REIHE INFORMATIK
10-2002

Application-Aware Distribution Trees for Application-Level Multicast

Jürgen Vogel, Jörg Widmer, Dirk Farin, Martin Mauve
Universität Mannheim
Praktische Informatik IV
L15, 16
D-68131 Mannheim

Application-Aware Distribution Trees for Application-Level Multicast

Jürgen Vogel, Jörg Widmer, Dirk Farin, Martin Mauve
Praktische Informatik IV, University of Mannheim, Germany
{vogel, widmer, farin, mauve}@informatik.uni-mannheim.de

***Abstract*— In this paper, we present an algorithm to construct application-aware distribution trees for application-level multicast. Unlike existing approaches, the distribution trees do not solely depend on network characteristics but also on the application semantics of the transmitted packets. In the presented algorithm, the application may specify an individual priority for each packet-receiver pair. The distribution tree is then constructed such that the higher the priority, the more direct the path from the sender to the packet's destination. This comes at the cost of an increase in link stress – the more direct a path, the less likely it is that it can be integrated efficiently into an overlay multicast distribution tree. Our algorithm takes this tradeoff into account and constructs efficient application-aware distribution trees. We demonstrate the performance and characteristics of the algorithm through extensive simulation.**

***Index Terms*— Application-Level Multicast, Multicast Routing, Distribution Tree, Distributed Interactive Applications.**

I. Introduction

Application-level multicast has received considerable attention over the past few years. The key idea of existing approaches [1-5] is to realize efficient multicast message delivery by using only the end-systems as nodes in a multicast distribution tree: the construction and maintenance of the tree is done at the application level without any support from the network. Routers within the network do not have to keep state information about group membership. Furthermore, application-level multicast can be deployed immediately without any changes to the network. This eliminates two key problems that have prevented native IP multicast from being widely adopted and makes application-level multicast a promising technology for group communication.

Existing approaches for application-level multicast focus on network characteristics (e.g., latency) to construct the multicast distribution tree. As long as those characteristics remain constant and no changes in the set of session members occur, all packets from a sender will take the same paths towards the destinations. This approach is well suited when all packets should be delivered to all receivers with the same priority (e.g., in a multi-destination file transfer). However, there exists a number of applications where the priority of a packet may be different for the receivers. In addition, this priority may change from packet to packet for some or all receivers.

In a networked computer game, for example, the action of a player is very important to competing players that are close by. These players should receive information about the action with a very low delay. Other players may be able to tolerate a higher delay, depending on their location and orientation within the game. Similarly, if sensor data is transmitted by a sender, this data may typically have a low priority for all receivers, unless some extreme sensor reading occurs which requires the transmission of a packet with very low latency to some destinations in order to handle an emergency situation.

In this paper, we propose to use a combination of application-level priorities and network characteristics in order to build and maintain a multicast distribution tree. Since multicast routing is handled at the application level, integrating application knowledge into the routing decision comes natural and introduces little overhead. The general idea of our approach is to allow the sending application to assign a priority to each pair of packet and receiver. The higher the priority, the more direct will be the path that the packet takes towards its destination. The cost for reduced latency is a possible increase in link stress (i.e., the number of copies of a packet that traverse the same link). The key challenge is to find an appropriate algorithm for the construction of a multicast distribution tree which takes this tradeoff into account.

The remainder of this paper is structured as follows: in Section II we briefly outline existing approaches for application-level multicast. The algorithm for the construction of multicast distribution trees which take application-level semantics into account is described in Section III. In Section IV we discuss practical issues such as the maintenance of a distribution tree that may change on a per-packet basis and how to efficiently distribute topology information to other nodes. Section V contains an evaluation of the presented algorithm by means of simulation, and we conclude the paper and give an outlook on future work in Section VI.

II. Related Work

Typically, application-level multicast algorithms construct their distribution topologies based on path characteristics such as (end-to-end) latency, available bandwidth and packet loss rates. Their aim is to build distribution trees that minimize the additional routing overhead compared to native IP multicast.

Yoid [1] creates a single multicast tree for all end-systems that participate in a session, independently of a specific sender. Each node v selects another node as parent, preferably a node with a low network delay to v. Receivers gather a list of possible parents on basis of periodic control messages and explicit queries. An initial list can be obtained from a so-called rendezvous host during the bootstrap phase. Apart from the network delays, the maximum number of children that can be attached to a potential parent (i.e., the fan-out) is considered in the choice of a parent node. Because the list of possible parents is usually incomplete and the fan-out is constrained, the resulting distribution tree may be suboptimal. As a consequence, nodes periodically ping other session members in order to find a better parent and optimize the tree structure. An alternative method to select a parent node in Yoid is provided for the transfer of large data files: nodes connect to the parent that caches the largest amount of data.

Overcast [2] is similar to Yoid in that a distribution tree is created by having each node select an appropriate parent node. However, the Overcast distribution tree is sender-specific. The main criterion for building the tree is to maximize throughput for each receiver which qualifies Overcast mainly for the distribution of bulk data.

Instead of constructing an application-level distribution tree directly, *Narada* [3] employs a two-step process. First, a mesh is built among the participating end-systems. For the actual data transport, Narada runs a distance vector protocol with latency and bandwidth [4] as the routing metrics on top of the mesh. The resulting tree is a sender-specific shortest path tree (SPT) based on the underlying mesh. The crucial factor in this approach is the quality of the mesh that must balance the number and the characteristics of the used unicast links. If there are too many links in the mesh, the resulting distribution topology will resemble a star of unicast connections from the sender to all receivers. As in Yoid, joining end-systems obtain a list of current session members by a bootstrap mechanism and connect to one or more listed nodes. Then, members periodically add links that improve the routing performance and remove links that are rarely utilized by a distribution tree.

Like Narada, *Gossamer* [5] also employs the tree-over-mesh approach where the mesh is constructed in order to minimize latencies of the distribution tree. The number of connections that a node can maintain at a certain point in time is explicitly restricted with Gossamer in order to take bandwidth limitations into account.

Approaches where application-level semantics are used for routing can be found in the area of content delivery networks. The common idea of *Bayeux* [6], *Chord* [7], and *Content Addressable Networks* [8] is to realize a scalable lookup service for objects (e.g., end-systems) where the responsibility for managing the object space is shared equally among a network of peer nodes. The multi-hop lookup path for a target object (e.g., the receiver of a message) is determined on basis of certain properties of the (hash-generated) destination address. For example, in Bayeux the current node uses the i-th digit of an object's address to resolve the next hop towards the destination. In contrast to the previously discussed application level multicast protocols, these content delivery networks base their routing decisions (almost) exclusively on application semantics. Consequently, the resulting distribution tree may be very inefficient with respect to end-to-end delay and link stress.

III. Application-Level Multicast Routing

As shown in Figure 1, information about the network delay between end-systems may be used to deduce a certain amount of information about the actual topology of the network: when node A has a high delay to both nodes B and C, and node B has a low delay to node C, then it is likely that a packet transmitted from A to B shares a significant portion of the physical link with a packet transmitted from A to C. With this assumption, a routing algorithm that wants to minimize the resource usage might construct a distribution tree where packets are routed from A to B and then further on from B to C.

Let us now assume that the pairwise unicast delays between all participants are known and that distribution trees are constructed from the table of unicast delays.[1] Two well known types of trees that can be constructed from this information are the minimum spanning tree (MST)

[1]We will ignore that unicast routing protocols may give suboptimal routes and assume that the underlying unicast routing algorithm causes direct paths to a node to be shorter than any indirect path over intermediate nodes.

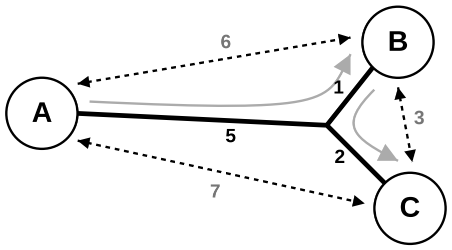


Fig. 1. Joint path to distant receivers

and the shortest path tree (SPT). The MST exhibits the desired characteristic that the resource usage for the forwarding of packets is minimal. The end-to-end delays in such a distribution tree, however, are not considered when constructing the tree and can become very large. Hence, using an MST for the distribution of packets is only reasonable when end-to-end delays are not an issue (e.g., for non-interactive data dissemination). When constructing an SPT instead of an MST from the table of unicast delays, the distribution tree will consist of separate unicast connections from the sender to each of the receivers (commonly, this would be regarded as 'normal' unicast rather than application-level multicast). The resulting end-to-end delays are optimal but the resource usage is very high.

Our aim is to construct application-aware distribution trees that balance the characteristics of MST and SPT: For each packet-receiver pair the application may provide a priority between 0 (lowest priority) and 1 (highest priority). Given this information, the path along which this packet is forwarded should gradually alter from the MST path (at a priority of 0) to the SPT path (at a priority of 1). In order to find an algorithm with this property, we first investigate well known metrics for the assessment of distribution trees. The optimization of *resource usage* leads to an MST when used as sole optimization criterion, while the optimization of the *cumulative end-to-end delay* leads to an SPT when used as sole optimization criterion. We combine those metrics by using one common application priority for the whole distribution tree. The optimization of the combined metric leads to a distribution tree where the paths to all destinations gradually change from the MST paths to the SPT paths as the application priority increases. In a second step, we generalize the metric such that one priority may be given for each destination. Its optimization leads to a tree where each path from the sender to a destination transitions from the MST path to the SPT path as the priority of the destination increases. Finally, we present an efficient algorithm which provides a very good approximation for the optimal distribution tree with respect to the last metric.

A. Distribution Tree Metrics

Let $G = (V; E)$ be a fully connected, directed graph, where $V = \{v_i\}$ denotes the set of nodes and $E = \{e_{ij}\}$ is the set of edges. We define the node $v_s \in V$ as source and the remaining nodes $R = V \setminus \{v_s\}$ as receivers. Edge weights $w(e_{ij})$ are assigned according to the delay over the corresponding link. For each distribution tree $T \subset E$, we can define two cost functions C_R and C_D:

- **Resource usage**, defined in [3] as the product of link stress and link delay, summed over all physical links of the underlying network. This sum is equivalent to the sum of all edge delays in the overlay distribution tree:[2]
$$C_R = \sum_{e_{ij} \in T} w(e_{ij}).$$
- **Cumulative end-to-end delay**, measuring the total network delays for the distribution of a packet from the source to all of the receivers. Let $r_t = \langle e_{sj_1}, e_{j_1 j_2}, \ldots, e_{j_n t} \rangle$ denote the route from the source v_s to the receiver v_t on the current distribution tree. In this case, the cumulative end-to-end delay is given by:
$$C_D = \sum_{v_t \in R} \sum_{e \in r_t} w(e).$$

The minimum cost distribution tree is equivalent to the minimum spanning tree (MST) in case costs are determined in terms of resource usage, and to the shortest path tree (SPT) in case of the cumulative end-to-end delay cost function.

B. Introducing Application-Level Semantics

For many applications, optimization of only one of the costs is not sufficient. While minimizing the total resource usage is desirable, overly large end-to-end delays reduce the utility of the application. Hence, some tradeoff between resource usage and end-to-end delay is required. Let $p \in [0; 1]$ be the application's priority with which it wants to deliver data, where 1 means that the end-to-end delay for the receivers should be as low as possible, while 0 denotes no special delay requirements. The balancing cost function is defined as follows:

$$C = (1 - p) \sum_{e_{ij} \in T} w(e_{ij}) \; + \; p \sum_{v_t \in R} \sum_{e \in r_t} w(e). \qquad (1)$$

To visualize the effect of the parameter p on the optimum distribution tree, we choose an example network as shown

[2]Given the distribution tree $A \longrightarrow B$ and $B \longrightarrow C$ in Figure 1, the resource usage in the underlying network is $5 + 2 * 1 + 2 = 9$, which is equal to the sum of edge weights in the overlay tree $6 + 3$. The link stress is implicitly contained in the end-to-end delays.

in Figure 2. The participants of the application-level multicast session are numbered from 1 to 6, while intermediate routers of the underlying network appear as unmarked nodes. The corresponding table contains the pairwise end-to-end delays for the participants. Let us assume that node 2 wishes to send a packet to the rest of the group. The resulting distribution trees that are optimal with respect to the cost function as given in the above equation are depicted in Figure 3. When p is increased, nodes farther away move up in the distribution tree, reducing the end-to-end delays to the sender, until for $p = 1.0$ a star-like shortest path tree is reached. As can be seen from the graphs, the number of possible distribution trees (that are optimal for the cost function as defined in (1)) for a small overlay network with only 6 nodes is very limited.

We can generalize the above cost function for the case of individual per-receiver priorities, where information may be of high importance to some receivers (and should therefore be delivered on a direct path) and of lower importance to other receivers. We therefore have per-node priorities $p : V \rightarrow [0; 1]$ for a sender v_s.

While the per-node priorities can easily be integrated into the C_D cost function, their integration into C_R is more difficult, since the costs are calculated as the sum of the edge weights of the tree and not per receiver. However, in an MST, the relevant cost for a receiver is the weight of the edge with which it is connected to the rest of the distribution tree. Consequently, the priority of a node can be assigned to this edge. We construct a combined cost function as a weighted sum of C_R and C_D taking node priorities into account:

$$C = \sum_{e_{ij} \in T} (1 - p(v_j)) w(e_{ij}) \; + \sum_{v_t \in R} p(v_t) \sum_{e \in r_t} w(e) \quad (2)$$

Note that this cost function specializes to C_D if $\forall v \; p(v) = 1$, and to C_R if $\forall v \; p(v) = 0$. This means that the node priorities determine the structure of the minimum cost tree with the extremes MST and SPT.

Direct optimization of this cost function is computationally complex. For the efficient minimization of the costs, we approximate the cost term in a way that allows us to directly modify edge weights and compute a minimum spanning tree based on these modified weights. In order to calculate the modified weights, the cost function needs to be based solely on the weights of the edges of the tree, and not on complete paths to individual receivers.

The general idea is to split the complete path r_t to a receiver into the last edge of the path e_{it} and all previous edges $\langle e_{sj_1}, e_{j_1 j_2}, \ldots, e_{j_n i} \rangle$. We can approximate the cost of the path from v_s to v_i with the cost of the direct edge e_{si}, where $w(e_{si})$ is a lower bound for the actual path

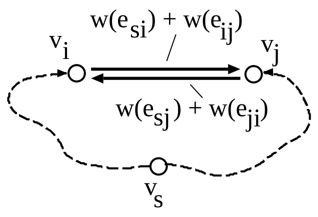


Fig. 4. MST has to be computed on the directed graph because edge weights differ for different edge orientations.

costs. This leads to a simplified approximate formulation for the global costs:

$$\begin{aligned} C &= \sum_{e_{ij} \in T} (1 - p(v_j)) w(e_{ij}) \; + \sum_{v_t \in R} p(v_t) \sum_{e \in r_t} w(e) \\ &\approx \sum_{e_{ij} \in T} (1 - p(v_j)) w(e_{ij}) + \\ &\qquad \sum_{v_t \in R, v_i \in V : e_{it} \in T} p(v_t) \big(w(e_{si}) + w(e_{it}) \big) \\ &= \sum_{e_{ij} \in T} w(e_{ij}) + p(v_j) w(e_{si}). \end{aligned}$$

The last equality follows from the property that a spanning tree of a graph has the same amount of edges as there are target nodes in the graph. Consequently, both sums are calculated over the same set of edges. In order to minimize the total cost term, we can apply a MST algorithm on the graph with modified weights. The new weights are set to

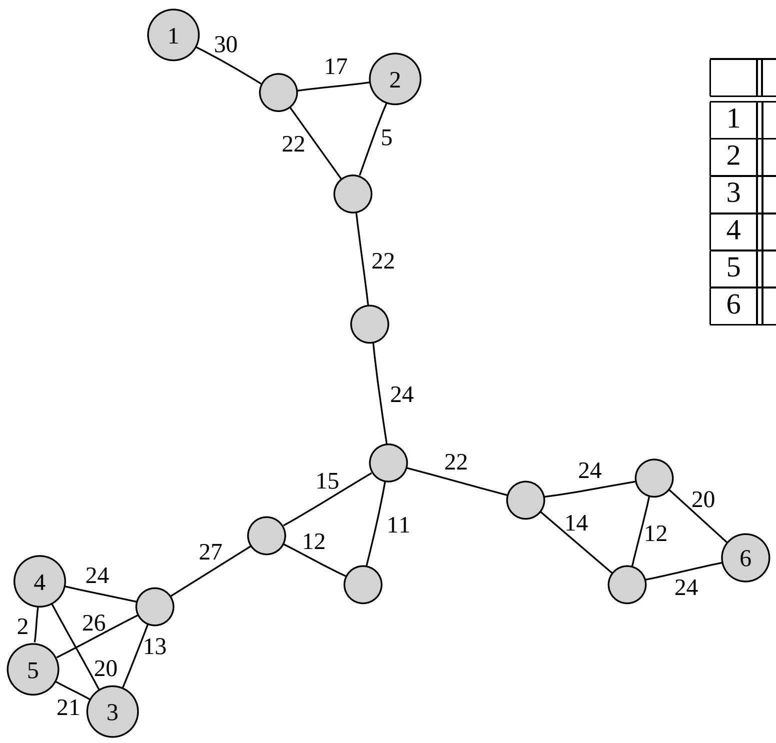
$$w'(e_{ij}) = w(e_{ij}) + p(v_j) w(e_{si}) \quad (3)$$

With increasing $p(v_j)$, indirect links to the target node v_j will become more and more expensive and eventually such links will be removed from the distribution tree. Note that a directed MST algorithm has to be applied to obtain correct results as it is not a priori known in which direction data is distributed over the edge. The costs for opposing directions may differ (see Figure 4). We term the combination of modified edge weights and directed MST computation *priority-based directed minimum spanning tree* (PST) algorithm.

Algorithms to construct MSTs in directed graphs have been described in [9], [10]. Pseudo code for the implementation that was used for the simulations can be found in Appendix A.

IV. Design Considerations

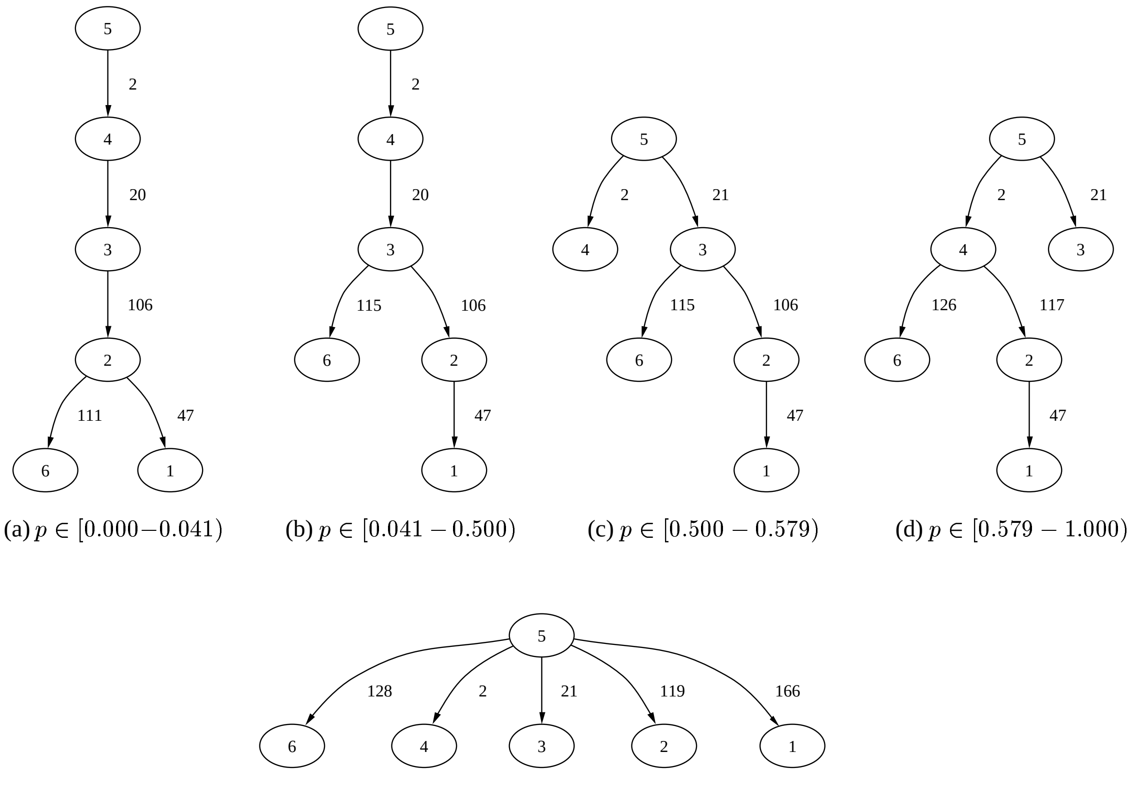
The presented PST algorithm improves the application's influence on the data distribution process through the inclusion of application semantics in the construction of the distribution tree. However, it may be very costly



	1	2	3	4	5	6
1	0	47	153	164	166	158
2	47	0	106	117	119	111
3	153	106	0	20	21	115
4	164	117	20	0	2	126
5	166	119	21	2	0	128
6	158	111	115	126	128	0

Distance table

Fig. 2. Example graph



(a) $p \in [0.000{-}0.041)$ (b) $p \in [0.041 - 0.500)$ (c) $p \in [0.500 - 0.579)$ (d) $p \in [0.579 - 1.000)$

(e) $p = 1.000$

Fig. 3. Optimal distribution trees

to recalculate the distribution tree for each packet. Moreover, nodes in a specific distribution tree need to know which other node or nodes to forward a packet to and thus require some information about the tree topology whenever the topology changes. This information has to be distributed to the nodes in an efficient way.

In this section, we will discuss how the PST algorithm can be integrated into applications while avoiding excessive calculations in the end-systems and message overhead through the distribution of topology information.

A. Maintenance of the Distribution Tree

The simplest form to maintain an up-to-date distribution tree is to rebuild the tree from scratch whenever topology information or application priorities change. This simple update mechanism can easily be improved to significantly reduce the number of necessary tree recomputations.[3]

The distribution tree will not change under the following conditions:

- The cost (delay) of a link that is not in the distribution tree (i.e., the directed MST) increases.
- The cost (delay) of a link that is in the distribution tree decreases.
- The priority for a receiver which is connected directly to the sender increases.
- The priority for a receiver which is connected indirectly via another receiver decreases.

In these cases, it is only necessary to update the link weights, but the distribution tree will not change. Furthermore, a change in receiver priorities or link delays may be too small to cause a tree change. As we will see later in the simulation section, very often only significant changes in the receiver priorities will actually result in changes in the tree topology, since the number of cost optimal distribution trees is limited.

An increase in link delay on a direct link between sender and receiver may cause the receiver to be connected through an indirect link (corresponding to a priority decrease). An increase in the delay of an indirect link may cause a node to be connected directly (corresponding to a priority increase). Similar considerations hold for a delay decrease on direct or indirect links. When computing a directed MST as specified in Appendix A, it is possible to record for each step of the algorithm by how much the cost of a link has to increase before it is excluded from the distribution tree, or by how much the cost of a link has to decrease before it will be included in the tree. With these considerations, rebuilding the tree can be limited to the cases where the tree structure will change.

[3]Note that some of the improvements in the update mechanism are only possible because the overlay graph is fully connected and because the relative weight increase on the last hop of an indirect path is based on the weight of the link from the sender to the start of the last hop link and not on the complete path to the receiver.

If changes to the tree structure are necessary, it is desirable to keep the number of updates small. When a number of link delays or priorities change simultaneously, recomputing the whole tree is reasonable. For minor changes, adjusting the existing tree can be much less costly.

Let us assume that the cost of a single link increases suffciently to cause a change in the distribution tree. We have to distinguish two cases:

- $w(e_{ij})$ increases for $v_i \in R$
- $w(e_{sj})$ increases for sender v_s

In the first case, $w'(e_{ij})$ is updated and node v_j is connected to the rest of the tree via a less expensive link. However, the link costs for all nodes in the tree below v_j as well as the tree structure remain unchanged. Because of the asymmetric links, it may be possible that it is now less expensive to connect v_i via e_{ji}, and so on. Hence, we have to reverse the direction of links on the path from v_j to v_s as long as the costs w' in the direction towards the sender are less expensive than the link costs in the opposite direction.

In the second case when the cost of a link e_{sj} from the sender increases, the change will also increase the costs of $w'(e_{jk})\ \forall v_k \in R \setminus \{v_j\}$. For all v_k with $e_{jk} \in T$, it is necessary to check whether the node can be connected to the rest of the tree via a less costly link (i.e., the rest of the tree may grow “into” the region with the increased link costs). The tree parts below the v_k will not be affected. Thus, in both cases only very limited parts of the tree will change.

The same calculations can be applied when link costs decrease. Moreover, priority changes affect the costs of all incoming links of a node but since only one of these links can be in the current distribution tree, the above statements are even valid for priority changes.

Lastly, even though a distribution tree may no longer be optimal given the current edge weights, it may be sufficient for data distribution as long as the changes are small (i.e., use a “fuzzy” update strategy where updates are triggered by significant weight changes).

B. Efficient Topology Distribution

For the forwarding process, a specific tree topology needs to be known by all nodes of the tree. Either, nodes may distribute their priority tables so that all other nodes can locally recalculate all distribution trees, or nodes may distribute the tree they already calculated. The second alternative seems much better suited for the task since the communication overhead is similar but much less calculations at the receivers are required. Furthermore, for the second alternative inconsistent delay information at the receivers will not result in routing loops.

In fact, nodes do not require the complete distribution trees, but only need to know which node or nodes to forward the packets to. This information is updated by a sender whenever its distribution tree changes. It is either possible to include the information in the data packet headers, or to send extra tree maintenance packets. The second alternative is preferable if a significant number of packets are sent along the same distribution tree.

If delays between nodes remain relatively constant and only some application priority patterns are valid, the number of different distribution trees is fairly limited.[4] In this case it may be more efficient to precompute all possible trees (or a limited subset of suitable trees), distribute this information to all the other nodes, and then only include an identifier for a specific distribution tree in the header of a data packet.

V. Simulations

We implemented a simple network simulator in order to evaluate the performance of our PST algorithm. The simulator is event-based and allows packet-level data distribution on arbitrary network topologies. A network topology is characterized by a set of nodes connected via edges with a certain delay. We do not consider other factors such as bandwidth, router load, and packet loss.

All network topologies were generated with the Georgia Tech Internetwork Topology Models (GT-ITM) [11] toolkit. The topologies use the transit-stub method without extra transit-stub or stub-stub edges. Edges between nodes are placed using the random model. To determine which nodes should act as end-systems and which as routers, we calculate how often a certain node is part of a path when connecting all nodes with each other using shortest path trees. The nodes with the least number of paths are designated as end-systems and the others as routers.

First, we evaluate the properties of priority-based directed MSTs for different network topologies when all receivers are assigned the same application-level priority. Following this, we give simulation results for realistic priorities based on a multi-player game.

[4]In our experience, even relatively large topologies can be covered efficiently by a handful of different sender-based distribution trees.

TABLE I

Routers	Links	End-Systems	Avg. # of Trees	Avg. # of Edge Changes
42	80	18	16	1.7
52	134	18	22	1.8
70	123	30	24	2.5
85	173	35	35	2.4
100	195	40	39	2.8
120	187	30	27	2.8
125	264	50	51	2.8
130	244	30	45	4.3
140	276	40	38	2.4
195	271	50	46	3.1

A. *Simulations for PSTs with a single priority*

In this section, we analyze how many distinct distribution trees are built by the PST algorithm and to what extent these trees differ. We define the application-level priority p to be equal for all receivers (according to Equation (1)) and calculate the set of trees T_i (i.e., $\forall p \in [0;1]$) for different network topologies. The results are listed in Table I. The first three columns describe the topology used in terms of the number of routers, the number of physical links, and the number of end-systems participating in the application-level multicast session. The average number of different distribution trees calculated by PST is given in column four. As can be derived from the table, this figure is correlated with the number of end-systems.

Next, we are interested in the topological difference between two successive distribution trees T_i and T_{i+1}, where T_{i+1} is the tree with the smallest priority $p_{i+1} > p_i$ with at least one changed edge compared to T_i. The average number of edge changes from one tree to the next is relatively small (see column five). Thus, the optimization of tree maintenance as described in Section IV is able to achieve a significant reduction in tree calculation costs.

B. *Simulations for a sample application*

In the following, we compare the characteristics of our PST algorithm to the delay-based MST and SPT approaches on basis of a realistic application scenario.

1) Simulation Setup: Realistic event patterns to determine application priorities for the simulations were generated by tracing a simple multi-player game [12]. In this game, each player controls a spaceship which can accelerate, decelerate, turn, and shoot at one another with a laser beam of a certain range. The rectangular game field allows players who approach one edge of the field to reappear at the opposite side. Each spaceship has a predefined

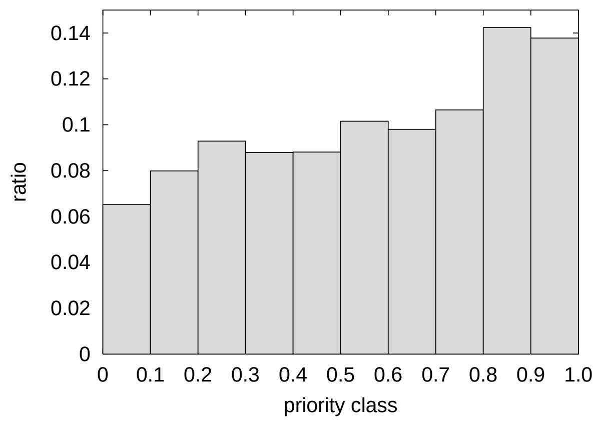


Fig. 5. Distribution of application priorities

amount of hit points: each time it is hit by a laser beam, one of the hit points is subtracted. If no hit points remain, the spaceship is removed from the game. User actions together with timestamps and information about the current game state were recorded for games with six and eighteen players. In our network simulation, each recorded user action led to one packet exchanged among end-systems.

The application priorities $p(v_i) \in [0;1]$ used for the tree building algorithm are based on the relative positions between the spaceships and their orientations. If the spaceship i of a player is within shooting range of another player's ship s, the end-system v_s of s sets $p(v_i) = 1$. We define that i is within shooting range of s if the distance between i and s is less than the maximum range of the laser beam and s is oriented in such a way that it can hit i after conducting at most one turn operation. The priority for all spaceships j outside the shooting range of s is calculated depending on their distance $d(s,j)$ to the sender: $p(v_j) = 1 - \frac{d(s,j)}{d_{max}}$, where d_{max} is the maximum distance that is possible on the game field.

A typical distribution of priorities for a game session with six players is depicted in Figure 5. Priorities close to 1 are common because the objective of the game is to score points by shooting other players and hence players will cluster together instead of spreading out evenly on the game field.

2) Simulation Results for a Small Topology: To evaluate the characteristics of our priority-based tree-building algorithm (PST), we compare it to the MST and the SPT, respectively. For the simulations, we use two network topologies of different sizes.

The first simulation scenario is based on a game session with six players. The session lasted for 140 seconds and during that time span a total of 2630 events were issued. The priority distribution that resulted from the spaceships' positions is depicted in Figure 5. Figure 2 shows the underlying network topology with end-to-end delays between 2 and 166 ms and an average value of 100 ms.

The delay properties of a specific distribution tree can be measured using the cumulative weighted end-to-end delays C_D^p (which is a modified version of C_D, including priorities):

$$C_D^p = \sum_{v_t \in R} p(v_t) \sum_{e \in r_t} w(e),$$

where R is the set of receivers and r_t is the path from the sender to a receiver v_t as determined by the corresponding tree. Figure 6(a) depicts the distribution of C_D^p for the SPT, the MST, and the PST, respectively. By definition, the SPT routing algorithm results in the best distribution of C_D^p, with 90% of all trees having a C_D^p of less than 440 ms. However, the difference between SPT and PST is comparatively small (12 ms at 90%), meaning that the end-to-end delays in the distribution trees constructed with the PST algorithm are on average only marginally higher than the delay on the direct paths. In comparison, distribution trees created with the MST algorithm result in a significantly higher difference for C_D^p (75 ms at 90%).

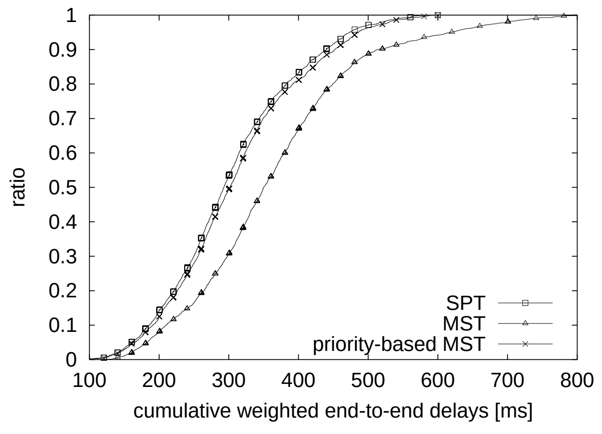
The receiver-specific end-to-end delays, defined as $\sum_{e \in r_t} w(e)$, resulted in the following 99% confidence intervals for this simulation scenario: SPT $[98.8; 101.2]$, MST $[118.1; 121.3]$, and PST $[103.9; 106.4]$.

The relative delay penalty (RDP) [3] is a measure for the optimality of the end-to-end delay:

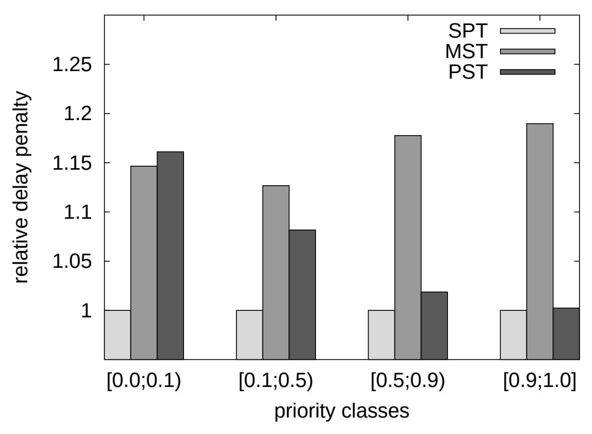
$$RDP(t) = \frac{\sum_{e \in r_t} w(e)}{w(e_{st})}.$$

The RDP compares the end-to-end delay of a receiver v_t to the smallest possible delay (i.e., the unicast delay from v_s to v_t). By definition, for the SPT distribution trees $RDP(t) = 1 \ \forall v_t \in R$. Figure 6(b) shows the average RDP values for different priority classes. In the case of the PST algorithm, the RDP decreases continuously with increasing application-level priorities from 1.16 for receivers v_t with $p(v_t) \in [0.0; 0.1)$ to 1.002 for v_t with $p(v_t) \in [0.9; 1.0]$. For application instances with a high priority, a delay close to the unicast latency can be achieved. The maximum range of the average RDP is relatively small (0.16) since only six end-systems participate in this simulation scenario and the distribution trees have paths with at most four hops.

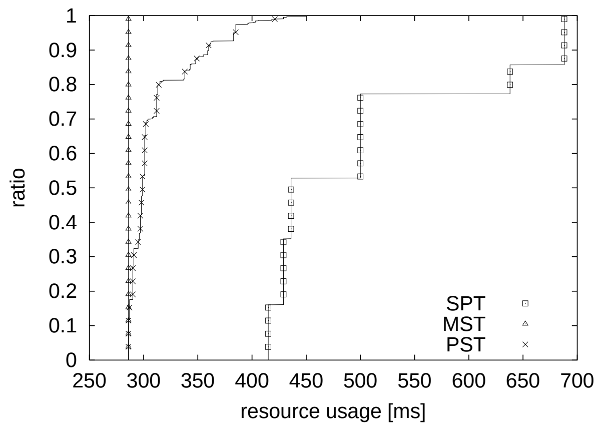
The load on the network caused by a certain distribution tree can be measured using the resource usage metric C_R as defined in Section III. C_R takes into account that more than one identical copy of a packet may be sent over the same physical link. The distribution of C_R is depicted in



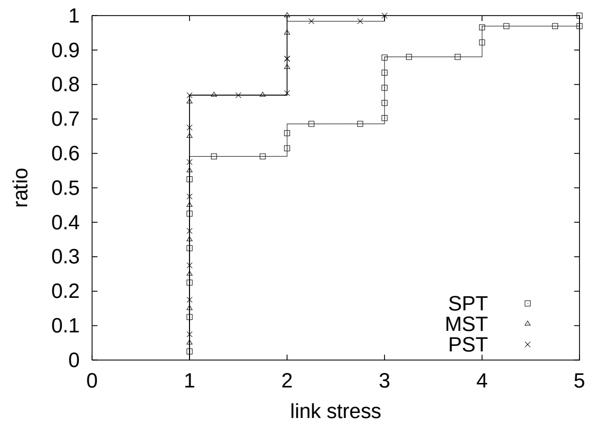
(a) Distribution of cumulative weighted end-to-end delays



(b) Average relative delay penalty



(c) Resource usage distribution



(d) Link stress distribution

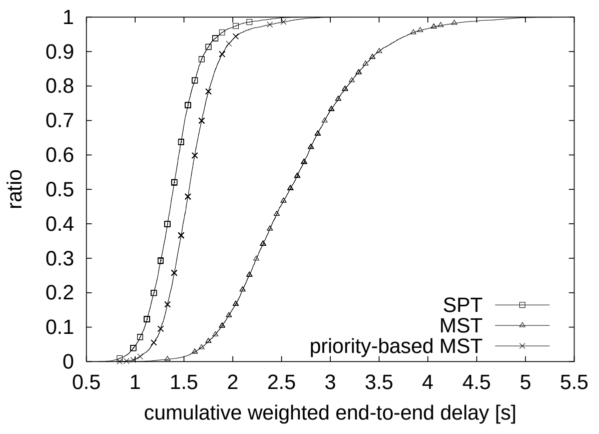
Fig. 6. Simulation results for the small topology

Figure 6(c). The MST algorithm always selects the same set of edges e_{ij} for its trees, independently of the source node. Thus, the MST's C_R has a constant value of 286 ms which is at the same time the lower bound for the resource usage of the other algorithms. 70% of all distribution trees built by the PST algorithm have a C_R between 286 ms and 307 ms which is close to the optimum and far better than the values obtained by SPT. Hence, the optimization of end-to-end delays for certain application instances by the PST causes only a slight increase in resource usage when compared to the MST.

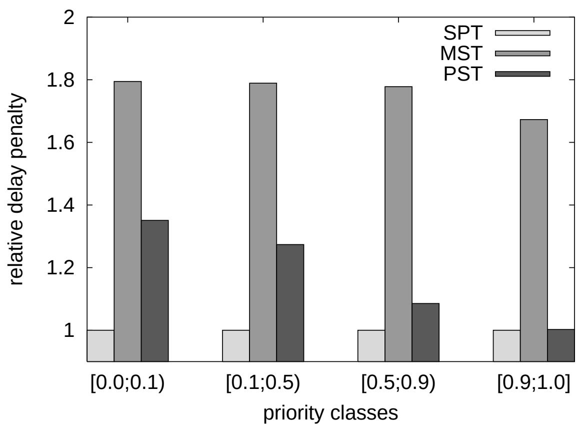
Link stress is another indicator for the network overhead caused by an application-level multicast tree. MSTs result in the lowest link stress with 77% of all distribution trees having a link stress of 1 and a maximum link stress of 2, as shown in Figure 6(d). Distribution trees constructed by the PST algorithm come close to these values with the only difference being that 1.7% of the trees have a link stress of 3. The link stress for the star-shaped SPT topologies lies between 1 and 5 and only 60% of the trees have a link stress of 1.

3) Simulation Results for a Large Topology: For the second simulation scenario, we created a more complex network topology with 42 routers, 80 links, and 18 end-systems participating in a virtual game session. The delays among end-systems lie between 16 ms and 268 ms with an average value of 145.5 ms. During the session's duration of 104 seconds, a total of 6564 events were issued by all players. The resulting application priorities are similar to the distribution shown in Figure 5.

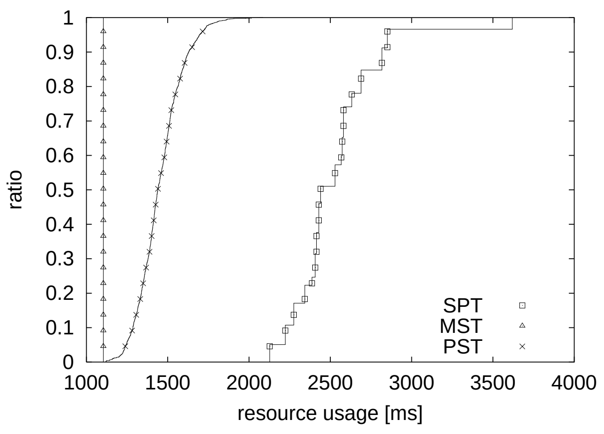
The distributions for the cumulative weighted end-to-end delay C_D^p are depicted in Figure 7(a). Because of the increased complexity of the application-level multicast trees (with up to 11 hops on paths of the PST), the difference in C_D^p between SPT and PST is larger (1721 ms to 1906 ms at 90%). However, the PST does achieve a good optimization of the latency from the source node to receivers with a high priority when compared to the values



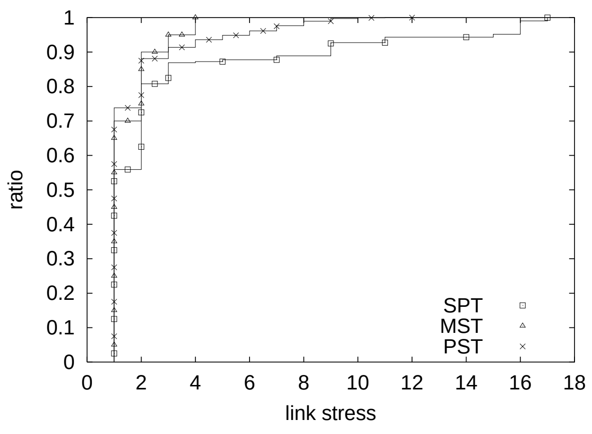
(a) Distribution of cumulative weighted end-to-end delays



(b) Average relative delay penalty



(c) Resource usage distribution



(d) Link stress distribution

Fig. 7. Simulation results for the large topology

of C_D^p for the MST algorithm (3496 ms at 90%).

The receiver-specific end-to-end delays resulted in the following 99% confidence intervals for the second scenario: SPT $[148.4; 149.2]$, MST $[281.5; 283.8]$, and PST $[175.2; 176.5]$.

The optimization of end-to-end delays becomes also visible in the average RDP values for application instances within different priority classes (see Figure 7(b)). For the PST algorithm, the RDP decreases from 1.35 to 1.002 for receivers with $p(v_t) \in [0.9; 1.0]$ which is close to the optimum RDP value. This is a significant improvement over multicast trees constructed using the MST, even for the receivers within the lowest priority class.

At the same time, priority-based minimum spanning trees cause a higher network load as can be seen from Figure 7(c). It shows the resource usage distributions for the three tree building algorithms: 90% of all PSTs have a resource usage that is up to 50% higher than C_R of the MST. Shortest path trees have a resource usage that is by far larger.

As in the first simulation scenario, the MST algorithm generates the lowest link stress with 90% of all distribution trees having a link stress of at most 2 and a maximum stress of 4 (see Figure 7(d)). The values for the PST algorithm are only slightly larger with 90% of all multicast trees having a link stress of at most 3 and a maximum link stress of 12. In comparison, the link stress of the SPT trees has a value of 9 at 90% and maximum link stress is 17.

Summing up, the simulation results show that the PST algorithm optimizes the end-to-end delay for application instances that have a high application priority. Even delays for end-systems with a lower priority are in most cases better than those that can be achieved when constructing multicast trees using the MST algorithm. At the same time, the increase in network load is kept at a tolerable level.

Besides the example simulation scenarios presented here, we evaluated a whole series of simulations with different network topologies. Although individual results may depend on the underlying topology and on the number of participating end-systems, the above conclusions hold true for all the scenarios we evaluated.

VI. Conclusions and Outlook

In this paper, we presented a routing algorithm for application-level multicast. Unlike previous approaches, the PST algorithm takes application information into account when constructing the distribution trees. The application may specify a priority for each packet-receiver pair which is used to alter from the MST-path to the SPT-path as the priority is increased from 0 to 1. We proposed a fast algorithm for the construction of the distribution tree and discussed how tree maintenance can be done efficiently by using incremental tree updates. The properties of the routing algorithm were investigated by means of extensive simulations with a real application. The simulations indicate that the PST algorithm builds multicast trees with end-to-end delays that are close to the optimum for receivers with a high priority. At the same time, the network load increases only slightly.

In the future, we plan to investigate the algorithm for other application scenarios such as networks between sensors and CSCW applications. Furthermore, we intend to perform more complex simulations taking bandwidth limitations and larger multicast groups into account. Finally, our aim is to integrate the routing algorithm into a real-world implementation and to perform measurements over the Internet.

References

[1] P. Francis, "Yoid: Extending the Internet Multicast Architecture," unrefereed report, available at http://www.icir.org/yoid/docs/yoidArch.ps.gz, Apr. 2000.

[2] J. Jannotti, D.K. Gifford, K.L. Johnson, M.F. Kaashoek, and Jr. J.W. O'Toole, "Overcast: Reliable Multicasting with an Overlay Network," in *4th Symposium on Operating Systems Design and Implementation (OSDI)*, San Diego, CA, USA, Oct. 2000.

[3] Y. Chu, S.G. Rao, and H. Zhang, "A Case For End-System Multicast," in *Proc. ACM SIGMETRICS*, Santa Clara, CA, USA, June 2000.

[4] Y. Chu, S.G. Rao, S. Seshan, and H. Zhang, "Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture," in *Proc. ACM SIGCOMM*, San Diego, CA, USA, Aug. 2001.

[5] Y. Chawathe, *Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service*, Ph.D. thesis, University of California, Berkeley, USA, Dec. 2000.

[6] S.Q. Zhuang, B.Y. Zhao, A.D. Joseph, R. Katz, and J. Kubiatowicz, "Bayeux: An Architecture for Scalable and Fault-tolerant Wide-area Data Dissemination," in *11th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Port Jefferson, NY, USA, June 2001.

[7] I. Stoica, R. Morris, D. Karger, M.F. Kasshoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," in *Proc. ACM SIGCOMM*, San Diego, CA, USA, Aug. 2001.

[8] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable and Content-Adressable Network," in *Proc. ACM SIGCOMM*, San Diego, CA, USA, Aug. 2001.

[9] Y.J. Chu and T.H. Liu, "On the Shortest Arborescence of a Directed Graph," *Science Sinica*, vol. 14, pp. 1396–1400, 1965.

[10] J. Edmonds, "Optimum Branchings," *J. Research of the National Bureau of Standards*, vol. 71B, pp. 233–240, 1967.

[11] K.L. Calvert, M.B. Doar, and E.W. Zegura, "Modeling Internet Topology," *IEEE Communications Magazine*, vol. 35, no. 6, pp. 160–163, 1997.

[12] M. Mauve, J. Vogel, V. Hilt, and W. Effelsberg, "Local-lag and Timewarp: Providing Consistency for Replicated Continuous Applications," *To appear in: IEEE Transactions on Multimedia*.

Appendix

A. Pseudo Code

Figure 8 gives the pseudo code to compute the PST on a graph $G = (V; E)$ for a sender v_s with priority function p. First, the weights $w'(e_{ij})$ of the directed graph are calculated as described in Section III. Second, the directed minimum spanning tree is determined according to the algorithm published by Edmonds [10]. This algorithm is designed to construct a branching [5] T with maximum total costs $C = \sum_{e_{ij} \in T} w(e_{ij})$ on basis of G. Thus, to build a minimum spanning tree, we define all weights w' to be negative and ensure that the branching contains $|V| - 1$ edges (maximizing C with negative weights is equal to minimizing C with positive weights).

The basic idea of Edmond's algorithm is to calculate an initial graph T by selecting for each node (except v_s) the incoming edge with maximum costs. While T contains any cycles, these are broken up by exchanging appropriate edges.

[5] A branching is a directed graph without cycle where each node has at most one incoming edge, i.e., a branching is not necessarily connected.

```
(1) Compute weights $w'(e_{ij})$ for all edges in E:
    • $\forall i, j, i \neq j : w'(e_{ij}) = -(w(e_{ij}) + p(j)w(e_{si}))$
(2) Compute the directed minimum spanning tree with source $v_s$ on $G$:
    • Discard all edges $e_{is} \in E$ entering the source node $v_s$.
    • $\forall$ nodes $v_i \in V, v_i \neq v_s$: select the edge $e_{ji} \in E$ with maximum weight $w'(e_{ji})$. Let $E'$ be the
      set of selected edges.
    • While $T := (V; E')$ contains a cycle $C := (W; F), W \subset V, F \subset E'$ do
        – Find the edge $e_{kl} \in F$ with minimum weight $w'(e_{kl})$.
        – Modify the weight $w'$ of each edge $e_{ij} \in \{e_{ij} | v_i \in V \setminus W, v_j \in W\}$:
              $w'(e_{ij}) := w'(e_{ij}) + w'(e_{kl}) - w'(e_{h(j)j})$, with $h(j) \in W$ being the predecessor node
          with edge $e_{h(j)j} \in F$.
        – Select the edge $e_{mn} \in \{e_{ij} | v_i \in V \setminus W, v_j \in W\}$ with maximum weight $w'(e_{mn})$, and set
          $E' := E' \cup \{e_{mn}\} \setminus \{e_{h(n)n}\}$.
        – Build a new graph $T$ by contracting all nodes $v_i \in W$ into a pseudo-node $\varphi$:
          $V := V \setminus W \cup \{\varphi\}$. Modify $E$ and $E'$ by replacing all edges $e_{ij}$ with tail node $v_i \in W$ or
          head node $v_j \in W$ by $e_{\varphi j}$ or $e_{i\varphi}$, and delete edges $\{e_{ij} | v_i, v_j \in W\}$. Create new weights
          $w'$ accordingly.
    • Replace all pseudo-nodes $\varphi \in V$ and the corresponding edges in $E'$ by the original nodes and
      edges. $T$ represents the directed MST with root $v_s$.
```

Fig. 8. Pseudo code for the computation of the optimum distribution tree