Multicast Trees for Collaborative Applications

Krzysztof Rzadca
School of Computer Engineering
Nanyang Technological University
Singapore
Email: krz@ntu.edu.sg

Jackson Tan Teck Yong
School of Computer Engineering
Nanyang Technological University
Singapore
Email: jacktty@ntu.edu.sg

Anwitaman Datta
School of Computer Engineering
Nanyang Technological University
Singapore
Email: anwitaman@ntu.edu.sg

***Abstract*—Current implementations of real-time collaborative applications rely on a dedicated infrastructure to carry out all synchronizing and communication functions, and require all end nodes to communicate directly with and through the central server. In this paper, we investigate an architecture, in which the most resource intensive functionality of continuous communication among collaborators to disseminate changes is decentralized, utilizing the end users as relays. We observe that communication characteristics of real-time collaboration makes use of existing multicast mechanisms unsuitable. As collaborative editing sessions are typically long, we are able to gather and then use additional parameters of nodes (their instabilities and frequency of sending updates) and communication links (latencies and average costs). We identify several criteria to determine the quality of a multicast tree: cost, latency and instability. We analyze the complexity of these problems and propose algorithms to optimize the communication topology. We also consider the multiobjective problem in which we search for a tree that results in a good trade-off between these measures. Validation of algorithms on numerous graphs shows that it is important to consider the multiobjective problem, as optimal solutions for one performance measure can be far from optimal values of the others.**

***Index Terms*—communication topology, collaborative applications, multi-objective optimization**

Complex projects are carried out in a collaborative manner involving multiple participants. There are numerous manners in which such collaborative work can be carried out, depending both on the application need as well as how consistency of shared objects is maintained. While some consistency maintenance mechanisms allow asynchronous collaboration (for example, cvs/svn) others like *operational transformation* [1], [2] significantly simplify collaboration tasks by facilitating real time group editing. Furthermore, techniques like *transparent adaptation* [3] facilitate adoption of diverse traditionally single user applications – from text editors like Word [4] to multimedia content creation tools like Maya [5] – into groupwares for real time collaborative editing. Such real-time collaboration groupware systems require four logical functions, namely a repository manager to store the shared content, a session manager to keep track of the members involved at any time (session) in the collaborative editing activities, a centralized synchronizer to carry out the operational transformations, and a communication mechanism among the users (and the central synchronizer). Current implementations of such groupware [4], [5] rely on a dedicated infrastructure to carry out all these functions, and require all end nodes to communicate directly with and through the central server.

The project is funded by A-STAR grant no: 072 134 0055

There are several motivations to move to a more decentralized approach. However, such a move brings about new challenges. In particular, real time collaborative editing is communication intensive. During a session when an object is being edited, all the members involved in the session need to communicate their updates to all the other users. If there are many object sessions and users using the infrastructure, it can generate heavy communication load at the server if all the communication messages need to go through such a server, as is the case with current implementations [4], [5]. However, such an overload is not only undesirable but also easy to avoid. What is essential is that each session at any time instant has a logically central synchronizer, taking care of consistency. But any member of the session itself can play the role of such a synchronizer, thus there is no fundamental need to burden the central infrastructure. All update related communication can likewise be confined within just the members of the session, without involving and overloading a central infrastructure.

The current implementations also assume that end users have reliable and fast connection, and membership changes in a session are infrequent. However, geographic distribution of groups, increased mobility of knowledge workers as well as proliferation of diverse portable devices like *Ultra Mobile PCs* (UMPC) require a more flexible support for nomadic collaboration. Such a flexible paradigm should take into account users' limited connectivity and other constraints. For instance, UMPCs typically have various connectivity options (bluetooth, wi-fi, GPRS, Ethernet). Wireless networks are much more varied than corporate LANs in availability, performance and costs.

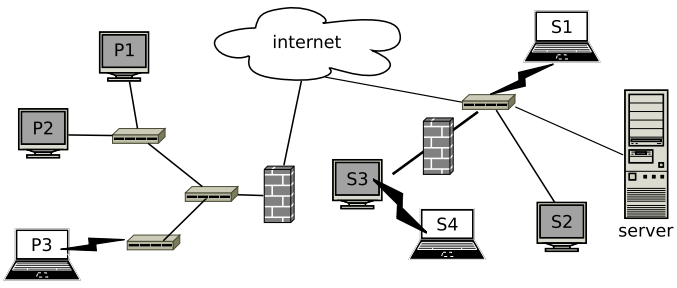

Fig. 1. Approximate physical network structure with peers grouped in two physical locations. Two groups of peers are behind firewalls.

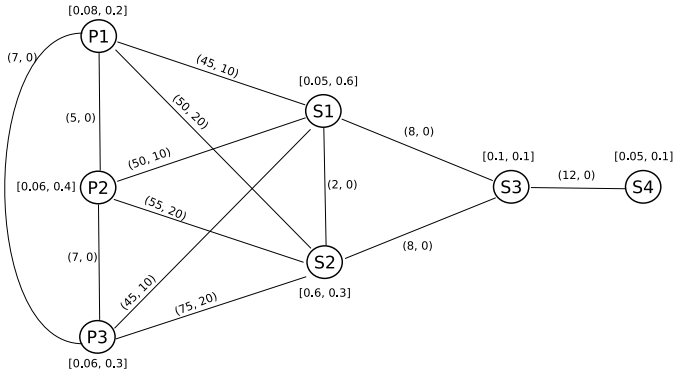

Fig. 2. Connectivity graph for the physical network structure presented in Figure 1. The values next to a node are its workload ρ (higher values denoting more frequent updates) and instability λ (higher values denoting more frequent disconnections). The values next to an edge are its latency l and its cost c.

These considerations motivate to design a more flexible communication mechanism, where a suitable topology can be chosen for communication within session members, where the optimality is determined by various considerations including the overall performance and cost.

In order to model the process of collaborative editing in a more accurate way, we propose to measure additional characteristics of nodes and edges (formally defined in Section I). Firstly, typically, some of the users contribute to a shared document more than others. The structure of the connectivity tree should take this into account by proposing an architecture that in which updates from frequent contributors have lower latency. We model the frequency of edits of node v by *workload* $\rho(v) \in [0, 1]$, with higher values denoting more frequent contributions. Secondly, nodes disconnect from the network with different rates, that depend e.g. on the quality of network connection (Ethernet vs. GPRS) or the type of the node (standard PC vs. smart phone). A disconnection of a node will force the nodes connected through that node to reorganize. We model the disconnection rate by nodes' *instability* λ, with higher values corresponding to more frequent disconnections. Thirdly, a communication channel between two nodes is characterized by the observed *latency*. As high latencies worsen the user experience of collaborative editing, nodes should use channels with low latencies. Finally, in a heterogeneous network infrastructure, some of the links (such as GPRS/3G) can have significant monetary *cost*, while others are free (local Ethernet or local wireless).

Consider for instance a geographically distributed team working at two physical locations P and S (Figure 1). The typical collaborative editing application uses a dedicated server as the hub of a logically star topology for communication. However, this would exclude the mobile node $S4$ from collaboration.

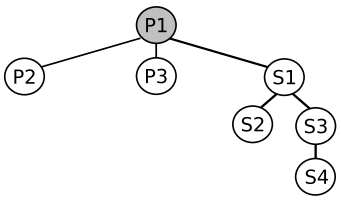

Fig. 3. A multicast tree in which, by using intermediate nodes, data on the backbone link is not replicated.

In a more dynamic collaboration scenario, one of the nodes takes also the role of the server (that can be delegated to another node, should the current server fail). Figure 2 presents all the possible connections at the application layer. Additionally, we present *workload* and *instability* for each node and *latency* and *cost* for each edge. Using this data, an alternative multicast tree (Figure 3) can be constructed, in which only one copy of data (instead of four) is sent over the cross geographic S-P link, which optimizes the usage of potentially costly connection. However, care should be taken in determining a good topology. For instance, in the example scenario, $S1$, the local root in S, uses a potentially unreliable wireless link, which is undesirable. Another problem may be that observed latency can be high, as node $S2$ is the source of a majority of the update operations.

In this paper, we consider the problem of building an optimal multicast tree for disseminating update information in collaborative applications. We take into account heterogeneous network structure and failing nodes.

We model the problem as a multi-objective optimization problem (Section I) that optimizes the stability, the latency and the cost of the multicast tree. We model the (in)stability as the average number of disconnections of a node in a unit of time. The latency takes into account both the uplink latency (on the path from a node to the synchronizing server) and the downlink latency (the maximum latency between the synchronizer and a node). Similarly, the cost considers both the cost of sending an update to the server and the cost of disseminating this update by the server to the nodes. Users' latencies and costs are weighted by the average participation of the user in the editing process. We analyze the boundary, monocriterion problems and propose exact algorithms (Section II). We validate the performance of our algorithms by simulation (Section III).

There is substantial research effort devoted to application level multicast, both in one-to-many and in many-to-many scenarios. One-to-many multicast typically models transmissions of contents that cannot be modified by the group (such as, e.g., Internet radio). Many-to-many multicast models more interactive applications, such as teleconferences. Collaborative applications are similar to many-to-many multicast. However, the main difference is that the messages must be validated (and, perhaps, modified by operational transformation) by a central synchronizer before other members can receive them, to guarantee consistency. Multicast trees for collaborative applications must thus optimize both the information collection and the dissemination. Another difference, thankfully to our advantage, is that a session to edit an object is typically small in realistic groupwares, for instance CoWord [4] supports a maximum of sixteen simultaneous users editing an instance of a document. We discuss related work and differentiate them from our work in Section IV before drawing our conclusions and identifying future research agenda in Section V.

I. Problem Definition

The connectivity graph $G = (V, E)$ represents all the *possible logical* connections $(u, v) \in E$ between peers (or nodes) $V = \{v\}$. If $(u, v) \in E$, peers u and v can establish a connection. We assume that $(u, v) \in E \Leftrightarrow (v, u) \in E$. However, if both peers are firewalled, or too far, there is no edge in G. We assume that the graph is connected.

We define the following cost functions over E:

- $c : E \to \mathbb{R}^+$: $c(u, v)$ is the monetary cost associated with sending a unit of data (e.g., 1kB) from u to v;
- $l : L \to \mathbb{R}^+$: $l(u, v)$ is the latency (in time units) over the edge (u, v).

In the most general model, the price and the latency of the link are not symmetric.

The amount of communication messages generated by a user of a collaborative application varies, depending on the type of the application and the usage pattern. In a boundary case, a very fast typist can generate about 30kB/s on CoWord configured to send one message for each pressed key. However, other applications do not produce as many messages, and CoWord can be reconfigured to group neighboring edits before sending them. Nevertheless, bandwidth limit can be introduced as a limit $deg_{\max}(v)$ over the number of immediate children of a node v (with a simplifying assumption that all the children share the same communication channel).

Each node v has the following characteristics:

- disconnection rate $\lambda(v_i) = \lambda_i$: the average number of disconnections from the network in the unit of time;
- sending rate $\rho(v_i) = \rho_i \in [0, 1]$: expresses the fraction of transmission time in which v sends data. $\rho(v_i)$ is normalized (we take into account only the time when there is at least one sender), so that $\sum \rho(v_i) = 1$. If more than one sender sends in parallel, we compute ρ as if all the sending operations were sequential.

In collaborative editing, the peers are well-identified and the collaboration sessions are long. Consequently, it is easy to collect the aforementioned parameters.

The objective is to build a spanning tree $T = (V, E_t)$ for G, with $E_t \subseteq E$. The tree corresponds to the data dissemination pattern in multicast scenario. We assume that the root $root(T)$ of the tree acts as a synchronizer.

When a peer v modifies the shared data (by modifying the local copy), the modification notification is sent to its parent $u : (u, v) \in E_t$, who forwards it to its parent, etc., until it reaches the root. However, the nodes along such a path do not consume the notification on the application level. The root accepts the modification (perhaps re-ordering it with other, concurrent modifications originating from other peers or marking it as a conflicting one). Then, the notification is multicasted to all the other nodes. The root sends the notification to its children. The children consume the notification and, in parallel, forward it further downstream to their children, etc., until the notification reaches all the nodes in the tree.

Note that this scenario is different from the usual multicast with multiple senders, like in e.g., peer to peer video conferences. Usually, it is assumed that neighbors of a sending peer can use the information as soon as they receive it. However, the collaboration framework we use requires a centralized repository, that also acts as a synchronizer and a mechanism to avoid and to resolve conflicts. For this reason remote peers (i.e., the peers who do not produce the modification) can accept messages only after they have been accepted and pre-processed by the root.

We assume that the tree used for collecting the updates is the same as the multicast tree. Alternatively, the update tree could have different topology. However, this would put the further burden on tree construction and maintenance.

We introduce the following notation that will simplify the formulation of the optimization criteria. The degree of a node $deg(u)$ is the number of (immediate) children node u has ($deg(u) = |\{v_i : (u, v_i) \in E_t\}|$). A path $\pi(u, v)$ in T is an ordered sequence of nodes $(v_0, v_1, \ldots, v_n)$ such that $v_0 = u$, $v_n = v$ and $\{(v_0, v_1), (v_1, v_2), \ldots, (v_{n-1}, v_n)\} \subseteq E_t$. The total latency $L_\pi = L(u, v)$ over path $\pi(u, v)$ is the sum of latencies of edges in the path, $L_\pi = \sum_{i=0,\ldots,n-1} l(v_i, v_{i+1})$. The total cost C_π is similarly defined as $C_\pi = C(u, v) = \sum_{i=0,\ldots,n-1} c(v_i, v_{i+1})$.

A. Optimization Criteria

We define the following optimization criteria over T:

1) tree's weighted end-to-end latency L: This criterion expresses the latency between the moment any of the peers sends an update and the moment when this update is received by the furthest peer in the network. The latency is averaged over all peers with weights equal to the sending rates $\rho(v_i)$ (so that the peers that send updates more frequently have more influence over the criterion). Thus,

$$L = \sum_i \rho(v_i)\big(L(v_i, root) + \max_{j \neq i} L(root, v_j)\big). \quad (1)$$

In the tree in Fig 3, L equals to:

$$L = 0.08\big(0 + (45 + 8 + 12)\big) + 0.06 \cdot 70 + 0.06 \cdot 72 + \\ + 0.05 \cdot 110 + 0.6 \cdot 112 + 0.1 \cdot 118 + 0.05\big(65 + (45 + 8)\big).$$

The first element computes the observed latency of node P1 as its workload (0.08) multiplied by the cost of the path to the root (0, as root is the highest node) and the latency to the furthest node (65). The following lines add the latency for nodes P2, P3, S1, S2, S3, S4. For S4, the downlink latency is the latency to the next furthest node (S3).

2) tree's aggregated cost C: The aggregated cost expresses the weighted cost of sending and receiving updates through the multicast tree. For each node v, the cost of sending the update is $C(v, root)$. Then, this update is multicasted to all the other nodes in the tree. Each edge in the tree is used, except $(parent(v), v)$ if v is a leaf node. By aggregating these costs over all the nodes and weighting it by $\rho(v)$, we obtain:

$$C = \sum_{v \in V} \rho(v) C(v, root) + \sum_{v \in V} c(parent(v), v) \\ - \sum_{u:leaf} \rho(u) c(parent(u), u). \quad (2)$$

In the tree in Fig 3, C equals to:

$$C = \big(0.05 \cdot 10 + 0.6 \cdot 10 + 0.1 \cdot 10 + 0.05 \cdot 10\big) + \big(0 + 0 + 10 + 0 + 0\big) + - \big(0.06 \cdot 0 + 0.06 \cdot 0 + 0.6 \cdot 0 + 0.05 \cdot 0\big).$$

The first line computes the uplink cost of nodes S1 to S4 as the product of their workload (0.05 for S1) multiplied by the total uplink cost (10 for S1). The second line adds the costs of all the edges. The last line subtracts the costs of leaf nodes, for instance for P2 its participation 0.06 is multiplied by the cost of the link to its parent, 0.

3) tree's instability Λ*:* a disconnection of a node u affects its children $\{v_i : (u, v_i) \in E_t\}$ that must find the new parent node. For the sake of the theoretical analysis, we make a usual assumption that nodes' failures are independent, and thus with high probability no two nodes disconnect at the same time. Thus, the instability of the tree can be computed as the average number of nodes affected by disconnection in a unit of time:

$$\Lambda = \sum_{v \in V} \lambda(v) deg(v). \tag{3}$$

In the tree in Fig 3, Λ equals to: $\Lambda = 0.2 \cdot 3 + 0.6 \cdot 2 + 0.1 \cdot 1$. Node's P1 impact on instability is its λ (0.2) multiplied by the total number of children (3). The formula also takes into account nodes S1 and S3.

B. Multiobjective Problem

The multiobjective optimization problem is defined as finding a *spanning tree* T^* on G that minimizes the latency L_{max}, minimizes the cost C and minimizes the instability Λ:

$$(\min L, \min C, \min \Lambda), \tag{4}$$

subject to:

$$\forall v \ deg(v) \leq deg_{\max}(v) \tag{5}$$

Note that the notation $(\min L, \min C, \min \Lambda)$ specifies only that the three functions are to be simultaneously minimized. However, it leaves open the meaning of the multiobjective minimization, and the kind of solution, or solutions, that are to be returned [6]. For instance, one possibility would be to return any *Pareto optimal* solution (i.e., a solution $\mathbf{y} = (l, c, \lambda)$ such that there is no solution $\mathbf{y}' = (l', c', \lambda')$ with values better for all the criteria $l' < l$, $c' < c$, $\lambda' < \lambda$). The other possibility is to return all the Pareto-optimal solutions. However, many multi-criteria combinatorial optimization problems are *intractable*, that is the number of all Pareto-optimal solutions is exponential. It is also possible to return a certain Pareto-optimal solution, being, for instance, the weighted average of the criteria.

C. Special Case Considered

In the rest of the paper, we restrict the general problem defined in the previous section to a class of problems that is realistic enough to model most of the real world scenarios and, at the same time, has boundary problems that can be solved by exact algorithms. We assume that all the latencies are symmetric ($l(u, v) = l(v, u)$). While symmetric latencies can be easily measured by measuring the round-trip time between nodes, it is hard to measure one-way latency. Similarly, costs are symmetric ($c(u, v) = c(v, u)$), which reflects usual network providers' charging schemes. We also assume that the collaborative application does not require significant bandwidth, thus there are no limits over the maximal degree of nodes $deg_{\max}(v) = \infty$.

II. Problem Analysis and Proposed Algorithms

A. Individual Subproblems

This section analyzes the complexity and the optimality for individual monocriteria subproblems, that is minimizing the maximal weighted end-to-end latency L_{max}, minimizing the aggregated cost C and the stability Λ.

1) Instability: $\min \Lambda$ problem is equal to the directed minimum spanning tree in a graph $G = (V, E)$, in which the cost $\lambda(v, w)$ of the edge (v, w) is equal to the disconnection rate $\lambda(v)$ of node v. Recall that $\Lambda = \sum_{v \in V} \lambda(v) deg(v)$, which, in such a spanning tree corresponds to $\sum_{(v,w) \in T} \lambda(v, w) = \sum_{(v,w) \in T} \lambda(v)$. Given a root r, such a tree can be found by Chu-Liu/Edmonds [7] algorithm in $O(|V|^2)$ for dense graphs. Moreover, as the following lemma states, the tree must be rooted at the most reliable node.

Lemma 1. *The instability is minimized by a spanning tree T_Λ rooted at the most reliable node $r_\Lambda = \arg\min_v \lambda(v)$.*

Proof: The proof is by contradiction. Assume that tree T, different than T_Λ is is optimal for Λ. Consequently, T must have a different root r' than T_Λ. Consider path $\pi(r', r_\Lambda)$ in T. Let us modify T into T' so that r_Λ becomes the new root and π is reversed. In T', the number of children of r_Λ increases by one and that of r' decreases by one. The number of children of all the other nodes is the same. Thus, the instability in T' changes by $\lambda(r_\Lambda) - \lambda(r') < 0$, which leads to a contradiction with the assumption that T is optimal for Λ. ■

Note that if the maximum degree of some nodes is bounded, this model becomes NP-hard ([8, ND3]).

2) Latency: The main difficulty in analyzing $\min L$ is caused by the fact that the same tree is used for upstream and downstream messages. $\min L$ problem is a special case of the optimal communication spanning tree (OCT) problem [9]. In OCT, given communication *requirements* $\rho(u, v)$ for each pair of nodes, the task is to find a spanning tree such that $\sum_{u,v \in V} \rho(u, v) L(u, v)$ is minimized. OCT is NP-hard, even for restricted cases when only some fixed number of senders $S \subset V$ ($|S| > 1$) communicate (i.e., $\rho(u, v) \neq 0$ only for $u \in S \subset V$). The combinatorial complexity is caused by the fact that senders communicate with *all* the other nodes.

In our problem, however, the set of communicating pairs is more limited. Using u' as the furthest node ($u' = \arg\max L(root, v)$) and u'' as the second furthest, L can be

rewritten as:

$$L = \sum_i \rho(v_i)\big(L(v_i, root)\big) + (1-\rho(u'))L(root, u') + \rho(u')L(root, u''). \quad (6)$$

The first element represent the upstream latency from node v_i to the root, the second one is the downstream latency between the root and the furthest node and the third one is an adjustment when u' is the source of the message (as it does not have to be informed). This OCT has a particular structure of requirements with only non-zero requirements being $\rho(u,v) = \rho(u)$ for $v = root$, $\rho(u,v) = 1-\rho(u')$ for $u = root$, $v = u'$ and $\rho(u,v) = \rho(u')$ for $u = root$, $v = u''$. However, in our problem the nodes u' and u'' are identified only *after* the tree is constructed.

Even in the case of symmetric latencies, the problem is still different from the usual Shortest Path Tree (SPT) problem, because of $\max$ element and non equal weights ρ. However, the following lemma shows that a SPT spanning tree is optimal.

Lemma 2. *If the latencies are symmetric ($l(v_i,v_j) = l(v_j,v_i)$), the SPT spanning tree with minimal L among all SPT spanning trees is optimal for the weighted end-to-end latency L.*

Proof: The proof is by contradiction. Let us denote as r^* the root node of the SPT spanning three T^* with minimal $L = L^*$. Assume that $L' < L^*$ for some spanning tree T' rooted at r'.

Denoting as k the furthest node from r ($k = \arg\max_i L(r, v_i)$) , we rewrite L as follows:

$$\begin{aligned} L &= \sum_i \rho(v_i)\big(L(v_i,r) + \max_{j\neq i} L(r,v_j)\big) \\ &= \sum_{i\neq k} \rho(v_i)L(v_i,r) + (1-\rho(v_k))L(v_k,r) + \\ &\quad \rho(v_k)\big(L(v_k,r) + \max_{i\neq k} L(v_i,r)\big) \end{aligned}$$

If the root of both trees is the same ($r^* = r'$), $L' < L^*$ implies that there is some node v_j, for which $L(v_j,r') < L(v_j,r^*)$, which means that L' has shorter path to v_j, which contradicts the assumption that the tree is an SPT tree. If the root is different, compare T' with a SPT $T^{SPT'}$ rooted at r'. By the same argument, it is not possible that the distance $L(v_j,r')$ in T' is less than the same distance in $T^{SPT'}$. Furthermore, $L^{SPT'} \geq L^*$, as T^* has the optimal latency among all the SPT trees. ∎

When the maximum degree of the tree is limited, this model becomes NP-hard (even with symmetric latencies), as SPT with bounded degree is NP-hard [10].

3) Cost: Equation (2) denoting the cost of a tree is composed of three components. $\min\sum_v \rho(v)C(v,root)$ is a weighted shortest path between v and r, optimized by shortest path spanning tree. $\min\sum_v c(parent(v),v)$ is the total cost of all the edges, optimized by the usual minimal spanning tree. Finally, $\max\sum_{u:leaf}\rho(u)c(parent(u),u)$ (denoted as Maximum Weighted Leaf Spanning Tree Problem, MWLSPT) is, itself, NP-hard, as it generalizes the Maximum Leaf Spanning Tree Problem (MLSPT) [8, problem ND4].

Lemma 3. *The decision version of $\sum_{u:leaf}\rho(u)c(parent(u),u)$ is NP-complete.*

Proof: We reduce the MLSPT to MWLSPT. From an instance of MLSPT $G_{MLSPT} = (V_{MLSPT}, E_{MLSPT})$ we construct an instance of MWLSPT $G = (V,E); c; \rho$ as follows. The set of vertexes and edges are the same ($G = G_{MLSPT}$. The costs of all edges are equal to 1. The participation rates ρ of all the vertexes are equal to $1/|V|$.

If there is a spanning tree with

$$\sum_{u:leaf}\rho(u)c(parent(u),u) \geq \frac{k}{|V|},$$

the same tree also solves MLSPT with at least k leafs, as

$$\sum_{u:leaf}\rho(u)c(parent(u),u) = \frac{1}{|V|}\sum_{u:leaf} 1.$$

∎

$\min\sum_v \rho(v)C(v,root) + \sum_v c(parent(v),v)$ problem is related to the NP-complete Minimum Diameter Spanning Subgraph [8, problem ND6], in which graph's diameter is minimized, given a budget on the spanning tree's cost. Another, similar problem is the problem of finding a tree that balances the total weight of the edges and the length of paths [11] (called LAST, Light Approximate Spanning Tree). In this problem, the goal is to find a tree with total weight of at most β times the total weight of the minimum spanning tree and which extends the length of path between the root and each vertex by at most α. [11] proposes a polynomial algorithm that adjusts Minimum Spanning Tree (MST). The algorithm does a depth-first search. For each vertex v, if the current path length breaks the shortest path requirement, the vertex is switched to its SPT path (by adding all the missing edges on SPT path between v and $root$). The algorithm is $(\alpha, 1+\frac{2}{\alpha-1})$ approximation of, accordingly, the length of each SPT path and the total weight of the minimum spanning tree. The main difference between LAST and $\min C$ is that $\min C$ optimizes the *weighted sum* of shortest paths and MST. Another related algorithm is MENTOR [12], a heuristics that combines Prim's MST and Dijkstra's SPT by modifying the edge scoring function. In Prim's MST, edge's (v,w) score is $c(v,w)$. In MENTOR, the scoring function adds the distance between the root node and the vertex multiplied by a constant $\alpha \in [0,1]$, so the score is $c(v,w) + \alpha C(r,v)$.

The problem of finding a polynomial algorithm optimizing C (or even $\min\sum_v \rho(v)C(v,root) + \sum_v c(parent(v),v)$) is still open. Currently, we use a heuristics that is a variant of LAST [11]. Our algorithm builds a MST and starts a depth-first search. Each vertex v is tentatively switched to its SPT path. For each vertex w on the SPT path between v and $root$, its parent on the SPT path replaces the current parent. Then, if the resulting tree has lower C, the tentative switch is accepted and

the current tree is modified; otherwise the process continues with the original tree. Finally, after visiting child w (and, thus, all its descendants), if w switched to its SPT parent, the original parent v tentatively becomes w's child (but only if it does not cause a cycle in the tree). If the resulting tree has lower C, such a change is accepted and the current tree is modified.

When the maximum degree of the tree is limited, this model also becomes NP-hard, as it involves both the spanning tree and the shortest path problems with bounded degree.

B. Multiobjective Problem

In order to solve the general multiobjective problem $(\min L, \min C, \min \Lambda)$, two issues must be addressed. Firstly, the meaning of multiobjective minimization and the kind of solutions we want to obtain. We propose to return a solution that is the closest to the *ideal* solution, constructed from the optimal values of monocriterion problems. Secondly, the problem of finding an algorithm that produce that solution. Currently, we propose to use exhaustive search over all possible spanning trees.

1) Approximating the ideal solution: We expect that the number of Pareto-optimal solutions is exponential, as the multiobjective problem involves bi-criteria versions of shortest paths and minimum spanning trees, which are intractable [6]. Thus, we chose only one Pareto-optimal solution as the solution to the general problem. This solution is defined as a solution minimizing the distance to the *ideal* solution $y^I = (L^*, C^*, \Lambda^*)$, where L^* is the optimal latency ($L^* = \min_T L(T)$), C^* is the optimal cost and Λ^* is the optimal stability.

Ranges of possible values of L, C and Λ can differ significantly. For instance, assume that $\Lambda^* = 0.1$ and the worst value of Λ for some tree is 10, whereas $C^* = 100$ and the worst cost is 1000. When using normal, euclidean distance over unscaled values, the distance from the ideal point $(C, \Lambda) = (100, 0.1)$ to $(100, 10)$ is 9.9, whereas to $(1000, 0.1)$ is 900. Thus, such a measure is much more sensible to changes in the function that has larger numerical values. However, the distance to these two solutions should be the same – the first point represents the solution minimal for C and maximal for Λ, whereas the second the opposite.

To make the distance measure unaffected by numerical values of functions, we scale the value of each function by the boundary values. However, in general case, the problem of determining the maximal value of some of the considered functions is hard (for instance, the longest path problem is NP-hard). That is why we scale each function $y \in \{L, C, \Lambda\}$ between its optimal value y^* and its approximated nadir $\widetilde{y}^N$. We define $\widetilde{y}^N$ as the maximal value of y observed in the optimal solutions for other functions [6]. For instance, denoting T_C as the tree minimizing C, and T_Λ as the tree minimizing Λ, the approximated nadir of the latency $\widetilde{L}^N$ is the maximum from the latencies of T_C and T_Λ, $\widetilde{L}^N = \max\big(L(T_C), L(T_\Lambda)\big)$. Note that it is possible that for some tree T, $L(T) > \widetilde{L}^N$.

After determining the approximated nadirs for each function, we scale functions to:

$$y'(T) = \frac{y(t) - y^*}{\widetilde{y}^N - y^*}. \tag{7}$$

Observe that, for the optimal tree T^* for y, the value of the scaled function is $y'(T^*) = 0$.

Thus, the function to minimize is:

$$dist(T) = \sqrt{\Big(\big(L'(T)\big)^2 + \big(C'(T)\big)^2 + \big(\Lambda'(T)\big)^2\Big)}. \tag{8}$$

2) Producing the approximated ideal solution: We propose to find a tree minimizing $dist(T)$ by an exhaustive search over all spanning trees. The number of spanning trees is exponential and equal to n^{n-2} [9] in the worst case of a fully-connected graph. Thus, the exhaustive search is clearly not efficient, especially for larger graphs. However, a session of collaborative editing usually has a few, rather than many, users. Consequently, exhaustive search is an acceptable algorithm for most of the scenarios of collaborative work.

III. Experimental Evaluation

We evaluated the algorithms on randomly-constructed connectivity graphs of $n \in [5, 10]$ vertexes. The graph models the logical, and not physical, connectivity between nodes, which makes most of the topology generators unfit for the task.

We group nodes into $s \sim U(2, 4)$ *sites*, that model different physical locations. Within each site, latencies are low ($\sim U(5, 20)$) and costs are negligible (0). There are two classes of nodes in a site: *peers* (at least one), that model standard PCs, and *leafs* that model mobile devices such as UMPCs or smart phones. A peer is able to connect to every other peer in its site. A leaf can connect only to its peer. Additionally, each peer is firewalled with *probability of* 0.8. Two firewalled peers from different sites cannot directly connect. We assumed that in the network there is at least one peer that is not firewalled. Connections between sites have non-zero costs ($\sim U(10, 20)$) and considerably higher latencies ($\sim U(100, 500)$). Cost and latencies between all pairs of peers from any two sites are similar. Instabilities $\lambda(v)$ are generated with $U(0, 1)$. Finally, sending rates $\rho(v)$ are generated with *normalized zeta distribution*, as we assume that users' contributions to a shared document vary substantially.

The results of experiments are summarized in Table I. In order to obtain meaningful comparisons between random graphs with different n, we present scaled values of each function (Eq. 7). Consequently, for each graph and each measure, 0 is the optimal value and 1 is the approximated nadir (corresponding to the maximum value of the measure on T_L^*, T_C^* and T_Λ^* for the graph). The table presents averages and standard deviations computed over 100 random graphs with the specified number of nodes n. T_L^* is the tree returned by optimal latency algorithm (Section II-A2), T_C^* – by adjusted LAST (Section II-A3), T_Λ^* – by MST on stability graph (Section II-A1). *multiobjective* is the solution chosen by the multiobjective algorithm (Section II-B). A number of phenomena can be observed.

TABLE I

RESULTS OF EXPERIMENTAL EVALUATION. THE TABLE PRESENTS SCALED PERFORMANCE MEASURES (EQ. 7) AVERAGED OVER 100 RANDOM GRAPHS FOR EACH n. T^*_L IS THE TREE RETURNED BY OPTIMAL LATENCY ALGORITHM (SECTION II-A2), T^*_C –BY ADJUSTED LAST (SECTION II-A3), T^*_Λ – BY MST ON STABILITY GRAPH (SECTION II-A1). *multiobjective* IS THE SOLUTION CHOSEN BY THE MULTIOBJECTIVE ALGORITHM (SECTION II-B).

	T^*_L				T^*_C				T^*_Λ				multiobjective					
	$C(T)$		$\Lambda(T)$		$L(T)$		$\Lambda(T)$		$L(T)$		$C(T)$		$L(T)$		$C(T)$		$\Lambda(T)$	
n	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
5	0.49	0.41	0.72	0.38	0.65	0.40	0.80	0.29	0.73	0.37	0.85	0.33	0.23	0.25	0.27	0.28	0.35	0.27
6	0.54	0.41	0.72	0.37	0.75	0.36	0.79	0.28	0.62	0.39	0.84	0.33	0.24	0.25	0.28	0.29	0.28	0.24
7	0.46	0.38	0.74	0.30	0.72	0.37	0.79	0.30	0.79	0.30	0.83	0.35	0.24	0.24	0.25	0.25	0.30	0.21
8	0.47	0.35	0.75	0.34	0.72	0.36	0.84	0.24	0.74	0.33	0.89	0.28	0.21	0.20	0.27	0.24	0.32	0.22
9	0.51	0.35	0.80	0.30	0.72	0.33	0.82	0.27	0.83	0.26	0.89	0.26	0.26	0.23	0.24	0.17	0.30	0.16
10	0.48	0.32	0.79	0.25	0.73	0.33	0.86	0.23	0.81	0.28	0.95	0.19	0.19	0.15	0.24	0.18	0.30	0.15

Firstly, the trees efficient for one objective tend to be inefficient for other objectives. Algorithms optimizing one objective have, on the average, score of 0.73 for the other objectives. The difference is especially visible when the absolute, and not rescaled, values of Λ are analysed (note that we do not present full results in the paper because of space constraints). On the average, Λ in trees optimal for L and C is 5.43 times worse than the optimal value, compared with 2.00 performance drop for L and 2.23 for C (averaged over trees optimal for C, Λ and L,Λ, respectively).

Secondly, the solutions returned by the multiobjective algorithm have a good trade-off between all the objectives. The average performance is better than the mono-objective optimization for all the objectives. Moreover, standard deviations are also lower, which denotes a more stable behavior of the algorithm.

Thirdly, the instability seems to be the hardest objective to optimize. The algorithms for C and L tend to be inefficient on Λ (0.78). Similarly, the trees optimal for Λ result in inefficient C and L (0.81). Additionally, the multiobjective algorithm has the worst average performance for Λ $(0.31$, compared with 0.23 for L' and 0.26 for $C')$.

Additionally, we checked how far are the costs of trees produced by LAST to the best trees in terms of C found by the exact algorithm (Table II). The table shows that LAST is suboptimal only in about 1% of instances. Additionally, even in these instances, the cost of the LAST tree is less than 5% from the optimal cost.

IV. RELATED WORK

Early approaches proposed to extend the Internet Protocol (IP) [13] for multicast. However, as the IP multicast has never been widely deployed and adopted, the only way to send the data to multiple receivers is to extend the application layer. We summarize some of the application layer multicast approaches below (see also [14] for comparison).

TABLE II

EVALUATION OF LAST FOR $\min C$ PROBLEM: PERCENTAGE OF INSTANCES IN WHICH LAST WAS NOT OPTIMAL AND THE AVERAGE RELATIVE ERROR TO THE OPTIMAL SOLUTION.

n	instances [%]	error [%]
5	0	0
6	1	4
7	3	3
8	0	0
9	1	0.2
10	2	2

The most straightforward way to construct a application-layer multicast tree is to optimize the latency in a greedy manner. In Yoid [15], a node connecting to a multicast tree chooses a random subset of nodes that are in the tree and connects to the one with minimal latency. Periodically, each node switches its parent, if the switch reduces the latency. In Overcast [16], a newly connecting node probes the latency to the root of the multicast tree and all of its immediate children. If the latency to one of the children is lower than to the root, the algorithm recursively probes that child's children, until either a leaf node is reached, or no child results in a lower latency. Fastcast [17] is similar, but considers the end-to-end latency over the whole path between the root and the newly connected node, not just the latency to any node in the tree.

In contrast, Narada [18] relies on a constantly-updated mesh, similar to what we define as a connectivity graph. Each node periodically adds links to other, randomly chosen nodes and drops existing, inefficient links. The data dissemination paths are produced by a standard, distance-vector algorithm.

Some other application-layer multicast methods are specific to a certain peer-to-peer overlay. For instance, Scribe [19] builds multicast trees based on reverse path forwarding in Pastry overlay. Bayeux [20] uses forward path forwarding, and Borg [21] combines the two approaches.

There are a number of differences between the aforementioned approaches and our proposal. Firstly, the way in which collaborative applications communicate differs significantly both from one-to-many and from many-to-many multicast. In one-to-many multicast, there is only one source of data. In the typical many-to-many multicast, many nodes can update the data, but the messages can be consumed by nodes that lie on the dissemination path. In collaborative applications, however, the messages must be firstly send to the node acting as a synchronization server, and only then multicasted to the group. Secondly, as collaborative sessions are long and peers can be well identified, we are able to collect and take into account more data on peers' behavior. Thus, we can explicitly take into account not only the latency, but also the cost of the links and the observed participation levels and disconnection rates.

On the theoretical level, the alternative to the spanning tree

is to model multicast as a Steiner tree. These approaches assume that some nodes only pass on the information. For instance [22] considers the problem of asymmetric latencies and [23] proposes a heuristics for bi-criteria optimization of latencies and costs.

V. Conclusions and Future Work

The paper models and proposes solutions to the problem of the network design to collect and disseminate updates to a shared document in the context of interactive collaborative applications. Collaborative applications require a central server that synchronizes users' actions. Consequently, messages containing updates are first aggregated at the synchronizer and then multicasted to other users.

We proposed to build a spanning tree to collect and disseminate the updates. We measure the performance of the tree by three metrics. The *latency* L expresses the delay between the moment a user updates her local copy of data and the change is propagated through the network. The *cost* C represents the (monetary) cost of sending information in the tree. Finally, the *instability* Λ represents the average number of nodes affected by disconnections from the network. The cost and the latency are weighted by the observed average participation of users in collaborative editing. The goal is to optimize the infrastructure so that the users who update the document more often have more influence over the performance measure.

We analyzed the boundary problems of minimizing latency, instability and cost, provided the complexity results and proposed exact, polynomial algorithms for the first two measures. We solved the general, multiobjective problem by producing a solution closest to the ideal solution, composed from the optimum values of all three functions. We performed an experimental evaluation of the algorithms that showed that it is important to consider the multiobjective problem, as optimal solutions for one performance measure can be far from optimal values of the other ones.

The paper summarizes the initial stage of the research. We considered static (steady-state) problem and proposed centralized algorithms. Clearly, there are many immediate extensions of this work.

In our subsequent work, we have implemented a Simulated Annealing algorithm for the multiobjective problem, which efficiently optimizes larger graphs. We are currently working on an implementation of the communication library that we plan to deploy in a real network environment. The prototype uses dynamic algorithms that locally adjust the spanning tree when a node connects or disconnects. Once the multiobjective performance of the tree becomes too far from the optimum, the original algorithm can be used to rebuild the whole tree.

Another interesting problem is the equity of the produced tree for individual users. Currently, the tree is biased towards heavy contributors, which may cause unacceptable performance for some users. In our implementation, we plan to address this issue by guarantees on the minimum performance for each user.

Acknowledgements Authors thank prof. C. Sun for his insightful remarks which we hope will help refine and make the current model more realistic in future.

References

[1] C. Sun and C. Ellis, "Operational transformation in real-time group editors: issues, algorithms, and achievements," in *CSCW, Proceedings*. ACM Press, 1998, pp. 59–68.

[2] M. Suleiman, M. Cart, and J. Ferrié, "Concurrent operations in a distributed and mobile collaborative environment," in *ICDE, Proceedings*, 1998.

[3] C. Sun, S. Xia, D. Sun, D. Chen, H. Shen, and W. Cai, "Transparent adaptation of single-user applications for multi-user real-time collaboration," *ACM TOCHI*, vol. 13, no. 4, pp. 531–582, 2006.

[4] S. Xia, D. Sun, C. Sun, D. Chen, and H. Shen, "Leveraging single-user applications for multi-user collaboration: The coword approach," in *ACM CSCW, Proceedings*, 2004.

[5] A. Agustina, F. Liu, S. Xia, H. Shen, and C. Sun, "Comaya: Incorporating advanced collaboration capabilities into 3d digital media design tools," in *ACM CSCW, Proceedings*, 2008.

[6] M. Ehrgott, *Multicriteria Optimization*. Springer, 2005.

[7] Y. Chu and L. Tseng-hong, "On the shortest arborescence of a directed graph," *Sci. Sin.*, vol. 14, pp. 1396–1400, 1965.

[8] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Potasi, *Complexity and Approximation*. Springer, 1999.

[9] B. Wu and K. Chao, *Spanning Trees and Optimization Problems*. Chapman & Hall/CRC, 2004.

[10] P. Jurcik and Z. Hanzalek, "Construction of the Bounded Application-layer Multicast Tree in the Overlay Network Model by the Integer Linear Programming," in *IEEE ETFA, Proceedings*, vol. 2, 2005.

[11] S. Khuller, B. Raghavachari, and N. Young, "Balancing minimum spanning trees and shortest-path trees," *Algorithmica*, vol. 14, no. 4, pp. 305–321, 1995.

[12] A. Kershenbaum, P. Kermani, and G. Grover, "MENTOR: an algorithm for mesh network topological optimization and routing," *Communications, IEEE Transactions on*, vol. 39, no. 4, pp. 503–513, 1991.

[13] S. Deering and D. Cheriton, "Multicast routing in datagram internetworks and extended LANs," *ACM Transactions on Computer Systems (TOCS)*, vol. 8, no. 2, pp. 85–110, 1990.

[14] M. Castro, M. Jones, A. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman, "An evaluation of scalable application-level multicast built using peer-to-peer overlays," in *INFOCOM, Proceedings*, 2003.

[15] P. Francis, Y. Pryadkin, P. Radoslavov, R. Govindan, and B. Lindell, "Yoid: Your Own Internet Distribution," Technical Report, ACIRI, 2000.

[16] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. O'Toole Jr, "Overcast: reliable multicasting with an overlay network," in *USENIX, Proceedings*, 2000.

[17] A. Wierzbicki, R. Szczepaniak, and M. Buszka, "Application Layer Multicast for Efficient Peer-to-Peer Application," in *Proc. of IEEE WIAPP*, 2003, pp. 126–130.

[18] Y. Chu, S. Rao, S. Seshan, and H. Zhang, "A case for end system multicast," *Selected Areas in Communications, IEEE Journal on*, vol. 20, no. 8, pp. 1456–1471, 2002.

[19] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, "Scribe: a large-scale and decentralized application-level multicast infrastructure," *Selected Areas in Communications, IEEE Journal on*, vol. 20, no. 8, pp. 1489–1499, 2002.

[20] S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiatowicz, "Bayeux: An Architecture for Scalable and Fault-tolerant Wide-area Data Dissemination," in *NOSSDAV, Proceedings*, 2001.

[21] R. Zhang and Y. Hu, "Borg: a hybrid protocol for scalable application-level multicast in peer-to-peer networks," in *NOSSDAV, Proceedings*. ACM Press, 2003, pp. 172–179.

[22] S. Ramanathan, "Multicast tree generation in networks with asymmetric links," *Networking, IEEE/ACM Transactions on*, vol. 4, no. 4, pp. 558–568, 1996.

[23] V. Kompella, J. Pasquale, and G. Polyzos, "Multicast routing for multimedia communication," *Networking IEEE/ACM Transactions on)*, vol. 1, no. 3, pp. 286–292, 1993.