Priority-Based Distribution Trees for Application-Level Multicast

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

ABSTRACT

In this paper, we propose a novel multicast routing algorithm that is based on application-level priorities and network characteristics: The application may specify an individual priority for each packet-receiver pair. The multicast distribution tree is then constructed such that the higher the priority, the more direct the path from the sender to the packet's destination and the lower the resulting end-to-end delay. This algorithm can be used to realize application-level multicast for delay-sensitive applications such as networked computer games. However, optimizing the multicast tree with respect to the end-to-end delay 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 distribution tree. Our algorithm takes this tradeoff into account and constructs efficient priority-based multicast trees. We demonstrate the performance and characteristics of the algorithm through simulation.

Keywords

Application-Level Multicast, Multicast Routing, Distribution Tree.

1. INTRODUCTION

Group communication, or multicast, is needed by distributed interactive applications to deliver data from one sender to multiple receivers. Examples are video conferences, groupware systems, and networked computer games. In many cases, realizing group communication by setting up a direct connection from a sender to each receiver is not a viable solution because of resource limitations. In the Internet, IP multicast provides efficient group communication by duplicating packets within the network routers so that data traverses physical links only once. But due to various technical and administrational reasons, IP multicast has not been widely deployed. A promising alternative is *application-level multicast (ALM)* [1-8]: The key idea is to use 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, ALM can be deployed immediately without any changes to the network. This eliminates two key problems of IP multicast.

NetGames 2003 May 22-23, 2003, Redwood City, CA, USA

Existing approaches for ALM 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, a number of applications exist where the priority of a packet may be different for the receivers. In a networked computer game, for example, the actions of a player is very important to competing players that are close by. These players should receive information about such actions with a very low delay. Other players may be able to tolerate a higher delay, depending on their location and orientation within the game. Furthermore, a packet's priority may change over time for some or all receivers. For example, if sensor data is transmitted by a sender, this data may typically have a low priority for all receivers, unless an extreme sensor reading occurs which requires the transmission of a packet with very low latency to some receivers. Traditional tree routing algorithms are unable to handle these situations.

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). Thus, 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 2 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 3. In Section 4 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 5 contains an evaluation of the presented algorithm by means of simulation. We conclude the paper and give an outlook on future work in Section 6.

2. 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 [7] 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. Aside 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.

Other examples for tree building ALM protocols are *HMTP* [14], *BTP* [8], and *Overcast* [9]. They all form self-organizing distribution trees where nodes select an appropriate parent, and they implement mechanisms for integrating new members, detecting loops and partitions, and for optimizing the tree by rearrangement. Unlike the other protocols, Overcast builds sender-specific trees instead of a single shared tree.

With *TMesh* [13], the authors propose to add additional links to the multicast tree. While these shortcuts reduce the number of hops on the way from a sender to the receivers, TMesh seeks to optimize the average end-to-end delays for the whole group and builds a rather stable ALM tree. Thus, TMesh seems not to be flexible and fast enough to facilitate delay optimization for certain receivers in an environment where priorities change dynamically.

Instead of constructing a tree directly, *Narada* [5] 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* [2] 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* [15], *Chord* [12], and *Content Addressable Networks* [11] 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.

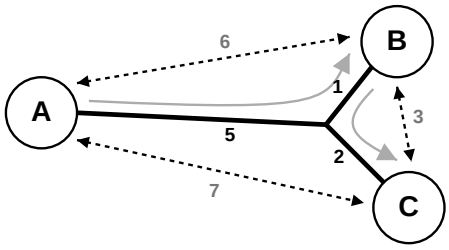


Figure 1: Joint path to distant receivers

3. APPLICATION-LEVEL MULTICAST ROUTING

An ALM routing algorithm builds a data distribution tree with the end-systems as nodes connected by unicast links. The resulting tree should use the resources of the underlying network efficiently. Since on the application level there is no direct access to network topology information, observable parameters (e.g., latency) may be used to deduce a certain amount of knowledge about the actual topology: When node A has a high delay to both nodes B and C, and B has a low delay to C, then it is likely that the route AB shares a significant portion of the physical link with route AC (see Figure 1). In the following, we will concentrate on unicast latencies as the network parameter that determines the ALM tree [1].

Two well-known types of trees are the minimum spanning tree (MST) and the shortest path tree (SPT). The MST optimizes the resource usage of the multicast tree but the path length is not considered and can cause very large end-to-end delays. Hence, using an MST is only reasonable when end-to-end delays are not an issue (e.g., for non-interactive data dissemination). When building an SPT from the unicast delays, the distribution tree will consist of separate unicast connections from the sender to each receiver (commonly, this would be regarded as 'normal' unicast rather than application-level multicast). With respect to end-to-end delay the SPT is optimal but it causes a very high consumption of network resources. Furthermore, building a SPT is not possible when the sender's bandwidth is not sufficient to serve all receivers simultaneously.

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. Depending on this priority, the path along which the packet is forwarded should gradually change from the MST path to the SPT path. 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, while the optimization of the *cumulative end-to-end delay* leads to an SPT. We combine those two metrics by using one common application priority for the whole distribution tree. The optimization of the combined metric allows the gradual transition from MST paths to SPT paths as the application priority increases. In a second step, we generalize the met-

[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.

ric 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 changes from the MST path to the SPT path. Finally, we present an efficient algorithm which provides a very good approximation for the optimal distribution tree with respect to the last metric.

3.1 Distribution Tree Metrics

Let $G = (V; E)$ be a fully connected, directed graph, where $V = \{v_i\}$ denotes the set of nodes and $E = \{e_{ij}\}$ the set of edges, where e_{ij} connects the nodes i and j. 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 of the corresponding link. For each distribution tree $T \subset E$, we can define two cost functions C_R and C_D:

- **Resource usage**, defined 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 delay for the distribution of a packet from the source to all receivers. Let $r_t = \langle e_{sj_1}, \ldots, e_{j_n t} \rangle$ denote the route from the source v_s to the receiver v_t on the current distribution tree. Then C_D is given by:

$$C_D = \sum_{v_t \in R} \sum_{e \in r_t} w(e).$$

When optimizing C_R, the minimum cost tree is equal to the MST, when optimizing C_D, it is equal to the SPT.

3.2 Introducing Application-Level Semantics

For many applications, selecting either of these two metrics as the optimization criterion does not give the desired result. 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. A balancing cost function C can be defined as follows:

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

Figure 2 visualizes the effect of p when building the optimum distribution tree (according to C) for a sample ALM session. The participants of the 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. Let node 2 be the sender. The resulting distribution trees that are optimal with respect to C are depicted in Figure 3. When p is increased, nodes farther away move up in the tree, reducing the end-to-end delays to the sender, until for $p = 1.0$ a star-like SPT is reached. As can be seen from the graphs, the number of possible trees for a small overlay network with only 6 nodes is very limited.

Following, we generalize the cost function C 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. Let $p : V \to [0; 1]$ be the per-node priorities for a sender v_s. They can easily be integrated into C_D, defining the cost function C_D^p:

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

Integrating the per-node priorities into C_R is more difficult since the costs are calculated over the edges of the tree and not per receiver. However, in an MST, the relevant cost for a receiver is the weight of the edge over which it is connected to the rest of the tree. Consequently, the priority of a node can be assigned to this edge. This leads to the following cost function C_R^p:

$$C_R^p = \sum_{e_{ij} \in T} (1 - p(v_j)) w(e_{ij}).$$

The total costs C are defined as

$$C = C_R^p + C_D^p. \qquad (2)$$

Note that C 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 SPT and MST.

Direct optimization of this cost function is computationally complex. Thus, we approximate the cost term in a way that allows us to directly modify edge weights and compute an MST 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 the path of 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 costs. This leads to a simplified approximate formulation for the global costs C:

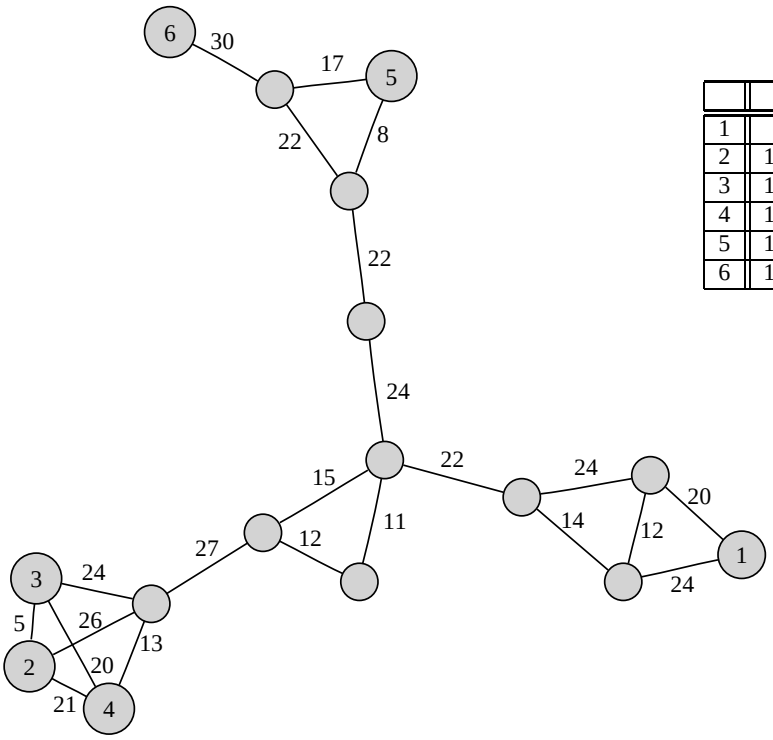
$$\begin{aligned} C &= \sum_{e_{ij} \in T} (1 - p(v_j)) w(e_{ij}) + \sum_{v_t \in R} p(v_t) \sum_{e \in r_t} w(e) \\ &\approx \sum_{e_{ij} \in T} (1 - p(v_j)) w(e_{ij}) + \\ &\qquad \sum_{v_t \in R, v_i \in V : e_{it} \in T} p(v_t) \big(w(e_{si}) + w(e_{it}) \big) \\ &= \sum_{e_{ij} \in T} w(e_{ij})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

The last equality follows from the property that a spanning tree of a graph has the same number 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 C, we can apply an MST algorithm on the graph with modified weights. The new weights are set to

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

With increasing $p(v_j)$, indirect links to the target node v_j will become 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 and the costs for

[2]Given the distribution tree $\{e_{AB}, e_{BC}\}$ in Figure 1, the resource usage in the underlying network is $1 * 5 + 2 * 1 + 1 * 2 = 9$, 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.



	1	2	3	4	5	6
1	0	128	126	115	114	158
2	128	0	5	21	122	166
3	126	5	0	20	120	164
4	115	21	20	0	109	153
5	114	122	120	109	0	47
6	158	166	164	153	47	0

End-to-end delays

Figure 2: Example graph

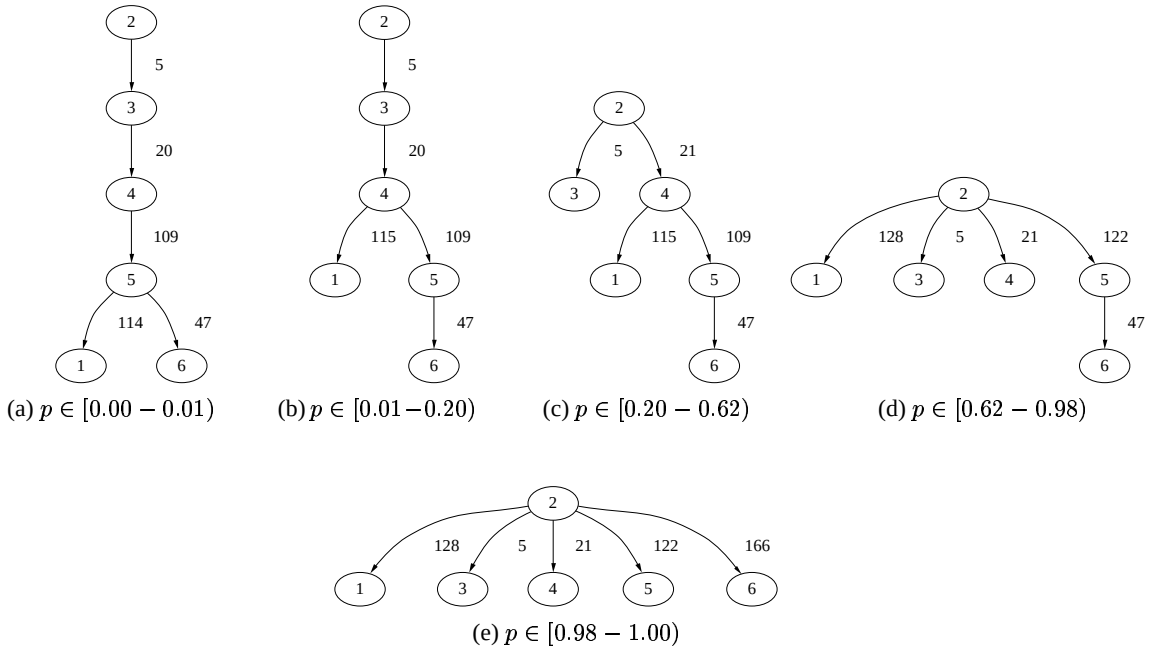


Figure 3: Optimal distribution trees

opposing directions may differ. We call 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 [3, 6]. Pseudo code for the implementation that was used for the simulations can be found in Appendix A.

4. 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 is very costly 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.

4.1 Maintenance of the Distribution Tree

Instead of rebuilding the distribution tree whenever topology information or application priorities change, an improved update mechanism can significantly reduce the number of necessary tree recomputations [3]. The tree will not change under the following conditions:

- the cost (delay) of a link that is not in the directed MST increases,
- the cost of a link within the directed MST decreases,
- the priority for a receiver which is connected directly to the sender increases, and
- 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. Furthermore, a change in receiver priorities or link delays may be too small to cause a tree change.

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, 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.

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.

[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.

Let us assume that the cost of a single link increases sufficiently 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 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 only).

4.2 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. 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.

5. SIMULATIONS

We implemented a simple network simulator in order to evaluate the performance of our PST algorithm compared to the delay-based MST and SPT approaches. The simulator is event-based and allows

Table 1:

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

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) [1] 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. End-systems were located on the network's edges.

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

5.1 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 1. 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 ALM session. The average number of different distribution trees 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 4 is able to achieve a significant reduction in tree calculation costs.

5.2 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.

5.2.1 Simulation Setup

Event patterns to determine application priorities for the simulations were generated by tracing a simple multi-player game [10]. 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 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.

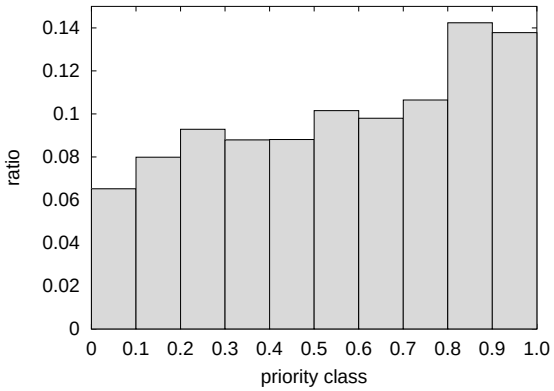


Figure 4: Distribution of application priorities

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. For players j outside the shooting range of s, p 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 possible distance.

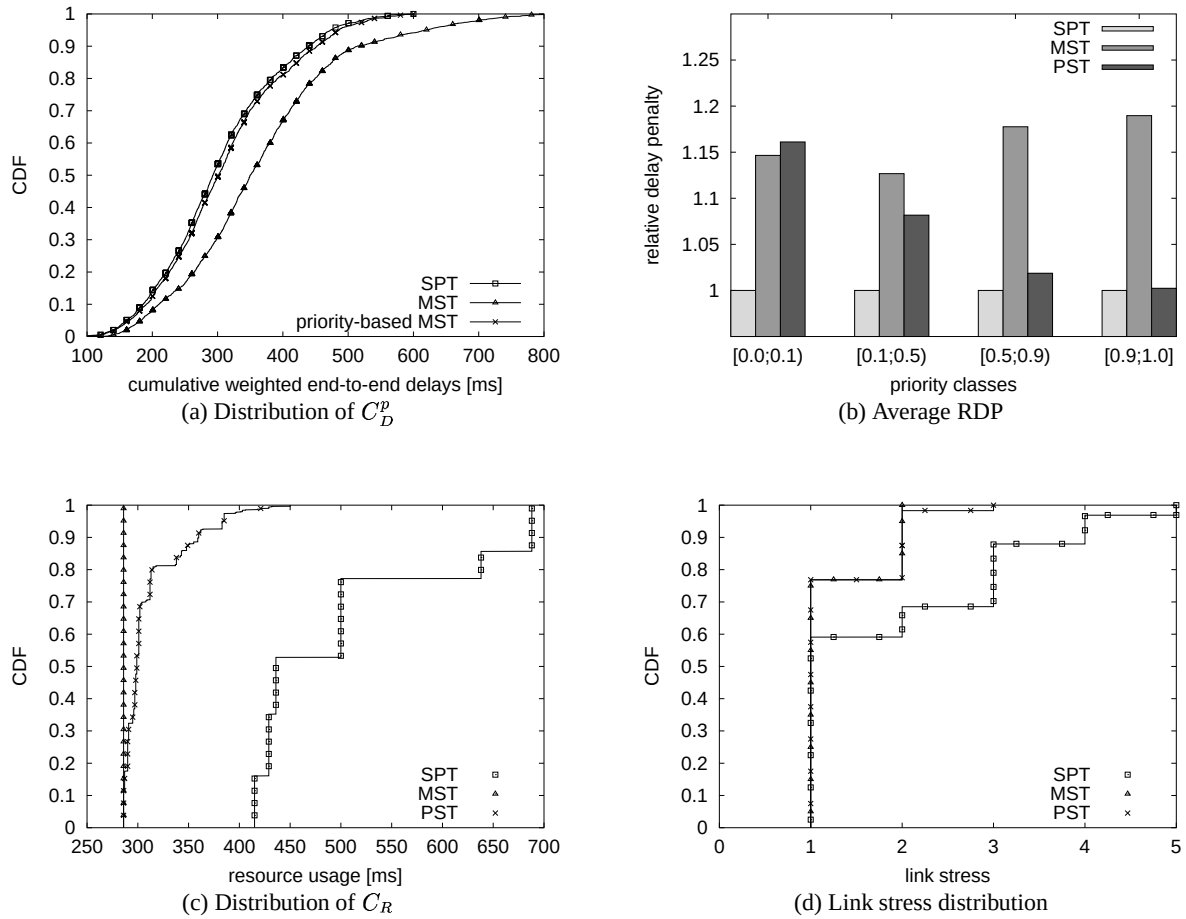
A typical distribution of priorities for a game session with six players is depicted in Figure 4. Priorities close to 1 are common because the objective of the game is to score points by shooting other players and hence players will cluster together instead of spreading out evenly on the game field.

5.2.2 Simulation Results for 6 End-Systems

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 4. Figure 2 shows the underlying network topology with end-to-end delays between 5 and 166 ms and an average value of 100 ms.

The delay properties of a specific distribution tree can be measured using the costs C_D^p (see Section 3). Figure 5(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%).



(a) Distribution of C_D^p

(b) Average RDP

(c) Distribution of C_R

(d) Link stress distribution

Figure 5: Simulation results for 6 End-Systems

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) is a measure for the optimality of the end-to-end delay:

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

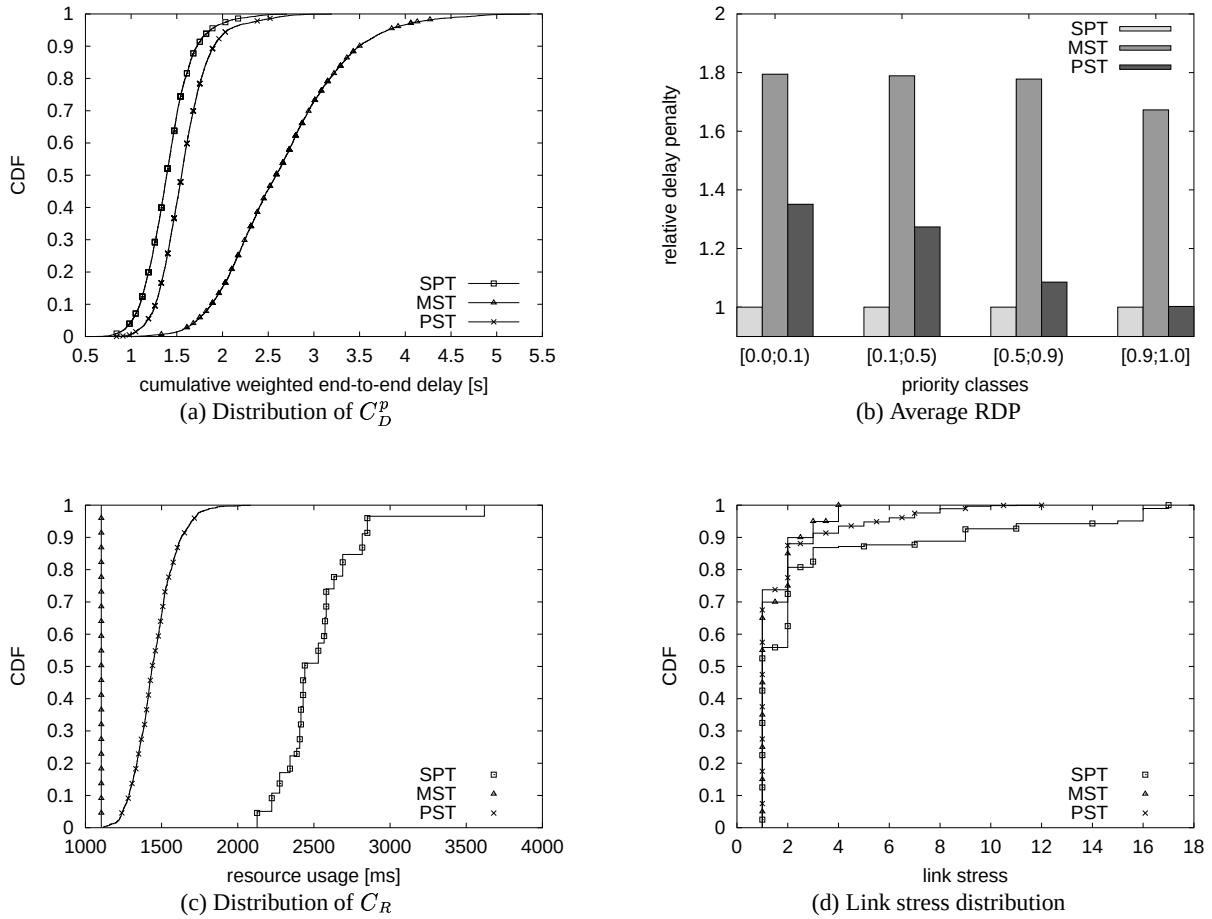
The RDP compares the end-to-end delay of a receiver v_t to the smallest possible delay (i.e., the unicast delay from v_s to v_t). By definition, for the SPT distribution trees $RDP(t) = 1 \ \forall v_t \in R$. Figure 5(b) shows the average RDP values for different priority classes. In the case of the 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 3. 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 Figure 5(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 ALM 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 5(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.

5.2.3 Simulation Results for 18 End-Systems

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-



(a) Distribution of C_D^p

(b) Average RDP

(c) Distribution of C_R

(d) Link stress distribution

Figure 6: Simulation results for 18 End-Systems

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 4.

The distributions for the cumulative weighted end-to-end delay C_D^p are depicted in Figure 6(a). Because of the increased complexity of the ALM 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 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: 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 6(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 6(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 6(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.

5.2.4 Introducing Uncertainty

All simulation results discussed above were calculated under the condition that the application always had full knowledge about the actual end-to-end delays. In a real network, delays fluctuate (depending on router load) and measurements give approximations only. Thus, we also conducted simulations for the PST algorithm when measured delays differ from the real values up to a certain percentage e. For $e = 20\%$, C_D^p degrades only by 128 ms at 90% when compared to the value given in Figure 6(a), and by 637 ms for $e = 50\%$. In the highest priority class, RDP increases only slightly to 1.04 for $e = 20\%$, and to 1.23 for $e = 50\%$. The ressource usage distribution for $e = 20\%$ is almost identical to the

one depicted in Figure 6(c). For $e = 50\%$, 95% of all trees have a C_R between 1120 ms and 1882 ms. These results indicate that the PST algorithm is fairly robust against inaccurate knowledge of delays.

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

6. CONCLUSIONS AND OUTLOOK

We have presented a novel priority-based routing algorithm for application-level multicast. It allows an application to influence the path that a packet takes from the sender to a receiver by specifying a priority for each packet-receiver pair. As the priority is increased from 0 to 1, the path changes gradually from MST to SPT. Thus, our PST can be considered as a generalization of the MST and the SPT. We have described an efficient algorithm for the construction of the distribution tree and discussed how tree maintenance operations can be minimized. Our simulation results 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 while the total network load increases only slightly.

In the future, we plan to investigate how to best select priorities on the basis of application-layer semantics for performance-critical multicast applications. Another important issue is to further reduce the computational complexity and to improve scalability. One solution might be to cluster adjacent (in respect to latencies and priorities) end-systems and to construct local PSTs. Also, we want to take capacity constraints on the links into account, and we intend to perform simulations for more complex topologies. Our final goal is to integrate the PST algorithm into a real-world multicast application, such as an Internet game, and to perform measurements over the Internet.

7. REFERENCES

[1] K. Calvert, M. Doar, and E. Zegura. Modeling Internet Topology. *IEEE Communications Magazine*, 35(6):160–163, 1997.

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

[3] Y. Chu and T. Liu. On the Shortest Arborescence of a Directed Graph. *Science Sinica*, 14:1396–1400, 1965.

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

[6] J. Edmonds. Optimum Branchings. *J. Research of the National Bureau of Standards*, 71B:233–240, 1967.

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

[8] D. Helder and S. Jamin. End-host Multicast Communication Using Switch-tree Protocols. In *Proc. GP2PC*, Berlin, Germany, May 2002.

[9] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. 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.

[10] 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*.

[11] 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.

[12] I. Stoica, R. Morris, D. Karger, M. 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.

[13] W. Wang, D. Helder, S. Jamin, and L. Zhang. Overlay Optimizations for End-host Multicast. In *Proc. NGC*, Boston, MA, USA, Oct. 2002.

[14] B. Zhang, S. Jamin, and L. Zhang. Host Multicast: A Framework for Delivering Multicast to End Users. In *Proc. IEEE INFOCOM*, New York, NJ, USA, June 2002.

[15] S. Zhuang, B. Zhao, A. 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.

APPENDIX

A. PSEUDO CODE

Figure 7 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 3. Second, the directed minimum spanning tree is determined according to the algorithm published by Edmonds [6]. This algorithm is designed to construct a branching [4] 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.

[4] 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 φ: $V := V \setminus W \cup \{\varphi\}$. 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.

Figure 7: Pseudo code for the computation of the optimum distribution tree