Multi-objective Optimization of Multicast Overlays for Collaborative Applications ☆

Krzysztof Rzadca, Jackson Tan Teck Yong, Anwitaman Datta

School of Computer Engineering
Nanyang Technological University
Singapore

Abstract

Current real-time collaborative application implementations use dedicated infrastructure to carry out all communication and synchronization activities. Specifically, they require all end nodes to communicate directly with and through the central server. In this paper, we investigate scenarios in which the most resource intensive functionality of continuous communication among collaborators to disseminate changes is decentralized, utilizing the end users themselves 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 and repeated, it is possible to gather and leverage node behavior (their instabilities and frequency of sending updates) and communication links (latencies and average costs). Several criteria to determine the quality of a multicast tree can be identified: cost, latency and instability. In this paper we analyze the complexity of the problem of finding optimal communication topologies, and propose approximate algorithms to optimize the same. We also consider the multiobjective problem in which we search for a topology that provides good trade-off between these sometimes conflicting measures. Validation of our proposed 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 for the other metrics. Finally, we briefly present an implementation of a communication library that uses the proposed algorithms to periodically adjust the dissemination tree.

Key words: communication topology; multicast; collaborative applications; multiobjective optimization

1. Introduction

Complex projects are carried out in a collaborative manner involving multiple participants. There are numerous ways in which such collaborative work can be supported, 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] or WOOT [3, 4] significantly simplify collaboration tasks by facilitating real time group editing. Furthermore, techniques like *transparent adaptation*[5] facilitate adoption of diverse traditionally single user applications—from text editors like Word [6] to multimedia content creation tools like Maya [7]—into groupware for real time collaborative editing.

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 synchronization mechanism — for example using a centralized synchronizer to carry out operational transformations, or vector clocks to carry out synchronization in a decentralized manner—and, finally, a communication mechanism among the users (and the central synchronizer, when applicable).

Collaborative applications may extend beyond traditional groupware. For instance, while very dif-

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

Email address: anwitaman@ntu.edu.sg (Anwitaman Datta)

ferent at the surface, multiuser groupware, social software, even massively multi-player online games (MMOGs) all are essentially multi-user collaborative applications. All these applications propagate users' actions and maintain consistency of the shared states, be it a document or a game environment. For the users' experience to be acceptable, these applications have stringent constraints on latencies and interruptions they may tolerate.

Current implementations of such groupware, from desktop based CoWord [6] and CoMaya [7], to Web based Google Wave [8] rely on a dedicated infrastructure to carry out all these functions, and require all end nodes to communicate directly with and through a central server.

While there are many advantages in supporting all these functions in a centralized manner using dedicated infrastructure, there are also several motivating factors for using decentralized or hybrid approaches.

Consider moderately large sized collaborative groups carrying out communication intensive realtime collaborative editing: for instance, in complex civil engineering designs (such as hospitals, airports), dozens of engineers, architects and specialists of different branches work on the same project in parallel. 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. Likewise, in massively multi-player online games (MMOG) the game universe is typically split into small regions in each of which tens to hundreds of users interact. The users in a game region are interested in actions and updates for the game region, rather than the whole game universe. Thus, all users within each game region may be considered as a single session. An alternative scenario is a smaller-scale online multiplayer game, involving tens of players in a single session, but many independent sessions played in parallel.

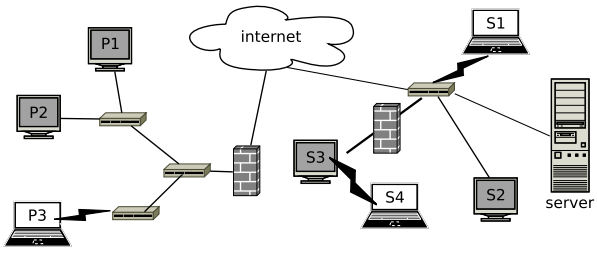

Figure 1: Approximate physical network structure with peers grouped in two physical locations. Two groups of peers are behind firewalls.

Even if the individual groups are not very large, if there are many object sessions and all users use the same infrastructure, it can generate heavy communication load at the server when all the communication messages need to go through such a server, as is the case with current implementations of several groupware [6, 7, 8].

Such an overload is not only undesirable but also easy to avoid, by allowing the users involved in a collaborative session to communicate in a peer-to-peer manner. In multi-player gaming, using peer-to-peer mechanisms to alleviate infrastructure load is already a relatively mainstream idea [9, 10, 11]. Another advantage of such a decoupling of the communication from infrastructure based solutions is that participants can collaborate in an ad-hoc manner, even if they do not have connectivity to the central infrastructure. For example, if all the users are traveling by train, or if the users are working within a university with fast LAN connection, but poor Internet connection.

We envision that in general, in this age of mobile knowledge workers, proliferated with sophisticated portable computing cum communication devices, it is thus essential to support nomadic collaboration, which is served well by supporting collaborative sessions even in absence of infrastructure. Such solutions may either be stand-alone or be compatible with infrastructure (hybrid solutions) as and when necessary or possible.

In order to decouple the collaborative session from dedicated infrastructure – it is essential not only to provide a communication mechanism to communicate with all members of the collaboration session, but also to synchronize their editing activities so that all participants have a consistent view of the collaboratively manipulated objects. Such synchronization can however be readily achieved by either adopting a centralized approach within the session, where an elected member acts as the synchronizer (from software engineering perspective, this is simpler), or can also be achieved in a decentralized manner by the use of vector clocks [12].

Current groupware implementations also assume that end users have reliable and fast connection, and membership changes in a session are infrequent, and are not optimized for various kinds of heterogeneity that occurs in practice. Geographic distribution of collaborators, increased mobility of knowledge workers as well as proliferation of diverse portable devices like *Ultra Mobile PCs* (UMPC) and smart-phones require a more flexible support

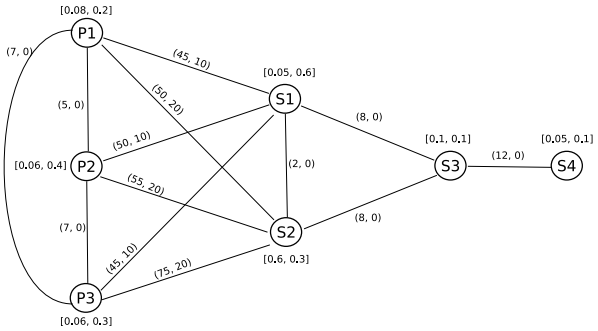

Figure 2: Connectivity graph for the physical network structure presented in Figure 1. Values next to a node are its workload ρ (higher values denoting more frequent updates) and instability λ (higher values denoting more frequent disconnections). Values next to an edge are its latency l and its cost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

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). Likewise, wireless networks are much more varied than corporate LANs in availability, performance and costs.

These considerations motivate the design of 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 2).

Typically, some users contribute to a shared document more than others. The structure of the connectivity tree should take this into account by proposing an architecture 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.

Nodes disconnect from the network with different rates, that depend on factors such as the quality of network connection (Ethernet vs. GPRS) or the type of the node (a standard PC vs. a 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.

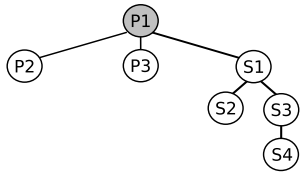

Figure 3: A multicast tree in which, by using intermediate nodes, data on the backbone link is not replicated.

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) may have significant monetary *cost*, while others are free/cheap (local Ethernet or local wireless), which should be taken into account.

Consider an example of 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.

In a more dynamic collaboration scenario, one of the nodes takes also the role of the server that synchronizes the rest of the nodes (this role 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, determining the best topology is often non-trivial, and 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 analyze the problem from both possible designs that can be used in collaborative applications: decentralized communication (as in many-to-many multicast), and centralized communication, in which all the messages must be sent

to a synchronizer, before being disseminated to the rest of the peers.

We model the problem as a multiobjective optimization problem (Section 2) that optimizes the stability, the latency and the cost of the multicast tree. Then, we formulate these performance measures in the two dissemination scenarios. In the centralized communication scenario, 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. In decentralized communication scenario, both measures are slightly modified: latency takes into account the longest path in the tree; and cost considers only the cost of the multicast. In both communication scenarios, we model the (in)stability as the average number of disconnections of a node in an unit of time. In both scenarios, 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 3). We then propose a multiobjective (approximate) algorithm based on a hill climbing heuristic (Section 3.3). We validate the performance of our algorithms by simulation (Section 4).

There is substantial research effort devoted to application level multicast, both in one-to-many and in many-to-many scenarios. We further discuss related work and differentiate them from our work in Section 7. One-to-many multicast typically models transmissions of contents that cannot be modified by the group (such as Internet radio or RSS feeds). Many-to-many multicast models more interactive applications, such as teleconferences. Collaborative applications are somewhat similar to many-to-many multicast.

The main difference from typical many-to-many multicast is that for many current implementations of collaborative applications, the messages must be validated (and, perhaps, modified by operational transformation) by a logically central synchronizer before other members can use them to guarantee consistency. Multicast trees for collaborative applications must thus optimize both the information collection and the dissemination (after necessary processing and reordering). In order to compare the classic many-to-many multicast with server-based collaboration, we formulate our criteria for tree optimality in both dissemination strategies. Due to different communication patterns, the formulation of criteria differ. However, the general approach, and the type of criteria used is the same.

Another fundamental difference of our approach is that the majority of existing multicast trees are built to optimize a single criterion, typically the latency. Our experiments show that single-criterion optimized topologies are far from the optimum with respect to the other criteria (such as cost and stability). The multiobjective approach proposed in this paper facilitates a good trade-off between all the criteria.

The fact that collaborative sessions last long, and same collaborators participate in repeated sessions allows for the collection of statistical information (such as the relative contribution of each of the users in the collaboration), which can be used to calculate the weighted versions of latency and cost of the dissemination tree, in which each users' criterion is weighted by the user's participation.

Another (final) difference, thankfully to our advantage, is that the number of members in a session to edit an object is small in typical groupwares: for instance CoWord [6] supports a maximum of sixteen simultaneous users editing an instance of a document, and even in massively multiplayer online games, each game region has tens upto around a hundred users. This allows the problem of finding a multiobjective optimized overlay to be computationally tractable.

To summarize, in many scenarios it is important to consider more than the latency in building multicast trees, as trees optimized for one objective tend to be inefficient for other objectives. We propose approximate algorithms to determine such topologies, and validate with simulations and experiments that a good compromise between the criteria is obtained, and that such topologies are fast to compute by our hill climbing algorithm, and at the same time, very close to the optimal ones determined by exhaustive search. The algorithms are used by MoMo, our implementation for optimized multicast communication, briefly described in Section 5. We provide the implemented MoMo communication library as an open source project (`http://code.google.com/p/momocomm/`).

2. 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 apart, 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 $\lambda(v)$: the average number of disconnections from the network in the unit of time;
- sending rate $\rho(v) \in [0, 1]$: expresses the fraction of messages (or the fraction of the total size of messages) sent by v (i.e., messages produced by v). It enables measuring the relative importance of v as a source in the multicast network: the higher $\rho(v)$, the more messages it emits, thus, the more important it is to optimize the uplink between v and the rest of the network.

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

The objective is to build a spanning tree $T = (V, E_t)$ for G, with $E_t \subseteq E$.

We consider two dissemination scenarios: centralized and decentralized. The *centralized* scenario models a typical collaborative application in which the root $r(T)$ 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 $parent(v) = u : (u, v) \in E_t$, who forwards it to its parent, and so on, until it reaches the root. Note that the nodes along such a path do/can not utilize the message at the application level. The root accepts the modification (possibly re-ordering it with other, concurrent modifications originating from other peers or marking it as a conflicting one). Then, the notification is multicasted by the root back to all the other nodes. This is done by reusing the overlay. That is to say, the root sends the notification to its immediate children. These children nodes use such received message at the application layer, and also forward it further downstream to their children, until the notification reaches all the nodes in the tree.

The *decentralized* scenario models a usual multicast with multiple senders, e.g., peer-to-peer video conferences, or when vector clocks are used for synchronization of coedited document. In this scenario, we assume that neighbors of a sending peer can use the information as soon as they receive it. Thus, the modification notification of a peer v is sent to its immediate parent $u = parent(v)$. The parent can immediately use the message at the application level, besides forwarding it further to its parent $parent(u)$ and all its other children apart node v itself $(w \neq v : parent(w) = u)$.

Even though collaborative applications can be realized in a decentralized pattern (using careful synchronization with vector clocks), many implementations (such as CoWord [6] or Google Wave [8]) rely on a central point. This point acts as a document repository, but also acts as a synchronizer and provides mechanisms to avoid and 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.

In summary, the fundamental difference between the two scenarios (de/centralized) we consider is this: We assume that in the centralized scenario the root node needs to process the aggregated messages (for example, re-order them) before all the other nodes can use the messages at the application layer, while in the decentralized scenario, no such centralized processing is carried out.

In both scenarios, we assume that the tree used for collecting the updates is the same as the multicast tree. In principle, the update (aggregation) tree could have been different from the dissemination tree. However, this would put a further burden on construction and maintenance of such a separate tree.

We use the following notation in the rest of the text. The degree of a node $\deg(u)$ is the number of (immediate) children node u has in the tree T ($\deg(u) = |\{v_i : (u, v_i) \in E_t\}|$). 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 $L(u, v)$ over path $\pi(u, v)$ is the sum of latencies of edges in the path, $L(u, v) = \sum_{i=0,\ldots,n-1} l(v_i, v_{i+1})$. The total cost $C(u, v)$ is similarly defined as $C(u, v) = \sum_{i=0,\ldots,n-1} c(v_i, v_{i+1})$.

2.1. Optimization Criteria in Centralized Dissemination

We define the following optimization criteria over T:

2.1.1. Tree's weighted end-to-end latency L

This criterion expresses the latency between the moment that 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,

$$\begin{aligned} L(T) &= \sum_{v \in V} \rho(v)\big(L(v, r) + \max_{w \in V} L(r, w)\big) \\ &= \max_{w \in V} L(r, w) + \sum_{v \in V} \rho(v) L(v, r), \end{aligned} \tag{1}$$

where r is the root. In the tree in Figure 3, L equals:

$$\begin{aligned} L = 65 &+ 0.08 \cdot 0 + 0.06 \cdot 5 + 0.06 \cdot 7 \\ &+ 0.05 \cdot 45 + 0.6 \cdot 47 + 0.1 \cdot 53 + 0.05 \cdot 65. \end{aligned}$$

The first element (65) is the total latency between the root P1 and the furthest node S4. The second element computes the observed uplink latency of P1 as its workload (0.08) multiplied by the cost of the path to the root (0, as P1 is the root). The following elements add the observed latencies for nodes P2, P3, S1, S2, S3, S4.

2.1.2. 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. By aggregating these costs over all the nodes and weighing it by $\rho(v)$, we obtain:

$$C(T) = \sum_{v \in V} \rho(v) C(v, r) + \sum_{v \in V} c(parent(v), v). \tag{2}$$

In the tree in Figure 3, C equals:

$$\begin{aligned} C = &\big(0.05 \cdot 10 + 0.6 \cdot 10 + 0.1 \cdot 10 + 0.05 \cdot 10\big) \\ &+ 10. \end{aligned}$$

The first line reflects the cost of the uplink of nodes S1, S2, S3 and S4 as the product of their workload (0.05 for S1) multiplied by the total cost of the link between each of the nodes and P1, the root (non-zero only for S1-P1 link). The cost of uplink of nodes P1, P2 and P3 is zero. The second line adds the costs of all the edges of the tree. In this case, only the link P1-S1 has non-zero cost.

2.1.3. Tree's instability Λ

A disconnection of a node v affects its children $\{u : (v, u) \in E_T\}$ that would have to find replacement parent node(s). For the sake of the theoretical analysis, we make an usual assumption that nodes' failures are independent, and thus with high probability no two nodes disconnect at the same time[1]. Thus, the instability of the tree can be computed as the average number of nodes affected by disconnection in an unit of time:

$$\Lambda(T) = \sum_{v \in V} \lambda(v) |\{u : (v, u) \in E_T\}| \tag{3}$$

In the tree in Figure 3, Λ equals:

$$\Lambda = 0.2 \cdot 3 + 0.6 \cdot 2 + 0.1 \cdot 1.$$

Node's P1 impact on instability is its λ (0.2) multiplied by the total number of children (3). The other nodes with children are S1 (adding $0.6 \cdot 2$) and S3 (adding $0.1 \cdot 1$).

[1] This assumption is unrealistic in case of network partitioning, when nodes share the same physical network link that gets disconnected. However such situations can be considered rare, compared with node churn.

2.2. Optimization Criteria in Decentralized Dissemination

When messages do not have to be processed by the root before being used by the other peers, the dissemination pattern, and thus the usage of the links changes. Thus, the criteria described in the previous section have to be modified accordingly.

2.2.1. Tree's weighted end-to-end latency L

In decentralized dissemination, each peer can immediately use a received message apart from forwarding the same to all its neighbors (except the source). Thus, for each sender, the observed latency is determined by the length of the longest path in the tree. This leads to the following formula for weighted latency:

$$L(T) = \sum_{v \in V} \rho(v) \max_{w \in V} L(v, w). \tag{4}$$

2.2.2. Tree's aggregated cost C

In decentralized dissemination, a message is not forwarded back to its source. Additionally, each message is routed to all other peers in the network. Note, however, that in the general case, $c(v, parent(v)) \neq c(parent(v), v)$.

The routing direction of a message on the $(v, parent(v))$ edge (and thus, the cost) depends on the original sender of that message. If v, or one of its descendants (denoted later as $T(v)$) is the sender, the message is routed "upward", or from v to $parent(v)$. If any other node in the graph is the sender, the message is routed "downward", or from $parent(v)$ to v.

In the first case (a message originates from $T(v)$), the routing results in cost $c(v, parent(v))$. Such messages are produced with probability $\sum_{w \in T(v)} \rho(w)$. Thus, the resulting weighted cost is equal to $c(v, parent(v)) \sum_{w \in T(v)} \rho(w)$.

In the second case (a message originates outside $T(v)$), the routing results in cost $c(parent(v), v)$, which results in a weighted cost equal to $c(parent(v), v)(1 - \sum_{w \in T(v)} \rho(w))$.

By summing over all edges $(v, parent(v))$ in tree T, we obtain the following formula:

$$C(T) = \sum_{v \in V} \Big(\big(c(v, parent(v)) \sum_{w \in T(v)} \rho(w)\big) + \big(c(parent(v), v)(1 - \sum_{w \in T(v)} \rho(w))\big)\Big). \tag{5}$$

2.2.3. Tree's instability Λ

Instability does not depend on the communication pattern, but on the protocol used to reconstruct the tree when a node fails. Thus, Eq. 3 is still valid in the decentralized dissemination pattern.

2.3. 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:

$$(\min L(T), \min C(T), \min \Lambda(T)), \tag{6}$$

Note that the notation $(\min L(T), \min C(T), \min \Lambda(T))$ 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 [13]. 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.

2.4. 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 ($l(u, v) = l(v, u)$). Thus, we assume that a link latency is equal to the half of the round trip time (RTT), which is not precise in general. However, RTTs can be easily measured, unlike latencies. 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 $\deg_{\max}(v) = \infty$.

3. Problem Analysis and Proposed Algorithms

3.1. Individual Subproblems in Centralized Dissemination

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 instability Λ.

3.1.1. Instability

The $\min \Lambda(T)$ problem is equivalent 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 $\Lambda(T) = \sum_{v \in V} \lambda(v) |\{w : (v, w) \in E_T\}|$, 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 [14] 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 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 problem becomes NP-hard ([15, ND3]).

3.1.2. Latency

The main difficulty in analyzing $\min L(T)$ is caused by the fact that the same tree is used for upstream and downstream messages. The $\min L$ problem is a special case of the optimal communication spanning tree (OCT) problem [16]. 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. OCT defined by Eq. 1 has a particular structure of requirements with only non-zero requirements being $\rho(u, v) = \rho(u)$ for $v = r$ (uplink latencies between each node u and the root r); and $\rho(u, v) = 1$ for $u = r$, $v = k$ (downlink latency between root r and the furthest node k). However, in our problem node k and r 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 ($\max_w L(r, w)$) and non equal weights ρ. However, the following lemma shows that a SPT spanning tree is optimal.

Lemma 2. *If the latencies are symmetric ($l(v, u) = l(u, v)$), the SPT spanning tree with minimal L among all possible SPT 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'.

Since latencies are symmetric, we rewrite $L(T)$ (Eq 1) as follows:

$$\begin{aligned} L(T) &= \max_w L(r, w) + \sum_v \rho(v) L(v, r) \\ &= \max_w L(w, r) + \sum_v \rho(v) L(v, r) \end{aligned}$$

If the root of both trees is the same ($r^* = r'$), $L' < L^*$ implies that there is some node v, for which $L(v, r') < L(v, r^*)$, which means that L' has shorter path to v, 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'. Using the same argument, it is not possible that the distance $L(v, r')$ in T' is less than the same distance in $T^{SPT'}$. Furthermore, $L^{SPT'} \geq L^*$, as T^* has the minimal latency among all the SPT trees. □

Optimal latency tree can be thus easily found by repeating Dijkstra's shortest path algorithm for every node (with total complexity $O(|V|^3)$) [17].

When the maximum degree of the tree is limited, this model becomes NP-hard (even with symmetric

latencies), since SPT with bounded degree is NP-hard [18].

3.1.3. Cost

Equation (2) denoting the cost of a tree is composed of two components. $\min \sum_v \rho(v)C(v,r)$ is a weighted shortest path between v and r, optimized by SPT (but taking into account the costs $c(v,u)$, and not the latencies $l(v,u)$, as in the previous section). $\min \sum_v c(parent(v),v)$ is the total cost of all the edges, optimized by the usual Minimal Spanning Tree (MST). Thus, a tree optimal for cost C is a trade-off between MST and SPT.

The $\min \sum_v \rho(v)C(v,root) + \sum_v c(parent(v),v)$ problem is related to the NP-complete Minimum Diameter Spanning Subgraph [15, problem ND6], in which a 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 [19] (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 paths between the root and each vertex by at most α. [19] proposes a polynomial approximation 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 r). 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 [20], 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 $\min \sum_v \rho(v)C(v,root) + \sum_v c(parent(v),v)$ is still open. Currently, we use a heuristic that is a variant of LAST [19]. 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 T' has lower cost ($C(T') < C(T)$), 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 T'' has lower cost ($C(T'') < C(T)$), 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.

3.2. Individual Subproblems In Decentralized Dissemination

In this section, we analyze the complexity of minimization of individual performance measures in the decentralized dissemination scenario. Analysis of the instability (Section 3.1.1) still holds, as the instability problem is the same in the centralized and in the decentralized communication pattern.

3.2.1. Latency

If the sending rates $\rho(v)$ are equal, the problem of minimizing $\sum_v \max_w L(v,w)$ becomes a special case of the k-Source Sum of Source Eccentricities Spanning Tree problem (k-SSET, [21]) in which all the vertices are sources. k-SSET is polynomial. The optimal tree is a SPT rooted at the midpoint of the diametral path (the shortest path connecting the pair of vertices furthest apart, i.e., $(v,w) : L(v,w) = max_{v_i,v_j} L(v_i,v_j)$) [21].

The following lemma shows that the weighted problem $\sum_v \max_w L(v,w)$ can be expressed by the unweighted k-SSET on a modified graph.

Lemma 3. *The problem of minimizing* $\sum_v \rho(v) \max_w L(v,w)$ *can be reduced to the problem of minimizing* $\sum_{v \in V'} \max_w L(v,w)$ *in a modified graph* (V', E').

Proof. Given a graph (V,E) and nodes with weighted sending rates, we will construct a modified graph (V', E'), in which *source* nodes have the same sending rates, the rest of the nodes is not sending, and the resulting latency is equal to the latency in the original graph (multiplied by a constant).

We defined the sending rates $\rho(v)$ as the fraction of transmission time in which node v sends data.

Thus, all $\rho(v)$ are rational numbers. We denote by ρ_0 their lowest common denominator (an integer).

We replace each node v in the original graph by $1 + \rho_0 \cdot \rho(v)$ nodes: the front node v_0 that has the same set of outgoing edges to the rest of the graph as v; and $\rho_0\rho(v)$ nodes $v_1, \ldots, v_{\rho_0\rho(v)}$, connected to v_0 with edges (v_0, v_i) with zero latency.

The set of senders S in the modified graph includes only nodes v_i (i.e., all the nodes excluding the front nodes).

In the modified graph, there are exactly ρ_0 times more messages leaving each front node v_0, compared with node v from the original graph. Thus, the relative rates of outgoing messages at front nodes are the same as in the original graph.

Thus, the latency in the modified graph $\sum_{v_i \in S} \max_w L(v, w)$ is equal to the latency in the original graph multiplied by the lowest common denominator $\rho_0 \sum_{v \in V} \rho(v) \max_w L(v, w)$.

□

Consequently, an algorithm optimizing $\sum_v \rho(v) \max_w L(v, w)$ can construct such a modified graph and then find a SPT rooted at the diametral path. However, a more straightforward approach is to construct all possible SPTs in the original graph and then choose the one resulting in the minimal latency. As the optimal tree in the modified graph is a SPT, in which all the added nodes are leafs, the best SPT in the original graph will have the same structure. Thus, again Dijkstra's shortest path algorithm can be repeated for all the nodes, with total complexity $O(n^3)$.

3.2.2. Cost

Symmetric costs ($c(u, v) = c(v, u)$) allow reformulating Eq 5 as:

$$C(T) = \sum_{v \in V} \Big(\big(c(v, parent(v)) \sum_{w \in T(v)} \rho(w)\big) + \big(c(v, parent(v))(1 - \sum_{w \in T(v)} \rho(w))\big)\Big)$$

Thus,

$$C(T) = \sum_{v \in V} c(v, parent(v)), \qquad (7)$$

which follows the intuition that, on each message, each edge is used exactly once. As cost in both directions is the same, we can just add the costs of all the edges in the tree.

The problem of cost minimization in decentralized dissemination is thus identical to the usual Minimal Spanning Tree (MST) problem with edges' weights equal to costs. MST can be found in $O(n^2)$ by Kruskal's (or Prim's) algorithm [17].

3.3. 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. We propose a heuristics algorithm based on hill climbing. Note that this part of the analysis can be applied both to decentralized and to centralized communication scenario.

3.3.1. 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 [13]. 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 the 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 [13]. 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:

$$y'(T) = \frac{y(T) - y^*}{\widetilde{y}^N - y^*}. \tag{8}$$

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:

$$dist(T) = \sqrt{\Big(\big(L'(T)\big)^2 + \big(C'(T)\big)^2 + \big(\Lambda'(T)\big)^2\Big)}. \tag{9}$$

3.3.2. Producing the approximated ideal solution

An obvious possibility to find a tree minimizing $dist(T)$ is an exhaustive search over all spanning trees. However, the number of spanning trees is exponential and equal to n^{n-2} [16] in the worst case of a fully-connected graph. Thus, the exhaustive search is clearly not efficient, especially for larger graphs. In our initial experiments, fully-connected graphs with $n = 10$ nodes took more than an hour to optimize.

We have implemented a heuristics based on stochastic hill climbing (HC). The algorithm minimizes the distance $dist(T)$ of the current tree T to the ideal solution by random modifications of T. If a modified tree results in lower $dist(T)$, it is accepted, otherwise it is rejected. Two types of modifications are possible: (1) swapping the root of the tree (with probability p, where p is computed from the formula given below); (2) swapping an edge with an unused edge from the graph (with probability $1 - p$). The probabilities are set in a way that makes small adjustments (such as swapping an edge connecting a small number of nodes) more probable than the large ones (swapping an edge close to the root, or swapping the root).

In root swapping, all the nodes are equally probable to be the new root. In edge swapping, the probability of removing an edge is inversely proportional to the number of descendants $desc(v)$ of the downstream node v of the edge (u, v), and equal to $p/(1 + desc(v))$ (except for *leaf* nodes that have only one connection to the rest of the graph, and thus cannot be disconnected). After removing an edge, the algorithm reconnects the tree by choosing an edge unused in T at random and checking whether the tree using this edge is connected.

Table 1: Comparison between the multiobjective hill climbing and the exhaustive search on small graphs.

	dist				instances [%]	
	multiobjective		exhaustive			
n	$\bar{x}$	σ	$\bar{x}$	σ	worse	pareto
5	0.63	0.25	0.61	0.25	12	0
6	0.58	0.28	0.56	0.26	13	0
7	0.57	0.23	0.55	0.22	16	1
8	0.57	0.24	0.55	0.23	14	1
9	0.55	0.2	0.53	0.19	20	1

4. Experimental Evaluation

4.1. Settings

We evaluated the algorithms on randomly-constructed connectivity graphs. We considered only the centralized dissemination pattern, as it is more used in existing collaborative applications and, at the same time, unexplored by many-to-many multicast approaches. The graph models the logical, and not physical, connectivity between nodes, which makes most 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 one of the peers. Additionally, each peer is firewalled with *probability of* 0.8 (in Coolstreaming [22], 20% of nodes are directly reachable or behind UPnP). 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 $U(0, 1)$ (note that our algorithms do not take into account the absolute values of functions, so the results for $U(0, 0.01)$ would be similar). Finally, sending rates $\rho(v)$ are generated with *normal-*

Table 2: Results of experimental evaluation based of the hill climbing multiobjective algorithm. The table presents scaled performance measures (Eq. 8) averaged over 100 random graphs for each n. T_L^* is the tree returned by optimal latency algorithm (Section 3.1.2), T_C^* –by adjusted LAST (Section 3.1.3), T_Λ^* – by MST on stability graph (Section 3.1.1). *multiobjective* is the solution chosen by the multiobjective algorithm (Section 3.3).

	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	0.46	0.39	0.73	0.37	0.65	0.4	0.79	0.29	0.72	0.38	0.86	0.32	0.22	0.26	0.27	0.28	0.34	0.27
6	0.5	0.39	0.73	0.37	0.75	0.37	0.78	0.29	0.62	0.39	0.84	0.34	0.25	0.26	0.26	0.28	0.29	0.24
7	0.44	0.37	0.74	0.3	0.72	0.37	0.79	0.3	0.79	0.31	0.83	0.35	0.24	0.27	0.23	0.24	0.31	0.22
8	0.45	0.34	0.75	0.34	0.72	0.36	0.84	0.25	0.74	0.33	0.9	0.28	0.22	0.24	0.26	0.24	0.31	0.21
9	0.5	0.35	0.8	0.3	0.72	0.34	0.82	0.26	0.83	0.25	0.9	0.26	0.23	0.22	0.21	0.16	0.34	0.21
10	0.46	0.32	0.79	0.25	0.73	0.33	0.86	0.23	0.8	0.29	0.96	0.18	0.19	0.16	0.25	0.21	0.31	0.17
12	0.41	0.29	0.8	0.29	0.73	0.32	0.8	0.26	0.83	0.26	0.98	0.08	0.18	0.14	0.2	0.14	0.32	0.18
15	0.38	0.29	0.85	0.21	0.69	0.32	0.8	0.28	0.88	0.2	0.93	0.23	0.23	0.19	0.17	0.15	0.3	0.17
20	0.37	0.25	0.86	0.21	0.73	0.3	0.8	0.28	0.89	0.18	0.96	0.17	0.16	0.13	0.17	0.11	0.26	0.12
25	0.32	0.21	0.85	0.21	0.65	0.33	0.85	0.22	0.94	0.15	0.98	0.1	0.14	0.1	0.13	0.08	0.24	0.13
30	0.3	0.22	0.84	0.23	0.67	0.31	0.84	0.22	0.93	0.14	0.99	0.03	0.12	0.07	0.14	0.11	0.21	0.12
40	0.32	0.22	0.84	0.21	0.68	0.29	0.84	0.24	0.95	0.12	0.98	0.11	0.12	0.1	0.12	0.08	0.2	0.12
50	0.27	0.2	0.87	0.18	0.59	0.29	0.84	0.2	0.97	0.09	0.99	0.07	0.1	0.06	0.09	0.07	0.18	0.1

ized zeta distribution, as we assume that users' contributions to a shared document vary substantially. We control the number of nodes n in the network, and, for each n, generate 100 random graphs.

We experiment with up to $n = 50$ nodes, which corresponds to collaborative sessions having more than three times as much users as the upper limit of CoWord. All the studied algorithms are feasible for this number of nodes: for instance, hill climbing takes about 3 minutes to produce optimized trees for 100 different random graphs of size 30 (or, about 1.8 seconds for each graph).

The experiments start with validation of the heuristics used for the multiobjective problem against exhaustive search (the hill climbing algorithm for the general problem in Section 4.2; and the adjusted LAST for cost minimization in Section 4.3). Then, we explore the nature of the solutions returned by the multiobjective problem (Section 4.4). We also analyze the sensitivity of the solutions by simulating inaccurate stability measurements (Section 4.5) and churn (Section 4.6). Finally, we compare the multiobjective solutions with alternative approaches: star topologies (Section 4.7); and a penalty- (or weight-) based algorithm (Section 4.8);

4.2. Validation of Hill Climbing

In the initial set of experiments, we validated hill climbing against exhaustive search on small graphs. Each execution of hill climbing tested 10,000 modifications of a tree. In our experiments, we measure the average distance from the ideal solution achieved by both approaches; the number of instances in which exhaustive search outperforms hill climbing in terms of score; and the number of instances in which the solution returned by the exhaustive search Pareto-dominates the solution returned by hill climbing. Table 1 presents the obtained results. Additionally, Table 11 and Table 12 show results of similar experiments in graphs with different configurations.

Table 3: 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	2	3
6	0	0
7	2	0.5
8	0	0
9	1	0.2

The results show that hill climbing in up to 20% of instances fails to produce the global optimum. However, the difference in average *dist* is less than 4%. Thus, it is relatively easy to find a solution close to the optimum; but hard to find the exact optimum. We think that such performance is satisfactory for an algorithm that, in a real distributed, networking environment, will work on imprecise data.

In another series of experiments, we compared hill climbing to simulated annealing (SA)-based heuristics. In smaller graphs ($n \leq 9$) SA outperforms hill climbing, as it finds the optimal tree in 98.8% of considered instances. However, in larger graphs (Table 13) hill climbing produces trees with smaller average *dist* and, often, Pareto-dominating SA, except when the initial temperature in SA is set to a very low value, which degrades SA into hill climbing. Hill climbing performs well in large graphs, as the local neighborhood of a solution is large—thus it is hard to become stuck in a local minimum.

4.3. Validation of LAST

Our multi-objective algorithm uses adjusted LAST [19] (Section 3.1.3) to produce a tree T_C^* whose cost $C(T_C^*)$ is used for the ideal solution (by which a candidate tree's solution is scaled according to Eq. 8). Adjusted LAST is a heuristics, thus there are no performance guarantees on its results.

In order to validate the use of adjusted LAST in the multiobjective algorithm, 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 3 summarizes the results obtained for small graphs . 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.

4.4. Assessment of multiobjective solutions

The results of experiments are summarized in Table 2. In order to obtain meaningful comparisons between random graphs with different n, we present scaled values of each function (Eq. 8). 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 3.1.2), T_C^* – by adjusted LAST (Section 3.1.3), T_Λ^* – by MST on stability graph (Section 3.1.1). *multiobjective* is the solution chosen by the multiobjective algorithm (Section 3.3). A number of phenomena can be observed.

Firstly, the trees efficient for one objective tend to be inefficient for other objectives. Algorithms optimizing one objective have, on the average, a score of 0.75 for the other objectives. The difference is especially visible when the absolute, and not rescaled, values of Λ are analyzed (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 5.59 times worse than the optimal value, compared with 2.08 performance drop for L and 2.14 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.

Table 4: Relative degradation of the instability of trees returned by the multiobjective algorithm running on graph with inaccurate nodes' instabilities $\lambda(v)$, in function of the level of the noise.

noise level	av. degr		inst. better	inst. same
(σ)	$\bar{x}$ [%]	σ [%]	%	%
0.05	4	13	16	51
0.1	8	18	14	37
0.15	12	22	13	28
0.2	17	27	13	23

Thirdly, the instability seems to be the hardest objective to optimize. The algorithms for C and L tend to be inefficient on Λ (0.81). Similarly, the trees optimal for Λ result in inefficient C and L (0.89). Additionally, the multiobjective algorithm has the worst average performance for Λ (0.30, compared with 0.21 for L' and 0.21 for C').

Finally, as the number of nodes in the graph is increased, the more specialized are the algorithms optimizing just one objective (i.e., the further are trees optimal for one objective from the optimum for other objectives). The multicriteria algorithm does not exhibit such behavior.

4.5. Sensitivity to Inaccurate Stability Measurements

One of the graph parameters used by algorithm is the instability $\lambda(v)$ of a node, which can be nontrivial to measure in a real system. In the following experiment, we quantify the impact of the inaccuracy of individual nodes' $\lambda(v)$ on the instability of the trees $\Lambda(T)$ returned by the multiobjective algorithm.

In order to simulate inaccurate $\lambda(v)$, we added to each $\lambda(v)$ a random variable distributed with the Normal distribution $N(0,\sigma)$ (σ, a parameter of the experiment, was set to $\sigma \in \{0.05, 0.10, 0.15, 0.20\}$). Thus, $\lambda(v') = \lambda(v) + N(0,\sigma)$. We then ran the multiobjective algorithm on the modified graph G'. Finally, we took the tree T' returned by the multiobjective algorithm; and calculated the instability $\Lambda(T')$ using the original (unmodified) nodes' instabilities $\lambda(v)$. We compared $\Lambda(T')$ to $\Lambda(T)$, the instability of the tree returned by the multiobjective algorithm running on the unmodified graph. Table 4 summarizes the obtained results (for graphs with $n = 30$ nodes). The table shows the average relative degradation (computed over instances in which the degradation was non-negative); and the percentage of instances in which the "noisy"

Table 5: Relative degradation of the score (without stability) of the trees returned by the multiobjective algorithm in two versions: optimizing stability, cost and latency; and optimizing only cost and latency. Degradation in function of the number of nodes removed from the tree.

nodes	with stability		without stability	
removed	$\bar{x}$	σ	$\bar{x}$	σ
1	0.26	1.15	0.86	4.93
2	0.18	0.97	1.77	5.70
3	0.35	1.26	2.92	7.30
4	0.63	1.84	3.12	7.19
5	0.70	1.98	4.02	7.30

tree T' was better than; and the same as the unmodified tree T.

The results show that the algorithm is, to some extent, prone to inaccurate measurements of nodes' instabilities, as with increased noise level the average degradation increases; similarly, the percentage of instances in which the returned tree is the same decreases. However, the average degradation does not increase strongly with the noise level; it is, approximately, of the same order as the standard deviation σ of the noise. There are also around 14% of instances in which the noised tree T' had lower instability Λ than the tree computed with original values T. This effect is due to the multiobjective nature of the optimization algorithm, and its non-determinism. Note that when taking these instances into account, the relative degradation drops from 2% for noise level $\sigma = 0.05$ to 14% for $\sigma = 0.20$.

4.6. Degradation under Churn

Tree topologies offer lower latency, cost, and instability in comparison with star topologies (Section 4.7), however they are harder to create and maintain in a dynamic system in which peers continuously join and depart. Although collaboration systems, due to their very nature, are more stable than p2p systems in general, these effects can be still significant. For instance, due to communication latencies, it can be impossible to create a completely new structure of a tree on each departure. Thus, in a dynamic system, the communication library should regenerate the tree only if the current performance is significantly worse than the performance of the optimal tree under current network conditions (as determined by the multiobjective algorithm).

In this series of experiments, we simulate the gradual degradation of the tree by removing a random node and then by fixing the tree locally: the removed node's children connect directly to the removed node's parent. We then measure the relative degradation of the cost and the latency of the resulting tree (we exclude the instability as a parameter related with the phenomenon). Afterward, we remove the next node, and so on, until 5 nodes are removed. We assume that the probability that a particular node v departs is proportional to its instability $\lambda(v)$. Table 5 summarizes the obtained results (for graphs with $n = 30$ nodes).

Table 6: Comparison of scores $dist(T)$ of the trees returned by the multiobjective algorithm with the best possible star topology.

	multiobjective		star	
n	$\bar{x}$	σ	$\bar{x}$	σ
5	0.63	0.21	1.22	0.62
6	0.54	0.21	1.32	0.58
7	0.50	0.18	1.39	0.61
8	0.47	0.15	1.36	0.45
9	0.42	0.15	1.35	0.39
10	0.41	0.14	1.38	0.41
12	0.37	0.12	1.43	0.40
15	0.31	0.12	1.39	0.34
20	0.28	0.12	1.47	0.37
25	0.24	0.09	1.45	0.26
30	0.22	0.08	1.51	0.29
40	0.21	0.07	1.49	0.27
50	0.19	0.07	1.50	0.27

Our multiobjective approach takes into account nodes' departures only indirectly—using the node instability $\lambda(v)$. Yet, the obtained results clearly show that degradation of cost and latency is considerably smaller than in the alternative scenario, when the instability in not taken into account in the initial optimization. Nevertheless, the degradation is still significant and it will thus be valuable to explore robust optimization techniques as future work.

4.7. Comparison with Star Topologies

The multiobjective approach commonly returns a tree, which, in a real system, is harder to maintain than a simple star topology (with all the nodes directly connected to a single coordinating and forwarding node). We wanted to quantify the gain of the tree-based infrastructure compared to the star topology. Of course, when leaf nodes are present, a star topology cannot be constructed, as such devices can directly connect only to specific (subset of) nodes: if such a node is not the root of the star, the leaf is bound to be disconnected from the network. Consequently, in this series of exper-

iments, the considered graphs did not contain any leaf nodes. Table 6 summarizes the obtained results, in which the tree returned by the multiobjective algorithm is compared to the best (according to $dist(T)$) star. It can be clearly seen that, overall, star topologies are much worse than the trees in terms of the distance to the ideal solution. Detailed results (Appendix, Table 9) show that star topologies are comparable (yet slightly worse) in terms of the instability $\Lambda(T)$, they are much worse for both cost and latency. High stability can be easily achieved in a star centered on the most stable node. However, a star topology uses costly inter-site links redundantly. In contrast, a tree commonly uses the inter-site link between any pair of sites just once. Moreover, a tree can choose the inter-site link that has the best latency/cost trade-off—which might not necessarily be the one that is available for the root of the tree.

4.8. Comparison with Weighted Approach

In our approach for the multiobjective optimization, we treated each objective separately until computing the distance $dist(T)$ to the ideal solution. An alternative approach is to aggregate all the parameters of nodes and edges into a penalty associated with using a particular edge and then to find a tree minimizing the overall penalty.

The penalty of an edge (u, v) is defined by:

$$pen(u, v) = w_C c(u, v) + w_L l(u, v) + w_\Lambda \lambda(u), \quad (10)$$

where w_C, w_L and w_Λ are the weights. We can assume that each weight w_y is composed of the scaling factor $w'_y = 1/(\max(y))$ (that normalizes the ranges of values of the parameters to $[0, 1]$), and a relative importance w''_y of the parameter y (with a constraint $w''_C + w''_L + w''_\Lambda = 1$).

The overall penalty of a tree T must take into account both the sum of the penalties of the downstream links (reflecting the cost of dissemination) and the sum of penalties of reaching the root r of the tree (reflecting the cost of updating the information), thus:

$$pen(T) = \sum_{v \in V} \rho(v) pen(v, r) + \sum_{v \in V} pen(v, parent(v)), \quad (11)$$

where $pen(v, r)$ is the aggregate penalty of the path between v and the root r. This equation has the same form as Eq. 2, and thus the analysis of the complexity of optimizing the cost (Section 3.1.3) is still valid. Note, however, that the penalty cannot capture precisely the nature of the downstream latency: the formulation used in Eq. 1 considers the maximum latency between the root and the furthest node; while here the sum of latencies is considered.

Table 7: Comparison of scores $dist(T)$ of the trees returned by the multiobjective algorithm with the ones returned by the penalty-based algorithm.

	multiobjective		penalty	
n	$\bar{x}$	σ	$\bar{x}$	σ
5	0.63	0.25	0.78	0.34
6	0.58	0.28	0.79	0.35
7	0.57	0.23	0.71	0.31
8	0.57	0.24	0.68	0.29
9	0.55	0.20	0.69	0.33
10	0.50	0.21	0.69	0.31
12	0.46	0.17	0.59	0.26
15	0.47	0.19	0.58	0.25
20	0.38	0.14	0.52	0.24
25	0.32	0.14	0.47	0.28
30	0.31	0.13	0.42	0.21
40	0.28	0.14	0.43	0.22
50	0.23	0.11	0.44	0.25

In order to compare the penalty-based approach with our multiobjective approach, we set equal relative weights $w''_C = w''_L = w_\Lambda = 1/3$ and used the same random hill climbing algorithm with the same number of iterations. The results are presented in Table 7 (and the detailed results in Appendix, Table 10).

The multiobjective approach returns trees that are closer to the ideal solution, compared to the penalty-based approach. Moreover, the performance is more stable, as the standard deviations are lower. In terms of multiobjective performance, the solution of the penalty-based approach Pareto-dominates the multiobjective solution in less than 1% of the instances. Two factors contribute to the observed performance. Firstly, as discussed above, the penalty-based approach is unable to correctly capture the latency of the tree. Detailed analysis (Table 10) reveals that indeed the penalty-based trees have the lowest performance for the latency. Secondly, the penalty-based approach uses weights based on the maximal values in a graph, and not—values from the approximate nadirs. Thus, for instance, a high latency link that is not used in any Pareto-optimal tree still reduces the scaling weight w'_L, and thus reduces the overall impact of the latency on the penalty. Note however, that in our experiments, the second problem is less visible, as all the values are generated from uniform, rather than

Table 8: Exploration of relative weights to obtain a penalty based overlay with better score (for a specific graph).

relative weights			penalty based approach			
w''_C	w''_L	w_Λ	$L(T)$	$C(T)$	$\Lambda(T)$	score
0.33	0.33	0.33	1.16	0.04	0.19	1.18
0.4	0.3	0.3	1.48	0.09	0.18	1.49
0.4	0.4	0.2	0.67	0.04	0.26	0.72
0.4	0.2	0.4	1.45	0.19	0.16	1.47
0.5	0.25	0.25	1.48	0.09	0.18	1.49
0.5	0.4	0.1	0.92	0.07	0.33	0.98
0.5	0.1	0.4	1.44	0.22	0.15	1.47
0.6	0.2	0.2	0.92	0.07	0.33	0.98
0.6	0.3	0.1	0.63	0.1	0.37	0.74
0.6	0.1	0.3	0.53	0.28	0.16	0.62
0.7	0.15	0.15	0.63	0.1	0.37	0.74
0.7	0.07	0.23	0.67	0.28	0.28	0.78
0.7	0.23	0.07	0.64	0.1	0.38	0.75
0.8	0.1	0.1	0.52	0.17	0.38	0.66
0.8	0.05	0.15	0.52	0.17	0.38	0.67
0.8	0.15	0.05	0.4	0.16	0.49	0.65
0.9	0.05	0.05	0.53	0.24	0.36	0.68
0.9	0.01	0.09	0.55	0.3	0.38	0.73
0.9	0.09	0.01	0.41	0.21	0.52	0.69

heavy-tailed, distributions. These problems could be mitigated by fine-tuning the relative weights for a particular graph and re-running the algorithm multiple times; however we believe that the less fine-tuning an algorithm requires, the better, especially as the weights would probably need to be adjusted after each modification of the underlying communication graph.

If the users perceive that one of the criterion is more important than the others, or alternatively, if the performance of one criterion needs to be significantly improved (e.g., say, cost and stability are within agreeable limits, but the latency is not), then in a specific scenario (a specific instance of the underlying graph), they may choose to use a different weight, penalizing more the criterion with stricter performance needs, to explore the parameter space, such that acceptable performance may be arrived at. Given the continuous range of penalty parameter choices, an exhaustive exploration of the parameter space is not feasible, and thus it is even hard to ascertain if a specific set of constraints can be met by exploring the penalty parameters. In contrast, in fact even exhaustive search of all multicast overlay configuration, despite the combinatorial problem, is still at least feasible to determine what are the best possible configurations, and our multiobjective optimization process can be seen as an approximation of the same. Nevertheless, out of curiosity, we investigate, what happens if we indeed try to explore a bit the parameter space. We do so by choosing random topologies, which had resulted in multicast configurations based on multiobjective optimization and using equal penalty that did not pareto-dominate each other. That means, the penalty based approach out-performed the multiobjective based approach along some criterion, and thus the problem was to find parameter choices which would hopefully help out-perform the multi-objective optimized overlay in the remaining aspect(s). In Table 8 we report the results from one specific such graph comprising of fifty nodes which had a diameter of four, an average network clustering coefficient of 0.2255 and an algebraic connectivity of 0.6442. For this graph, the multiobjective optimized multicast topology had a score of 0.42, with $C(T) = 0.19$, $L(T) = 0.2$ and $\Lambda(T) = 0.32$. We observe, albeit from a coarse-grained exploration of the parameters, that the score of the penalty based tree obtained with the best choice of relative weights (shaded row) is 0.62, which while much better than what was obtained when equal relative weightage was given, is still worse than the one obtained with multiobjective optimization. Furthermore, the very low latency achieved by multiobjective optimization (which is the reason for it's low score) was never realized with any choice of the parameters we tried.

4.9. Summary of Experimental Evaluation

The experiments consisted of three parts. Firstly, we demonstrated (Sections 4.2 and Section 4.3), that the heuristics (hill climbing and adjusted LAST) are reasonable, as the performance of the produced trees is close to the performance of the optimal trees found by exhaustive search.

Secondly, we analyzed the solutions returned by the multi-objective algorithm, both in terms of the values of the objectives problem (Section 4.4) and also sensitivity to inaccurate stability measurements (Section 4.5) and churn (Section 4.6).

Finally, we validated the efficacy of the multiobjective approach by comparing it to alternatives (star topologies in Section 4.7); and a penalty- (or weight-) based algorithm in Section 4.8). The results in this part clearly indicated that the trees produced by the multiobjective approach outperform the alternatives.

5. Multicast Communication Library

Using optimization algorithms described in Section 3.3.2, we implemented a multi-threaded communication library for collaborative applications, which we call MoMo (which stands for Multiobjective Optimized Multicast Overlay). The library organizes participating peers in a tree. This section outlines design principles and some of the protocols used in the library.

The library creates two logical networks having different objectives and different topologies. The *multicast* network is used to send normal multicast messages. The *status* network is used as a robust communication channel to manage the multicast network.

The multicast network is used for the standard multicast communication in the collaborative application. The topology of the network is derived by the optimization algorithm described in Section 3.3.2.

The status network is used to send notifications of node arrivals and departures, to measure network parameters (such as latencies) and to send messages based on which nodes reconfigure the topology of the multicast tree. In the status network, peers are interconnected, creating a complete graph (whenever it is possible). The status network works independently of the multicast network in order to create a reliable communication channel even if the multicast tree is being reorganized or down due to nodes' failures.

In both networks, every message contains the actual data, a unique identifier, the sender's identifier and a vector clock.

As the logical structure of the multicast network depends on parameters of peers (workload $\rho(v)$, instability $\lambda(v)$) and links (cost $c(u, v)$, latency $l(u, v)$), the multicast network should be reconfigured when these parameters change or new peers join or leave the system. In the current implementation, the optimization algorithm is run periodically, and, in addition, on each join and leave.

5.1. Routing

In the status network, notifications are broadcasted to each connected peer (similar to gossiping). Messages with the same identifier are discarded, to reduce the overhead.

In the multicast network, messages are routed up to the root of the tree. The root, upon receiving a message forwards it back downstream to each of its children. Each child node can use such a message at the application layer, besides further forwarding it to all of its children.

The root acts as a free synchronizer – the order of messages received by each peer is the same once it has been ordered by the root. Reception of one's own message indicates that the message has been correctly received by the root, and has been broadcast back in an order which will be consistent globally.

5.2. Message Handling

In a dynamic network, even when a central root exists, the order of messages can be easily changed. A state vector, added to each message, partially handles this issue. Using the logical clock value for a peer, messages originating from the same peer can be reordered to the original order in which they have been sent. This is similar to the receiving buffer in TCP, although we have separate buffers for each peer, instead of one, global buffer.

Messages are delivered to the application only in the correct order. If an arriving message has the expected (next) sequence number, it is delivered immediately to the application. Out-of-sequence messages are buffered by the communication library. If the missing messages do not arrive for some time, the receiving peer explicitly requests their retransmission. Finally, if the messages are still missing, the library informs the application about serious and persistent problems on the communication layer. In this case, the states of the shared document should be synchronized on the application, rather than in the communication layer.

5.3. Measurements

Four variables need to be measured to be used by the optimization algorithm. They are gathered in a distributed manner and are stored in every connected peer. Additionally, when a new peer joins the network, its parent sends the current values of these measures.

After measurement completes, the newly measured value ($\rho'(u)$) is used to update the stored value ($\rho(u)$) using an aging process, e.g., $\rho(u) = \kappa \cdot \rho(u) + (1 - \kappa) \cdot \rho'(u)$ (we use $\kappa = 0.8$ as the default).

Latencies $l(u, v)$ are measured by round-trip time (RTT) between each pair of peers. This measurement is performed in the status network. Peers that are persistently unreachable are considered as firewalled or behind a NAT.

Cost $c(u, v)$ is given by the application (which, in turn, obtains it as user input).

Workload rate $\rho(u)$ is measured as the fraction of data sent by u. Thus, each peer v, upon receiving a multicast message from u, updates its counter $count(v, u)$ with the amount of data received. Periodically, peer v updates $\rho(u)$ by $\rho'(u) = count(v, u)/\sum_w count(v, w)$.

The stability $\lambda(u)$ of the peer u is measured indirectly, by the inverse of its average session length. After u disconnects, other peers update u's average session length, and then compute the stability.

5.4. Reconfiguration of Multicast Network

Periodically, an elected peer runs the optimization algorithm in order to ensure that the multicast network structure is still optimal. If the derived structure differs from the actual, the elected peer coordinates the new network formation by informing all the peers about their respective new parent. These messages are sent over the status channel.

In order to elect such a peer, MoMo uses bully election algorithm. There is another timeout that starts after the election starts. This timeout is used as a preventive measure against the situation where the elected peer leaves in course of the reconfiguration process. In such a pathological scenario, the election process is restarted.

While the multicast network is reconfigured, MoMo enters a transient state, in which all the messages are delayed locally at peers in order to prevent message loss.

5.5. Handling arrivals and departures

MoMo handles node arrivals, node failures and node departures. Node failures are considered as node departures.

5.5.1. Node Arrival

MoMo assumes that a peer joining the network must know another, already connected, peer. Initial contact is established through the status port of the connected peer. The connected peer becomes the parent of the new peer and creates a bi-directional link in the multicast network and in the status network. Once all the links are successfully established, other peers are informed about the join through the status network.

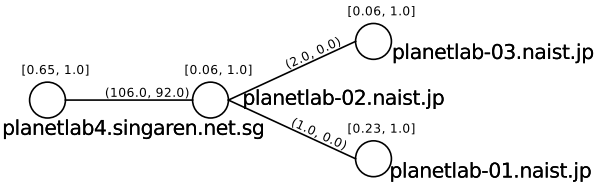

Figure 4: Network topology derived by the multiobjective algorithm for 4 planet-lab nodes. Values next to a node are its workload ρ (higher values denoting more frequent updates) and instability λ (higher values denoting more frequent disconnections). Values next to an edge are its latency l and its cost c.

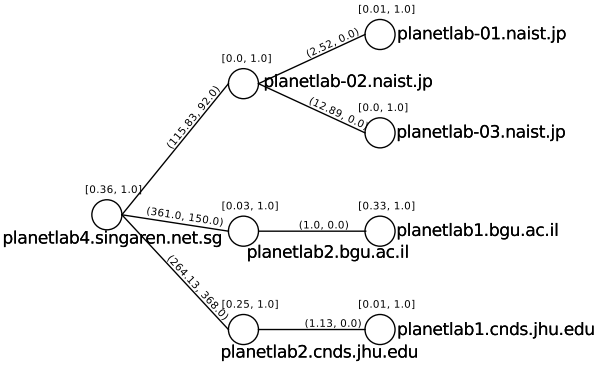

Figure 5: Network topology derived by the multiobjective algorithm for a configuration comprising 8 planet-lab nodes, which is obtained by adding four new nodes to the previous experiment.

5.5.2. Node Departure

When a node leaves the network, its parent and its children inform other nodes using the status network. In the current version of the library, the network is reconfigured immediately after each leave. However, in future we plan to defer the reconfiguration until the network degrades beyond acceptable threshold: and temporarily connect the children to the parent of the departing node whenever possible.

5.5.3. Node Failure

Each communication link uses TCP sockets. Abrupt departure of a node or a failure of a link is easily detected by the node's direct parent and the node's children. Node failure is considered as node departure and is handled in the same manner.

6. Experimental Validation of the Communication Library

We tested a prototype of the communication library on the PlanetLab infrastructure using an example application that multicasts short text messages. PlanetLab is not an ideal testbed for such

an experiment since it is a dedicated infrastructure, so we contrive some of the settings with artificial inputs (e.g., cost based on geographic distance), mixing it with observed parameters. Likewise, despite PlanetLab being a dedicated infrastructure based testbed, there are high fluctuations in the observed parameters, influencing the results and their reproducibility.

In our experiments, some of the observed parameters come inherently from the environment (such as latency), while others, like the workload, are generated using experimental parameters. The sending rate of messages varied among different nodes between 2 and around 120 messages per minute; the distribution roughly followed Pareto distribution with $s = 0.5$. Each node measured the sending rate ρ of all other nodes by the number of messages received. During the communication, every two minutes, the current root of the tree performed the optimization algorithm and reconfigured the tree (if the optimum differed from the current tree). Despite these considerations, we obtain reasonably meaningful results from the conducted experiments.

Figure 4 shows a network topology derived by the communication library for 4 planet-lab nodes. The node located in Singapore (`planetlab4.singaren.net.sg`) is the root of the multicast tree, as it sends most messages (its real sending rate is about 120 messages per minute). All the nodes located in Japan (`planetlab-0x.naist.jp`) are connected using only one Singapore-Japan network link, to minimize the cost of the tree.

Figure 5 shows a tree derived for topology with two additional nodes in Israel (`planetlabx.bgu.ac.il`) and two in the US (`planetlabx.cnds.jhu.edu`). The tree minimizes the usage of costly intra-country links, as each site has only one node that communicates with the root.

We also carry out separate larger scale experiments using up to 28 nodes. Figure 6 shows a tree derived for topology with twenty eight nodes spread across several continents. We see that again nodes in Asia form (several) sub-clusters, while nodes in Europe and North America form their own sub-clusters. We also notice some interleaving of Hong Kong, Singapore and again Hong Kong nodes in the overlay. While we are not completely certain about the reasons for such configurations, one speculation is that some of the specific nodes (e.g., `plab2.cs.ust.hk`,`planetlab2.ie.cuhk.edu.hk`) are too slow to respond or the link saturates, so that these nodes are then pushed to the periphery of the overlay.

Repeating the experiments with the same nodes over different time periods when the observed variables had naturally changed in the system (which is an artefact of PlanetLab's non-determinism), we obtained different configurations (for example, Figure 7). Nevertheless, the overall qualitative behavior of forming country and continent specific sub-clusters remained the same.

7. Related Work and Discussion

There have previously been proposals to extend the Internet Protocol (IP) [23] to support multicast natively. However, since IP multicast has never been widely deployed and adopted, the practical way to send data to multiple receivers is to extend the application layer. We summarize some of the application layer multicast approaches below (see [24] for a more exhaustive comparison).

The most straightforward way to construct an application-layer multicast tree is to optimize the latency in a greedy manner. In Yoid [25], 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 [26], 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 [27] 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 [28] 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.

Massively multiplayer online games (MMOGs) [9, 10, 11], use the notion of groups of players (similar to our members of a session), who need to communicate among each other. Our approach can be used to optimize these intra-group communications.

Some other application-layer multicast methods are specific to a certain peer-to-peer overlay. For instance, Scribe [29] builds multicast trees

based on reverse path forwarding in Pastry overlay. Bayeux [30] uses forward path forwarding, and Borg [31] combines the two approaches. All these systems do not consider multiobjective optimization; moreover, such multi-layered overlay based forwarding involves nodes (as relays) who are not even interested in the message.

There are several differences between the aforementioned approaches and the proposal presented here.

First of all, the way in which collaborative applications communicate differs significantly both from classic one-to-many as well as 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 act as data source, but messages are directly used by nodes that lie on the dissemination path. In collaborative applications, in contrast, the messages must first be sent to and processed by a node acting as a synchronization server, and then be multicasted to the group members. Note that such dissemination pattern is similar to multicast with rendez-vous nodes, such as Scribe [29].

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, our approach can explicitly take into account not only the latency, but also the cost of the links and the observed participation levels and disconnection rates. This allows optimizing the structure of the mutlicast tree (in contrast to Scribe, that does not perform optimization).

On the theoretical level, the alternative to the spanning tree is to model multicast as a Steiner tree. These approaches assume that there are some nodes that only pass on the information. For instance [32] considers the problem of asymmetric latencies and [33] proposes a heuristic for bi-criteria optimization of latencies and costs.

In multiobjective optimization, we treat all the objectives symmetrically (as minimization goals). However, for some applications, optimization of the (network) latency matters only to a certain extent, beyond which other latencies (GUI, user reaction, etc.) are more noticeable. Cost and stability can have similar properties—for instance, system's reliability can be optimized until it reaches, e.g., five nines. Our multiobjective approach can be easily extended for such mixed objectives through, e.g., scaling based on reservation-aspiration points [34], applied on each objective before mixing the distance function; or by choosing different reference points than the ones corresponding to the trees optimal for cost, latency and stability.

8. Conclusions and Future Work

In this paper we model and propose 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 usually 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 in two scenarios: the classic, decentralized (many-to-many) multicast and the sender-server-multicast used in collaborative applications. We provided the complexity results and proposed exact, polynomial algorithms for optimal instability, latency and, in decentralized dissemination, cost.

Then, we tackled the general, multiobjective problem. Among many Pareto-optimal solutions, we proposed to chose the one closest to the ideal solution, composed of the optimum values of all three objectives determined individually. To find such a multiobjective optimized solution, we implemented a hill climbing algorithm, which if effective and efficient even for reasonably large graphs.

We performed experimental evaluation of the algorithms that showed that it is important to consider the multiobjective problem, as individual optimal solutions for single performance measure can be far from optimal for the others.

Then, we described a communication library (MoMo) that uses the proposed algorithms to opti-

mize the multicast tree. The protocols used in the library respond to dynamic changes in the network structure. We performed basic tests of the library on the Planet-Lab infrastructure. The MoMo library is in fact currently used as the communication layer for SharedMind [35] (`http://code.google.com/p/sharedmind`), an open-source collaborative mind-mapping application.

In our subsequent work, we plan to make the library even more responsive to local changes, by locally adjusting the spanning tree when a node joins or disconnects. Once the performance of the tree diverges too far from the multiobjective based optimum, our 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 additional constraints of guarantees for the minimum performance for each user.

References

[1] C. Sun, 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, J. Ferrié, Concurrent operations in a distributed and mobile collaborative environment, in: ICDE, Proceedings, 1998.

[3] G. Oster, P. Molli, S. Dumitriu, R. Mondejar, UniWiki: A Collaborative P2P System for Distributed Wiki Applications, in: Proc. of IEEE WETICE, IEEE Computer Society, 2009, pp. 87–92.

[4] G. Oster, P. Urso, P. Molli, A. Imine, Data consistency for P2P collaborative editing, in: Proc. of CSCW, ACM, 2006, p. 268.

[5] C. Sun, S. Xia, D. Sun, D. Chen, H. Shen, W. Cai, Transparent adaptation of single-user applications for multi-user real-time collaboration, ACM TOCHI 13 (4) (2006) 531–582.

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

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

[8] Google wave, `wave.google.com` (2009).

[9] B. Knutsson, H. Lu, W. Xu, B. Hopkins, Peer-to-peer support for massively multiplayer games, in: IEEE INFOCOM, Vol. 1, Citeseer, 2004, pp. 96–107.

[10] Badumna network suite, `http://www.badumna.com/badumna/technology.html` (2010).

[11] S. Kulkarni, Badumna Network Suite: A Decentralized Network Engine for Massively Multiplayer Online Applications.

[12] M. Raynal, M. Singhal, Logical time: Capturing causality in distributed systems, Computer 29 (2) (1996) 49–56.

[13] M. Ehrgott, Multicriteria Optimization, Springer, 2005.

[14] Y. Chu, L. Tseng-hong, On the shortest arborescence of a directed graph, Sci. Sin. 14 (1965) 1396–1400.

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

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

[17] T. Cormen, C. Leiserson, R. Rivest, C. Stein, Introduction to algorithms (2001).

[18] P. Jurcik, 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.

[19] S. Khuller, B. Raghavachari, N. Young, Balancing minimum spanning trees and shortest-path trees, Algorithmica 14 (4) (1995) 305–321.

[20] A. Kershenbaum, P. Kermani, G. Grover, MENTOR: an algorithm for mesh network topological optimization and routing, Communications, IEEE Transactions on 39 (4) (1991) 503–513.

[21] H. Connamacher, A. Proskurowski, The complexity of minimizing certain cost metrics for k-source spanning trees, Discrete Applied Mathematics 131 (1) (2003) 113–127.

[22] B. Li, Y. Qu, Y. Keung, S. Xie, C. Lin, J. Liu, X. Zhang, Inside the new coolstreaming: Principles, measurements and performance implications, in: Proc. of IEEE INFOCOM, 2008.

[23] S. Deering, D. Cheriton, Multicast routing in datagram internetworks and extended LANs, ACM Transactions on Computer Systems (TOCS) 8 (2) (1990) 85–110.

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

[25] P. Francis, Y. Pryadkin, P. Radoslavov, R. Govindan, B. Lindell, Yoid: Your Own Internet Distribution, Technical Report, ACIRI (2000).

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

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

[28] Y. Chu, S. Rao, S. Seshan, H. Zhang, A case for end system multicast, Selected Areas in Communications, IEEE Journal on 20 (8) (2002) 1456–1471.

[29] M. Castro, P. Druschel, A. Kermarrec, A. Rowstron, Scribe: a large-scale and decentralized application-level multicast infrastructure, Selected Areas in Communications, IEEE Journal on 20 (8) (2002) 1489–1499.

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

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

Table 11: Comparison between the multiobjective hill climbing and the exhaustive search on graphs with no leaf nodes and the firewall ratio of 0.5.

	score				instances [%]	
	multiobjective		exhaustive			
n	$\bar{x}$	σ	$\bar{x}$	σ	worse	pareto
5	0.62	0.22	0.61	0.22	8	0
6	0.54	0.22	0.51	0.20	17	1
7	0.50	0.20	0.49	0.19	11	0
8	0.46	0.16	0.44	0.15	27	3
9	0.43	0.15	0.41	0.14	31	2

Table 12: Comparison between the multiobjective hill climbing and the exhaustive search on graphs with no leaf nodes and the firewall ratio of 0.0 (no nodes firewalled).

	score				instances [%]	
	multiobjective		exhaustive			
n	$\bar{x}$	σ	$\bar{x}$	σ	worse	pareto
5	0.63	0.21	0.60	0.19	16	1
6	0.56	0.20	0.53	0.18	22	3
7	0.49	0.18	0.48	0.18	17	2
8	0.47	0.15	0.45	0.14	27	1
9	0.46	0.17	0.43	0.15	28	4

179.

[32] S. Ramanathan, Multicast tree generation in networks with asymmetric links, Networking, IEEE/ACM Transactions on 4 (4) (1996) 558–568.

[33] V. Kompella, J. Pasquale, G. Polyzos, Multicast routing for multimedia communication, Networking IEEE/ACM Transactions on) 1 (3) (1993) 286–292.

[34] A. Wierzbicki, A mathematical basis for satisficing decision making, Mathematical Modelling 3 (5) (1982) 391–405.

[35] S. Ang, K. Rzadca, A. Datta, SharedMind: A tool for collaborative mind-mapping, in: ICME 2010 (Demo paper), Proceedings

Additional Experimental Results

Table 13: average *dist* to the ideal solution for hill climbing and various variants of Simulated Annealing (SA). Graphs with $n = 50$ nodes, no leaf nodes and no firewalled nodes.

settings	$\bar{x}$	σ
hill climbing	0.19	0.07
SA, $t_0 = 1.0$, $f_t = 0.95$	0.51	0.12
SA, $t_0 = 0.1$, $f_t = 0.97$	0.41	0.09
SA, $t_0 = 0.01$, $f_t = 0.99$	0.29	0.07

Table 9: Detailed comparison of trees returned by the multiobjective algorithm to the best possible star topology.

	$L(T)$				$C(T)$				$\Lambda(T)$			
	mult		star		mult		star		mult		star	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
5	0.22	0.24	0.54	0.40	0.29	0.22	0.80	0.60	0.39	0.23	0.41	0.51
6	0.22	0.20	0.75	0.52	0.24	0.17	0.79	0.37	0.35	0.20	0.44	0.53
7	0.21	0.16	0.72	0.38	0.24	0.17	0.95	0.42	0.32	0.17	0.46	0.62
8	0.20	0.16	0.79	0.37	0.20	0.13	0.93	0.36	0.32	0.13	0.35	0.42
9	0.18	0.14	0.69	0.31	0.20	0.12	0.93	0.34	0.28	0.13	0.47	0.45
10	0.16	0.12	0.75	0.31	0.17	0.11	0.93	0.29	0.29	0.14	0.50	0.49
12	0.15	0.10	0.82	0.28	0.16	0.08	0.90	0.28	0.28	0.12	0.52	0.52
15	0.12	0.07	0.82	0.23	0.13	0.07	0.90	0.25	0.24	0.11	0.49	0.46
20	0.11	0.07	0.89	0.20	0.11	0.06	0.98	0.29	0.22	0.11	0.51	0.41
25	0.11	0.05	0.91	0.19	0.09	0.05	0.98	0.22	0.18	0.09	0.41	0.37
30	0.10	0.04	0.93	0.15	0.08	0.04	0.99	0.24	0.17	0.09	0.49	0.43
40	0.11	0.05	0.92	0.16	0.06	0.03	0.99	0.19	0.16	0.08	0.49	0.41
50	0.10	0.05	0.91	0.15	0.06	0.03	0.97	0.18	0.14	0.06	0.54	0.45

Table 10: Detailed comparison of trees returned by the multiobjective algorithm with the trees returned by the penalty-based algorithm.

	$L(T)$				$C(T)$				$\Lambda(T)$			
	mult		pen		mult		pen		mult		pen	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
5	0.22	0.26	0.27	0.36	0.27	0.28	0.33	0.34	0.34	0.27	0.41	0.35
6	0.25	0.26	0.34	0.39	0.26	0.28	0.31	0.35	0.29	0.24	0.37	0.37
7	0.24	0.27	0.35	0.37	0.23	0.24	0.23	0.29	0.31	0.22	0.34	0.30
8	0.22	0.24	0.34	0.35	0.26	0.24	0.24	0.27	0.31	0.21	0.33	0.26
9	0.23	0.22	0.37	0.38	0.21	0.16	0.25	0.27	0.34	0.21	0.33	0.25
10	0.19	0.16	0.38	0.34	0.25	0.21	0.22	0.27	0.31	0.17	0.34	0.28
12	0.18	0.14	0.31	0.29	0.20	0.14	0.20	0.20	0.32	0.18	0.31	0.23
15	0.23	0.19	0.33	0.27	0.17	0.15	0.16	0.20	0.30	0.17	0.30	0.23
20	0.16	0.13	0.37	0.28	0.17	0.11	0.12	0.14	0.26	0.12	0.23	0.16
25	0.14	0.10	0.33	0.29	0.13	0.08	0.12	0.12	0.24	0.13	0.22	0.16
30	0.12	0.07	0.31	0.23	0.14	0.11	0.10	0.10	0.21	0.12	0.18	0.14
40	0.12	0.10	0.33	0.23	0.12	0.08	0.10	0.14	0.20	0.12	0.16	0.13
50	0.10	0.06	0.37	0.27	0.09	0.07	0.05	0.07	0.18	0.10	0.16	0.11

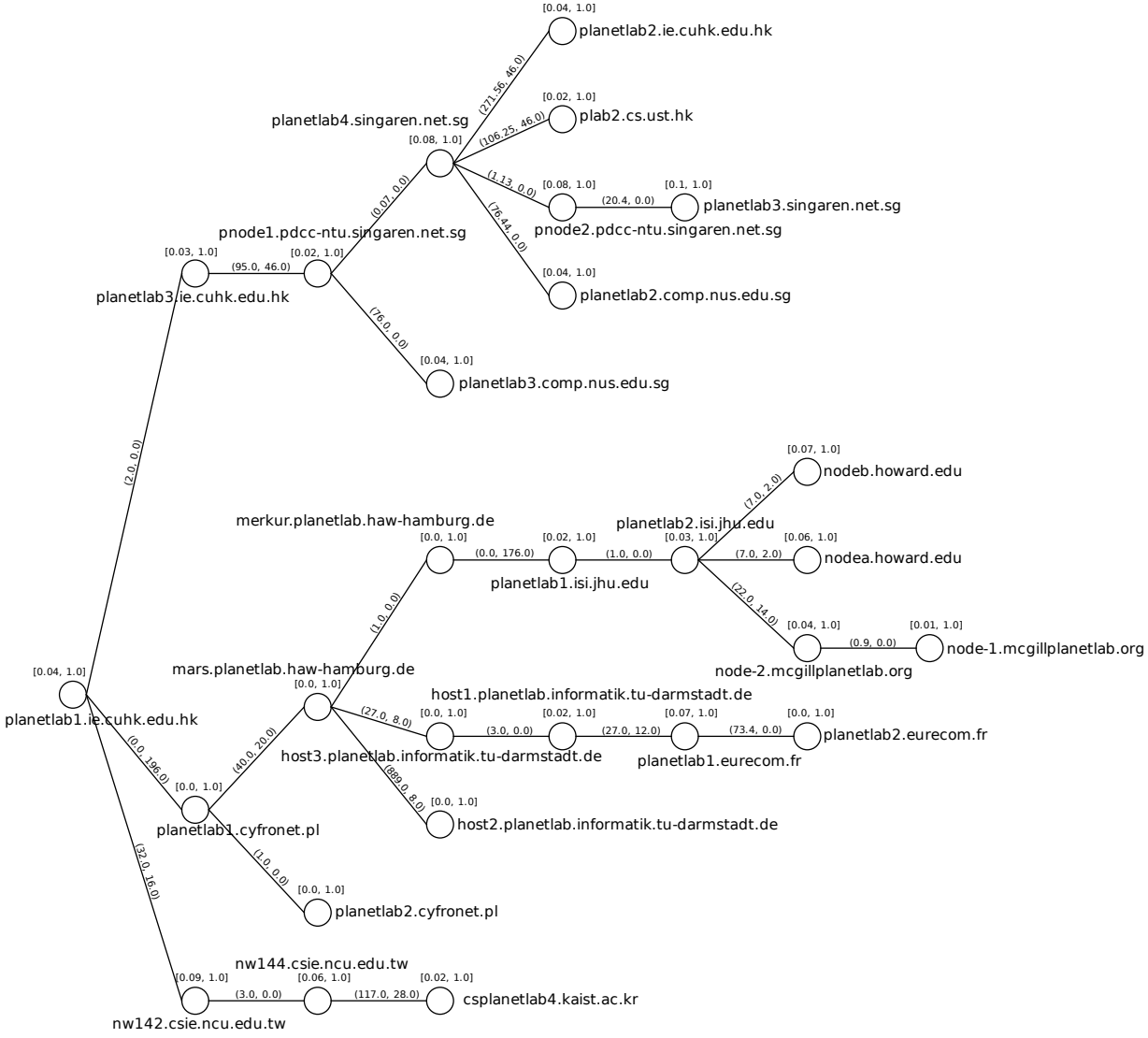

Figure 6: Network topology derived by the multiobjective algorithm for 28 planet-lab nodes.

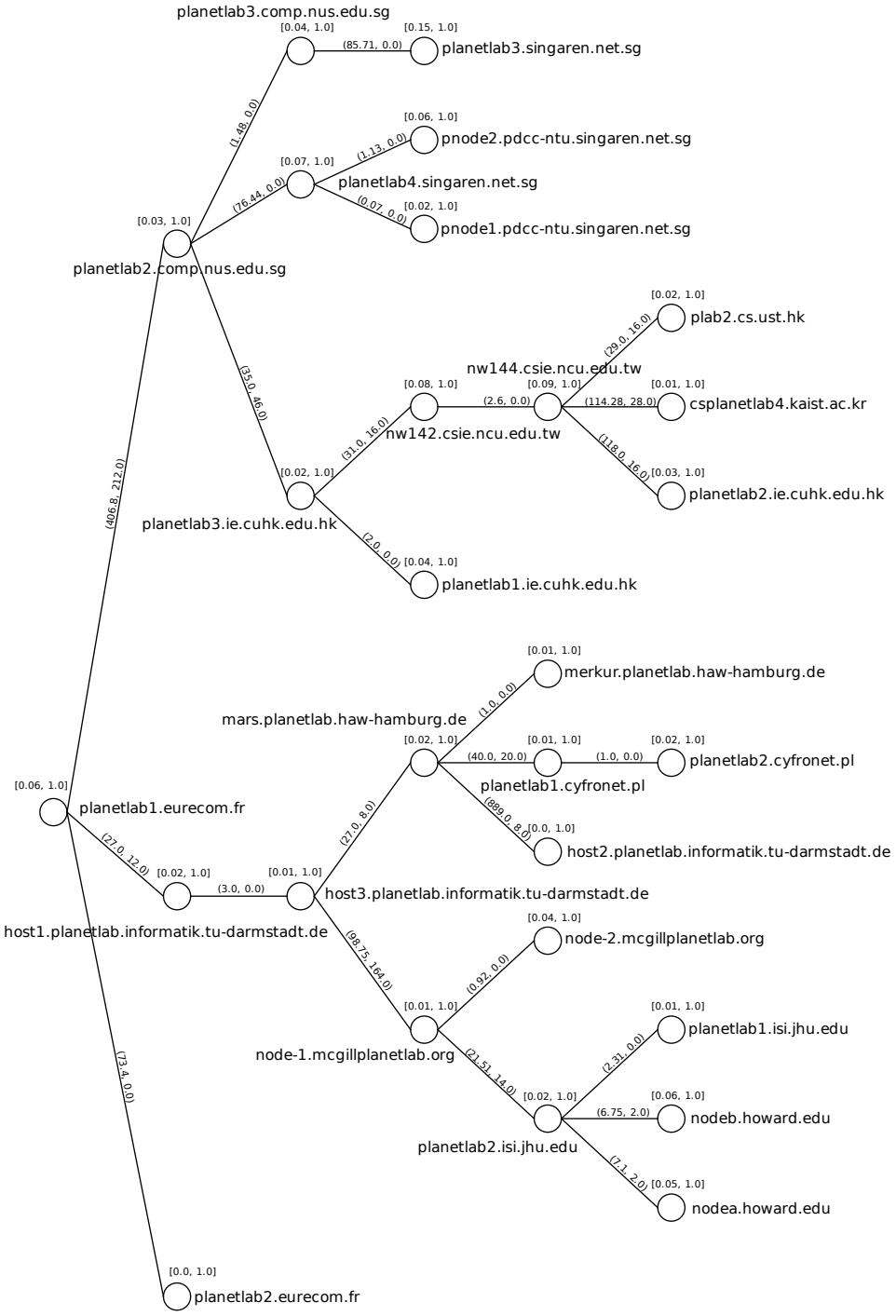

Figure 7: A different network topology derived by the multiobjective algorithm for the same 28 planet-lab nodes as in Figure 6, but at a different time. Such a different configuration is an artefact of PlanetLab's non-determinism. Nevertheless, the basic qualitative characteristics of the overlay are similar. Note also that the same Hong Kong nodes again form a somewhat unexpected configuration, which we speculate is because of these nodes being very slow to respond and thus being chosen as leaf nodes in the system by our algorithm.