Dynamic Multicast in Overlay Networks with Linear Capacity Constraints

Ying Zhu, Baochun Li, Ken Q. Pu

Abstract—In a peer-to-peer overlay network, the phenomonon of multiple overlay links sharing bottleneck physical links leads to correlation of overlay link capacities. We are able to more accurately model the overlay by incorporating these linear capacity constraints (LCC). We formulate the problem of maximizing bandwidth in overlay multicast using our LCC model. We show that finding a maximum-bandwidth multicast tree in an overlay network with LCC is NP-complete. Therefore, an efficient heuristics algorithm is designed to solve the problem. Extensive simulations show that our algorithm is able to construct multicast trees that are optimal or extremely close to optimal, with significantly higher bandwidth than trees formed in overlays with no LCC. Furthermore, we develop a fully distributed algorithm for obtaining near-optimal multicast trees, by means of gossip-based algorithms and a restricted but inherently distributed class of LCC (node-based LCC). We demonstrate that the distributed algorithm converges quickly to the centralized optimal and is highly scalable.

Index Terms—Peer-to-peer overlay networks, multicast, algorithms, network protocols.

1 Introduction

We consider the problem of maximizing the bandwidth of application-layer multicast in overlay networks. Overlay links map to paths in the low-level network. When overlay links map to paths that share a common physical link, the *sum* of the capacities of the overlay links is constrained by the capacity of the shared physical link. We say these overlay links are *correlated* in capacity. Some existing work, however, view the overlay as a regular network graph where capacities of links are simply scalars which are their unicast bandwidths. We refer to this as the independent overlay model, as it models overlay link capacities as uncorrelated, independent entities. Other related work considers *location-aware* or *topology-aware* overlays (*e.g.*, [1], [2]). They either focus on end-to-end latencies, or assume that the underlying IP-layer topology is known. In comparison, we seek to focus on end-to-end bandwidth, and believe that it is prohibitively expensive — and thus not feasible — to discover the entire AS-level topology using overlay probing techniques (such as `traceroute`), just for the sake of improving overlay performance.

We introduced *linear capacity constraints* (LCC) as a formulation of link correlations in overlays. Instead of scalars, the overlay links are assigned capacity variables and linear constraints of these variables express the hidden link correlations. For instance, suppose two overlay links u and v map to paths that share the physical link e with capacity $c(e)$, this link correlation would give rise to the linear capacity constraint $x_u + x_v \leq c(e)$, where x_u, x_v are the respective capacity variables for u, v. The *LCC-overlay* is simply an overlay graph with variables for the link capacities together with a set of LCC.

We now use a simple example to illustrate how better multicast trees can be obtained in an LCC-overlay than in the traditional independent overlay. Consider the simple overlay network in Fig. 1(a). There are two planes: the upper plane is the overlay network; the lower plane is the low-level IP network with paths that connect the overlay nodes ($r1, r2$ are routers). A maximum-bandwidth tree one obtains in the independent overlay (i.e., upper plane) is shown in Fig. 1(b). The actual achievable bandwidth of this tree is only 50, because overlay links (u, v) and (u, w) share the low-level bottleneck link $(r1, r2)$. Now we consider the LCC-overlay. We use the LCC's $c(u, v)+c(u, w) \leq 100$ and $c(v, w) \leq 99$ (where $c(u, v)$ is the capacity variable for link (u, v), etc.). A maximum-bandwidth tree found this way is shown in Fig. 1(c); it has the achievable bandwidth of 99. By assigning variables instead of scalars to overlay links and using LCC's, we were able to find a tree with higher bandwidth.

The above example is extremely simplified, it only serves to motivate our endeavor in this paper. For the remainder of the paper, we will study the problem of optimizing bandwidth in constructing overlay multicast trees. We define three metrics to measure the performance and quality of an overlay multicast tree. We show that the optimal multicast tree found in an independent overlay cannot reliably achieve high bandwidth, due to its fundamental weakness of being unable to accurately represent the true network topology.

We formulate the new problem of overlay multicast in the LCC-overlay: Given an overlay with a set of LCC, how do we find a highest-bandwidth multicast tree?

Y. Zhu is with the Faculty of Business and Information Technology, University of Ontario Inst. of Tech. E-mail: ying.zhu@uoit.ca
B. Li is with the Department of Electrical and Computer Engineering, University of Toronto. E-mail: bli@eecg.toronto.edu
K. Pu is with the Faculty of Science, University of Ontario Inst. of Tech. E-mail: ken.pu@uoit.ca

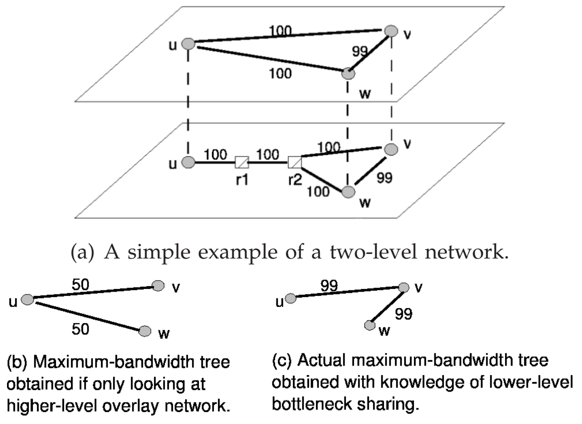

100
u
v
99
100
w
u
100
100
100
v
r1
r2
99
100
w
(a) A simple example of a two-level network.
50
u
v
50
w
(b) Maximum-bandwidth tree obtained if only looking at higher-level overlay network.
99
u
v
99
w
(c) Actual maximum-bandwidth tree obtained with knowledge of lower-level bottleneck sharing.

Fig. 1. Simple example illustrating the problems caused by lower-level bottlenecks.

We show that this problem is NP-complete. We propose a heuristics algorithm to solve this problem. Through extensive simulations, we compare the performance of multicast trees constructed in the independent overlay, and those found by our algorithm using LCC-overlays. Our algorithm always obtains near-optimal trees, converges quickly, and is scalable for increasing network sizes.

Furthermore, we designed a fully distributed algorithm. The algorithm ensures the preservation of the tree topology as nodes executes protocols in an entirely distributed and asynchronous manner. The algorithm, moreover, does not rely on the assumption of any global information. Gossip-based algorithms are utilized to efficiently estimate and gather global statistics. Then, each node acts independently and asynchronously to improve the tree bandwidth. Through simulations, we demonstrate that the distributed algorithm consistently performs as well as its centralized counterpart.

The remainder of the paper is organized as follows. We discuss related work in Sec. 2. In Sec. 3, we present definitions of LCC and the metrics. The problem of finding high-bandwidth multicast trees is presented in Sec. 4, as well as the (centralized) heuristics algorithm. We evaluate this algorithm in Sec. 5. An overview of the distributed algorithm is presented in Sec. 6; its formal presentation is given in Sec. 7, Sec. 8, and it is evaluated in Sec. 9. Sec. 10 concludes the paper.

2 Related Work

Distributed algorithms for general-purpose overlay construction were proposed by Young *et al.* in [3] and by Shen in [4]. Application-specific proposals have been made for various overlay services, including overlay multicast [5], [6], content distribution [7], [8], [9] and multimedia streaming [10], [11], [12], [13], [14]. In [7], a distributed algorithm is proposed to find an optimal set of content distribution trees; the technique of distributing different file pieces on multiple trees is used. A mesh-based peer-to-peer streaming mechanism is proposed in [10] for the delivery of live content that incorporates swarming content delivery. An empirical investigation of the Coolstreaming peer-to-peer live streaming system is presented in [11] Also relevant is previous work by Ratnasamy *et al.* [1]; a distributed binning scheme is designed for forming unstructured overlay networks, with the focus on latency.

All of the above proposals view and treat overlay links as independent. However, we have studied the new problem of high-bandwidth data dissemination in a multicast tree within the framework of overlays with linear capacity constraints. In our previous paper [15], we only studied problem of overlay unicast (widest path and maximum flow), and did not consider the problem of overlay multicast. As well, in this paper (as opposed to [15]), the metrics are defined differently with respect to multicast trees, and the stress metric defined here is new.

Recent work by Tsang *et al.* [16] identified a number of data transfer patterns that cause suboptimal bandwidth usage of bottleneck links. They proposed a solution which involves topologically close receivers to request data in a coordinated way. Another recent work by Kim *et al.* [17] proposes a protocol to eliminate probed bottlenecks in an overlay multicast tree. Our work is distinct from theirs in formalizing link correlations using linear capacity constraints and thus being able to find an efficient and distributed algorithm based on node-based LCC. We also rigorously define the metrics of accuracy, efficiency and stress, which not only quantitatively measure performance, but also provide crucial insight into the quality of overlay networks in general. By formalizing the LCC-overlay model, and by conceptually separating LCC-overlay construction and finding multicast trees using LCC, we are able to (1) prove that finding the optimal overlay multicast tree is NP-complete, and (2) develop fully distributed algorithms which consistently achieve performance that is remarkably close to the optimal. Our evaluation of the performance is based on comparing against a rigorously determined upper bound of the optimal.

3 Linear Capacity Constraints in Overlay Networks

An overlay is a hierarchical network comprising two levels: the lower level is the IP network of routers and physical links, while the higher level is an application-layer network of end-systems (overlay nodes) and virtual links (overlay links). A virtual link is the unicast connection between two overlay nodes; it maps to a path of physical links in the low-level network, which is determined by IP routing. It is possible that two overlay links map to paths which share a bottleneck physical link. Obviously, the sum of the capacities of these overlay links is bounded by the capacity of the underlying bottleneck link. We say these two overlay links are *correlated* in capacity. Our proposal is to model the overlay by explicitly expressing link correlations as linear capacity constraints (LCC). We will show that the *LCC overlay model* is necessary for ensuring overlay quality, in particular, for yielding the highest bandwidth in overlay data dissemination.

Many previous proposals of algorithms and protocols for overlay networks have adopted, implicitly or explicitly, a model of the overlay as a traditional flat network rather than hierarchical. That is, they do not specifically take into consideration overlay link correlations; they assume that overlay links are all independent. Henceforth we will refer to this as the *independent overlay* model. In the independent model, an overlay network is viewed as a weighted graph where the edges are overlay links weighted by their unicast bandwidth and delay. A representative generic scheme of constructing an overlay with the aim of optimizing bandwidth is: each node selects its neighbors by choosing k adjacent links with the highest unicast bandwidth.

We use the example in Figure 2 to illustrate the disadvantage of the independent model, and to introduce and show the advantage of the LCC model.

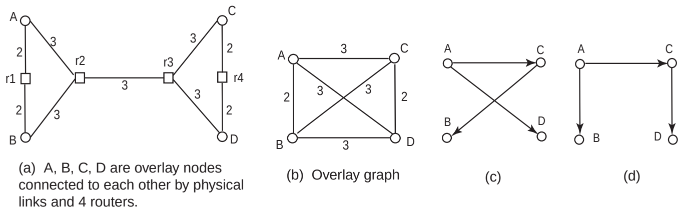

A
B
C
D
r1
r2
r3
r4
2
3
3
2
2
3
3
2
3
(a) A, B, C, D are overlay nodes connected to each other by physical links and 4 routers.
(b) Overlay graph
(c)
(d)

Fig. 2. A simple example illustrating the disadvantage of the independent model and the advantage of linear capacity constraints.

The example network in Figure 2(a) contains four overlay nodes, $A-D$, and four low-level routers, $r1-r4$. The overlay graph is given in Figure 2(b), in which the overlay links are labeled by their independent unicast bandwidths. Figure 2(b) is the case when k is 3 in the above construction scheme based on the independent model; it is not hard to see that the following reasoning will also hold for $k=2$ or 1.

In the independent model, the highest-bandwidth multicast tree in this overlay graph[1] is shown in Figure 2(c). Although the predicted bandwidth of the tree is 3, it can be seen that the achievable bandwidth of the tree is 1 because all three links in the tree share a single bottleneck physical link $(r2, r3)$ with capacity 3.

In the LCC model, instead of links labeled by numbers, the capacity of each overlay link is represented by a variable and a set of linear constraints are used to formulate link correlations. For instance, the above link correlation among the three overlay links can be easily captured by the constraint $x_{AC}+x_{AD}+x_{CB} \leq 3$, where x_{AC} is the capacity variable for link (A, C) and so on.

1. The tree can be found by an all-widest paths algorithm that is a variant of Dijkstra's all-shortest paths.

The complete set of linear capacity constraints (LCC) is given below in matrix form:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{AB} \\ x_{AC} \\ x_{AD} \\ x_{BC} \\ x_{BD} \\ x_{CD} \end{pmatrix} \leq \begin{pmatrix} 2 \\ 3 \\ 2 \end{pmatrix} \qquad (1)$$

The LCC-overlay is the overlay graph with link capacity variables together with the set of LCC. To find a highest-bandwidth multicast tree in this LCC-overlay, one can devise a simple variant of the algorithm for regular graphs. The difference is that every time a link is considered for selection, its capacity (instead of a fixed number) is computed as its fair share of the capacities in the LCC with the set of already selected links. The tree thus obtained is given in Figure 2(d). In this case, the achievable bandwidth is 2, the same as the predicted bandwidth.

The highest-bandwidth tree obtained in the LCC-overlay has twice the bandwidth of that in the independent overlay. Moreover, it is easy to see that the LCC tree in fact achieves the optimum multicast tree bandwidth. From this example, we can observe that the achievable bandwidth of the independent overlay is low because it lacks accuracy in representing the true network topology. In the LCC-overlay, however, the incorporation of link correlations results in an accurate representation of underlying topology, and hence an optimal multicast tree can be found.

3.1 Formal definitions of LCC and metrics

We now present formal definitions of LCC-overlay and several metrics that measure the performance and quality of overlay multicast trees.

The two-level hierarchy of an overlay network can be formulated as consisting of:

- A low-level (IP) graph $G=(V,E)$; each low-level link $e \in E$ has a capacity of $c(e) \geq 0$.
- A high-level overlay graph $\widehat{G}=(\widehat{V},\widehat{E})$, where $\widehat{V} \subset V$;
- A mapping P of every overlay edge $(\widehat{v}_1,\widehat{v}_2) \in \widehat{E}$ to a low-level path $P(\widehat{v}_1,\widehat{v}_2) \subset G$ from $\widehat{v}_1$ to $\widehat{v}_2$.

The formulation of capacity constraints in the overlay graph $\widehat{G}$ is where LCC-overlay departs from independent overlay. The independent overlay is defined as follows.

Definition 1 (Independent overlay): The *independent overlay* is a pair $(\widehat{G},\widehat{c})$, where $\widehat{c}$ is a capacity function such that each link $\widehat{e} \in \widehat{E}$ has a nonnegative capacity $\widehat{c}(\widehat{e}) \geq 0$.

The LCC-overlay is defined as follows.

Definition 2 (LCC-overlay): The *LCC-overlay* is a triplet $(\widehat{G}, C, b)$ where

- The capacity of each link $\widehat{e}$ in $\widehat{G}$ is a variable $x_{\widehat{e}}$.
- (C,b) represent a set of m linear capacity constraints $Cx \leq b$:
  - C is a 0-1 coefficient matrix of size $m \times |\widehat{E}|$;

– x is the $|\widehat{E}| \times 1$ vector of link capacity variables;
– $b \in \mathrm{R}^m$ is the capacity vector.

Each row i in (C, b) is a constraint of the form $\sum_{\widehat{e}:C(i,\widehat{e})=1} x_{\widehat{e}} \leq b(i)$.

3.1.1 Overlay (or predicted) bandwidth

Given any tree T in either the independent overlay or the LCC-overlay, we observe that there exists the notion of the *overlay bandwidth* of T. The overlay bandwidth of a tree can be thought of as its *predicted bandwidth* in a certain overlay model. Because tree construction algorithms in overlays rely solely on predicted bandwidth to optimize tree formation, this is an crucial concept to clarify and study in these two distinct overlay models.

For a tree T in the independent overlay $(\widehat{G}, \widehat{c})$, the overlay bandwidth of T is the minimum bandwidth link in T as given by $\widehat{c}$, or formally, $\min\{\widehat{c}(\widehat{e}) : \widehat{e} \in T\}$.

As for a tree T in an LCC-overlay $(\widehat{G}, C, b)$, the overlay bandwidth of T can be obtained by the following procedure. For each constraint (C_i, b_i) (i-th row in C, b), compute the bandwidth allocated to every link $j \in T$ for which $C_{i,j} = 1$: $\gamma_i(j) = b_i / \sum_{k \in T} C_{i,k}$. That is, $\gamma_i(j)$ is the bandwidth allocated to j in T by the i-th constraint. Considering all constraints together, the allocated bandwidth of link j in T is $\gamma j = \min\{\gamma_i(j) : i = 1 \ldots m\}$, where m is number of rows in C. Finally, the overlay bandwidth of T is $\sigma(T) = \min\{\gamma(j) : j \in T\}$.

3.1.2 Achievable bandwidth $\sigma_G(T)$

Another notion of interest is: Given any overlay tree T spanning all overlay nodes, what is the *achievable bandwidth* of T? Let $\sigma_G(T)$ denote the achievable bandwidth of T in overlay $\widehat{G}$ residing on top of G. First, for each low-level edge e, equal shares of its capacity are allocated to overlay edges in T. It follows that an overlay edge $\widehat{e}$ has a bandwidth allocation from every low-level edge to which $\widehat{e}$ maps. The achievable bandwidth of $\widehat{e}$ is the minimum of all these allocations. Finally, the minimum of achievable bandwidths of all overlay edges in T is the achievable bandwidth of T, $\sigma_G(T)$. The procedure for obtaining $\sigma_G(T)$ is summarized in Figure 3. The following verifies the correctness of the procedure.

Proposition: Given an overlay $\langle \widehat{G}, G, P \rangle$ and any overlay tree $T \subseteq \widehat{G}$, the highest bandwidth that T can achieve is $\sigma_G(T)$ obtained by the procedure in Fig. 3.

Proof: For each overlay edge $\widehat{e}$ in T, its bandwidth $\gamma_T(\widehat{e})$ is assigned the minimum of its allocations from low-level edges mapped from $\widehat{e}$. Consequently, at each low-level edge e, the γ_T of each mapped overlay edge is no greater than its allocation, hence the sum of γ_T's cannot exceed the capacity of e. Since $\sigma_G(T)$ is the minimum of the γ_T's, it is obvious that $\sigma_G(T)$ does not violate any low-level capacity and is hence feasible.

Because the bandwidth metric is concave and the bandwidth of a tree is the minimum bandwidth of its edges, the achievable bandwidth of T must be no greater than the minimum of all allocations given out by low-level edges. To maximize bandwidth of T, one is actually maximizing the minimum of the allocations by low-level edges. It is easy to see that to maximize the minimum, a low-level edge should assign equal allocations. Therefore, the $\sigma_G(T)$ obtained by the procedure is the highest bandwidth T can achieve. □

```
for each $e \in E$
   Allocate $c(e)$ equally among
      $\{\widehat{e} : e \in P(\widehat{e})$ and $T(\widehat{e}) > 0\}$,
   let each allocation be denoted by $\gamma_T^e(\widehat{e})$
   for each $\widehat{e} \in \widehat{E}$
     if $T(\widehat{e}) > 0$     $\gamma_T(\widehat{e}) \leftarrow \min\{\gamma_T^e(\widehat{e}) : e \in P(\widehat{e})\}$
     else            $\gamma_T(\widehat{e}) \leftarrow 0$
   $\sigma_G(T) \leftarrow \min\{\gamma_T(\widehat{e}) : \gamma_T(\widehat{e}) \neq 0$ and $\widehat{e} \in \widehat{E}\}$
```

Fig. 3. The procedure for obtaining achievable bandwidth of a tree T in G, $\sigma_G(T)$.

We proceed now to present definitions of three metrics: accuracy, efficiency and stress. Accuracy is the ratio of the overlay or predicted bandwidth of a tree over its achievable bandwidth. Efficiency is the ratio of achievable bandwidth of a tree over the optimum tree bandwidth. Stress is defined for low-level links that are mapped by overlay tree links; it is essentially the load placed on a low-level link by the overlay tree, normalized by the link capacity. The formal definitions are given below.

Definition 3 (Accuracy): The *accuracy* of a multicast tree T in an overlay network $\langle G, \widehat{G}, P \rangle$, is

$$\frac{\text{overlay bandwidth of } T \text{ in } \widehat{G}}{\sigma_G(T)}. \tag{2}$$

Definition 4 (Efficiency): The *efficiency* of a multicast tree T in an overlay network $\langle G, \widehat{G}, P \rangle$, is

$$\frac{\sigma_G(T)}{\sigma_G(T_{\text{opt}})}, \tag{3}$$

where T_{opt} is an optimum multicast tree in $\langle G, \widehat{G}, P \rangle$.

Definition 5 (Stress): The *stress* of a low-level edge $e \in E$ due to a multicast tree T in an overlay network $\langle G, \widehat{G}, P \rangle$, is

$$\frac{|\{\widehat{e} : \widehat{e} \in \widehat{E} \text{ and } \widehat{e} \in T\}|}{c(e)}, \tag{4}$$

where $c(e)$ is the capacity of e. Our definition of *stress* is original; unlike the stress metric in [5], our stress metric normalizes the number of overlay links by the capacity of the physical link. The reasoning is that a physical link with a huge capacity can provide high bandwidth to each overlay link mapped to it, even if there are many.

4 Multicast Tree with Linear Capacity Constraints

4.1 MTC is NP-complete

We showed above that by using LCC, the bandwidth of multicast trees can be significantly increased. Now we are faced with the question of how to obtain the highest-bandwidth multicast tree in an LCC-graph.

We state the problem of highest-bandwidth multicast tree with LCC (MTC) as a decision problem as follows.

INSTANCE: An LCC-graph (G, C, b), where $G = (V, E)$ and (C, b) are LCC; a positive integer B.

QUESTION: Is there a multicast tree T for G such that the bandwidth of T is $\geq B$?

Theorem: MTC is NP-complete.

Proof:

MTC is in NP because given an LCC-graph (G, C, b), a positive integer B and a multicast tree T for G, the bandwidth T can be computed by the polynomial-time procedure in Sec. 3.1 for obtaining the overlay bandwidth of T in an LCC-overlay.

To show that MTC is NP-complete, we reduce the NP-complete problem Degree-Constrained Spanning Tree (DST) [18] to MTC; the problem DST is defined as:

INSTANCE: Graph $G = (V, E)$, positive integer $K \leq |V|$.

QUESTION: Is there a spanning tree for G in which no vertex has degree larger than K?

Given an instance of DST, $G = (V, E)$ and K, we transform it to an instance of MTC. We let G remain the same and add some LCC. For every node $u \in V$, a constraint is formed: $\sum_{e \in \text{inc}(u)} x_e \leq K \cdot B$, where inc$(u)$ denotes the set of edges incident to u. This transformation can obviously be done in polynomial time.

Suppose a spanning tree T exists in G for which no node has degree larger than K. That is, $|\text{inc}(u)| \leq K, \forall u \in V$. Then assigning B to each edge in T implies that $x_e = B$ or 0. It follows that $\sum_{e \in \text{inc}(u)} x_e \leq K \cdot B, \forall u \in V$. Thus T is a multicast tree that satisfies all the LCC and $\sigma(T) = B$.

In the other direction, suppose a multicast tree T has bandwidth $B' \geq B$. If a node $u \in T$ has degree greater than K, *i.e.*, $|\text{inc}(u)| = d > K$, then $\sum_{e \in \text{inc}(u)} x_e \geq d \cdot B' > K \cdot B$, which violates the constraint imposed. Therefore no node has degree greater than K. Clearly, T is a solution of DST. □

4.2 Heuristics Algorithm

In Sec. 4.1 above, we showed that the problem of finding the highest-bandwidth multicast tree with LCC is NP-complete. Therefore, we propose a heuristics algorithm to solve the problem in this section. The evaluation of the algorithm will be deferred to Sec. 5.

The input is an LCC-graph $(\widehat{G}, C, b)$. The goal is to find a high-bandwidth multicast tree. The main idea of the algorithm is to begin with an initial tree, and make incremental improvements by replacing, at each iteration, the lowest-bandwidth edge with a higher-bandwidth one, thereby increasing bandwidth of the tree. For every edge replacement, the algorithm preserves the topology of a multicast tree, *i.e.*, a tree spanning all receiver nodes.

The initial multicast tree is found by first forming a regular graph G with edges weighted with independent

```
HMTC($\widehat{G}, C, b$)
obtain $G$ with independent edge
  capacities from $\widehat{G}$ and LCC $(C, b)$
$T_0 \leftarrow$ highest-bandwidth tree in $G$
$(C_{T_0}, b) \leftarrow$ LCC of $T_0$
$R \leftarrow$ edges in $T_0$ in ascending order
  of bandwidths allocated by $(C_{T_0}, b)$
$T_{hi} \leftarrow T_0$
iter $\leftarrow 1$
while iter $\leq$ max_num_iterations and $R \neq \varnothing$
  $r \leftarrow$ 1st edge in $R$
  $R \leftarrow R - \{r\}$
  $T_{sub} \leftarrow$ subtree under $r$
  $T \leftarrow T_{hi} - \{r\}$
  for each edge $e \in T - T_{sub}$
    $T' \leftarrow T \cup \{e\}$
    $C_{T'} \leftarrow$ LCC of $T'$
    if bandwidth$(T')$ > bandwidth$(T_{hi})$
      $T_{hi} \leftarrow T'$
      $R \leftarrow$ edges in $T_{hi}$ in ascending
       order of allocated bandwidths
      break out of for-loop
  iter $\leftarrow$ iter $+ 1$
return $T_{hi}$
```

Fig. 4. Summary of heuristics algorithm *HMTC*.

edge capacities (numbers). This is easily done by setting $\min\{b_i : C(i, e) = 1\}$ to be the capacity for each edge $e \in \widehat{G}$. Note that this corresponds exactly to an independent overlay with unicast bandwidths for the edges. A highest-bandwidth multicast tree is found in G and set to be the initial tree, let it be denoted T_0. The algorithm then incrementally improves the tree by always replacing the worst edge with a better edge, if possible, while maintaining the spanning tree structure.

The algorithm, which we call *HMTC* (High-bandwidth Multicast Tree with LCC), is summarized in Table 4.

We re-use the example overlay network in Figure 2 from Sec. 3 to illustrate the algorithm. The overlay graph along with the LCC (C, b) are in Figure 5(a). For simplicity of presentation, we let UV denote the capacity variable of edge (U, V).

In the graph in Figure 5(a), the numbers labeling the edges are the independent edge capacities. For the graph with independent edge capacities, a highest-bandwidth multicast tree is found: T_0 in Figure 5(b). The *LCC of* T_0 is listed on the right of T_0 in the figure. The LCC of T_0, C_{T_0}, is LCC (of the graph) (C, b) intersected with T_0 so that they only contain capacity variables of edges from T_0. In this instance, there is only one constraint in LCC of T_0: $AC + AD + BC \leq 3$. Each T_0 edge is thus allocated an equal bandwidth of 1. The bandwidth of T_0 is the minimum of allocated bandwidths of its edges, in this case, 1.

The worst edge, one with the lowest bandwidth allocated by the LCC of T_0, is selected to be replaced. Since all three edges have lowest bandwidth, any one can be selected, say edge AC. We want to find another edge XY such that by removing AC and adding XY to form T_1, T_1 is a multicast tree and the bandwidth of T_1 (allocated by the LCC of T_1) is higher than the bandwidth of T_0. In other words, XY connects the subtree under AC, *i.e.*,

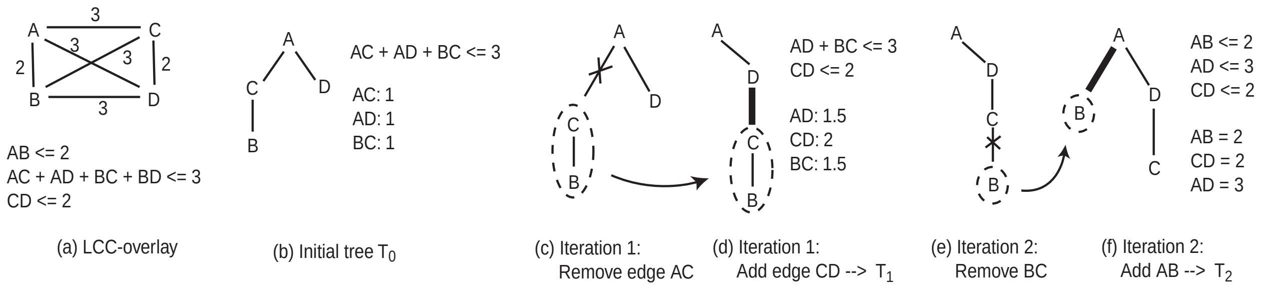


Fig. 5. An example of the heuristics algorithm being executed.

subtree rooted at node C, with the rest of the tree. The resulting new tree also has improved bandwidth.

Edge CD satisfies the above conditions; the removal of AC and the new tree T_1 formed by adding CD can be seen in Figure 5(c) and (d), respectively. The LCC of T_1 and the bandwidths allocated to the edges are given as well. The bandwidth of T_1 is 1.5, an improvement on T_0.

Now the above edge replacement procedure is repeated for T_1. The new tree T_2 is shown in Figure 5(f); its bandwidth is 2. At this point, none of the three edges can be replaced by another edge to improve the tree bandwidth and the algorithm terminates.

4.3 Effectiveness of HMTC algorithm in minimizing stress

It is not hard to see that with our definition of stress (Eq. 4), minimizing stress maximizes bandwidth. In a multicast tree, the overlay link with the lowest bandwidth determines the bandwidth of the tree, i.e., receiver bandwidth. Therefore, to maximize multicast bandwidth, one should minimize the maximum stress in the underlying network.

The *HMTC* algorithm attempts to accomplish exactly this: minimizing maximum link stress of underlying physical network. To analyze the ability of *HMTC* to minimize stress, we *visually* examine low-level link stress. For ease of visual representation, we find it convenient to slightly modify the definition of stress. Recall that the stress of a low-level link e is (number of overlay links mapped to e)/(the capacity of e). The modified definition is essentially *stress ratio*: for e, it is the ratio of stress of e over the minimum stress of all low-level links with at least one overlay link mapped to them, rounded to the nearest integer not greater than it. Stress ratio measures stress levels, 1 being the lowest; for instance, 3 means the links has three times the stress of a link with stress ratio 1. Henceforth we will refer to stress ratio as stress, since the former is only a certain normalized form of the latter.

Visual representation of link stress is accomplished by *stress graphs*. A stress graph is a graph of low-level links that have overlay links mapped to them, and for each link between a pair of nodes (u, v), the number of edges connecting u and v in the stress graph is the stress of link (u, v).

Just to illustrate how *HMTC* incrementally minimizes the maximum link stress after each iteration of the algorithm, we revisit our example in Fig. 5 from Sec. 4.2. In Fig. 6, we show the stress graphs of the trees constructed at each iteration of *HMTC*. It can be seen that the worst link stress progressively decreases as the algorithm progresses.

We shall elaborate more on the relationship between reducing stress and increasing efficiency (i.e., multicast bandwidth) in Sec. 5, where we evaluate the performance, convergence and scalability of the algorithm.

5 Evaluation of Algorithm Performance and Convergence

In this section, we evaluate the performance and convergence of our heuristics algorithm *HMTC* that finds a high-bandwidth multicast tree in an LCC-overlay. Two types of LCCs are considered: complete LCC — the complete set of LCC that expresses all link correlations, and node-based LCC — every constraint contains only links incident to the same node. We also consider optimal (*i.e.*, highest-bandwidth) multicast trees in independent overlays (i.e., without LCC), without using *HMTC*. We compare these three cases: optimal independent tree, *HMTC* with complete-LCC and *HMTC* with node-LCC. To be succinct, we will refer to optimal independent tree as *Independent*, *HMTC* with complete LCC as *Complete-LCC* and *HMTC* with node-based LCC as *Node-LCC*. We will moreover evaluate the convergence and scalability of *HMTC*.

For the purpose of generating realistic network topologies, we use an Internet topology generator, BRITE [19], which is based on power-law degree distributions[2]. Networks of various sizes are generated using BRITE and a fraction of the nodes with the lowest degrees (mostly 1 or 2) are selected to be the overlay nodes. Random distribution is used for assigning bandwidth to low-level links.

2. A seminal paper [20] revealed that degree distribution in the Internet is a power-law.

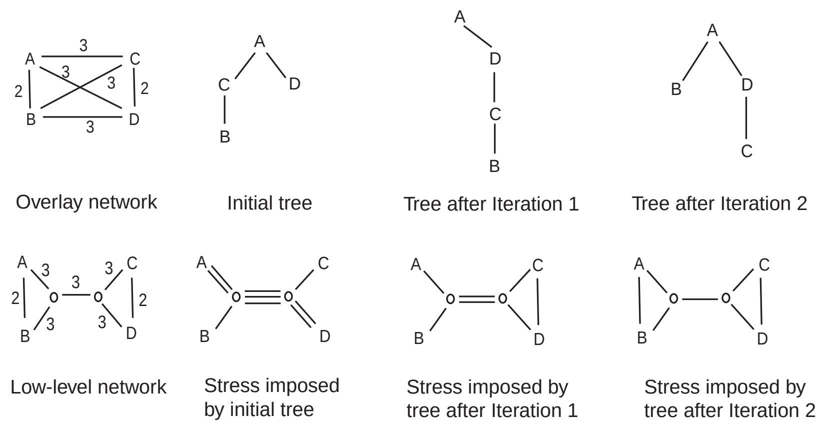

Overlay network
Initial tree
Tree after Iteration 1
Tree after Iteration 2
Low-level network
Stress imposed by initial tree
Stress imposed by tree after Iteration 1
Stress imposed by tree after Iteration 2
A
B
C
D
2
3

Fig. 6. Incremental improvement in stress throughout execution of *HMTC* on the example network. The top four graphs are overlay and the bottom four are stress graphs of the underlying network.

5.1 Performance metrics

For measuring performance and quality, the metrics we use are efficiency, accuracy and stress; their definitions are given previously in Sec. 3.1. For stress, we continue to use stress ratio (and stress graph) for evaluation purposes, as in Sec. 4.3. Recall that the *efficiency* of a tree T measures how close the bandwidth of T is to the optimal, specifically, it is the ratio of achievable bandwidth of T over optimal multicast tree bandwidth. However, as shown in Sec. 4.1, finding optimal-bandwidth multicast trees in graphs with link capacity correlations — *i.e.*, in overlay networks — is NP-complete. Thus to compute the efficiency metric, instead of the exact optimal which is hard to find, we use an *upper bound* of the optimal: the optimal *independent* tree bandwidth.

To summarize, we evaluate the performance of *HMTC* using the following calculations of the efficiency and accuracy metrics:

- Efficiency of high-level tree T is its achievable bandwidth over the optimal independent tree bandwidth.
- Accuracy of T is its achievable bandwidth over its predicted bandwidth.
- Stress ratio of low-level link e is its stress over the minimum nonzero stress of low-level links.

Again, we will be using the terms 'stress' and 'stress ratio' interchangeably.

5.2 Efficiency and Accuracy

We compare the efficiency and accuracy of three types of multicast trees: optimal independent trees (Independent), trees given by *HMTC* with complete LCC (Complete-LCC) and with node-based LCC (Node-LCC).

First, we study the effect of different ratios of overlay size to low-level size, by fixing low-level network size to 300 and varying the percentage of overlay nodes from 10% to 80%.

Efficiency for the three types of trees is plotted against the ratio of overlay to low-level size, in Figure 7(a). The efficiency of Complete-LCC is almost always optimal; this can be attributed to the effectiveness of the heuristics algorithm *HMTC*.

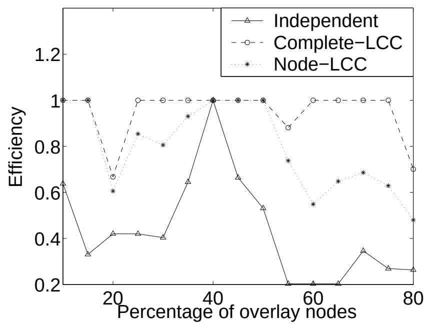
Node-LCC follows Complete-LCC remarkably closely in efficiency for all overlay-to-network ratios of less than 60%. In realistic scenarios, such as in the Internet, overlay nodes are always greatly outnumbered by low-level nodes. Therefore, we can say that for all realistic overlay percentages, Node-LCC is near-optimal.

Efficiency of Independent trees is by far the lowest. Its curve has a narrow peak at 40% overlay nodes and sharply declines on both sides. The shape can be explained. When overlay nodes are few, they have long paths between them, containing more links, hence higher likelihood of link sharing. Yet higher densities of overlay nodes imply more overlay links, therefore more collisions on the same physical links. Both scenarios result in high link correlation and thus lower bandwidth allocated to overlay links. Intuitively, there may exist a middle ground where paths are not so long and overlay links are not so dense, such that the overloading of bottleneck links is minimized. This optimum point seems to be at 40% here.

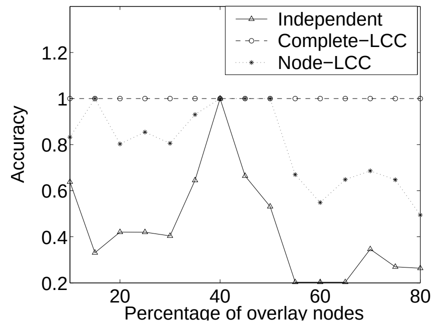
The plot of accuracy versus overlay percentage can be seen in Figure 7(b). Complete-LCC is always perfectly accurate, due to its complete link correlation information. An encouraging observation here is that Node-LCC accuracy is greater than 80% for all networks with less than 60% overlay nodes, i.e., for all realistic overlays. For Independent trees, accuracy is exactly the same as efficiency, with respect to the slightly modified definition of efficiency above in Sec. 5.1.

The next parameter we vary is the total network size, from 100 to 6000 nodes, while keeping the percentage of overlay nodes at a constant 10%, a value which we believe is realistic. Efficiency is shown in Figure 8(a) and accuracy in Figure 8(b).

In this experiment, the efficiency and accuracy graphs are almost indistinguishable. The efficiency of Complete-LCC is again almost always optimal. Node-LCC effi-

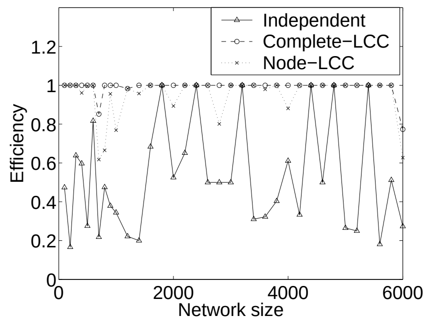


(a) Efficiency versus overlay ratio, low-level size = 300.

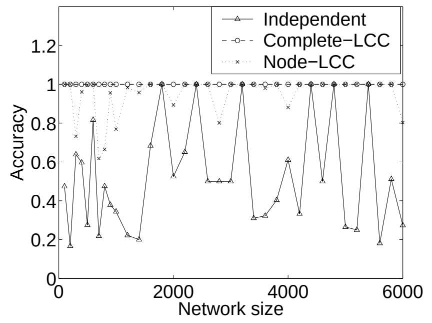


(b) Accuracy versus overlay ratio, low-level size = 300.

Fig. 7. Efficiency and accuracy versus overlay ratio, low-level size = 300.



(a) Efficiency versus network size, 10% overlay nodes.



(b) Accuracy versus network size, 10% overlay nodes.

Fig. 8. Efficiency and accuracy versus network size, 10% overlay nodes.

ciency is also optimal for a vast majority of network sizes, and when it is not, it is within 20% of the Complete-LCC efficiency. The Independent efficiency generally demonstrates poor performance for most network sizes.

It can be concluded from the good performance of both Complete- and Node-LCC that, the heuristics algorithm *HMTC* demonstrates effectiveness in exploiting link correlation information (in the form of LCC) to obtain optimal trees. In contrast, the Independent overlay rarely exhibits good performance because it ignores link correlations, leading to inaccuracy of representing the real network and hence cannot be consistent in finding the optimal tree. The extreme similarity between respective efficiency and accuracy curves implies that it is crucial to strive for accuracy by explicitly taking into consideration link correlations.

5.3 Stress

To further investigate the underlying cause of poor performance of the Independent overlay and the good performance of LCC-overlays, we visually examine low-level link stress — its severity and distribution — imposed by the trees for Independent, Complete-LCC and Node-LCC.

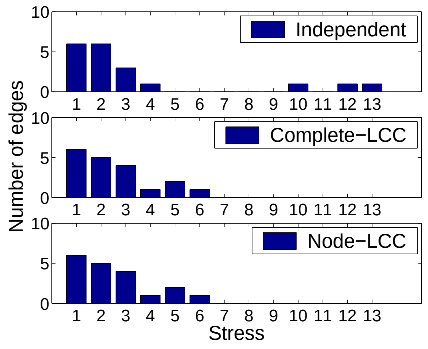
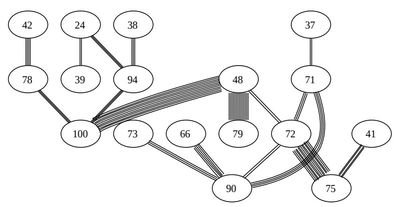


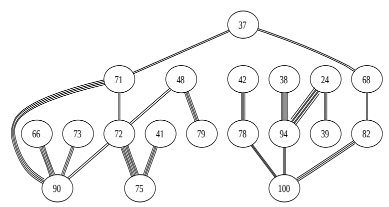
Fig. 10. Distributions of link stress for Independent, Complete-LCC and Node-LCC, network size = 100, 10% overlay nodes.

With a fixed network size and overlay-to-low-level ratio, for each of the three trees (Independent, Complete- and Node-LCC), its constituent (overlay) links are mapped to the low-level and the stress placed by them on all the low-level edges are visually presented in the stress graph. Recall that the stress graph contains only low-level edges with at least one overlay tree link mapped to it. For every low-level edge (u, v) with stress s, in the stress graph, s multiple edges are drawn between node u and node v. Recall that stress as defined above in Sec. 5.1 takes on integer values greater than or equal to 1. In this manner, we are able to visualize link stress at a glance. Higher link stress easily shows in more multiple edges or a thicker line between two nodes.

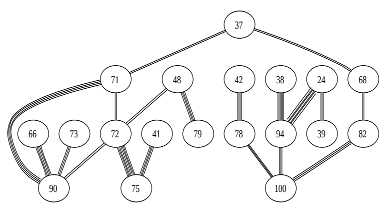
Consider the scenario of network size 100 and overlay ratio 10%; it corresponds to the 100 point on the x-axis



(a) Stress graph for Independent.



(b) Stress graph for Complete-LCC.



(c) Stress graph for Node-LCC.

Fig. 9. Stress graphs, network size = 100.

of the efficiency graph in Figure 8(a). The stress graphs are shown in Fig. 9. The Independent stress graph of Figure 9(a) clearly shows three thick trunks, indicating heavy stress. This is confirmed by the distribution of stress shown in the top graph of Figure 10: three links have higher stress. The stress graphs of Complete-LCC and Node-LCC are in Figure 9(b) and (c), respectively. In this particular scenario, they look the same, containing only relatively thin lines between nodes, *i.e.*, links have relatively low stress. Their distributions of stress, bottom two graphs in Figure. 10, show that the maximum stress is half of that for Independent.

Herein lies the reason for optimal efficiency of Complete-LCC and Node-LCC and for poor efficiency of Independent, as seen in Figure 8(a) at the value of 100 on x-axis (the first point on the curves). The maximum link stress placed by the Independent tree is twice as high as the maximum link stress caused by the two LCC trees. Incidentally, the fact that Independent efficiency is roughly half of that of the two LCC is a convincing indication that our definition of stress is good in representing the load placed on a link relative to its capacity.

We observe that the virtue of *HMTC* with LCC is its success in choosing overlay links that spread load evenly among the low-level links, minimizing stress placed on any single link. The success is prominent even with the much restricted Node-LCC.

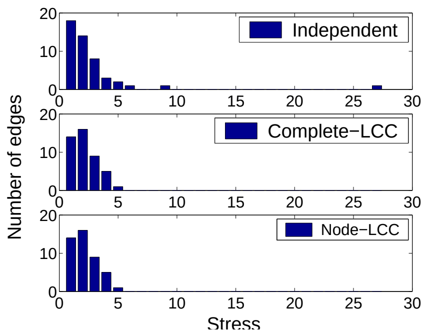


Fig. 12. Distributions of link stress in Independent, Complete-LCC and Node-LCC, network size = 200, 10% overlay nodes.

In a different scenario of 200 network nodes, 10% of which are overlay nodes, the three stress graphs are given in Figure 11. It is obvious that a low-level link in the Independent stress graph, in Figure 11(a), has much higher stress than all the rest — visually, it is an extra thick trunk in the right part of the graph. The distribution of stress in the top graph of Figure 12 shows that there is a link with stress greater than 25, whereas maximum stress for both LCC's is 5. Correspondingly, the Independent efficiency is less than $1/5$ of the efficiency of both Complete-LCC and Node-LCC, as given by the second point on the curves in Figure 8(a).

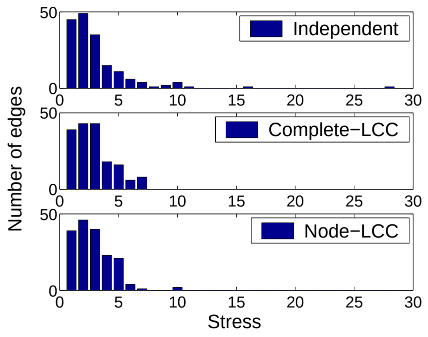
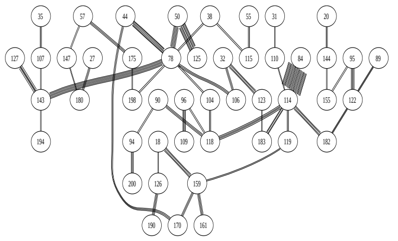


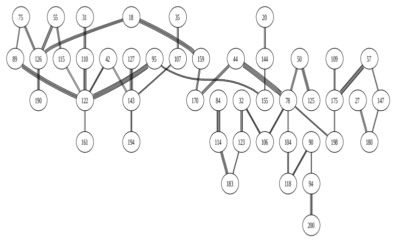
Fig. 14. Distributions of link stress for Independent, Complete-LCC and Node-LCC, network size = 700, 10% overlay nodes.

We examine a third scenario where the efficiency of all three trees are different, where network size is 700 with 10% overlay nodes; it corresponds to the 7th point on the efficiency curves in Figure 8(a). The stress graphs in Figure 13 still show Node-LCC and Complete-LCC to be similar, while stress for Independent is clearly higher in a few links. Figure 14 informs that the highest stress for Node-LCC is between those for Complete-LCC and Independent, leaning more to Complete-LCC — this is reflected in their respective efficiencies.

5.4 Convergence and Scalability

We analyze the convergence of our algorithm *HMTC* in Figure 15(a) (for various overlay ratios) and Figure 15(b) (for various network sizes). For a fixed network of 300 nodes and increasing overlay ratios, the (normalized) tree bandwidth attained is plotted against the number of rounds that has been executed in the algorithm. At each round, a link is selected to be potentially replaced; it is replaced only if another link can be found to improve the bandwidth. This is why there are instances where


(a) Stress graph for Independent.


(b) Stress graph for Complete-LCC.

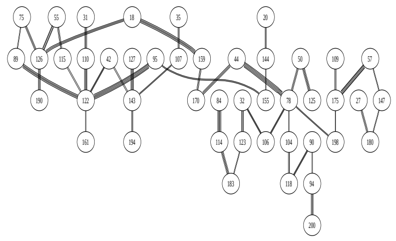
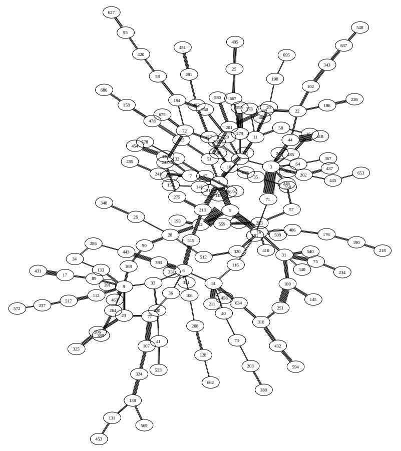
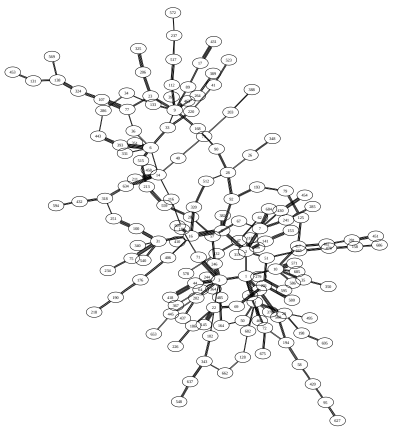

(c) Stress graph for Node-LCC.

Fig. 11. Stress graphs, network size = 200.


(a) Stress graph for Independent.


(b) Stress graph for Complete-LCC.

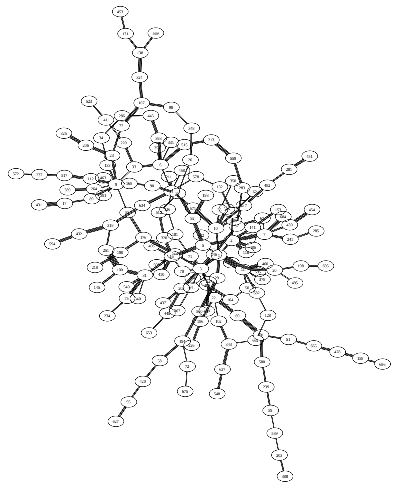

(c) Stress graph for Node-LCC.

Fig. 13. Stress graphs, network size = 700.

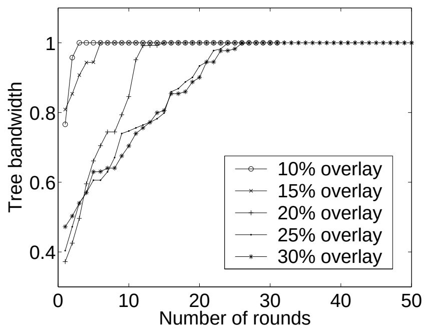
even though the number of rounds has increased, the tree bandwidth does not improve.

It can be seen from the convergence graph that the final bandwidth is always reached in a small number of rounds, for all overlay ratios. Although the number of rounds required to converge increases slightly for higher overlay ratios, it remains small. We do not show the cases of higher overlay ratios to maintain clarity in the graph, but for all ratios up to 80%, the algorithm converges in less than 50 rounds. Scalability of the algorithm is supported by the fact that similar convergence occurs for network sizes up to the high thousands. Moreover, in our simulations, we set the maximum number of rounds to be 100. Hence, the algorithm never runs for more than 100 rounds and still yields the near-optimal efficiency for all network sizes.

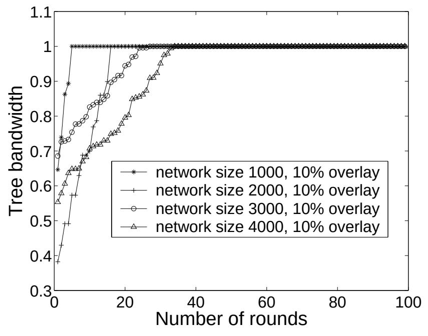
6 Overview of Distributed Algorithm

In this section, we present an overview of the completely *distributed* algorithm we propose that is based on the tree-improving algorithm. The formal treatment of the distributed algorithm is deferred to Sec. 8. Several challenges arise when extending the centralized algorithm to a distributed one to achieve the same objective:

1) Each node can no longer be assumed to possess global knowledge of Complete-LCC.



(a) Convergence of heuristics algorithm for various overlay ratios, low-level size = 300.



(b) Convergence of heuristics algorithm for various network sizes with 10% overlay nodes.

Fig. 15. Convergence of heuristics algorithm, low-level size = 300.

2) Due to lack of global topology information, it is not immediately obvious how to determine the link with the worst bandwidth in the tree. More generally, how does a node decide whether an adjacent link has too low a bandwidth and thus initiate a tree restructure by replacing it?
3) In the centralized algorithm, tree restructuring actions taken by different nodes are synchronized so that no cycles are introduced and hence the tree topology is preserved. However, in the distributed case, nodes initiate tree restructuring *asynchronously*, which may result in the forming of cycles and loss of the tree structure.

6.1 Constructing an LCC-overlay

The first difficulty is resolved by using (inherently decentralized) node-based LCC instead. To be precise, every node has only the LCC that contain links adjacent to itself. We have previously proposed, in [15], a decentralized scheme for constructing an LCC-overlay. In this scheme, every overlay node strategically probes to obtain its own set of node-based LCC. That is, for each node, its set of LCC contains only capacity variables of its incident links. Because this is out of the scope of this paper and also due to space constraint, we do not reproduce the details of our LCC-overlay construction scheme. In brief, it is based on an existing efficient technique for detecting shared bottlenecks, proposed by Katabi et al. in [21], [22]. In our scheme, the node-based LCC are obtained in iterations of increasing refinement. Our simulation results showed that nearly complete node-based LCC can be discovered in a small number of refinement stages.

The second and third problems list above at the beginning of this section are considerably more complicated to solve. We elaborate on how to address them in the remainder of this section as well as the next two sections.

6.2 Gossip-based Node Decision for Tree Restructure

Every node must decide, in a decentralized fashion, whether it should initiate a tree restructuring. The decision procedure should satisfy the following requirements:

- A node decides to restructure to replace an adjacent link only if that link has one of the lowest bandwidths.
- The number of concurrent restructuring nodes should be controlled to be less than a certain percentage, ρ, of the total number of peer nodes.
- Any global statistics used in the decision procedure should be obtainable in a distributed, efficient and scalable way.
- Every node should make the decision entirely on its own, locally.

Now we proceed to present a distributed decision procedure for restructuring which satisfies all of the above requirements. The global statistics needed for the decision are gathered using gossip-based algorithms [23], [24], [25], [26] that estimate aggregate information in large-scale peer-to-peer networks.

GATHERING GLOBAL STATISTICS

Kempe et al. proposed gossip-based algorithms [23] for computation of global aggregate information. In particular, the algorithms are able to estimate sums and quantiles (through random sampling) of numerical elements stored at the nodes. In a network of n nodes with each node holding a value x_i, the gossip-based algorithm can find the sum and quantile of these values. We use the sum estimation to obtain the total number of peers. This can be trivially done by storing at each node the integer 1. The completely decentralized, gossip-based algorithm to compute the total number of nodes is given in Fig. 16; it is a slightly modified version of the algorithm proposed in [23] for computing averages. In [23], it was shown that in at most $O(\log n + \log \frac{1}{\epsilon} + \log \frac{1}{\delta})$, with probability at least $1 - \delta$, the relative error in the estimate is within ϵ.

```
Estimation of Number of Peers
Each node $i$ maintains a sum $s_{t,i}$ and a weight $w_{t,i}$
At round 0: If node $i$ is the root node,
  initialize $s_{0,i} := 1$, $w_{0,i} := 1$. For every other
  node $i$, initialize $s_{0,i} := 1$, $w_{0,i} := 0$.
  Each node $i$ sends the pair $(s_{0,i}, w_{0,i})$ to itself.
At subsequent rounds, each node $i$ does as follows:
1: Let $\{(\hat{s}_r, \hat{w}_r)\}$ be all pairs sent to $i$ in rnd $t-1$
2: Let $s_{t,i} := \sum_r \hat{s}_r, w_{t,i} := \sum_r \hat{w}_r$
3: Choose a target node $u_t(i)$ uniformly at random
4: Send the pair $(\frac{1}{2}s_{t,i}, \frac{1}{2}w_{t,i})$ to $u_t(i)$ and $i$ (itself)
5: $\frac{s_{t,i}}{w_{t,i}}$ is the estimate of the total number of
   peers in round $t$
```

Fig. 16. Decentralized gossip-based algorithm for estimating the total number of nodes.

We use the quantile computation to estimate the lowest and highest link bandwidths in the tree. To do this, each node holds a set M_i of values which are the measured bandwidths of all its adjacent links. The algorithm from [23] essentially finds a value that is close to the largest, with high probability, in a decentralized way. At the start, the entire set of values – i.e, the union of all M_i's – are considered. In each round, a pivot value is chosen from among the remaining values uniformly at random. Then the number of values larger than the pivot is counted, and the algorithm recurses in this subinterval. This estimates the highest link bandwidth; the same algorithm can be used to estimate the lowest link bandwidth. The number of rounds required is $O((\log m + \log \frac{1}{\delta}) \cdot (\log n + \log m + \log\log \frac{1}{\delta}))$, where m is the size of the union of all M_i's, and $1 - \delta$ is the probability of finding the desired value.

The lengths of all messages are bounded by the largest number of bits to encode the values being held at each node, which is $O(\log n + k \log b)$, where n is the total number of nodes, k is the highest number of neighbors a node may have and b is the largest link bandwidth.

This should be multiplied by the number of rounds to obtain the total message overhead, which is the sum of the number of rounds for estimation of number of nodes and estimation of the highest and lowest link bandwidth, as given above.

RESTRUCTURING CANDIDATES

Let B_{min} and B_{max} denote, respectively, the lowest and highest link bandwidth obtained by the gossip algorithms. Each node decides whether it is a candidate for restructuring, by comparing its adjacent link bandwidths with B_{min} and B_{max}. At this stage, the node assumes a uniform distribution of bandwidth bounded by B_{min} and B_{max}. The node considers itself a restructuring candidate if an adjacent link has bandwidth less than $B_{\mathsf{min}}+\rho\cdot(B_{\mathsf{max}}-B_{\mathsf{min}})$, i.e., within ρ percent of the lowest bandwidth.

However, there may be too many candidate nodes, if the bandwith distribution is too skewed from uniformity. To control the number of concurrent restructuring nodes, we must take additional measures.

CONTROLLING PERCENTAGE OF RESTRUCTURING NODES

First, we estimate the number of restructuring candidates. Each node stores 1 if it is a candidate and 0 otherwise. Using the gossip-based algorithm for sums, each node can obtain an estimate of the total number of candidates, N_0.

Let N be the estimated total number of peer nodes. To keep the percentage of restructuring nodes less than ρ, each candidate node simply computes a bias $\beta = \min(\rho \cdot \frac{N}{N_0}, 1)$. Then it tosses a β-biased coin, and only proceeds with restructuring if the outcome is head. It is easy to see that even for large N_0, the maximum number of restructuring nodes is $\beta \cdot N_0 = \rho \cdot N$, as desired.

6.3 Preserving Tree Topology with Asynchronous Tree Restructures

Once a node has determined that it should replace an adjacent link and initiate a tree restructure, it still needs to take measures to avoid introducing cycles, and equivalently, disconnecting the multicast graph.

When a node u decides to initiate a tree restructure, we call u a *restructor* and the subtree of which u is the root *u's subtree*.

After a node u decides to undertake a tree restructure by replacing the link uv, where v is u's current parent in the tree, the procedure of a complete tree restructure is broken down as follows. (i) u selects a neighbor w such that link uw is better than uv. (ii) u disconnects itself and hence its subtree from its old parent v. (iii) u reconnects to the tree by adding the link uw and thus attaching its subtree to its new parent w.

Some thought will reveal the possible causes of cycles forming when different nodes asynchronously decide to restructure the tree. A simple example of a scenario in which cycles can form is illustrated in Fig. 17. Both node B and node C decide to initiate a tree restructure. Node B replaces edge AB with edge EB, i.e., disconnecting from the old parent A and reconnecting to the tree by attaching to the new parent E. Simultaneously (or nearly simultaneously), C changes its parent in the tree from A to D. Unfortunately, because B's tree restructuring makes it a descendant of C, when C's tree restructuring makes C a descendant of B, a cycle is formed (as well as making the "tree" disconnected).

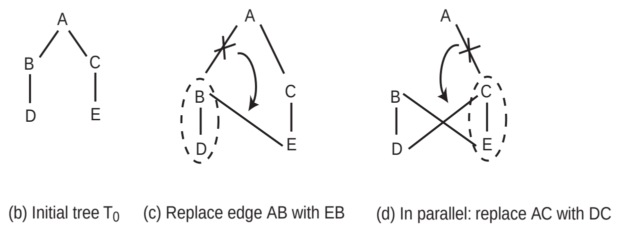

A
B
C
D
E
A
B
C
D
E
A
B
C
D
E
(b) Initial tree T_0
(c) Replace edge AB with EB
(d) In parallel: replace AC with DC

Fig. 17. Example of possible scenario of cycle forming.

It is easy to see that the cycle (and disconnectivity) is caused by B and C both attaching to each other's descendants. However, the (connected tree topology is preserved if only B attaches to C's subtree (or vice versa), or if both attach to respective new parents not in their subtrees.

Generalizing this reasoning leads to the observation that during the tree restructure process of any node u, as long as it does not allow another restructor to attach to its (u's) subtree, no cycle will be formed.

Thus, we enhance the tree restructure procedure with mechanisms to protect against cycles. We give a high-level overview below and defer the formal presentation to the next section.

When a node u initiates a tree restructure (u is a restructor) to replace its adjacent link uv (it is easier to think of v as u's parent, from now on):

(1) u *locks* its subtree.
   - $\Rightarrow$ Nodes in u's subtree is in *locked* mode and cannot accept any requests from other nodes (restructors) to connect to them, i.e., a locked node cannot be the new parent of any restructor.

(2) u selects a target t such that replacing uv with link ut improves tree bandwidth.

(3) u sends a *connect* request to t, and waits for t's reply.
   - If t is locked, due to its ancestor being in the tree restructure process, then t rejects u's request.

(4) Repeat (2) and (3) until u finds a node w that accepts the *connect* request, then u disconnects from v and connects to its new parent w.

(5) u *unlocks* its subtree.
   - $\Rightarrow$ Nodes in u's subtree can now accept requests from restructors to connect to them.

To lock its subtree, u does the following:

- u multicasts a *lock* message to its subtree, i.e., by sending the message to its children who forward it to their children, etc.

- Each descendant (node in subtree) sends a *ready* reply up the subtree, and goes into *lock* mode.
- Once u is notified (via the *ready* replies) that its subtree is locked, it proceeds to select target nodes for connecting to.

The message overhead for each tree restructure consists of $O(n)$ (including all *lock*, *ready* and *unlock* messages) messages transmitted, where n is the number of nodes in the tree. However, only a few bits are needed to encode these messages. The total message overhead is hence $O(n)$ messages times the number of tree restructures. The number of tree restructures required is the number of simultaneously restructuring nodes times the number of iterations for the distributed algorithm to converge to a steady tree throughput. The former is a parameter that can be varied (in the simulations, we vary it from 1.5 to 15 percent); the latter is given in the simulation results in Section 9 and is on the order of 10.

7 State Transition Diagrams

Before we formally present our distributed tree restructuring algorithm (Sec. 8), we first give an explication of state transition diagrams. In particular, concurrent state diagrams are what we use to model the distributed algorithm.

7.1 Single state diagram

The interpretation of state diagrams is completely event based. Each state diagram consists of a finite number of states representing serving as memory of the algorithm. Directed transitional edges are labeled by events. Events are classified into *active* and *passive* events. Active events correspond to function calls and initiation of network requests. Passive events correspond to network requests received from remote peers and conditions resulted from function calls. The semantics of the transitions is that active events can be triggered by the state diagram without delay, while passive events occur spontaneously with nondeterministic delays. Each state can have at most one active event and possible multiple passive events. Mathematically, a state diagram is a unmarked automata: $A = (X, x_0, \Sigma_{\mathsf{act}} \dot{\cup} \Sigma_{\mathsf{pas}}, \delta)$, where $x_0 \in X$ is the initial state, Σ_{act} and Σ_{pas} the active and passive events, respectively. Define, $\Sigma = \Sigma_{\mathsf{act}} \dot{\cup} \Sigma_{\mathsf{pas}}$, the transition function $\delta : X \times \Sigma \to X$ is a partial function [3] mapping a state and a transition (x, e) to the next state $\delta(x, e)$. A *run* is simply a string of events $s \in \Sigma^*$ belonging to the language $\mathcal{L}(A)$ of the automata [27]. We say that an event $e \in \Sigma$ is admissible by A at state x if $(x, e) \in \mathrm{dom}(\delta)$.

7.2 Concurrent state diagrams

[The Synchronous Approach to Designing Reactive Systems]

3. We assume that state diagrams are deterministic automata.

The interpretation of the execution of multiple state diagrams is formalized by synchronous product of automata. A collection of state diagrams can synchronize on *shared* events. For simplicity, suppose that we have two state diagrams $A_i = (X_i, \Sigma_i, \delta_i), i = 1, 2$, which are synchronized on common events in $\Sigma_1 \cap \Sigma_2$. The execution of A_1 is partially affected by A_2. Suppose that A_1 is at state $x_1 \in X_1$ and A_2 at state $x_2 \in X_2$. An event $e \in \Sigma_1$ can be triggered if and only if:

- $e \in \Sigma_1 - \Sigma_2$, i.e. it is not shared by A_2, or
- $e \in \Sigma_1 \cap \Sigma_2$, *and* e admissible by A_2 at state x_2.

In case e is a shared event and is triggered in one state diagram (say A_1), then the other state diagram (A_2) *must* also trigger the transition labelled by e.

8 Tree Restructuring Algorithm

We model the distributed tree restructuring using multiple concurrent state diagrams. Each peer executes three state diagrams: locking , mk-restruct and resp-restruct . The state diagram, mk-restruct , describes the protocol for initiating a connection to remote peers for tree restructuring. The state diagram, resp-restruct , describe the protocol for accepting a restructuring request from a peer. Finally, a third state diagram, locking , is used to describe the locking mechanism used so that a peer does not involve itself in two concurrent restructuring sessions.

8.1 Making restructuring connections

The state diagram mk-restruct is shown in Fig. 18. Every node u executes this state diagram concurrently. We now trace, in detail, the states and transitions between them.

The *initial* state of u is the ready state, shown in the diagram as the top leftmost circle with the entrance arrow going into it. In this state, u is ready to initiate a new tree restructuring.

When the recv-restruct event occurs, i.e., u decides to do a tree restructure, u signals self-busy so that u does not initiate another restructure before this one is completed. Then u locks its subtree by first sending a message to its subtree (snd-lock-st) and waiting for the subtree to be locked (rcv-st-ready).

The next step is to select a target peer node w such that for u to connect to w, the tree bandwidth will improve as a result. Two events may occur to forbid u to connect to w. (1) w is busy — in the process of restructuring itself, or in a subtree locked by an ancestor. (2) w is a descendant of u; u connecting to w would create a cycle.

After the event of connecting successfully to a target peer, u unlocks its subtree (snd-unlock-st) and signals itself to be ready (sig-self-ready) for potentially another tree restructure.

8.2 Responding to restructuring connections

The state diagram resp-restruct is shown in Fig. 19(a). It models the protocol that a node u follows when it

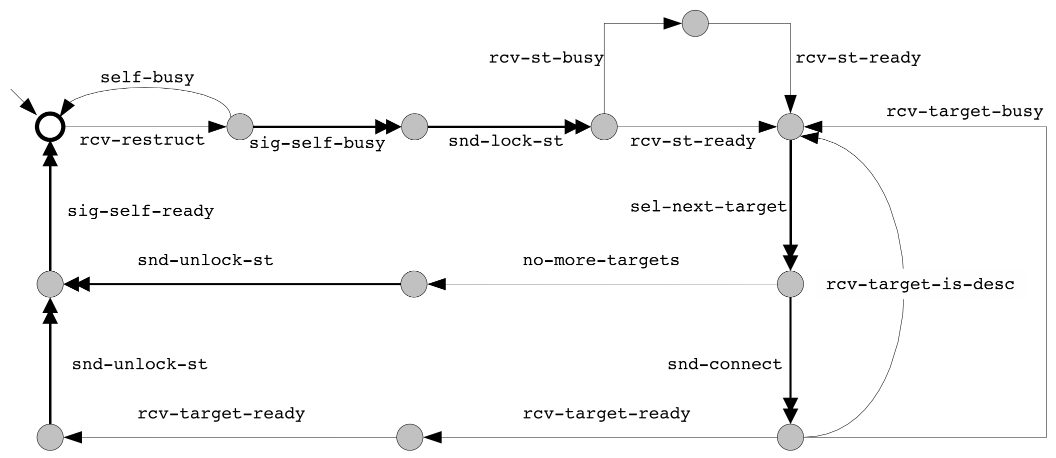


Fig. 18. Protocol for initiating restructuring connections

receives a connect request from a peer, i.e., u is the target node that another node v selects to connect to during v's restructuring. The entry state is the bold circle on the left in the diagram. Note that this entry state could correspond to any state from the other two state diagrams mk-restruct and locking .

At the event of receiving a connect request (rcv-connect) from v, if u is in the self-busy state, then it will reply to v that it is busy snd-target-busy). Otherwise, u checks if it is a descendant of v. This is straightforward: Since v must have previously sent a message to lock its subtree (which includes u), v just piggybacks its node ID on the lock message, and its descendants simply record v's ID. Therefore u can directly check whether it is a descendant of v; if it is, reply accordingly to v (snd-target-is-desc).

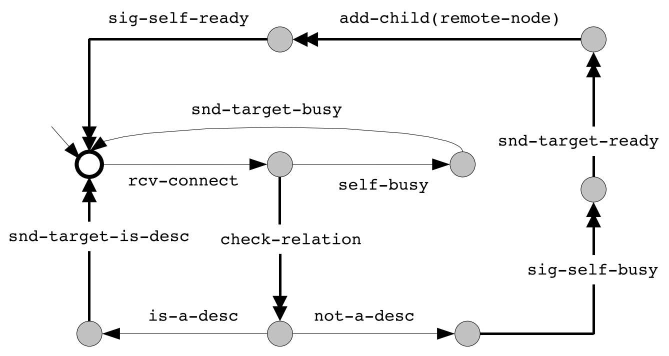
If u is neither busy nor in v's subtree, then u will accept the connection from v and return to the entry state.

8.3 Node lock control

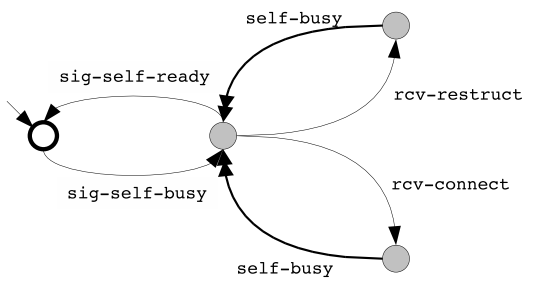
The state diagram locking is shown in Fig. 19(b). It models the protocol for a node being locked by one of its ancestors. At the receipt of a locking message, the node signals itself to be self-busy. In this locked/busy state, any subsequent event of receiving a connect request from a peer (rcv-connect) is rejected. Also, every subsequent event of deciding to initiate a tree restructure (rcv-restruct) is also not implemented, while in the locked/busy state.

9 Simulation Results for Distributed Algorithm

We present here performance evaluation of our distributed tree restructuring algorithm. We implemented, using Java, an event-based simulator to conduct simulation experiments of the algorithm. The network topology is again generated by the *BRITE* topology generator. The overlay network is constructed by simply selecting a subset of nodes with the smallest degrees.



(a) Responding to restructuring requests



(b) Locking node

Fig. 19. State transition diagrams for responding to restructuring and locking node.

In Fig. 20, the convergence rate of the centralized algorithm using Node-LCC is compared with the convergence rate of the distributed algorithm using Node-LCC. For the distributed algorithm, convergence rates are shown for different values of the percentage of restructuring nodes: $1.5\%, 5\%, 10\%$ and 15%. It can be seen from the graph that for the lowest percentage of 1.5% and the highest percentage of 15%, the distributed algorithm does not converge to the steady throughput of the centralized algorithm (but converges to a lower throughput). However, the distributed algorithm does converge to the centralized steady throughput for both 5% and 10%, values in the middle between the low

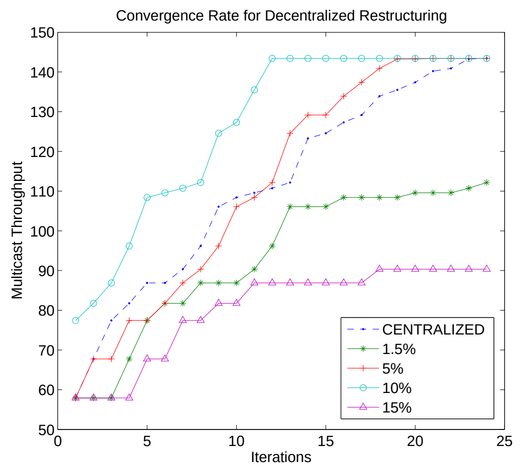


Fig. 20. Convergence rates of distributed restructuring, compared with centralized restructuring.

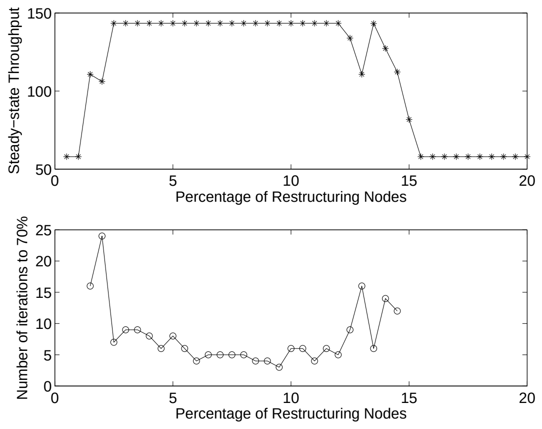


Fig. 21. Convergence rates versus percentage of restructuring nodes.

(1.5%) and the high (15%). For 10%, the distributed convergence is even faster than the centralized. This is a result of the inherent parallelism in the distributed algorithm: Multiple nodes can restructure at the same time, thereby increasing the rate of converging to the steady (highest) throughput.

The objective is clearly to achieve the highest possible steady throughput, therefore it is of interest to investigate further the effect of the percentage of restructuring nodes on the steady throughput. We vary the percentage from 1 to 20 and plot the steady throughput achieved by the distributed algorithm, shown in the top graph in Fig. 21. Clearly, the low and the relatively high percentages do not do as well as those in the middle, e.g., from 3% to 12%. The encouraging observation is that the throughput performance is the best for all the percentages between 3 and 12, that is, the best performance can be consistently achieved for a relatively large range of percentage values. Therefore, percentages can be selected for the distributed algorithm from that range, without concerns of the performance being sensitive to

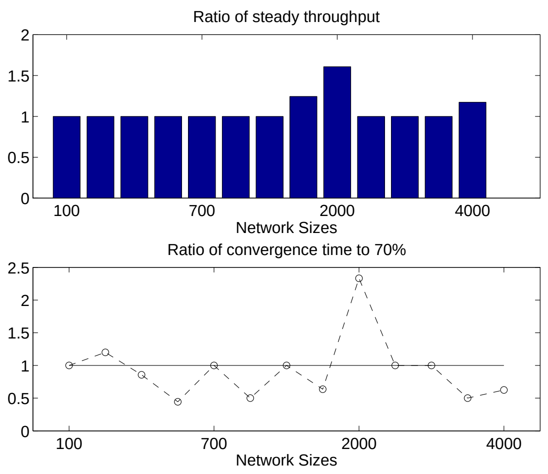


Fig. 22. Ratio of steady throughput versus network size.

small perturbations in the selected parameter value.

The lower graph of Fig. 21 plots the number of iterations needed to reach 70% of the steady-state throughput versus the percentage of restructuring nodes. It demonstrates that for the range of percentages $[3, 12]$, the number of iterations needed is quite low, around 5.

Finally, we experimented with various network sizes, setting the percentage of restructuring nodes to 10%. In the top graph of Fig. 22, the network size is varied from 100 to 4000, and we plot the ratio of distributed steady-state throughput over centralized steady throughput. For all network sizes, the centralized steady throughput is reached by the distributed algorithm. In a few cases (e.g., network size = 2000), the distributed steady throughput even exceeds the centralized. This demonstrates that the convergence and throughput performance of the distributed algorithm is consistently good for all network sizes and is not affected by increasing network sizes.

We compare convergence times of the distributed and centralized algorithms to achieve 70% of the steady-state throughput. The ratio of distributed over centralized convergence time is plotted against the network size. For all but two network sizes, the distributed algorithm converges faster than the centralized algorithm (to 70% of steady-state throughput). The slower convergence at network size of 2000 can probably be explained by the higher distributed steady-state throughput for that size, as seen in the top graph in Fig. 22.

10 Conclusions

In this paper, we have studied the problem of high-bandwidth overlay multicast in overlay networks with linear capacity constraints. We showed that the problem is NP-complete and proposed a heuristics algorithm. Extensive simulations showed that our heuristics algorithm is able to attain near-optimal performance using LCC, even with the restricted node-based LCC. The algorithm also demonstrated fast convergence and scalability. Using node-based LCC, we also developed a fully distributed, scalable, fast-converging algorithm for obtaining multicast trees that have as high steady-state throughput as for the centralized algorithm.

REFERENCES

[1] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-Aware Overlay Construction and Server Selection," in *Proc. of the IEEE INFOCOM*, 2002.

[2] M. Kwon and S. Fahmy, "Path-aware Overlay Multicast," *Computer Networks*, vol. 47, no. 1, pp. 23–45, January 2005.

[3] A. Young, J. Chen, Z. Ma, A. Krishnamurthy, L. Peterson, and R. Wang, "Overlay Mesh Construction Using Interleaved Spanning Trees," in *Proc. of INFOCOM*, 2004.

[4] K. Shen, "Structure Management for Scalable Overlay Service Construction," in *Proc. of NSDI*, 2004.

[5] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang, "A Case for End System Multicast," *IEEE Journal on Selected Areas in Communications*, pp. 1456–1471, October 2002.

[6] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," in *Proc. of ACM SIGCOMM*, August 2002.

[7] X. Zheng, C. Cho, and Y. Xia, "Optimal Peer-to-Peer Technique for Massive Content Distribution," in *Proc. of INFOCOM 2008*, 2008.

[8] J. Byers and J. Considine, "Informed Content Delivery Across Adaptive Overlay Networks," in *Proc. of ACM SIGCOMM*, August 2002.

[9] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh," in *Proc. of ACM SOSP*, 2003.

[10] N. Magharei and R. Rejaie, "PRIME: Peer-to-Peer Receiver-drIven MEsh-based Streaming," in *Proc. of INFOCOM 2007*, 2007.

[11] B. Li et al., "Inside the New Coolstreaming: Principles, Measurements and Performance Implications," in *Proc. of INFOCOM 2008*, 2008.

[12] D. Ren, Y. H. Li, and S. G. Chan, "On Reducing Mesh Delay for Peer-to-Peer Live Streaming," in *Proc. of INFOCOM 2008*, 2008.

[13] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai, "Distributing Streaming Media Content Using Cooperative Networking," in *Proc. of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2002)*, Miami Beach, Florida, May 2002.

[14] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-Bandwidth Multicast in Cooperative Environments," in *Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP 2003)*, October 2003.

[15] Y. Zhu and B. Li, "Overlay Networks with Linear Capacity Constraints," in *Proceedings of the Thirteenth IEEE International Workshop on Quality of Service (IWQoS 2005)*, 2005.

[16] M. Tsang, C. Wang, K. Tsang, and F. Lau, "A Receiver-coordinated Approach for Throughput Aggregation in High Bandwidth Multicast," in *Proc. of INFOCOM 2007*, 2007.

[17] M.S. Kim, Y. Li, and S.S. Lam, "Eliminating Bottlenecks in Overlay Multicast," in *Networking 2005*, 2005.

[18] M.S. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.

[19] A. Medina, A. Lakhina, I. Matta, and J. Byers, *BRITE: Boston University Representative Internet Topology Generator*, http://www.cs.bu.edu/brite.

[20] C. Faloutsos, M. Faloutsos, and P. Faloutsos, "On Power-Law Relationships of the Internet Topology," in *Proc. of ACM SIGCOMM*, August 1999.

[21] D. Katabi and C. Blake, "Inferring Congestion Sharing and Path Characteristics from Packet Interarrival Times," Tech. Rep., Laboratory of Computer Science, Massachusetts Institute of Technology, 2001.

[22] D. Katabi, I. Bazzi, and X. Yang, "A passive approach for detecting shared bottlenecks," in *Proc. of ICCCN '01*, 2001.

[23] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-Based Computation of Aggregate Information," in *Proc. IEEE FOCS*, 2003.

[24] M. Zaharia and S. Keshav, "Gossip-Based Search Selection in Hybrid Peer-to-Peer Networks," *J. Concurrency and Computation: Practice and Experience*, 2007.

[25] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Gossip Algorithms: Design, Analysis and Applications," in *Proc. IEEE Infocom*, 2005.

[26] Dimitrios Psaltoulis, Dionysios Kostoulas, Indranil Gupta, and et al., "Practical algorithms for Size estimation in Large and Dynamic groups," in *http://www.cs.cornell.edu/Info/Projects/Spinglass/public_pdfs/size_estimation.PDF*.

[27] J.E. Hopcroft, R. Motwani, and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation (2nd Edition)*, Addison Wesley, 2000.

Ying Zhu Ying Zhu is an Assistant Professor in the Faculty of Business and Information Technology at the University of Ontario Institute of Technology, Canada. She received her PhD in Computer Engineering at the University of Toronto. She has a MSc in Numerical Analysis also from the University of Toronto and a BSc in Computer Science and Mathematics from Dalhousie University. Her main research interests include performance optimization in distributed systems such as peer-to-peer overlay networks and wireless networks, and pervasive systems involving sensors and RFID technology.

Baochun Li Baochun Li is a Professor in the Department of Electrical and Computer Engineering at the University of Toronto. He received his M.S. and Ph.D. degree in Computer Science from the University of Illinois at Urbana-Champaign, and his B.E. degree in Computer Science from Tsinghua University, China. His research interests are in application-layer algorithms in a wide variety of distributed systems, including peer-to-peer and overlay networks, wireless networks, and sensor networks.

Ken Q. Pu Ken Qian Pu is an Assistant Professor in the Faculty of Science at the University of Ontario Institute of Technology, Canada. He received his PhD in Computer Science at the University of Toronto. He has a MSc in Electrical Engineering and a BSc in Engineering Science also from the University of Toronto. His main research interests are in information management and computer networks. His current focus is on search algorithms in information systems, and distributed processing and dissemination of streamed data.