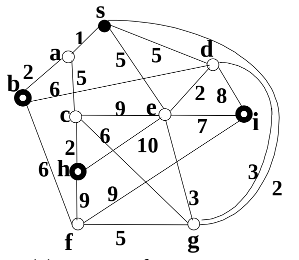
After the tree is constructed, ASPH will continue to run the tree node failure checking and recovery steps to repair the delay constrained multicast tree if node failures occur during an ongoing multicast session.

4.2. Details of the proposed algorithm

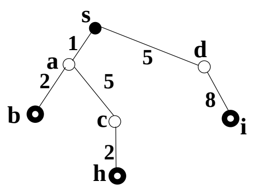
Eight types of messages are used in ASPH, which are
open - opening a multicast connection;
setup - setting up a shortest path from the tree to a non-tree node;
fork - forking a new branch from the tree node that is closest to the selected non-tree node;
completion - notifying the termination of the multicast tree setup;
break - notifying its parent to break a loop;
reject - rejecting the invitation to join the tree as either the constraint may be violated or a loop may be formed;
remove - removing a sub-tree from the tree;
destination - adding destination back to the uncovered destination list;

Among these messages described above, the first four types of messages are used in DSPH as well, while the last four types of messages are newly introduced for ASPH. It is assumed that node failures are detected by a lower level protocol.

Interleaved with the detailed description of the proposed algorithm, an example network shown in Figure 1 is used to illustrate the algorithm. For clarity of the diagrams, the same integer number is used to represent both cost and delay values. Node *s* is the source node and dark nodes *b*, *h* and *i* are destination nodes. Δ is 14 and node *e* fails after node *b* and *h* are covered.



(a) a network



(b) a multicast tree for (a)

dest	*cost*	*tree_ node*	*tag*
b	3	s	no
h	8	s	no
i	12	s	no

(c) initial T2D table

dest	*cost*	*tree_ node*	*tag*
b	3	s	yes
h	7	a	no
i	12	s	no

(d) T2D table updated by node *a*

Figure 1: An example for ASPH Heuristic

The details of the proposed algorithm are as follows:

1. When a node receives a request (*open* message) for opening a multicast connection with parameters *S* and *Δ*, it becomes the source node *s* of the multicast connection (i.e., the only tree node in the multicast tree).
   In Figure 1, when node *s* receives an *open* message, it becomes the first node in the tree.
   1.1 The source node then calculates an initial T2D (tree to destination) table. For each destination $d_i \in S$, T2D table records the following information: *cost* - the cost from the tree to d_i, *tree_node* - the tree node closest to d_i and *tag* - indicating if d_i is in the tree or not. Obviously, the *cost*, *tree_node* and *tag* fields for each destination in T2D table will be initially set to the distance of the shortest path from the source node *s* to the destination, source node *s* and "*no*", respectively.
   1.2 The destination closest to the tree is selected and its *tag* in T2D table is marked as "*yes*". A *setup* message is sent to the neighbor towards the selected destination to include all nodes on the shortest path into the tree. This setup message carries T2D table and the accumulated delay and the number of hops (called *tree level*) from the source node *s*.
   In Figure 1, node *b* will be selected as the closest destination based on Figure 1(c).

2. When a node *v* receives a *setup* message, it includes itself into the tree and modifies T2D table if a lower cost from itself to a destination is found, the destination is not yet in the tree and the delay from itself to the destination plus the accumulated delay from the source node *s* is under Δ.

   In Figure 1, when node *a* receives the setup message, node *a* will update T2D table as in Figure 1(d).

   2.1 If the addition of node *v* to the tree will introduce a loop (which means that node *v* is already in the tree), node *v* will do the following to avoid the loop:

      2.1.1 If the accumulated delay along the existing path from the source node *s* to the destination node is within the delay bound, then node *v* sends a *reject* message to the sender of the setup message. So, the sender won't include node *v* in the tree.

      2.1.2 Else if the sender node has higher or equal tree level than node *v* and the new accumulated delay from the source node *s* to node *v* via the sender is less than the old accumulated delay, then node *v* sends a *break* message to its parent to break the existing tree path and accept the new one.

      2.1.3 Otherwise, field *tree_node* in T2D for the destination under consideration will be set to the source node *s* to force the algorithm to re-build the tree path to the destination from the source node *s*.

   2.2 If node *v* is not the destination yet, node *v* simply passes the *setup* message to the neighbor towards the destination with the possibly modified T2D table and the adjusted accumulated delay from the source node *s*.

   2.3 If all destinations are included in the tree (i.e., all tags in T2D table are "*yes*"), node *v* sends a *completion* message to the source node *s*.

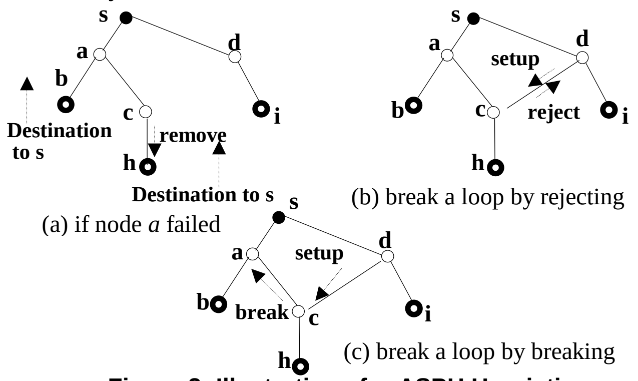
   2.4 If node *v* is the destination itself, it selects next closest destination and sends a *fork* message to *tree_node* that is recorded in T2D table. The *fork* message carries T2D table.

   In Figure 1, node *b* will send a *fork* message to node *a* to include the next selected destination *h*.

3. When node *v* receives the *fork* message, if node *v* is in the tree, the destination closest to the tree is selected and its *tag* in T2D table is marked as "*yes*". Node *v* then creates a *setup* message and forwards it to the neighbor towards the destination with T2D table and the adjusted accumulated delay and the tree level from the source node *s*. If node *v* is not in the tree, field *tree_node* in T2D for the destination under consideration will be set to the source node *s* to force the algorithm to re-build the tree path to the destination from the source node *s*. Then it re-forks as in Transaction 2.4.

   The final multicast tree for the example is shown in Figure 1(b).

4. When node *v* in the tree detects that its child node fails, node *v* removes the child node from the tree.
5. When node *v* in the tree detects that its parent node fails or receives the *remove* message, node *v* removes itself from the tree. If node *v* is not a leaf node, it will forward the *remove* message to all of its children. If node *v* is a destination node, it sends a *destination* message to the source node *s* so that the source node *s* will add the destination node back to the uncovered destination list when the merging of the uncovered destination list occurs.
6. When the source node *s* receives the *destination* message, it adds the sender node to a "sync list" which is the uncovered destination list.
7. When the source node *s* receives the *completion* message, if "sync list" is not empty, the *cost*, *tree_node* and *tag* in T2D table for each destination $d_i \in$ "sync list" will be re-set to the distance of the shortest path from the source node *s* to the destination, source node *s* and "*no*", respectively. Then it re-forks as in Transaction 2.4. If "sync list" is empty, it announces that a delay-constrained multicast tree is ready for use.



(a) if node *a* failed
(b) break a loop by rejecting
(c) break a loop by breaking

Figure 2: Illustrations for ASPH Heuristic

Figure 2(a) illustrates the procedure described in transactions 4 to 6, where *remove* and *destination* messages are sent. Figure 2(b) illustrates the case described in 2.1.1, where the *setup* message is rejected to avoid loops in the tree, while Figure 2(c) illustrates the case described in 2.1.2, where the loop can be broken by breaking the existing tree branch.

It is shown that: **a)** The delay-constrained multicast tree built by ASPH is loop free; **b)** When ASPH terminates with a multicast tree, the multicast tree is constrained; that is, the tree satisfies the delay constraint; **c)** ASPH's message complexity is O(*kmn*) and time complexity is O(*kmn*) in the worst case, where *k*-1 is the number of node failures occurred in the network during the construction of a delay constrained multicast tree and the on-going multicast session.

5. Performance analysis

In this section, we compare the performance of ASPH with DSPH in the average case under the condition that a node failure occurs during the construction of a delay constrained multicast tree or during the on-going multicast session. DSPH is going to use the naïve fault recovery approach; that is, the algorithm will be re-run from scratch when a node failure occurs. To the best of our knowledge, there are no other fault recovery approaches that have been published for the delay constrained multicast tree problem. Otherwise, we would include them in the comparison as well.

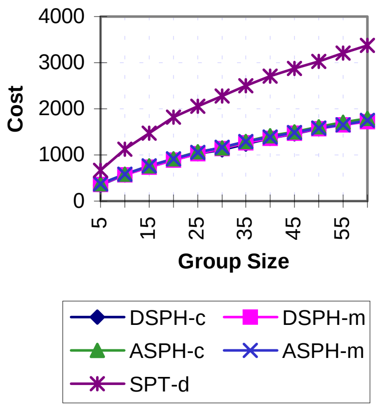


Figure 3: Tree cost versus group size when node failure occurs

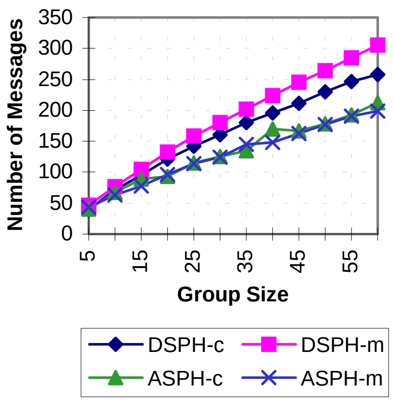


Figure 4: Number of messages versus group size when node failure occurs

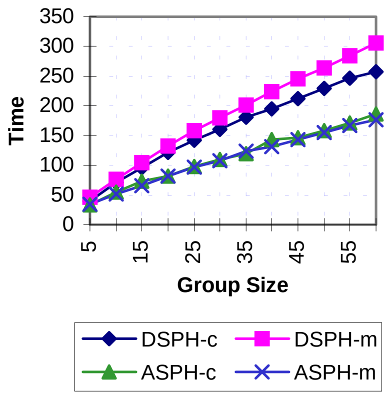


Figure 5: Time versus group size when node failure occurs

In order to compare the performance of ASPH with DSPH, a series of simulations have been performed by applying both DSPH and ASPH to networks generated by the Waxman's approach [23]: the nodes are randomly distributed over a rectangular coordinate grid. Each node is placed at a location with integer coordinates. The Euclidean metric is then used to determine the distance between each pair of nodes. A link between two nodes u and v is added with a probability that is given by the function $P(u, v) = \beta \exp(-d(u, v)/ \alpha L)$, where $d(u, v)$ is the distance from u to v, L is the maximum distance between any two nodes, and $0 < \alpha \leq 1$, $0 < \beta \leq 1$. Larger values of β result in graphs with higher link densities, while small values of α increase the density of short links relative to longer ones. The cost of a link is assigned to a value which is uniformly distributed over the range between 0 and 60. The delay of a link (u, v) in the graph is the distance between nodes u and v on the rectangular coordinate grid. Graphs are generated and tested until the graph is a two-connected network, which has at least two paths between any pair of nodes. The random graphs do have some of the characteristics of an actual network. It has been shown by simulation [15] that the performance of a multicast routing algorithm when applied to a real network is almost identical to its performance when applied to a random two-connected network.

In the simulations, node failures are injected into networks in order to see how the multicast routing algorithms perform in a network where node failures occur. There are two types of experiments: one is for the case when node failures occur during the construction of a multicast tree, and the other is for the case when node failures occur during the on-going multicast session. For the first type of experiments, it is assumed that at most one node failure may occur during the construction of the multicast tree. The timing for a node failure is randomly selected so it could occur randomly among the different stages of the construction of the multicast tree. The failed node is randomly selected among the nodes in the multicast tree built so far that are neither the source node nor the destination nodes when node failure occurs, since the failure of the source node means that there will be no multicast tree to be built and the failure of a destination node means that the constructed multicast tree will not be comparable with other multicast trees which cover all destination nodes. DSPH will be re-run when a node failure occurs.

The number of messages exchanged, the convergence time and the cost of the multicast tree are measured by their average value in a total of 100 simulation runs on a network with 200 nodes. Note that an exchanged message will only be counted once from its sender to its receiver no matter how many intermediate nodes are walked through as long as the algorithm running on the node does not interpret the message. Meanwhile, the convergence time is counted by taking one message exchange as a time unit. However, within one time unit, there may be several message exchanges occurring in the network. Thus, multiple messages exchanged within the same time unit will only be counted once in the convergence time.

For the second type of experiments, DSPH will have to be re-run to re-build the entire multicast tree when a node failure occurs during an ongoing multicast session. In the experiments, one multicast tree node will be randomly selected as a failure node during a multicast session for the type of networks that we study. Like the first type of experiments, the number of messages exchanged, the convergence time and the cost of the multicast tree are measured by their average value in a total of 100 simulation runs on a network with 200 nodes.

As DSPH will be re-run when a node failure occurs while ASPH will always return a multicast tree no matter whether a node failure occurs or not, ASPH has a better success rate than DSPH.

Figure 3, Figure 4 and Figure 5 show the simulation results when Δ is set to $d_{\max}$ + (3/8)$d_{\max}$, where $d_{\max}$ = max({d_u | for any $u \in S$: d_u is the delay on the shortest path from s to u}), and the group size changes between 5 and 60 in 200-node networks. In the figures, suffix "-c" means during the construction of a multicast tree, suffix "-m" means during the on-going multicast session and "SPT-d" means the delay-based SPT. The delay-based SPT could be considered as the delay constrained multicast trees without any optimization on tree costs.

Figure 3 shows that the costs of the trees generated by ASPH are almost identical to those by DSPH. This result is very encouraging. DSPH calculates the multicast tree based on the consistent current network topology information after it re-runs while ASPH might use different network topology information for different parts of a delay constrained multicast tree. Intuitively, it could be expected that the costs of the trees generated by ASPH should be noticeably higher than those by DSPH. But, the simulation results show that it is not the case. It is also shown in Figure 3 that the delay-constrained multicast tree algorithms generate trees with much better cost performance than the algorithms without considering optimization on the tree cost such as SPT-d. This means that it is worth using the delay-constrained multicast tree algorithms rather than using SPT-d directly.

Figure 4 shows that the number of messages required by DSPH is up to 20% more than that required by ASPH during the construction of a delay constrained multicast tree, and is up to 55% more than that required by ASPH during the on-going multicast session. This result is surprising as we know that doing fault recovery normally costs extra number of messages. Intuitively, one would expect that because DSPH is so efficient on using messages, any fault recovery approach that tries to merge the list of uncovered destinations in the failed sub-tree with the list of uncovered destinations in the active node will have a worse number of messages than DSPH even though DSPH has to run twice. Contrary to this expectation, ASPH has a significantly better message performance than DSPH. This is due to the fact that ASPH uses the approach that refrains from sending messages on node failure. The previous analysis shows that the delay on sending node failure messages does not have a negative effect on the quality of the generated multicast trees.

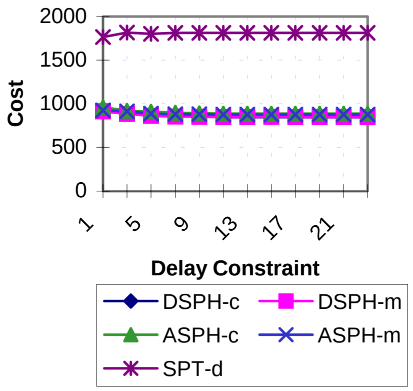


Figure 6: Tree cost versus delay constraint when node failure occurs

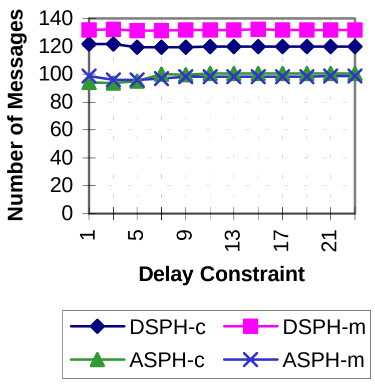


Figure 7: Number of messages versus delay constraint when node failure occurs

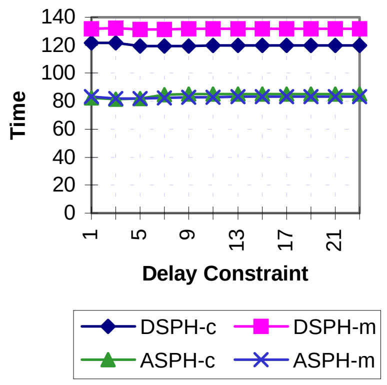


Figure 8: Time versus delay constraint when node failure

Figure 5 shows that the convergence time required by DSPH is up to 50% and 75% more than that by ASPH. This result confirms what has been expected. Through the localized recovery approach taken by ASPH, it is expected that the convergence time can be reduced by ASPH.

When the group size is fixed at 20 in 200-node networks and Δ varies between $d_{\max}$ + (1/8)$d_{\max}$ and $d_{\max}$ + (23/8)$d_{\max}$, where $d_{\max}$ = max({d_u | for any $u \in S$: d_u is the delay on the shortest path from s to u}), Figure 6, Figure 7 and Figure 8 give the corresponding results. Figure 6 shows that the costs of the generated trees by two algorithms are almost identical. Also, confirming what has been expected, when the delay constraint is tight, the cost of the generated tree is slightly higher than that when the delay constraint is relaxed since the algorithm has less options to optimize the tree. Figure 7 and Figure 8 show that the message and time performance for ASPH is much better than that for DSPH. For DSPH, they show that the tighter the delay constraint, the higher the number of messages and time. This could be due to the fact that when the delay constraint is tight, DSPH tends to fork more tree paths directly from the source node. However, ASPH behaves contrary to DSPH. This may be due to the localized recovery approach used in ASPH, which tends to try less alternatives to expand the tree under tighter delay constraints.

In conclusion, ASPH performs better than DSPH in terms of the number of exchanged messages and the convergence time in addition to the advantage that ASPH has local recovery from the node failures without re-building the entire tree. Re-building the entire multicast tree will cause the interruption of the multicast session, which is avoided in ASPH. Meanwhile, the costs of the generated multicast trees by two algorithms are almost identical, which means that the quality of the multicast trees generated by ASPH is as high as that generated by DSPH even though ASPH conducts the fault recovery actions.

6. Conclusions

In this paper, we proposed a new distributed SP-based delay constrained multicast routing algorithm which takes into account the changes in the topology of the network. The proposed algorithm can recover from node failures during the construction of a delay constrained multicast tree, and during an on-going multicast session without requiring the rebuilding of the entire multicast tree. Furthermore, compared with the existing distributed SP-based delay constrained multicast routing algorithm DSPH that uses the naïve fault recovery approach, the proposed algorithm gives better performance in terms of the number of exchanged messages and the convergence time when applied in a network where node failures occur.

Acknowledgment

The authors would like to thank Dr. G. Luo of Nortel Networks for many insightful discussions, and Prof. X. Jia at City University of Hong Kong, Prof. D.S. Reeves at North Carolina State University, and Prof. B.M. Waxman at Southern Illinois University for providing their simulation software to us. This work is supported in part by the Natural Sciences and Engineering Research Council of Canada under grant number OGP0000976.

References

[1] A. Ballardie, “Core based trees (CBT version 2) multicast routing protocol specification,” RFC 2189, Internet Engineering Task Force, September 1997.

[2] A. Ballardie, B. Cain and Z. Zhang, “Core based trees (CBT version 3) multicast routing protocol specification,” Internet Draft draft-ietf-idmr-cbt-spec-v3-01, Internet Engineering Task Force, August 1998.

[3] F. Bauer and A. Varma, “Distributed algorithms for multicast path setup in data networks,” *IEEE/ACM Trans. Networking*, vol.4, no.2, April 1996, pp.181-191.

[4] D. Bertsekas and R. Gallager, *Data Networks*, Second Edition, Englewood Cliffs: Prentice-Hall, 1992.

[5] K. Carlberg and J. Crowcroft, “Building shared trees using a one-to-many joining mechanism,” *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 1, January 1997, pp. 5-11.

[6] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, Cambridge, Ma: MIT, 1992.

[7] S. Chen, K. Nahrstedt, and Y. Shavitt, “A QoS-aware multicast routing protocol,” *IEEE J. Select. Areas Commun.*, vol.18,no.12, December 2000, pp.2580-2592.

[8] M. Faloutsos, A. Banerjea, and R. Pankaj, “QoSMIC: Quality of service sensitive multicast internet protocol,” in *Proceedings of ACM SIGCOMM’98*, Sept. 1998, pp.144-153.

[9] X. Jia, "A distributed algorithm of delay-bounded multicast routing for multimedia applications in wide area networks," *IEEE/ACM Trans. Networking*, vol. 6, Dec. 1998, pp. 828-837.

[10] R.M. Karp, “Reducibility among combinatorial problems,” in *Complexity Computer Communications*, R.E. Miller and J.W. Thatcher, Eds., New York: Plennum, 1972.

[11] L. Kou, G. Markowsky, and L. Berman, “A fast algorithm for Steiner trees,” *Acta Informatica*, no.15, 1981, pp.141-145.

[12] V.P. Kompella, J.C. Pasquale, and G.C. Polyzos, "Multicasting for multimedia applications," in *Proc. IEEE INFOCOM’92*, 1992, pp.2078-2085.

[13] V.P. Kompella, J.C. Pasquale, and G.C. Polyzos, "Multicast routing for multimedia communication," *IEEE/ACM Trans. Networking*, vol.1,no.3, June 1993, pp.286-292.

[14] V.P. Kompella, J.C. Pasquale, and G.C. Polyzos, "Two distributed algorithms for the constrained Steiner tree problem," in *Proceedings of the Second International Conference on Computer Communications and Networking*, June 1993, pp.343-349.

[15] C.A. Noronha and F.A. Tobagi, “Evaluation of multicast routing algorithms for multimedia streams,” in *Proceedings of IEEE International Telecommunication Symposium*, August 1994.

[16] Sanjoy Paul, *Multicasting on the internet and its applications*, Kluwer Academic Publishers, 1998.

[17] R. Prim, “Shortest connection networks and some generalizations,” *Bell Syst. Tech. J.*, vol.36, Nov.1957, pp.1389-1401.

[18] L. Schwiebert and R. Chintalapati, “Improved fault recovery for core based trees,” *Computer Communications* 23, 2000, pp.816-824.

[19] H.F. Salama, D.S. Reeves, and Y. Viniotis, "Evaluation of multicast routing algorithms for real-time communication on high-speed networks," *IEEE J. Select. Areas Commun.*, vol.15, no.3, April 1997, pp.332-345.

[20] Q. Sun and H. Langendoerfer, “Efficient multicast routing algorithm for delay-sensitive applications,” in *Proceedings of 2nd international Workshop on Protocols for Multimedia Systems (PROMS’95)* , 1995, pp.452-458.

[21] Q. Sun and H. Langendoerfer, “An efficient delay-constrained multicast routing algorithm,” *Journal of High-Speed Networks*, vol.7, no.1, 1998, pp.43-55.

[22] H. Takahashi and A. Matsuyama, “An approximate solution for the Steiner problem in graphs,” *Mathematica Japonica*, vol. 24, no. 6, 1980, pp. 573-577.

[23] B.M. Waxman, "Routing of multipoint connections," *IEEE J. Select. Areas Commun.*, vol.6, no. 9, Dec. 1988, pp. 1617-1622.

[24] R. Widyono, “The design and evaluation of routing algorithms for real-time channels,” Technical Report TR-94-024, Tenet Group, University of California at Berkeley, 1994.

[25] Q. Zhu, M. Parsa, and J.J. Garcia-Luna-Aceves, "A source-based algorithm for delay-constrained minimum-cost multicasting," in *Proc. IEEE INFOCOM'95*, 1995, pp.377-385.