ARTICLE IN PRESS

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Computer Communications xxx (2006) xxx–xxx

computer communications

www.elsevier.com/locate/comcom

Explicit routing in multicast overlay networks

Torsten Braun [a,*], Vijay Arya [b], Thierry Turletti [b]

[a] *University of Bern, Neubrückstrasse 10, CH-3012 Bern, Switzerland*
[b] *INRIA, 2004 route des Lucioles, B.P. 93, F-06902 Sophia Antipolis, France*

Received 10 February 2005; received in revised form 16 February 2006; accepted 21 February 2006

Abstract

Application Level Multicast is a promising approach to overcome the deployment problems of IP level multicast by establishing delivery trees using overlay links among end systems. This paper presents algorithms to support traffic engineering, to improve the reliability of multicast delivery, and to facilitate secure group communications. First, we introduce the so-called backup multicast tree algorithm to compute a set of $n - 1$ backup multicast delivery trees from the default multicast tree. Each backup multicast tree has exactly one link of the default multicast tree that is replaced by a backup link from the set of available links. The algorithm can calculate this set of trees with a complexity of $O(m \log n)$, which is identical with the complexity of well known minimum spanning tree algorithms. The so-called reduced multicast tree algorithm is based on the backup multicast tree algorithm and can calculate a tree from the default multicast tree by removing a particular node and by replacing the links of the removed node. Using the algorithms trees can be calculated individually by each of the nodes but it requires global topology knowledge. We therefore discuss distributed versions of the algorithms.

Keywords: Multicast; Overlay networks; Explicit routing

1. Introduction

Application level multicast also known as end system multicast or overlay multicast has become a very popular research topic during the last years due to the deployment problems of IP multicast. Typically, application level multicast approaches apply similar concepts as IP multicast such as running multicast routing protocols and building multicast delivery trees, but with the difference that these operations are performed on application rather than on network level. Application level multicast avoids multicast deployment problems in the Internet and can be used to bypass routes established by underlying routing protocols that do not consider the current load or congestion level for the routing decision. Application level multicast is based on the establishment of overlay networks. Multicast packets are forwarded between the end systems via such overlay networks.

Mechanisms for explicit path selection are not included in most multicast distribution concepts. With explicit path selection, the sender of a multicast packet can explicitly select the distribution path (usually a tree) of a single multicast packet. This allows a sender selecting individual multicast trees for each single packet in order to react on events such as link breaks, node failures, congested links, and group member leaves. We propose that a sender of a multicast packet can select a backup multicast tree instead of the default multicast tree by inserting a fixed size identifier to the multicast packet. A multicast delivery tree is typically established by multicast routing protocols in case of IP multicast and by peer-to-peer protocols in case of application level multicast. Such a multicast delivery tree is then used for the distribution of multicast data. The selected backup multicast tree can then be used to immediately react on link failures without any delay caused by reestablishing a new multicast delivery tree for the new topology. Load balancing can be achieved by using different trees simultaneously and

[*] Corresponding author. Tel.: +41 31 631 4994; fax: +41 31 631 3261.
E-mail addresses: braun@iam.unibe.ch (T. Braun), Vijay.Arya@sophia.inria.fr (V. Arya), Thierry.Turletti@sophia.inria.fr (T. Turletti).

0140-3664/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2006.02.022

can be applied when a particular link of the default multicast tree becomes congested or for increasing throughput.

Another usage of explicit path selection might be preventing particular nodes of a multicast group to receive a multicast message. This might be useful in secure group communications. In such cases, group keys must be updated whenever a node joins or leaves [1]. For joining nodes, the new key can be distributed along the old multicast delivery tree and by explicitly sending the new key to the joining member. However, in case of a leaving node, the old multicast delivery tree can not be used, because the leaving node would receive the multicast message with the new group key. A naïve approach for key distribution is to distribute the group key via point-to-point connections between the root generating the new key and the individual group members, but this approach is not scalable. Other tree based and hierarchical approaches form sub-groups within the group and distribute the new group key along the sub-group trees. Only a small part of the sub-groups must be re-established for a joining member and most of the established multicast distribution trees for the various sub-groups can be used for efficient key distribution [1,2].

In Section 2 we review related work on explicit path selection and on application level multicast, in particular concepts to improve reliability. Section 3 introduces our concept of explicit path selection in multicast overlay networks and presents an appropriate signaling protocol. The proposed algorithm for constructing $n-1$ backup trees out of a single default multicast tree is described in Section 4. Also the complexity and performance of this algorithm is evaluated. We will show that the complexity for calculating $n-1$ backup multicast trees for each of the links of the default multicast tree has a similar complexity as calculating a single minimum spanning tree. The result is confirmed by an implementation of the backup multicast tree algorithm. This algorithm is one of the key contributions of this paper. Section 5 presents an algorithm for constructing a multicast delivery tree, but with the condition that a particular node is removed from the default multicast tree. This algorithm is based on that one introduced in Section 4 and can be used to calculate additional n alternative multicast delivery trees. Section 6 presents our proposed encoding scheme to specify within a multicast packet, which of the alternative multicast delivery trees shall be used for multicasting the packet. The encoding scheme is based on a cardinal representation of trees and is another main contribution of the paper. Section 7 presents distributed versions of the algorithms presented in Sections 4 and 5. The signaling protocol introduced in Section 3 is extended accordingly. Section 8 concludes the paper.

2. Related work

2.1. Explicit path selection

Explicit path selection can be implemented by explicitly specifying the nodes to be traversed, e.g., using the routing header in IPv6 [3] or by describing the multicast group member addresses in explicit multicast [4]. Since specifying all the nodes to be traversed does not scale for large multicast groups, it has been proposed for small groups only. As an alternative, packets can be marked with a unique path identification (ID) such as a label like in multi-protocol label switching (MPLS) [5]. The label must be assigned with a certain path using label distribution protocols, which adds significant overhead in terms of signaling bandwidth and delay. MPLS-like approaches add hard states to the involved protocol entities.

In contrast to MPLS, the BANANAS concept [6] provides path IDs for IP level unicast forwarding without introducing a special signaling protocol. The path ID is derived from a link state database, which must be known in advance within a routing domain, and is encoded as a concatenation of local link IDs of the routers to be traversed. Four bits are sufficient per router with up to 15 interfaces. In this case, a 128 bit path ID can encode paths with a length of up to 32 hops. In order to eliminate the signaling overhead, BANANAS proposes to calculate alternative paths in a distributed manner such that each node calculates the same set of paths. The concept is based on the assumption that a limited set of possible alternative paths can be calculated in reasonable time. It is proposed that each node i calculates the k_i shortest paths to each destination. Since the calculation is performed at each node independently from each other, a distributed validation process in order to harmonize the calculations is required. According to [7], k shortest paths of a graph with n vertices and m edges can be calculated with complexity $O(m + n \log n + k)$.

2.2. Application level multicast

Several Application level multicast schemes have been proposed by other researchers. Some of them provide mechanisms to support load balancing or reliability in case of error situations such as node failures and broken links. However, none of them supports the delivery of multicast packets to exactly $n-1$ group members, which can be helpful for efficient key distribution in multicast groups.

Scribe [8] is built on Pastry [9], a generic peer-to-peer object location and routing substrate. Scribe generates a tree routed at a rendezvous point, which corresponds to the node with the closest node ID to the group ID of the multicast group. A node can join the group by sending a join message towards the root of the tree. Forwarding entries are created in forwarder nodes as a result of join messages. Nodes wishing to leave a group transmit leave messages, which result in removing the forwarding entries in the forwarder nodes. In case of a parent node or link failure, a node must retransmit a join message towards the tree in order to repair its branch.

Bayeux [10] is an application level multicast system on top of Tapestry [11] routing. Multicast receivers are organized in a tree with a single root. Load balancing can be achieved by replicating root nodes. A member joins by sending a join message towards the root of the tree.

The root replies with a tree message. When receiving a tree message, nodes on the path between the root and the joining member add the new member as a node they have to serve. Similarly, leave messages trigger prune messages in case a member leaves a group.

NICE [12] arranges end systems into a hierarchy. Each end system is assigned to a certain level in the hierarchy. Members of the same level are grouped into different clusters, which are controlled by a cluster leader. The cluster leader is selected such that it has a minimum distance to the other cluster members. The hierarchy is used to define different structures for control message and data delivery. Control messages are required for cluster management. A joining host is mapped to a cluster, such that it has rather close neighbors. Each member of a cluster on level n must be the head of a cluster on level $n-1$. Clusters exceeding a certain size are split or merged if the cluster size violates some upper or lower bounds.

A mechanism called Probabilistic Resilient Multicast [13] is based on forwarding data not just once but multiple times to randomly selected nodes. Obviously, this introduces some bandwidth overhead. Negative acknowledgements are used to detect losses.

Narada [14,15] constructs a multicast distribution tree in two steps: first, a mesh is established out of a set of possible links between two nodes based on continuous performance measurements between two nodes. This mesh is then the basis for the spanning tree construction in the second step by applying a reverse shortest path mechanism. Spanning trees are constructed for each potential sender in order to optimize trees for each source. In case of node or link failures, new overlay links need to be added to the mesh. This results in some delay to repair a failure.

The HostCast protocol [16] establishes an overlay data delivery tree and a corresponding control mesh. Both cover all group members. Reliability is achieved by establishing specific secondary links between nodes and their grandparent nodes as well as their uncle nodes. Broken links of the primary data delivery tree can then be replaced by the secondary links. HostCast requires establishing a control mesh. Moreover, the number of secondary links is quite large ($>2n$).

Network Coding and distribution of specially encoded multicast streams over a redundant multicast graph has been proposed in [17] The concept not only increases the throughput that can be achieved by distributing data to receivers that are reachable via various paths. If different streams have to pass the same links, they can be combined using network coding mechanisms. Each node has two redundant paths to the source, but this does not protect from arbitrary link failures.

3. Explicit routing in multicast overlay networks

3.1. Overview

The concept of explicit routing in multicast overlay networks as introduced in this paper proposes path IDs for multicast data delivery. This allows selecting a certain multicast tree for explicit delivery of a multicast packet. We propose to select one multicast delivery tree for application level multicast packet distribution from a set of up to n trees, where up to $n-1$ backup multicast trees are constructed from the default multicast delivery tree. Each of the $n-1$ backup multicast trees has $n-2$ links in common with the default multicast tree and differs in exactly one link. A sender of a packet can then choose among the n trees to distribute the packet. The chosen tree must be identified by an ID within each multicast message. Since we assume a limited ID space, we have to limit the set of possible trees among which a sender can choose. We present the backup multicast tree algorithm that calculates the $n-1$ backup multicast trees belonging to a single default multicast delivery tree. Based on this algorithm we present the reduced multicast tree algorithm computing multicast delivery trees that include all nodes of a group except a single particular node to be removed from the multicast group. For each node to be removed its links will be replaced, if possible by links calculated for the $n-1$ backup multicast trees mentioned above.

Alternatively, a tree that is as disjoint as possible to the default multicast tree could be calculated. However, such a single disjoint tree can not be used for the construction of trees that exclude particular nodes. Moreover, our algorithm for the backup multicast trees minimizes the number of backup links that are required to build the $n-1$ backup multicast trees.

Since we believe that application level multicast will be a promising basis for future multicast services and applications, we develop our concept within this context. For our work, we assume some kind of overlay network such as CAN [18], Chord [19], RON [20] or Tapestry [11] on top of which the application level multicast protocol can run. Our concept is independent of the underlying protocols. We only assume that the underlying protocols establish a mesh of links between nodes, not necessarily a full mesh. A certain connectivity is also required to be able to identify backup links that shall replace the links of the default multicast tree in certain conditions.

Our concept is independent from specific applications and could support applications such as streaming, audio/video conferencing, games and computer-supported collaborative work. It is based on the calculation of a spanning tree for multicast data delivery. Spanning trees can be used for both any source multicast and source-specific multicast. Scalability is limited by the overhead to distribute topology information and to compute a spanning tree based on this information. However, by introducing hierarchical structures as proposed in NICE [12], the concept should be able to support large numbers of group members. We therefore think that our proposed mechanisms could be integrated into NICE. However, the concept should also be applicable to other protocols that are generating spanning trees for multicast data delivery, e.g. Narada [14].

3.2. Signaling support

The mechanism for computing backup multicast trees can be used in three modes:

1. ***Independent mode***. Each node must get a complete view of the multicast overlay topology, in order to calculate the backup multicast trees independently from any other node. This is a similar requirement as in [6], where each node needs the complete knowledge of a domain's topology. In our case, each multicast overlay node performs exactly the same algorithm and computes exactly the same set of backup multicast trees for a given default multicast tree. The algorithm is described in Section 4. In order to get this complete overlay topology view, the exchange of topology information is required. In this section, we propose a simple signaling protocol that supports the distribution of topology information.
2. ***Distributed mode***. The exchange of complete topology information can be avoided by a distributed version of the proposed mechanism. This allows reducing the amount of exchanged information. It also allows a node to know only the local neighborhood, but it requires a sophisticated signaling protocol, which is tailored to support the backup multicast tree algorithm. This protocol is described in Section 7 in more detail.
3. ***Central mode***. The algorithm can also be used at the root of the multicast delivery tree only. In this case, only the root calculates an appropriate tree for multicast data delivery and specifies the tree using a self-describing specification of the multicast tree in the multicast data packet. This can be done using a cardinal representation as described in Section 6. However, such a cardinal representation might exceed the space available for a tree description and might be applicable to limited group sizes only.

An important task to enable multicast data distribution is the management of a multicast group and the multicast delivery tree establishment. We propose to use a simple protocol in order to exchange complete topology information among the multicast overlay network and to support the independent mode (1.) as described at the beginning of this sub-section. The signaling protocol makes use of three signaling messages: join, leave and tree. Those signaling messages can be encrypted or signed depending on security requirements.

The ***join*** message is sent by any node that wants to join the multicast group. The join message contains information about the connectivity of the new member to other peers and is forwarded towards the root of the multicast tree. In order to limit the join implosion problem in case of a large group, we can have multiple root nodes that individually serve a certain subset of multicast member nodes. In that case, these root nodes have to organize themselves on a higher level such that each root node gets all multicast packets sent to the multicast group. Naturally, the root nodes of the various sub-trees perform the same protocol but just one level higher. This two-tier architecture also corresponds to peer-to-peer networks with *super peers*. Super peers are peers with special characteristics such as higher access network bandwidth or higher processing power. They are ideal candidates to serve as root nodes for sub-trees. Such a two-tier architecture as depicted in Fig. 1 can also preserve the scalability of the approach in case of large groups. Note that similar as proposed in [12] more than two levels can be formed. In that case, a node must only know the topology of its own sub-tree.

In response to a join message, the root sends a ***tree*** message to the group members possibly after the root has checked whether the node is allowed to join the group. The purpose of the tree message is to inform the other group members about newly joined nodes and to update the connectivity information of that peers. We assume that the exchange of tree messages ensures that the peers are always aware of the connectivity within the multicast overlay network. If the overlay network does not provide information about the quality of the links, the nodes might measure parameters like round trip times or available bandwidth to other nodes themselves. In case of a super peer based network, each peer only needs to know the connectivity of the peers belonging to the same sub-tree served by a super peer. After receiving the tree message each node updates its information about the peer-to-peer network. It also calculates the alternative multicast delivery trees, i.e. the default multicast tree, e.g. using a minimum spanning tree algorithm, and the backup multicast trees using the backup multicast tree algorithm presented in Section 4. The alternative multicast delivery trees are assigned to some unique tree ID as discussed in Section 6. Each multicast message must carry this tree ID. A node receiving a message can derive from the tree ID and the knowledge about the topology how to forward the message.

If a peer node wants to leave a group, it sends a ***leave*** message towards the root. In this case, the root updates its member list as well as the overlay network topology and sends a tree message to the group in order to update

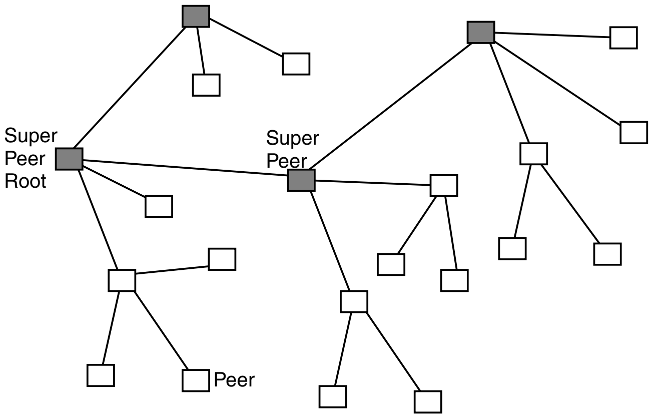

Fig. 1. Two-tier application level multicast architecture.

the group membership and topology information. All group members have to update their group membership and topology information accordingly and have to recalculate the alternative multicast delivery trees including their tree IDs.

Tree messages are sent in response to join or leave messages, but should also be sent periodically (typically in the range of several seconds or tens of seconds as usual in link state routing protocols) in order to refresh group membership and topology information. In case of a refresh the tree messages can use an already existing multicast delivery tree and can be sent by the root with the corresponding tree ID. The same is true in the case of a leave message. The leaving node should take this tree message as an acknowledgement that the root has received the leave message. For the other nodes, the tree message serves as a message to indicate that a node has left the group. Recalculation of the alternative multicast delivery trees and updating the tree IDs should occur after forwarding the tree message to the next node. If the tree message is sent in response to a join message, it can be sent using a previously used tree ID identifying the multicast delivery tree without the new node. In addition, a separate tree message with complete group membership and topology information can be unicast to the new peer.

The transmission of a multicast message can be performed always via the root node in order to allow admission control for group messages. If an incoming message contains an unknown tree ID, the message should be forwarded as a broadcast message to the neighbor nodes of the peer. Broadcast messages should be kept in memory for a certain duration in order to detect and discard duplicated broadcast messages.

4. Backup multicast trees

4.1. Overview

The main motivation of explicit multicast routing is to enforce packet forwarding along pre-selected alternative paths. Alternative paths can be used in several situations such as link failures, congestion, and leaving nodes. The corresponding multicast delivery trees need to be calculated in advance and each node must be able to assign a tree ID.

We assume that n nodes (vertices) of an overlay network are interconnected by a mesh of m links (edges). For a full mesh with n vertices, the number of edges is $m = n\,(n-1)/2$. However, usually application level multicast approaches do not establish full meshes, but only certain links between nodes. For example, Narada [14] proposes to select the best links that fulfill certain quality requirements. Moreover, since overlay networks are established between nodes behind firewalls, we have to assume that not each node can connect arbitrarily to any other node. On the other hand, in most approaches each node tries to establish a certain number of links to other nodes. This also improves the reliability of the overlay network.

Out of the finally resulting mesh a huge number of possible multicast delivery trees could be calculated. We assume that multicast delivery trees are spanning trees consisting of n vertices and $n-1$ edges. Since the tree ID is limited to a certain size, we need an algorithm that restricts the number of possible multicast delivery trees. We propose to restrict this number to n and to compute $n-1$ backup edges that can replace each of the $n-1$ edges of the default multicast tree in case of link breaks or congestion situations. The basic idea of our approach is compute a backup edge from the set G–T (G: set of edges of the graph, T: set of default multicast tree edges) for each edge of the default multicast tree T. Replacing each of the $n-1$ edges of T by its corresponding backup edge results in $n-1$ backup multicast trees. Note that a single edge can serve as backup edge for more than one default multicast tree edges. The default multicast tree and the $n-1$ backup multicast trees result in n alternative multicast delivery trees that can be used for a delivery over a multicast overlay network.

In this section, we present the algorithm for modes 1 and 3 as described at the beginning of Section 3.2. We assume that each node knows the default multicast tree. There are several ways how to build such a default multicast tree. It can be built based on multicast routing protocols or by using a minimum spanning tree algorithm. Assuming Prim's minimum spanning tree algorithm [21] with a complexity of $O\ (m \log n)$, one could naïvely calculate $n-1$ minimum spanning trees for the $n-1$ graphs G–e_i with $e_i = \text{edge } i$ of the minimum spanning tree, $i = 1,\ldots,n-1$. The complexity for calculating $n-1$ backup multicast trees is $O(m\,n \log n)$ in this case. Our intention is not to calculate backup multicast trees with optimal link weights, but those that can be calculated at low cost. Another goal is to determine a rather small set of edges that can serve as backup edges for the edges of the default multicast tree. Moreover, we select backup paths in such a way that they have minimum overlap with the default path.

The idea for the proposed algorithm to calculate the $n-1$ backup multicast trees for a given default multicast tree is taken from the observation that one broken edge of the default multicast tree can either be repaired with a single replacement of this edge by a backup edge that is not in the default multicast tree or it can not be repaired at all. A backup edge can repair all other edges of the default multicast tree with which the backup edge forms a cycle. For example, in Fig. 2 edges (C, E), (E, I), (I, M), (M, N), (C, H), and (H, L) can be repaired by edge (N, L) or (L, N). In this case, edges (N, L) or (L, N) generate a cycle with the other edges that can be repaired by that edge. Also, (J, L) forms a circle with the edges (A, C), (C, H), (H, L), (A, B), (B, D), (D, F), and (F, J). Again (J, L) or (L, J) can repair all these edges by a single replacement. From these examples we also see that both (C, H) and (H, L) can be repaired by either (N, L) or (J, L). Our algorithm selects the backup edge that has the lowest match with the path to be repaired. This is motivated by the goal to circumvent the edge to be replaced as far as pos-

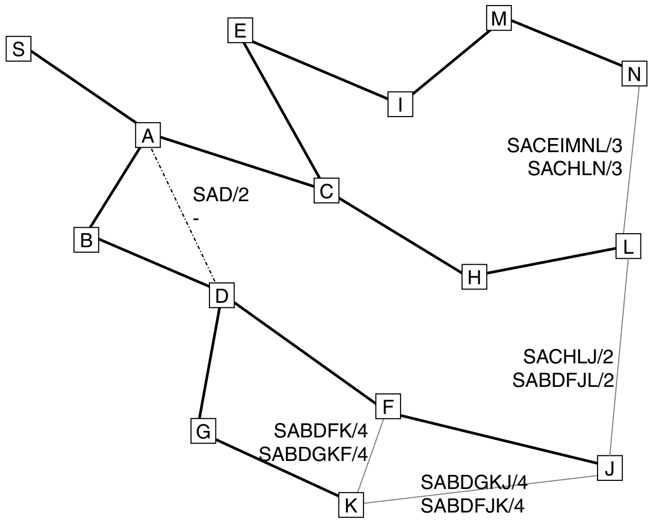

Fig. 2. Default multicast tree with backup edges.

sible. In our example, (H, L) will be replaced by (J, L), but not by (N, L), because (J, L) generates the path SABDFJL from S to L, while (N, L) generates the path SACEIMNL. SABDFJL has only the first two nodes in common with the original path SACHL, while SACEIMNL has three nodes in common with SACHL. Note that in Fig. 2, edge (S, A) can not be repaired.

4.2. Backup multicast tree algorithm

In the following we describe an algorithm for uniquely determining the backup edge for each single default multicast tree edge using a C-like pseudo code. We assume that the complete graph is stored in a linked data structure of vertices with pointers to their edges and neighbor vertices as it would result from a minimum spanning tree calculation. We also assume that the default multicast tree edges are labeled as such and that each vertex has a vertex ID.

In the first part, we calculate the path from the root to each vertex in T and store this path with each vertex (path_from_root). We also store the distance (distance_to_root) of each vertex to the root. We begin with the root as first vertex and add all vertices that can be directly reached from the root to set V. The path from the root as well as the distance is stored at all the vertices that can be directly reached from the root. After processing all direct neighbors of the root, we select all the vertices of set V after each other and perform the same operations as for the root. This is repeated until all nodes of the tree have been processed.

In the second part, we calculate for each edge in G–T the resulting path from the root via the first vertex to the second vertex of the edge (backup_path). That path is called backup path for the second vertex, since it allows reaching a vertex via an alternative path other than the default path along the default multicast tree. The backup paths can be used to reach vertices in case of edge failures. For example, edge (L, N) stores backup path SACHLN (S → L, N), while (N, L) stores backup path SACEIMNL (S → N, L). Edge (A, D) stores backup path SAD (S → A, D), while edge (D, A) stores SABDA (S → D, A), which will later be detected as not valid, because A occurs twice in it. This means that a vertex can be reached via several paths: First via the path along the default multicast tree and in addition via several backup paths. For example, node L can be reached via the default multicast tree, but also via a backup path via J and another backup path via N. For all backup paths we also determine at which node the backup path and the path along the default multicast tree begin to differ. For example, node L can be reached via the backup path SACEIMNL (S → N, L) and by the path along the default multicast tree SACHL (S → L). These two paths begin to differ in the fourth vertex (E vs. H), but the first three vertices (SAC) are identical. Therefore, we store the value of 3 (also called common path length) with the backup path (SACEIMNL/3) at (N, L). Fig. 2 shows the result of the first two parts of the algorithm.

In the third part, we copy the backup paths from the edges in G–T to the leaf edges of the default multicast delivery tree. After the copy operation we extend the backup path by that node of the edge that is closer to the root of the tree. If a leaf edge connects to two edges of G–T, we only keep one backup path, in particular that path with the lowest common path length. For example, edge (H, L) connects to two edges in G–T: (N, L) and (J, L). We only keep backup path SABDFJLH/2 from (J, L), but do not keep SACEIMNLH/3 from (N, L) due to the lower common path length at (J, L)h. This approach selects among several alternative backup paths that one with the lowest number of common nodes shared with the path from the root to a node along the default multicast delivery tree. We consider that backup path as the best choice. The inner while loop of part 3 is executed twice, once for all edges in T and once for all edges in G–T. Only the "best" backup path is copied from edge *f* to the considered edge *e*. After processing the inner while loop twice, the considered edge becomes labeled. If the upstream edge of $e(g)$ then only has labeled downstream edges, *g* is added to set E. The set E contains all edges that are ready to be processed by the inner while loop.

Algorithm.

```
vertex_set N, V;
vertex root, n, v, x;
edge_set E, F, G, T;
```

```
edge e, f, g, h;
V := {root}; root.path_from_root := (root); root.distance_to_root := 0;
while (V != ∅) /* part 1 : O(n) */
{
  v := first_element(V); V := V - v;
  N := {all vertices x | edge (v, x) ∈ T};
  while (N != ∅) {
    n := first_element(N); N := N - n;
    n.path_from_root := (v.path_from_root, n);
    n.distance_to_root := v.distance_to_root + 1;
    V := V + n;
  }
}
/* here, for all vertices x: x.path_from_root */
/* describes the path from the root to x, */
/* x.distance_from_root describes the number of */
/* hops from root to x */
F := G-T;
while (F != ∅) {/* part 2: O(m log n) */
  f := first_element(F); F := F - f;
  f.backup_path := (f.x.path_from_root, f.y);
  x := (lowest common parent of f.x and f.y);
    /* we apply binary search: complexity O(log n)*/
  f.backup_path_common := x.distance_to_root + 1;
}
/* e.x, e.y are the vertices of e (e.x -> e.y) */
/* at this point each edge e from G-T stores the */
/* path from the root to e.x plus edge e.y */
E := {edges of T with leaf vertices};
/* leaf vertex: vertex with degree 1 */
while (E != ∅){/* part 3: O(m)*/
  e := first_element(E); E := E - e;
  e.backup_path = ();
  e.backup_path_common = MAXINT;
  for (j := 2; j > 0; j–){
    if (j == 2)
      F := {all edges f | f ∈ T && f.x == e.y};
    else
      F := {all edges f |f ∈ G-T && f.y == e.y};
    while (F != ∅){
      f := first_element(F); F := F - f;
      if ((e.y is not twice in f.backup_path) & &
        (f.backup_path_common
          < e.backup_path_common)
        e.backup_path := (f.backup_path, e.x);
        e.backup_path_common := f.backup_path_common;
    }
  }
  e.labelled := TRUE;
  if (all edges g ∈ T with g.x == e.x
    are labeled) {
    h := (edge ∈ T with h.y == e.x);
    E := E + h;
  }
}
```

Fig. 3 shows the state after copying the backup paths from the edges in G–T to the leaf edges. We consider all the leaf edges as labeled. After that the copy operations are performed on the other edges of T, which are not yet labeled.

Fig. 4 shows the final result of the algorithm. After processing leaf edges in the first round, edges (D, F), (D, G), (C, H), and (I, M) are processed in the second round. Basically, the backup paths are copied towards the root of the tree. For edge (D, F) the backup path copied from edge (K, F), i.e. SABDGKFD/4 is copied, but becomes removed, because the other backup path (SACHLJFD/2) from edge (F, J) has a lower common path length (2 vs. 4). In a later round, edge (A, C) is considered. From the backup paths copied from edges (E, C) and (H, C) only the one from (H, C) is appropriate (SABDFJLHCA/2). The other one from edge (C, E) SACHLNMIECA/3 contains C twice and must therefore be removed. When edge 1 (S, A) is considered, all backup paths from edges (A, C) and (A, B) contain A twice. Therefore, (S, A) can not be repaired and is marked as not repairable.

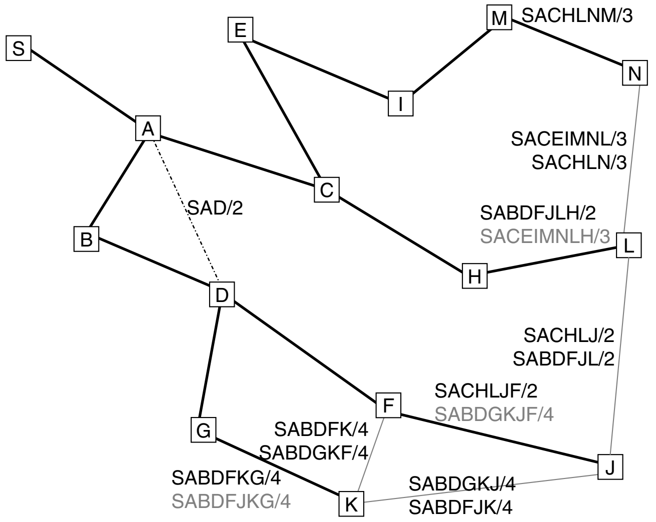

Fig. 3. Result of first round of backup multicast tree construction.

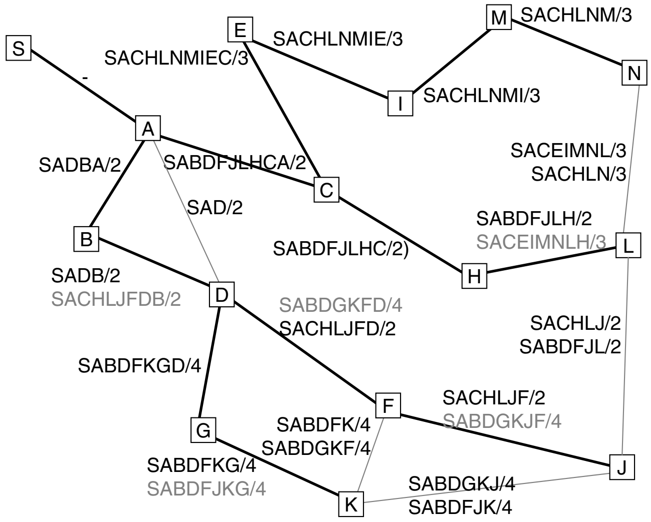

Fig. 4. Final result of backup multicast tree construction.

4.3. Complexity analysis

4.3.1. General case

The complexity of the first part is $O(n)$. We basically calculate for each edge $(x, y) \in \mathrm{T}$ the path from the root of the tree to y and the distance of y to the root.

The while loop in the second part is executed for each edge in G–T, i.e. with $O(m)$. The determination of the lowest common parent can be performed in $O(\log\ n)$ steps, if we apply binary search. For example, when we calculate the lowest common parent of vertices F and K, we have to consider the paths from the root to F and K, respectively. These paths are SABDF and SABDGK. The lowest common parent is D and D's distance to the root is 3. In our example, where the lowest common parent for edge (F, K) is calculated, we could have selected position 3 (B), then position 5 (G vs. F), and finally position 4 (D). Applying binary search, searching the lowest common parent of the two nodes of an edge has a complexity of O (log D), with $\mathrm{D} =$ the depth of the default multicast tree. Since $\mathrm{D} \leqslant n$, the total complexity O (m log n).

In the third part of the algorithm the inner loop is executed once for each edge in G. This results in a complexity of O(m). This also means that the overall complexity of the backup multicast tree algorithm is $O(m \log n)$.

4.3.2. Binary tree with full mesh

We now assume that the default multicast tree is a complete binary tree and the graph is a full mesh, i.e. each node is connected to any other. For the complexity analysis we assume that we perform the determination of the lowest common parent by sequential search in the paths starting at the root of the tree. We now take one edge (x, y) from G and perform the comparison described above for the two paths $\mathrm{S} \to x$ and $\mathrm{S} \to y$. The probability that the two paths along the binary tree to the two nodes x and y differ at the second node is at least 1/2. This means that for $m/2$ edges (x, y) of G a single basic comparison operation (comparison whether two nodes are different) is sufficient to find a difference in the two paths $\mathrm{S} \to x$ and $\mathrm{S} \to y$. The probability that the two paths along the binary tree to the two nodes x and y are equal at the second node but differ at the third node is at least 1/4. The probability that the two paths along the binary tree to the two nodes x and y are equal at the i^{th} node but differ at the $i+1^{\mathrm{th}}$ node is at least $(1/2)^{\mathrm{i}}$.

In case of a complete binary tree for the default multicast tree and a full mesh for the complete graph an upper limit for the total number of path comparisons to find a difference in the two paths $\mathrm{S} \to x$ and $\mathrm{S} \to y$ for all m edges (x, y) of the mesh is given by the following formula:

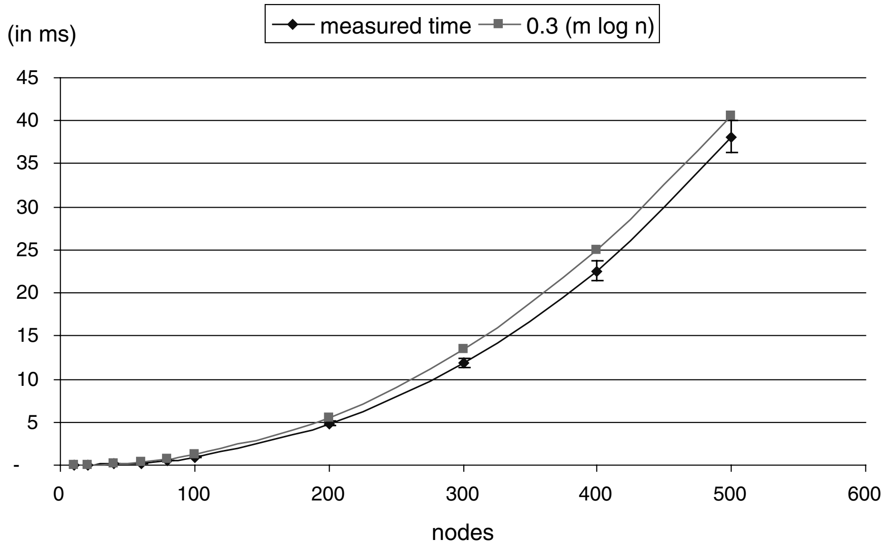

Fig. 5. Backup multicast tree performance (each node has connections to 20% of the other nodes).

$$\frac{1}{2}\cdot m\cdot 1+\frac{1}{4}\cdot m\cdot 2+\frac{1}{8}\cdot m\cdot 3+\frac{1}{16}\cdot m\cdot 4 + \cdots + \frac{1}{2^{D-1}}\cdot m\cdot (D-1) = m\sum_{i=1}^{D-1}\frac{i}{2^i}$$

$$\lim_{D\to\infty} m\sum_{i=1}^{D-1}\frac{i}{2^i} = m\sum_{i=1}^{\infty}\frac{i}{2^i} = m\sum_{j=1}^{\infty}\sum_{i=j}^{\infty}\frac{1}{2^i} = m\sum_{j=1}^{\infty}\frac{1}{2^{j-1}} = m\sum_{j=0}^{\infty}\frac{1}{2^j} = m\frac{1}{1-\frac{1}{2}} = 2m.$$

This means that in a full mesh with a complete binary tree as default multicast tree the total number of comparisons to find the lowest common parent of the two nodes of any edge is limited by 2 m. The total complexity of part 3 becomes $O(m)$. In that case, calculating $n-1$ backup links can even be performed with a lower complexity than computing the minimum spanning tree.

4.4. Performance measurements

Fig. 5 shows the performance of an implementation of the backup multicast tree algorithm using gcc on an Intel™ Xeon 3.06 GHz CPU with 512 KB cache and 1 GB main memory. Random topologies with up to 500 nodes have been generated. For each node, links to 20% of other nodes randomly selected are generated ($m = 0.2\ n^2$). A topology with $n = 500$ nodes has $m = 500 * 500 * 0.2 = 50{,}000$ links. Calculating all backup links for the $n-1$ default multicast tree links takes less than 40 ms for a 500 node topology. The graph shows the measured time using 95% confidence intervals compared with the function $f(m, n) = 0.3\ (m \log n)$. We see that the curve for $f(m, n)$ is growing faster than the curve representing the computing time of the backup spanning tree algorithm. Fig. 6 shows the performance

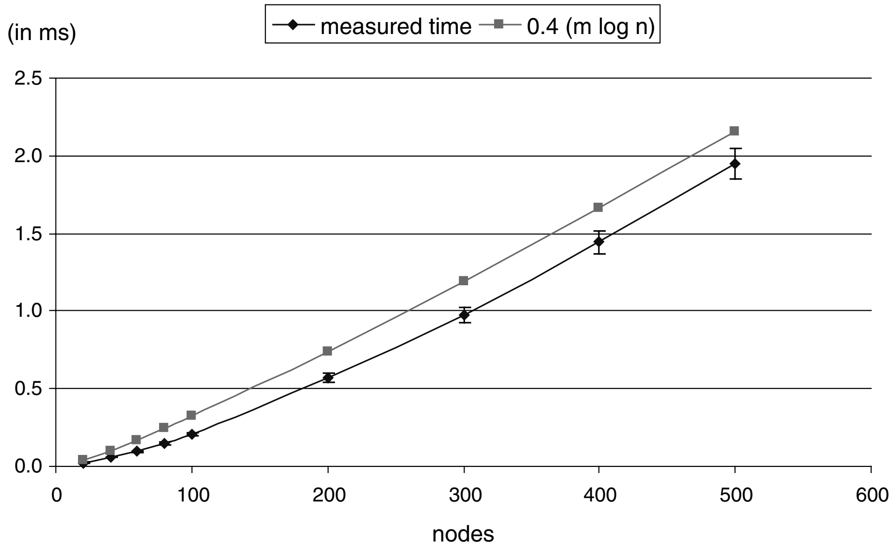

Fig. 6. Backup multicast tree performance (each node has four connections to other nodes).

when each node establishes only 4 links to other nodes ($m = 4n$). In addition, the function $f(m, n) = 0.4$ (m log n) is plotted. Again this indicates that our determined complexity of $O(m \log n)$ is correct.

Although we did not optimize the implementation, we compare the performance numbers with related work. Since we are not aware of a similar algorithm, we have to compare the performance with shortest path computation algorithms. In [22], a fast shortest path algorithm has been implemented and the processing time per edge has been presented. Values >5 µs per edge are reported, however using less powerful hardware than for our evaluation. Our implementation is running on a computer with a four times faster CPU and with four times more memory. It needs below 1 µs per edge. Note that a shortest path algorithm would have to run n times to get n backup trees.

5. Removing nodes from a multicast tree

5.1. Overview

Backup multicast trees can also support situations, where nodes leave the multicast group and new group keys need to be distributed among the remaining group members efficiently, but such that the leaving node does not receive the key. Our goal is to construct from the default multicast tree a new tree that covers all nodes except the leaving node. This new tree is also called *reduced multicast tree* hereafter. A reduced multicast tree can be derived from a single backup multicast tree, only if a node leaving the tree is not a branching point. In that case and assuming that x is the leaving node, w is the upstream node (the next node from x to the root), and y is the downstream node (the directly connected child of x), a backup multicast tree should be constructed by replacing edge (x, y) by the corresponding backup edge and by eliminating edge (w, x). Given the example of Fig. 4 and assuming that node F leaves the group, we can construct a reduced multicast tree by replacing edge (F, J) by backup edge (L, J) and by removing edge (D, F).

However, a single backup multicast tree is not able to support leaving nodes that are branching points of the tree. In the following we describe a general mechanism to construct a so-called reduced multicast tree. This tree can be used to reach all nodes of the default multicast tree, but not a single node that shall be removed from the tree. To remove a node from a default multicast tree, we have to remove the upstream edge of that node and to replace the downstream edges of that node by other links. The replacement is required to re-connect the vertices further down the tree. The replacement of different downstream edges can be supported by the reduced multicast tree algorithm presented in Section 5.2. A downstream edge can be replaced by its backup edge, if the following condition is fulfilled: The path $S \rightarrow y$ via the backup edge of (x, y) does not lead via x. It is important to note that if there exists such a backup edge that can replace the downstream edge, the algorithm presented in Section 4.2 will find such a backup edge.

If we consider again the scenario in Fig. 4 and assume that node C leaves the group and needs to be removed, the backup edge for downstream edge (C, H) is edge (J, L). The resulting path from S to H is SABDFJLH. This path meets the condition above and does not lead via the leaving node C. However, the backup edge for edge (C, E) is edge (L, N) and the resulting path from S to E is SACHLNMIE. This path does not meet our condition and leads via node C. Therefore, backup edge (L, N) is not appropriate to replace downstream edge (C, E). We have to emphasize here that if any of the vertices along this sub-tree (E, I, M, N) would have had an edge to another vertex not in the sub-tree of C, this edge would have been found as a backup edge for downstream edge (C, E) by the algorithm presented in Section 4. This means that if this sub-tree (E I, M, N) can be connected to the default multicast tree without going via C, there must either exist a connection via the other sub-tree beginning at C or it cannot be connected at all to the default multicast tree. This observation leads to the following algorithm for removing a node x from a default multicast tree and for constructing a reduced multicast tree.

5.2. Reduced multicast tree algorithm

In the first part of the algorithm given in C-like pseudo code below, we process all edges of T. If the backup path for edge (x, y) leads via vertex x, vertex y is colored red, otherwise it is colored green. If a vertex y is colored green, we can replace edge (x, y) by its backup edge for the construction of the reduced multicast tree.

In the second part, we process each edge of G–T. All these edges might be needed to connect the sub-trees of the red vertices to the reduced multicast tree. For each edge (x, y) of T we create an edge set. Then, we map each edge of G–T to one of these edge sets. An edge (w, z) is mapped to the edge set $D_{x,y}$, if x is the lowest common parent of w and z in T and if w is a child of y in T. The edges in an edge set $D_{x,y}$ are candidates to connect the sub-tree below y to the reduced multicast tree.

The third part processes all edges mapped to one of the edge sets in the second part. First, we select a node x to be removed and store all direct children of x in sets GREEN or RED depending on their color from the first part. We take one green node y after another and check whether an edge of its set $D_{x,y}$ connects to a sub-tree of a vertex in set RED. If so, the red vertex becomes green and the edge is added to set I_x (set of interconnection edges for x). At the end of the algorithm, a reduced multicast tree can be constructed for all nodes x with only green direct children y. For constructing the reduced multicast tree, we take the default multicast tree and remove all edges that include x. We add all edges of sets B_x and I_x. This results again in a spanning tree, which includes all vertices of the default multicast tree except *x*.

If a node *x* has one or more directly connected red children, it is not possible to construct a reduced multicast tree. In this case, several group members become even disconnected from the multicast group. It is not possible to build a new default multicast tree that includes those vertices with the current set of edges. New edges (overlay links) need to be established in this case.

Algorithm.

```
vertex x, y, a, b;
vertex_set RED, GREEN, Y;
edge e, f;
edge_set E, F, G, T, $\forall$(x, y) $\in$ T: $D_{x,y}$, $\forall$x $\in$ T: $I_x$, $B_x$;
E := T; /* part 1 : O(n)*/
while (E != $\emptyset$) {
  e := first_element(E); E := E - e;
  x := e.x;
  if (x is in e.backup_path) {
    e.y.color := red;
  else {
    e.y.color := green;
    $B_x$ := $B_x$ + backup_edge(e);
  }
}
E := G-T; /* part 2: O(m) */
order(E); /* order all edges according to */
          /* some predefined criteria */
while (E != $\emptyset$) {
  e := first_element(E); E := E - e;
  x := (lowest common parent of e.x and e.y);
  a := (next node from x towards e.x);
  $D_{x,a}$ := $D_{x,a}$ + e;
}
V := {all vertices of T}; /* part 3: O(m) */
while (V != $\emptyset$) {
  x := first_element(V); V := V - x;
  Y := {all vertices y |
    $\exists$ edge e in T with e.x == x, e.y == y}
  GREEN := {all green vertices of Y}
  RED := Y - GREEN;
  while (GREEN != $\emptyset$) {
    y := first_element(GREEN); GREEN := GREEN - y;
    F := $D_{x,y}$;
    while (F != $\emptyset$) {
      f := first_element(F); F := F - f;
      b := (next node from x towards f.y);
      if (b.color == red) {
        $I_x$ := $I_x$ + f;
        b.color := green;
        GREEN := GREEN + b;
        RED := RED - b;
      }
    }
  }
}
```

Fig. 7 illustrates the reduced multicast tree construction process with a given default multicast tree consisting of the solid line edges. X is the node to be removed. The sub-trees of vertices Y_1 and Y_5 have backup edges that connect their sub-trees to the default multicast tree without going via X. The backup edges are added to B_X. Therefore, vertices Y_1 and Y_5 are colored green initially, all others (Y_2, Y_3, Y_4) become red. In the second part, the various links of G–T are mapped to edge sets. For example, (W_1, Z_2) and (Z_1, Z_2) are mapped to edge set $D_{(X, Y1)}$. One of these two edges is processed first (let us assume (W_1, Z_2)) and it is discovered that this edge connects to the sub-tree of the red vertex Y_2. Y_2becomes green and (W_1, Z_2) is added to set I_X. Later edges (W_2, Z_3) and (Z_5,W_4) are also added to I_X. In order to get a reduced multicast tree for node X we have to remove all edges with X as a vertex from the default multicast tree and add the edge sets I_X and B_X.

Note that if no reduced multicast tree for node X can be constructed by the given algorithm the nodes downstream of a leaving node X will be disconnected from the graph and no tree that includes all of them exists. In this case, the underlying peer-to-peer network must solve the connectivity problem. This problem can be avoided if each node establishes a certain amount of links to other overlay nodes. It might also be helpful, if not only links to close peers are established. Otherwise, network partitioning might occur with a higher probability in case of link failures.

Instead of distributing a multicast message via a single reduced multicast tree, one could also use several trees, if a single reduced multicast tree cannot be constructed.

In the case of multiple leaves, we have to serialize the leaves and construct the reduced multicast trees accordingly. In this case, for a hierarchical scenario as depicted in Fig. 7 the algorithm can be performed concurrently in each sub-tree.

5.3. Complexity analysis

The loop in part 1 of the reduced multicast tree algorithm is performed once for each edge in T. This results in a complexity of $O(n)$. The second part has a complexity of $O(m)$, because the loop is performed for each edge in G–T. Also part 3 has a complexity of $O(m)$. The inner loop is performed once per edge in G–T. Each edge is in one of the sets $D_{x,y}$. This means that an overall complexity of $O(m)$ is required to construct n reduced multicast trees, if we assume that all backup edges are known in advance.

6. Tree IDs for multicast delivery trees

In the previous sections we have presented concepts to calculate backup and reduced multicast trees from a given default multicast tree. In particular, $n - 1$ backup multicast trees and $n - 1$ reduced multicast trees can be calculated in order to support failures or cases with leaving nodes. The idea is that the sender (the root of the tree in case of source-specific multicast) selects an appropriate multicast delivery tree among several alternatives to distribute the multicast message. We propose to calculate for each alternative multicast delivery tree a unique identification called tree ID which allows a forwarding node to discover how a message must be forwarded. Such an ID should have the following characteristics:

- The tree IDs of two consecutive default multicast trees (before and after a node joins or leaves) should differ, because some messages might be delayed and messages that shall be forwarded along different trees might be present simultaneously. Also the protocol operations presented in Section 3.2 require that a tree ID remains valid for some time after a tree has been changed.

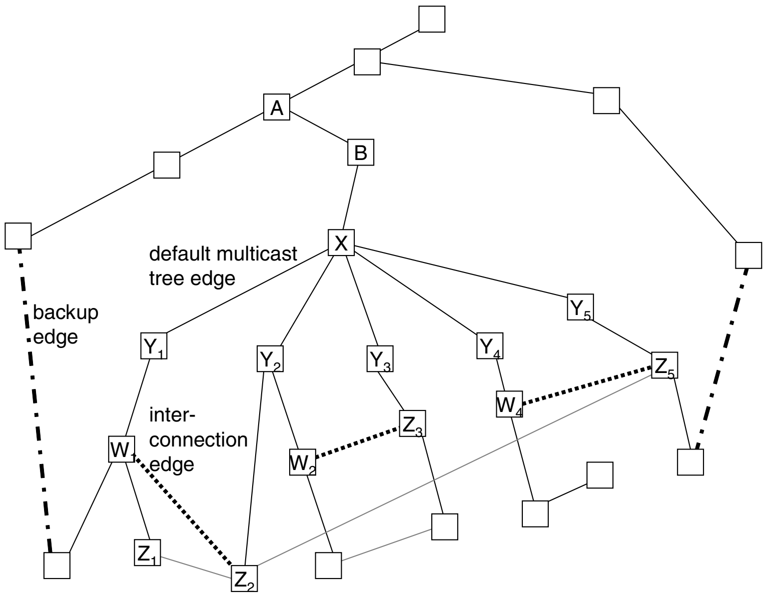

Fig. 7. Reduced multicast tree construction.

- The tree IDs of all backup and reduced multicast trees should be different in order to be able to distinguish which trees associated with a default multicast tree shall be used to distribute a multicast message.
- The size of such a tree ID should be limited in order to scale for large groups.

Based on this discussion, we propose to use three fields for specifying the selected multicast distribution tree:

1. a default multicast tree ID for specifying the currently used default multicast tree used for the multicast group,
2. a type ID specifying whether the default multicast, one of its $n - 1$ backup multicast trees or one of its $n - 1$ reduced multicast trees shall be used,
3. a node ID specifying in case of a reduced multicast tree which node shall be excluded from the default multicast tree and in case of a backup multicast tree which upstream link of the specified node shall be replaced by its backup link.

For the calculation of the default multicast tree ID we propose a combination of a cardinal representation and MD5 [23]. A link-based scheme as used in [6] is not appropriate for our case, because links among peers will change more frequently than router links. Moreover, in case of large groups a tree ID consisting of all traversed link IDs might become too large.

The default multicast tree ID is based on a cardinal representation [24,25], which encodes the structure of the tree and its IDs separately. The structure is represented using a balanced parenthesis representation obtained by a *pre-order* traversal wherein a "(" is output when we enter a node and a ")" when we leave a node. This is then combined with the pre-order traversal of the node IDs. This representation takes ($n \log n + 2n$) bits to encode a tree of n nodes. Further, it can be used to represent arbitrary sub-trees since the nodes are simply listed in pre-order. At each forwarding node, a single traversal of the encoding is sufficient to find the children and construct the encodings of the sub-trees. Thus at each node the number of bits in the encoding reduce by a significant amount. For the tree given in the example below the following path ID is calculated at root S: ((((())())(()(()))))SABGDFCKEJ. Since this path ID is variable in length and can easily become very long in large groups, we have to map it to a constant length identifier. We propose to apply a hashing mechanism on the cardinal representation and use the resulting hash value as path ID. In cases where the cardinal presentation is short enough, hashing could be avoided. Also in this case, the pre-computation of $n - 1$ backup trees or $n - 1$ reduced trees would not be required, since the sender of the multicast packet is able to completely specify the path to be taken by the multicast packet. In this case, our presented algorithms can be used to efficiently calculate alternative routes for single link breaks and node leaves. However, cardinal presentations are not limited in size and may exceed a given maximum size value [25].

Example:

```
S-----A-------C--------E----J
      ¦       ¦
      B---F   K
      ¦
      G---D
```

.

7. Distributed algorithms

For the calculation of the backup and reduced multicast trees, we assumed that each node knows the default multicast tree and the complete mesh topology. Based on this knowledge, the backup and reduced multicast trees can be computed by each node independently according to the algorithms presented in Sections 4 and 5. An issue to be investigated in this section is whether the algorithms can be performed without that each node knows the complete topology. In the following sub-sections we show that this is possible, but under the constraint that additional signaling between the nodes is introduced.

For both sub-sections we assume a more light-weight basic signaling mechanism than described in Section 3.2 based on tree, join, and leave messages. A joining node sends a join message towards the root of the multicast delivery tree. The root in turn confirms the inclusion of the joining node by a tree message and inserts the path from the root to the joining node along the default multicast tree. This way, each node can learn the path from the root node to itself.

7.1. Backup multicast tree

For the distributed algorithm of the backup multicast tree algorithm, all the nodes need to know to which other nodes they connect to and which of these links are used for the default multicast tree. The algorithm makes use of two additional signaling messages in addition to tree, join, and leave messages:

- Backup Path Establishment (BPE)
- Backup Path Termination (BPT)

The protocol begins with the transmission of BPE messages over links that are not part of the default multicast tree. Initially, the node originating a BPE message will put the IDs of the nodes along the default multicast tree path from the root to itself into the BPE message. We assume that each node has learnt that path before, e.g. by a tree message.

A leaf node receiving a BPE message will then select from all received BPE messages that one that includes the best backup path as defined for the backup multicast tree algorithm in Section 4. The node will append its own ID to the backup path in the BPE message and forward it to its upstream node towards the root of the default multicast tree. A forwarded BPE indicates that a node is part of a backup path in order to replace a link. For all BPE messages that are not selected by a leaf node, a BPT message is sent back towards the originator of the BPE message. The node creating a BPT message is also called terminating node hereafter. BPT messages are forwarded in the reverse direction as the corresponding BPE messages until the final destination (the originating node of a BPE message) has been reached. BPT messages contain the recorded path information from the root via the node originating node of the BPE message to the terminating node. If a node receives a BPT message it can learn from the included backup path, for which links the backup path (and the backup link) have been selected. In particular, these are all the links between the nodes originating and terminating the BPE message.

A node, which is not a leaf node in the default multicast tree, will receive BPE messages from links that are not part of the default multicast tree, but also from its downstream nodes via default multicast tree links. Processing of BPE messages is performed in exactly the same way as in the case of leaf nodes. Forwarding of BPE messages to upstream nodes shall be performed periodically and a node has to consider BPE messages received from all other links.

For example, we consider Fig. 4. Node L issues a BPE message with path SACHL to node N. N forwards the BPE message via M, I, and E to C. C terminates this BPE message and returns a BPT message including the backup path SACHLNMIEC. Backup link (L, N) can serve as backup link for the default multicast tree links between C and L. These are (C, E), (E, I), (I, M), and (M, N). Later, any node between the root and C might detect or be informed (via additional failure notifications) that there exists a problem on link (C, E). Such a node might simply insert a tree ID into the multicast packet indicating that the packet shall be forwarded via the backup multicast tree instead of the default multicast tree. This means that the packet shall be forwarded via the backup link (L, N) instead of link (C, E). When node C receives such a message, it does not forward the packet via link (C, E). The packet arrives also at node L, which has learned from the exchange of BPE and BPT messages that link (L, N) serves as backup link for link (C, E). Therefore, L forwards the multicast packet via the backup link to N. Then, the packet travels to E along the branch of the default multicast delivery tree, but in the opposite direction.

The signaling overhead associated with this procedure depends on the number of edges (m). There is not more than one BPE message on each of the m links. Since the BPT messages travel exactly on the reverse path back to the originators of BPE messages, also not more than m BPT messages are generated. Note that in a dynamic environment these messages should be distributed periodically. In that case one BPE and one BPT message occur on each link per interval.

7.2. Reduced multicast tree

With the signaling protocol described in Section 7.1 each node learns the backup links and the backup paths for all links, to which it is directly connected. With this information a node $Y_i (i = 1, \ldots, 5)$ in Fig. 7 can derive, whether there exists a backup path for link (X, Y_i) that does not lead via node X. If such a backup path exists node Y_i is a green node and can be connected to the multicast tree via the backup link for (X, Y_i). If this is not the case, node Y_i is a red node and has to search for a connection to one of the other sub-trees of X that are represented by the nodes Y_i.

This search can be supported by another signaling protocol extension. Each node Y_i that can not be connected via backup links to the multicast tree has to send an ***explore*** message towards the children along the default multicast tree. The message contains the link (X, Y_i) as a parameter. The message is flooded on the complete sub-tree of Y_i, and each node on this sub-tree that has a link from (G–T) forwards the message to its neighbor. The neighbor receives this message and discards it, if X is not on the path from the root to itself. If X is on the path from the root to itself, the message is forwarded towards X and will be received by a child of X that is directly connected to X. This direct child is one of the downstream nodes, for example Y_j. If Y_j is a green node, it returns an *interconnect* message to Y_i in the reverse direction than the explore message. The message passes one link that is not in T. We call this link interconnection link. The nodes of the interconnection link learn that this link is required to reach node Y_j, if node X becomes removed and will later forward multicast packets that contain a tree ID for the reduced multicast tree for node X. After node Y_j has received the interconnect message, it becomes a green node and can also return interconnect messages for incoming explore messages from other red nodes.

In our example given in Fig. 7, Y_1 and Y_5 are green nodes, the other nodes Y_2–Y_4 are red. Y_2 sends an explore message, because it has learned that there is no backup link for (X, Y_2) that does not lead via X. The explore message is forwarded by W_2 via Z_3 to Y_3, which is a red node too and does not respond the explore message. However, the explore message is forwarded by Z_2 via Z_1 and W_1 to Y_1. Y_1 responds with an interconnect message that travels back to Y_2. W_1 learns that (W_1, Z_2) will be required to reach Y_2 if node X becomes removed. Y_2 receives the interconnect message, becomes red and may later respond to explore messages from other red nodes such as Y_3 with interconnect messages.

The signaling mechanism to support reduced multicast trees requires an exchange explore/interconnect messages for each red node of the tree. The overhead can be reduced by piggy-backing the various explore and interconnect messages that are traveling up and down the sub-tree below the node to be removed.

8. Conclusions

In this paper, we have proposed a concept for explicit routing in multicast overlay networks. In particular, it allows specifying a particular distribution tree for the transmission of multicast data. We proposed to calculate $n - 1$ backup multicast trees for a given default multicast tree (e.g., a minimum spanning tree) that interconnects the n nodes belonging to a group. The complexity for calculating these backup multicast trees is slightly above the complexity for calculating a minimum spanning tree but can be even lower with certain types of graphs. The performance measurements of the backup multicast tree algorithm implementation confirm the determined complexity. In addition, an algorithm has been presented that allows calculating a reduced multicast tree by discarding a particular node, e.g. a node leaving the multicast tree, from the default multicast tree. Unique IDs for the alternative (default, backup, or reduced) multicast trees are required in order to specify which of the trees shall be used for multicast message forwarding. An appropriate encoding scheme has been developed and discussed. We also described a distributed version of the algorithms avoiding that each node needs to be aware of the full topology. This requires introducing a light-weight signaling protocol.

Acknowledgements

We thank Prof. Shivkumar Kalyanaraman, who introduced us to the problem of explicit multicast for overlay networks. The second author has worked with him to design encoding schemes for multicast trees [25].

References

[1] C.W. Kong, M. Gouda, S. Lam, Secure group communications using key graphs, IEEE/ACM Transactions on Networking 8 (1) (2000) 16–30.

[2] S. Banerjee, B. Bhattacharjee, Scalable secure group communication over IP multicast, IEEE Journal on Selected Areas in Communications 20 (8) (2002) 1511–1527.

[3] S. Deering, B. Hinden, Internet Protocol, Version 6 (IPv6) Specification, RFC 2460, December 1998.

[4] R. Boivie, N. Feldman, Y. Imai, W. Livens, D. Ooms, O. Paridaen, Explicit Multicast (Xcast) Basic Specification, Internet Draft, work in progress, August 2003.

[5] E. Rosen, A. Viswanathan, R. Callon, Multiprotocol Label Switching Architecture, RFC 3031, January 2001.

[6] H.T. Kaur, S. Kalyanaraman, A. Weiss, S. Kanwar, A. Gandhi, BANANAS: an evolutionary framework for explicit and multipath routing in the internet, in: ACM SIGCOMM 2003 Workshop on Future Directions on Network Architectures (FDNA), Karlsruhe/Germany, August 25–27, 2003.

[7] D. Eppstein, Finding the k Shortest Paths, 35th Symposium on Foundations of Computer Science, IEEE, November 1994.

[8] M. Castro, P. Druschel, A.M. Kermarrec, A. Rowstron, Scribe: a large-scale and decentralized application-level multicast infrastructure, IEEE Journal on Selected Areas in Communications 20 (8) (2002).

[9] A. Rowstron, P. Druschel, Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-peer Systems, IFIP/ACM Middleware 2001, Heidelberg/Germany, November 2001.

[10] S. Zhuang, B. Zhao, A. Joseph, R. Katz, J. Kubiatowicz, Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination, in: 11th International Workshop on Network and Operating System Support for Digital Audio and Video, Port Jefferson/USA, June 2001.

[11] B. Zhao, L. Huang, J. Stribling, S. Rhea, A.D. Joseph, J.D. Kubiatowicz, Tapestry: a resilient global-scale overlay for service deployment, IEEE Journal on Selected Areas in Communications 22 (1) (2004).

[12] S. Banerjee, B. Bhattacharjee, Ch. Kommareddy, Scalable application layer multicast, in: ACM SIGCOMM 2002, Pittsburgh/USA, August 19–23, 2002.

[13] S. Banerjee, S. Lee, B. Bhattacharjee, A. Srinivasan, Resilient multicast using overlays international conference on measurements and modelling of computer systems, in: ACM SIGMETRICS 2003, San Diego/USA, June 9–14, 2003.

[14] Y. Chu, S. Rao, S. Seshan, H. Zhang, Enabling conferencing Applications on the Internet using an overlay multicast architecture, in: ACM SIGCOMM 2001, San Diego, August 27–31, 2001, pp. 55–67.

[15] Y. Chu, S. Rao, S. Seshan, H. Zhang, A case for end system multicast, IEEE Journal on Selected Areas in Communications 20 (8) (2002).

[16] Zhi Li, P. Mohapatra, Hostcast: a new overlay multicasting protocol, in: IEEE International Conference on Communications (ICC) 2003, Anchorage/USA, May 11–15, 2003.

[17] Y. Zhu, B. Li, J. Guo, Multicast with network coding in application-layer overlay networks, Journal on Selected Areas in Communications 22 (1) (2004) 1–13.

[18] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A Scalable content addressable network, in: ACM SIGCOMM 2001, San Diego/USA, August 27–31, 2001.

[19] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaeshoek, F. Dabek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup protocol for internet applications, IEEE/ACM Transaction on Networking 11 (1) (2003).

[20] D. Andersen, H. Balakrishnan, M. Kaashoek, R. Morris, The case for resilient overlay networks, in: Proceedings of the Eighth Annual Workshop on Hot Topics in Operating Systems (HotOS-VIII), Schloss Elmau, Germany, May 2001.

[21] R. Prim, Shortest connection networks and some generalizations, Bell System Technical Journal 36 (1957) 1389–1401.

[22] S. Pettie, V. Ramachandran, S. Sridhar, Experimental evaluation of a new shortest path algorithm, in: Proceedings of the Fourth Workshop on Algorithm Engineering and Experiments, LNCS vol. 2409, pp. 126–142, 2002.

[23] R. Rivest, The MD5 Message-Digest Algorithm, RFC 1321, April 1992.

[24] D. Benoit, E. Demaine, J. Munro, V. Raman, Representing trees of higher degree, in: Proceedings of the Sixth International Workshop on Algorithms and Data Structures (WADS'99), pp. 169–180, 1999.

[25] V. Arya, T. Turletti, S. Kalyanaraman, Encodings of multicast trees, in: IFIP Networking Conference, Waterloo, Canada, May 2–6, 2005, pp. 992–1004.

Torsten Braun got his diploma and Ph.D. degrees from the University of Karlsruhe, Germany, in 1990 and 1993, respectively. From 1994 to 1995 he was a guest scientist with INRIA Sophia Antipolis, France. From 1995 to 1997 he worked at the IBM European Networking Center, Heidelberg, Germany. Since 1998 he has been a full professor of computer science at the Institute of Computer Science and Applied Mathematics, heading the Computer Networks and Distributed Systems research group. He has been a board member of SWITCH, the Swiss national research network, since 2000. He spent parts of his sabbatical in 2004 at INRIA, Sophia-Antipolis.

Vijay Arya received his B.E. in Computer Science from University of Mysore, India in 1997 and M.Tech in Computer science from Indian Institute of Technology (IIT) Delhi, India in 2001. Since October 2001, he is a doctoral student in Planete team at Inria Sophia Antipolis, under the direction of Thierry Turletti. His research interests include peer to peer networks, measurements and modeling, and network tomography.

Thierry Turletti received the M.S. (1990) and the Ph.D. (1995) degrees in computer science both from the University of Nice - Sophia Antipolis, France. He has done his Ph.D. studies in the RODEO group at INRIA Sophia Antipolis. During the year 1995–96, he was a postdoctoral fellow in the Telemedia, Networks and Systems group at LCS, MIT. He is currently a research scientist at the Planète group at INRIA Sophia Antipolis. His research interests include multimedia applications, congestion control and wireless networking.

Dr. Turletti currently serves on the Editorial Board of *Wireless Communications and Mobile Computing (WCMC) and Wireless Networks (WINET)* journals.