Near-Optimal Schedules for Simultaneous Multicasts*

Bernhard Haeupler, D Ellis Hershkowitz, David Wajc

Carnegie Mellon University, {haeupler,dhershko,dwajc}@cs.cmu.edu

Abstract

We study the store-and-forward packet routing problem for simultaneous multicasts, in which multiple packets have to be forwarded along given trees as fast as possible.

This is a natural generalization of the seminal work of Leighton, Maggs and Rao, which solved this problem for unicasts, i.e. the case where all trees are paths. They showed the existence of asymptotically optimal $O(C + D)$-length schedules, where the congestion C is the maximum number of packets sent over an edge and the dilation D is the maximum depth of a tree. This improves over the trivial $O(CD)$ length schedules.

We prove a lower bound for multicasts, which shows that there do not always exist schedules of non-trivial length, $o(CD)$. On the positive side, we construct $O(C + D + \log^2 n)$-length schedules in any n-node network. These schedules are near-optimal, since our lower bound shows that this length cannot be improved to $O(C + D) + o(\log n)$.

*Supported in part by NSF grants CCF-1527110, CCF-1618280, CCF-1814603, CCF-1910588, NSF CAREER award CCF-1750808 and a Sloan Research Fellowship.

1 Introduction

We study how to efficiently schedule multiple simultaneous multicasts in the store-and-forward model.

Unicasts and multicasts are two of the most basic and important information dissemination primitives in modern communication networks. In a unicast a source sends information to a receiver and in a multicast a source sends information to several receivers. Typically, many such primitives are run simultaneously, causing these primitives to contend for the same resources, most notably the bandwidth of communication links.

The store-and-forward model has been the classic model for developing a clean theoretical understanding of how to most efficiently schedule many such primitives contending for the same link bandwidth. In the store-and-forward model, a network is modeled as a simple undirected graph $G = (V, E)$ with n nodes. Time proceeds in synchronous rounds during which nodes trade packets. In each round a node can send packets it holds to neighbors in G, but at most one packet is allowed to be sent along an edge in each round. Nodes can copy packets and send duplicate packets to neighbors, again subject to the constraint that at most one packet crosses an edge each round.[1]

The store-and-forward model, in turn, enables a formal definition of the problem of scheduling many simultaneous multicasts or unicasts. A simultaneous multicast instance is given by a set of rooted trees $\mathcal{T}$—one for each multicast—on a store-and-forward network G. Each $T_i \in \mathcal{T}$ has root r_i and leaves L_i along with a packet (a.k.a. message) m_i, initially only known to r_i. A schedule instructs nodes what packets to send in which rounds, subject to the constraint that m_i can only be sent over edges in T_i. The quality of a schedule is its *length*; i.e., the number of rounds until all nodes in L_i have received m_i for every i. A simultaneous unicast instance is the simple case of a simultaneous multicast where all T_i are paths. The goal of past work and this work is to understand the length of the shortest schedule.

The most important parameters in understanding the length of the shortest schedule has been the *congestion* $C = \max_e |\{T_i \ni e\}|$, i.e., the maximum number of packets that need to be routed over any edge in G and the *dilation* $D = \max_i \text{depth}(T_i)$, i.e., the maximum depth of any multicast-tree or the maximum length of any path in the case of simultaneous unicast. It is easy to see that any schedule requires at least $\max(C, D) = \Omega(C + D)$ rounds: a tree with depth D requires at least D rounds to deliver its message and any edge with congestion C requires at least C rounds to forward all packets that need to be sent over it. On the other hand, any instance can easily be scheduled in $O(CD)$ rounds in a greedy manner: in each round and for each edge $e = (u, v)$, forward m_i from u to v where T_i is an arbitrary tree such that $e \in T_i$ and u knows m_i but v does not; it is easy to verify that this schedule takes $O(CD)$ rounds.

Classic results of Leighton, Maggs, and Rao [27] improve upon this trivial $O(CD)$ bound for the case of simultaneous unicast. They showed that introducing a simple independent random delay for each packet at its source suffices to obtain schedules of length $O(C + D \cdot \log n)$ or $O((C + D) \cdot \frac{\log n}{\log \log n})$. A similar strategy can be shown to also work for simultaneous multicasts [13]. More surprisingly, Leighton et al. show how an intricate repeated application of the Lovász Local Lemma [1] proves the existence of length $O(C + D)$ for any simultaneous unicast instance. This seminal paper initiated a long line of followup work [2, 5, 6, 13, 26, 30–32, 34, 35, 39], some of which even showed these $O(C + D)$-length schedules are efficiently computable [28], even by *distributed* algorithms [30, 34].

In contrast, essentially nothing beyond the above trivial $O(CD)$ and simple random delay bounds of $O(C + D \cdot \log n)$ and $O((C + D) \cdot \frac{\log n}{\log \log n})$ is known for simultaneous multicast, despite ample practical

[1]The assumption that nodes can copy and broadcast packets reflects how standard IP routing works in practice [9].

and theoretical motivation. In particular, simultaneous multicast forms an important component of practical content-delivery systems [8, 24], as well as recent theoretical advances in distributed computing [10, 14–20].

The length of the optimal simultaneous multicast schedule is made all the more intriguing by a recent, award-winning work of Ghaffari [13]. This work studied a natural generalization of simultaneous multicast, namely how to schedule many simultaneous distributed algorithms, which corresponds to scheduling the routing of messages on directed acyclic graphs (DAGs). Ghaffari showed that in this setting no $O(C+D)$ schedules exist and, in fact, (up to $O(\log\log n)$ factors) the random delay upper bound of $O(C+D\cdot\log n)$ is the closest that one can get to an $O(C+D)$ bound. Given that multicasts are more general than unicasts but less general than DAGs, it has remained an interesting open question whether an $O(C+D)$ schedule comparable to those for unicasts is also possible for multicasts or whether, like for DAGs, a multiplicative $O(\log n)$ overhead is required.

1.1 Our Contributions

We show that, unlike in the unicast setting where $O(C+D)$ schedules are possible, for multicasts the trivial $O(CD)$ upper bound cannot be improved without introducing a dependence on the number of nodes, n.

Theorem 1.1. *For any positive integers C, D there exists a simultaneous multicast instance with congestion C and dilation D whose optimal schedule requires at least $\frac{CD}{2}$ rounds.*

We note that our lower bound also implies a new lower bound of $\Omega(CD)$ for the DAGs case studied by Ghaffari [13] since the DAGs case generalizes simultaneous multicasts.

On the positive side, we show that if one allows a schedule's length to depend on n then, unlike in the DAGs case where $O(C+D\cdot\log n)$ is the closest one can get to $O(C+D)$, one can get $O(C+D)$ with a mere *additive* $O(\log^2 n)$.

Theorem 1.2. *Each simultaneous multicast instance with congestion C and dilation D in an n-node network admits a schedule of length at most $O(C+D+\log^2 n)$.*

We also verify that these schedules are efficiently computable both by a deterministic, centralized polynomial-time algorithm and by a distributed algorithm in the CONGEST model.

Complementing our proof that shows the existence of $O(C+D+\log^2 n)$ schedules, we extend our lower bound to show that any schedule with purely additive dependence on C, D and any function of n incurs at least an additive $\Omega(\log n)$ term. This implies that the additive $\log^2 n$ in Theorem 1.2 is essentially optimal.

Theorem 1.3. *Suppose there is a function f such that for any simultaneous multicast instance with congestion C and dilation D, there is a schedule delivering all packets in $O(C+D)+f(n)$ steps. Then $f(n)=\Omega(\log n)$.*

In summary, our results give an essentially optimal characterization of what simultaneous multicast schedules are possible and cleanly separate the complexity of simultaneous multicast schedules from those of simultaneous unicasts and DAGs.

1.2 Related Work

We will take this section to summarize additional related work.

Existence of Good Simultaneous Unicast Schedules. The seminal work of Leighton et al. [27] initiated a series of works aimed at showing short simultaneous unicast schedules exist. For example, [31, 36] improved the constants in the $O(C + D)$ schedules of Leighton et al., with [31] also generalizing this result to edges with non-unit transit times and bandwidth. Rothvoß [35] presented a simplified proof compared to that of [27] by way of the "method of conditional expectations", and also increased the constant in the $\Omega(C + D)$ lower bound.

Scheduling Other Simultaneous Algorithms. In addition to the mentioned work of Ghaffari [13], there is a variety of work in scheduling of specific distributed algorithms. A classic result of Topkis [41] shows that h-hop broadcast of k messages from different sources can be done in $O(k + h)$ rounds. This is a special case of simultaneous multicast, where k multicast instances are to be scheduled along edges of trees with congestion $C \leq k$ and depth D. So, for this special case of simultaneous multicast a $O(C + D)$-length schedule always exists. More recently, Holzer and Wattenhofer [21] showed that n BFSs can be performed from different nodes in $O(n)$ rounds. This was generalized by Lenzen and Peleg [29] who showed that k many h-hop BFSs from different sources can be done in $O(k + h)$ rounds.

Algorithmic Results. Another line of work on simultaneous unicast and related problem focused on *computing* optimal or near-optimal schedules efficiently, starting with work of Leighton et al. [28]. There has been work on simultaneous unicast focused on "local-control" or distributed algorithms, where at each step each node makes decisions on which packets to move forward along their paths, based only on the routing information that the packets carry and on the local history of execution. The $O(C + D \cdot \log n)$ algorithm of Leighton et al. [27], for example, is such a distributed simultaneous unicast algorithm. Rabani and Tardos [34] improved this bound to $O(C) + D \cdot (\log^* n)^{O(\log^* n)}$ rounds, which was then further improved by Ostrovsky and Rabani [30] to $O(C + D + \log^{1+\epsilon} n)$ rounds for any constant $\epsilon > 0$. Another series of works also studied centralized algorithms for simultaneous unicast where the source and sink pairs are fixed but the algorithm is free to choose what paths it uses to deliver packets from sources to sinks. Notably, Srinivasan and Teo [39] gave a constant approximation for this problem. Bertsimas and Gamarnik [5] then provided an asymptotically-optimal algorithm, outputting a schedule of length $OPT + (\sqrt{n \cdot OPT})$; i.e., $OPT(1 + o(1))$ for sufficiently large OPT. Lastly, there has been work in computing schedules for single multicasts [4, 11] and even simultaneous multicasts [22, 23] in models fundamentally different from the store-and-forward model we study.

2 Intuition and Overview of Techniques

We now give an overview of and intuition for the techniques we use in our main results.

2.1 $\Omega(CD)$ Lower Bound

The goal of our lower $\Omega(CD)$ lower bound construction is to repeatedly "accumulate" delays by combining together already delayed multicast trees. Here, we build some intuition for this strategy .

Consider a simultaneous multicast instance consisting of two trees S and T using a single edge as in Figure 1. Since at most one message crosses this edge each round, we know that after one round at least one of our trees' messages will be delayed by 1 round, i.e., will not have crossed the edge. More generally, if C trees all use a single edge e then for any fixed schedule one of these trees will require at least C rounds until its message crosses e.

If we knew, a priori, for any C-congested edge which multicast tree was delayed by C, producing a hard multicast instance would be easy as we could repeatedly combine together the multicast trees delayed by C

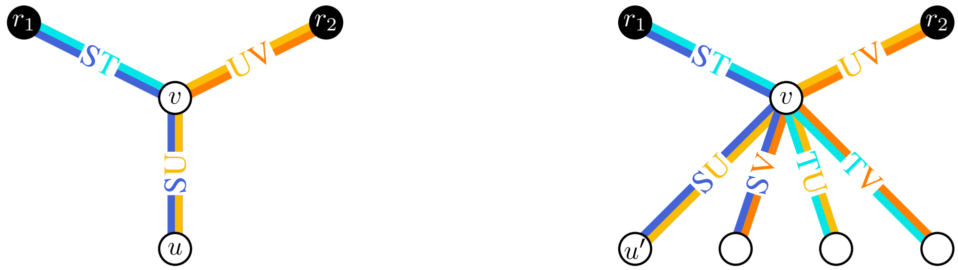
Figure 1: A congested edge example on multicast trees S and T. Root of both trees given by black node. Each multicast tree given in a different color and edges labeled by which multicast trees use them.

in each congested edge. For instance, consider the following example, illustrated in Figure 2a where $C = 2$. We have four multicast trees S, T, U and V where S and T have root r_1 and U and V have root r_2. Both roots connect to a vertex v where (r_1, v) is used by S and T and (r_2, v) is used by U and V. If we knew that after a single round T and V used edges (r_1, v) and (r_2, v) respectively, then we could "combine" S and U into a new edge (v, u). Then, the messages for S and U wouldn't arrive at v until at least two rounds have passed and since both S and U use the edge (v, u), one of the messages of either S or U wouldn't arrive at u until four rounds have passed, despite the fact that u is only two hops from the root of each tree. We might hope, then, to recursively repeat this strategy, combining together such gadgets to accumulate larger and larger delays.

However, we, of course, do not always know which trees are delayed and so combining together the most delayed tree is not a feasible strategy. That is, we must provide a construction which requires many rounds for *every possible* simultaneous multicast schedule, not many rounds for one fixed schedule.

We overcome this challenge by using the fact that trees, unlike paths, branch. In particular, we will use the branching of trees to "guess" which tree was delayed for every congested edge. As a concrete example of this strategy consider the simultaneous multicast instance given in Figure 2b. We have the instance as in Figure 2a but now instead of vertex u, we have four vertices, one for each possible guess of which pair of elements in $\{S, T\} \times \{U, V\}$ are delayed at (r_1, v) and (r_2, v). Now notice that for any fixed simultaneous multicast schedule for this instance we know that after one round only one of S and T's messages will cross (r_1, v) and only one of U and V's message will cross (r_2, v). Without loss of generality suppose S and U do not cross (r_1, v) and (r_2, v) respectively in the first round. We then know that one of the edges (v, u') corresponding to one of our guesses—in this case the edge used by S and U—is such that the trees which use this edge will not deliver the their messages to v until two rounds have passed. Similarly, we know that at most one of S and U's messages arrive at u' by the third round—without loss of generality U's message. Thus, S will not successfully deliver its message to all leaves until at least four rounds have passed, despite the fact that all leaves of S are only two hops from S's root. More generally, if we repeated this construction with a larger congestion C we would have that some multicast tree requires at least $2C$ rounds to deliver its message to all leaves, despite the fact that $C + D = C + 2$.

Our lower bound construction will recursively stack trees like those in Figure 2b to guess which multicast trees a schedule chooses to delay and accumulate a larger and larger delay by combining together these delayed trees. We will guarantee that some sub-graph is always correct in its guesses. We will also make use of the observation that if C trees all use a single edge e then by Markov's inequality at least $\frac{C}{2}$ of these trees will require $\frac{C}{2}$ rounds until their message crosses e to reduce the amount of guessing we must do; this will allow us to expand the possible values of C and D we can use when constructing our lower bound graph which will aid in the proof of Theorem 1.3. We elaborate on our $\Omega(CD)$ construction in Section 3 and then extend it in Section 5 to show that additive $\Omega(\log n)$ is necessary for length $O(C + D)$ schedules.



(a) Accumulating delays if we know S and U delayed. (b) Guessing the delayed trees.

Figure 2: Illustration of how one can "guess" which trees are delayed. Roots given by black nodes. Each multicast tree given in a different color and edges labeled by which multicast trees use them.

2.2 Existence of $O(C + D + \log^2 n)$-Length Simultaneous Multicast Schedules

The main intuition underlying our $O(C + D + \log^2 n)$-length simultaneous multicast schedules is that every instance of multicast can be reduced to a series of unicast instances and, as Leighton et al. [27] showed, unicast instances admit schedules of length linear in their congestion and dilation. Our goal then is to gracefully reduce a simultaneous multicast to a series of simultaneous unicasts.

Here, we discuss two natural approaches for such a reduction, argue that they fail and extract intuition for our upper bound from this failure. In the first approach, for each multicast tree T_i we define $|L_i|$ unicast instances, where for each leaf $l \in L_i$ we have a unicast instance on the root-to-leaf path from r_i to l. While this simultaneous unicast instance has dilation $D' = D$, it also has congestion potentially as high as $C' = \Omega(n)$: unicasts corresponding to the same tree are run independently, and each edge in T_i is contained in every root-to-leaf unicast path. Relying on the existence of schedules of length $O(C' + D')$ guaranteed by [27], then, could yield schedules of length as bad as $\Omega(D + n)$. In the second approach, we define a separate unicast instance for each edge in each T_i. We then run a simultaneous unicast schedule for all edges from roots of multicast trees to their children, then from roots' children to their children, and so on and so forth. Here we have at least obtained a sequence of simultaneous unicast instances with lower dilation—$D' = 1$—and congestion no larger than what we started with—$C' \leq C$. Leighton et al. [27] guarantees the existence of schedules of length $O(C' + D')$ for each such simultaneous unicast instance. Unfortunately, we must concatenate together the schedules of D such simultaneous unicast instances to solve the simultaneous multicast instance, which would yield schedules of length $\Omega(D(C'+D')) = \Omega(CD)$; i.e., no better than the trivial schedule.

Thus, the challenge in reducing simultaneous multicast to simultaneous unicast is finding a suitable way of balancing between these two approaches. In the first reduction, we were able to solve a single simultaneous unicast problem with dilation D but one whose congestion was much larger than the congestion of the simultaneous multicast problem with which we started. In the second extreme, we were able to solve simultaneous unicast instances with dilation and congestion only 1 and C but we had to solve many such problems.

Our goal, then, is to find a way of reducing simultaneous multicast to simultaneous unicast in a way that keeps the dilation and congestion of the resulting simultaneous unicast instances small but does not require solving too many simultaneous unicasts. We strike such a balance by computing what we call a $(\log n, \log n)$-*short path decomposition* of each multicast tree. This decomposition is based on subdividing paths in the heavy path decompositions of Sleator and Tarjan [38]. By using such a decomposition on each multicast tree along with random delays determining when to schedule each path in the decomposition, we obtain a sequence of $\frac{C}{\log n} + \frac{D}{\log n} + \log n$ many simultaneous unicast instance whose congestion C' and dilation D' are both at most $O(\log n)$ with high probability. Relying on the $O(C' + D')$ schedules guaranteed by Leighton et al. [27] for these simultaneous unicast instances, we find that every simultaneous multicast instance admits a schedules of length $O(C + D + \log^2 n)$. We elaborate on this in Section 4. We also provide centralized and distributed algorithms for the computation of these schedules in Appendix B.

3 $\Omega(CD)$ Lower Bound

This section is dedicated to the proof of our $\Omega(CD)$ lower bound. We begin this section by providing the family of instances we use to show this lower bound. We proceed to show how this family requires $\Omega(CD)$ rounds, showing that $O(C + D)$ simultaneous multicast schedules are generally impossible and that the trivial $O(CD)$ schedule is the best simultaneous multicast schedule without a dependence on n. Specifically, we prove the following.

Theorem 1.1. *For any positive integers C, D there exists a simultaneous multicast instance with congestion C and dilation D whose optimal schedule requires at least $\frac{CD}{2}$ rounds.*

3.1 Multicast Instance

We will describe how our instance is constructed in a top-down manner. For the remainder of this section we fix a desired congestion C and dilation D. We will recursively construct a graph in which every edge receives C "labels" where the graph induced by each label is a distinct multicast tree.[2] As each label corresponds to a multicast tree, each label will also have a root corresponding to it which will be the root of the corresponding multicast tree. Ultimately, our instance corresponding to a fixed C and D will contain $C \cdot 2^{D-1}$ multicast trees and so throughout this section we will imagine we have $C \cdot 2^{D-1}$ distinct labels. Throughout this section we will also let capital letters correspond to labels; e.g. $\{S, T, U, V, W, X, Y, Z\}$ is a set of 8 labels. Before moving onto specific details, we refer the reader to Figure 5 for a visual preview of our lower bound construction.

3.1.1 Interleaving Labels

In order to rigorously define what it means to guess which multicast trees are delayed, we introduce the idea of "interleaving" the sets of labels corresponding to our multicast trees.

Given sets S_1 and S_2, each consisting of C labels, we let the interleaving of S_1 and S_2 be $I(S_1, S_2) := \{S'_1 \cup S'_2 : S'_i \subseteq S_i, |S'_i| = C/2\}$ be all subsets which take $C/2$ labels from S_1 and $C/2$ labels from S_2. For example, if $C = 2$ and $S_1 = \{S, T\}$ and $S_2 = \{U, V\}$ then $I(S_1, S_2) = \{\{S, U\}, \{S, V\}, \{T, U\}, \{T, V\}\}$. S_1 and S_2 will correspond to two adjacent edges, each in a disjoint set of C multicast trees each and so $I(S_1, S_2)$ will correspond to all ways of guessing which C trees, taking $C/2$ trees from one edge and $C/2$ trees from the other edge, are delayed among the $2C$ multicast trees which use one of the two edges.

Let $\mathcal{S} = (S_i)_{i=1}^{2^{D-1}}$ be a tuple partitioning our $C \cdot 2^{D-1}$ distinct labels into sets of size C. That is, each S_i is a set (with associated index i) containing C distinct labels and $S_i \cap S_j = \emptyset$ for $i \neq j$. We call two sets in $\mathcal{S}$ adjacent if the index of one is $2i - 1$ and the index of the other is $2i$ for some $i \in \mathbb{Z}_{\geq 1}$. Finally, we let

$$I(\mathcal{S}) := \bigtimes_{i=1}^{|\mathcal{S}|/2} I(S_{2i}, S_{2i+1})$$

[2] The graph induced by a label χ on graph G is the subgraph of edges from G labeled χ.

be all possible interleavings of adjacent sets in $\mathcal{S}$. This tuple $\mathcal{S}$ will correspond to all edges of our construction at height D while a pair of adjacent sets S_{2i-1} and S_{2i} will correspond to two adjacent edges, each in disjoint sets of C multicast trees; thus $\mathcal{I}(\mathcal{S})$ will correspond to all possible ways to guess how, among all pairs of adjacent edges at height D, which C trees, taking $C/2$ trees from one edge and $C/2$ trees from the other edge, are delayed in each pair.

We give a concrete example of our notation where $C = 2$ and $D = 3$. Let $\mathcal{S} = (S_1, S_2, S_3, S_4) = (\{S,T\}, \{U,V\}, \{W,X\}, \{Y,Z\})$. Then $I(\mathcal{S})$ corresponds to all ways of combining S_1 and S_2 by taking one element from each and all ways of combining S_3 and S_4 by taking one element from each. In particular, we have

$$\begin{aligned} I(\mathcal{S}) &= I(S_1, S_2) \times I(S_3, S_4) \\ &= \big\{\{S,U\}, \{S,V\}, \{T,U\}, \{T,V\}\big\} \times \big\{\{W,Y\}, \{W,Z\}, \{X,Y\}, \{X,Z\}\big\}, \end{aligned}$$

which is

$$\begin{aligned} = \big\{&(\{S,U\},\{W,Y\}), && (\{S,U\},\{W,Z\}), && (\{S,U\},\{X,Y\}), && (\{S,U\},\{X,Z\}), \\ &(\{S,V\},\{W,Y\}), && (\{S,V\},\{W,Z\}), && (\{S,V\},\{X,Y\}), && (\{S,V\},\{X,Z\}), \\ &(\{T,U\},\{W,Y\}), && (\{T,U\},\{W,Z\}), && (\{T,U\},\{X,Y\}), && (\{T,U\},\{X,Z\}), \\ &(\{T,V\},\{W,Y\}), && (\{T,V\},\{W,Z\}), && (\{T,V\},\{X,Y\}), && (\{T,V\},\{X,Z\})\big\}. \end{aligned}$$

Notice that each $\mathcal{S}' \in I(\mathcal{S})$ is a tuple of sets, each of C labels, and the number of distinct labels in $\mathcal{S}'$ is precisely half of the labels in $\mathcal{S}$. Each $\mathcal{S}' \in I(\mathcal{S})$ will correspond to a single recursive call in our construction.

3.1.2 Our Instance

With the above notation in hand, we now describe how we recursively construct our multicast instance for a fixed C and D. Let $\mathcal{S} = (S_i)_{i=1}^{2^{D-1}}$ be an arbitrary partition of our $C \cdot 2^{D-1}$ distinct labels into sets of size C as above. Our recursion will be on D; that is, we will recursively construct several instances of simultaneous multicast with dilation $D-1$ and congestion C and then combine together these instances into a single instance with congestion C and dilation D (in fact, every edge will have congestion exactly C and every tree will have depth exactly D). We let $M_{\mathcal{S}}$ be the simultaneous multicast instance we construct from $\mathcal{S}$ and let $G_{\mathcal{S}}$ be its corresponding graph. As can easily be seen by induction on our recursive depth, each root will inductively only be incident on a single edge and so we let $\chi(r)$ be the set of labels of the one edge incident to root r.

- **Base case ($D = 1$):** In this case we have $\mathcal{S} = (S_1)$. We let $M_{\mathcal{S}}$ consist of one edge (r, v) which receives every label in S_1 and let r be the root of every label/tree.
- **Inductive case ($D > 1$):** We construct $M_{\mathcal{S}}$ in three steps. See Figures 3, 4 and 5 for an illustration of the results of steps one, two and three respectively. In the example in these figures $C = 2$, $D = 3$ and $\mathcal{S} = (S_1, S_2, S_3, S_4) = (\{S,T\}, \{U,V\}, \{W,X\}, \{Y,Z\})$; roots of multicast trees are indicated by black nodes and edges are colored according to their label/tree.
    1. First, for each pair of adjacent sets S_{2i-1} and S_{2i} in $\mathcal{S}$ we introduce vertices r_{2i-1}, r_{2i} and v_i and edges $e_{2i-1} = (r_{2i-1}, v_i)$ and $e_{2i} = (r_{2i}, v_i)$. We let r_j be the root of all trees with a label in S_j and e_j receive all labels in S_j for every j (Figure 3).

Figure 3: The result of step one of our lower bound construction. Notice that we have an edge for each set S_i and each pair of adjacent S_i are joined together at a vertex. We outline in red and green v_1 and v_2 respectively. Left-to-right black nodes are r_1, r_2, r_3 and r_4.

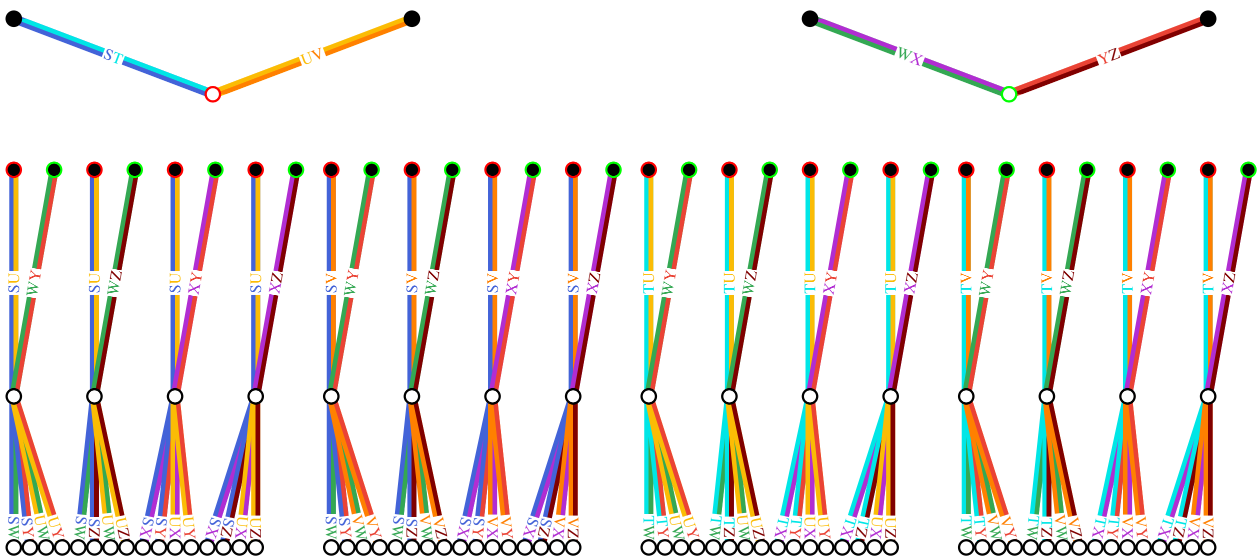


Figure 4: The result of step two of our construction. Notice that we now have a new connected component for each $\mathcal{S}' \in I(\mathcal{S})$, each of which corresponds to a guess for which trees will be delayed at v_1 and v_2. We outline in red and green the vertices which in step three we identify with v_1 and v_2 respectively.

2. Next, we "guess" which trees will be delayed on the e_js. In particular, we add to our graph the disjoint union of $G_{\mathcal{S}'}$ for each $\mathcal{S}' \in \mathcal{I}(\mathcal{S})$; each of the new connected components in our graph at this point corresponds to some instance $M_{\mathcal{S}'}$; each edge inherits the C labels it received in $M_{\mathcal{S}'}$ (Figure 4).
3. Finally, we connect up our guesses to the corresponding parents. In particular, for each vertex r that was a root in $M'_{\mathcal{S}}$ if $\chi(r) \in I(S_{2i-1}, S_{2i})$, we identify r and v_i as the same vertex (Figure 5).

It is easy to verify by induction on our recursive depth that, indeed, each label and its root induce a tree in the returned graph and so $M_{\mathcal{S}}$ is an instance of simultaneous multicast; see Appendix A for a proof.

Lemma 3.1. *Let $\mathcal{S} = (S_i)_i^{2^{D-1}}$ be a partition of $C \cdot 2^{D-1}$ distinct labels as above. Then each label in $\bigcup_i S_i$ induces a rooted tree in $G_{\mathcal{S}}$.*

3.2 Proof of $\Omega(CD)$ Lower Bound

An induction on D demonstrates that our simultaneous multicast instance has the appropriate congestion and dilation. We defer a proof of this simple lemma to Appendix A.

Lemma 3.2. *$M_{\mathcal{S}}$ has congestion C and dilation D.*

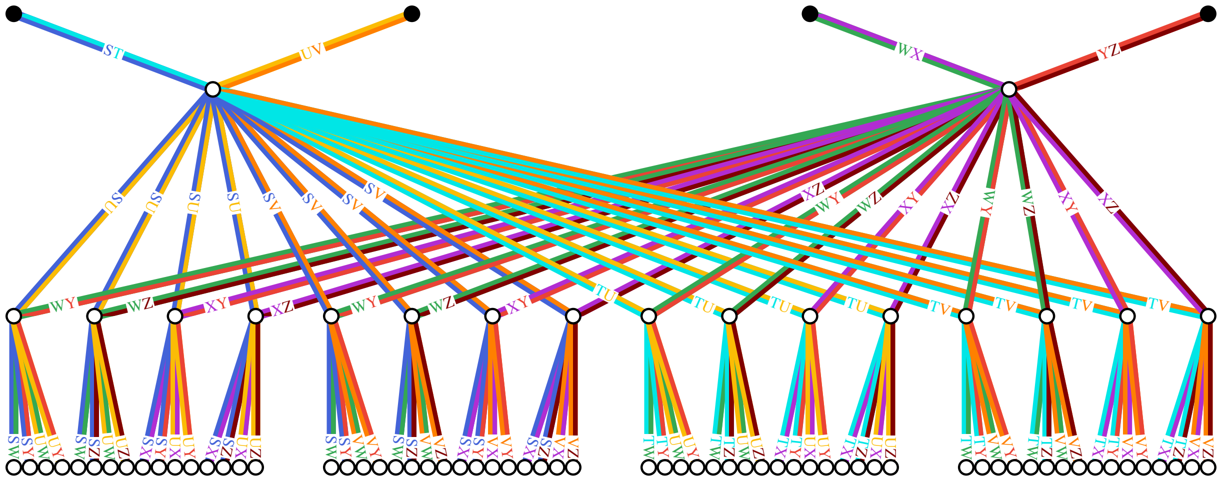


Figure 5: Our construction (i.e. the result of step three) for $\mathcal{S} = (S_1, S_2, S_3, S_4) = (\{S,T\},\{U,V\},\{W,X\},\{Y,Z\})$, $C = 2$ and $D = 3$. Notice that we have modified the graph in Figure 4 by adding one edge for each S_{2i-1} (resp. S_{2i}) colored by the labels in S_{2i-1} (resp. S_{2i}) going from root r_{2i-1} (resp. r_{2i}) to v_i.

Having established the basic properties of our instance, we now argue that (asymptotically) the best one can hope for on our instance is the trivial $O(CD)$-round schedule. As discussed in Section 2.1, we will prove this by arguing that for any fixed schedule, some sub-graph in G was correct in "guessing" which multicast trees were slowed down. In particular, we will argue that for any fixed schedule there is some smaller instance of simultaneous multicast which this schedule must solve as a sub-problem which takes at least $\frac{C(D-1)}{2}$ rounds but which the schedule does not start making progress towards solving until at least $\frac{C}{2}$ rounds have passed.

Lemma 3.3. *The optimal schedule on $M_{\mathcal{S}}$ is of length at least $\frac{CD}{2}$.*

Proof. Fix an arbitrary simultaneous multicast schedule. We will prove by induction on D that $M_{\mathcal{S}}$ requires at least $\frac{CD}{2}$ rounds. The base case of $D = 1$ is trivial, as in this case $M_{\mathcal{S}}$ is a single edge with congestion C and so clearly requires at least $C \geq \frac{CD}{2}$ rounds.

For the inductive step, $D > 1$, suppose that for any partition $\mathcal{S}'$ of $C \cdot 2^{D-1}$ distinct labels into sets of size C, we have that $M_{\mathcal{S}'}$ requires at least $\frac{C(D-1)}{2}$ rounds. By definition of $M_{\mathcal{S}}$, any schedule which solves $M_{\mathcal{S}}$ can be projected in the natural way onto $M_{\mathcal{S}'}$ as a schedule which solves $M_{\mathcal{S}'}$ for any $\mathcal{S}' \in I(\mathcal{S})$. For example, any schedule which solves the instance in Figure 5 induces a schedule which when projected onto Figure 4 solves $M_{\mathcal{S}'}$ for each of the recursively constructed $M_{\mathcal{S}'}$. Even stronger, notice that $M_{\mathcal{S}}$ is created by combining the union of all $M_{\mathcal{S}'}$ for $\mathcal{S}' \in I(\mathcal{S})$ in such a way that any schedule which solves $M_{\mathcal{S}}$ must also send all messages from roots of trees in $M_{\mathcal{S}}$ to corresponding roots of trees in $M_{\mathcal{S}'}$ and solve $M_{\mathcal{S}'}$. That is, let r' be an arbitrary root for $M_{\mathcal{S}'}$ and let $\chi(r')$ be the labels associated with the one edge for root r' in $M_{\mathcal{S}'}$. Then, if we identify r' with v_i when constructing $M_{\mathcal{S}}$ then a schedule for $M_{\mathcal{S}}$ must both send r' the C messages of $\chi(r')$ from r_{2i-1} and r_{2i} and solve $M_{\mathcal{S}'}$. Thus, clearly the time our schedule takes is at least the time it takes to send one message in $\chi(r')$ to r' from r_{2i-1} and r_{2i} for some root in $M_{\mathcal{S}'}$ plus the time it takes to solve $M_{\mathcal{S}'}$. For example, if we let $\mathcal{S}' = (\{S,U\},\{W,Y\})$ then any schedule which solves Figure 5 solves $M_{\mathcal{S}'}$ but before doing so must clearly send at least one message of $\{S,U\}$ to v_1 or at least one message $\{W,Y\}$ to v_2.

We now define $\bar{\mathcal{S}}'$ where $\bar{\mathcal{S}}' \in I(\mathcal{S})$ so that $M_{\bar{\mathcal{S}}'}$ is an instance of simultaneous multicast embedded in $M_{\mathcal{S}}$ which our fixed schedule must solve in order to solve $M_{\mathcal{S}}$ but which it does not start start solving until at least $C/2$ rounds have passed. In particular, consider what our fixed schedule does in the first $C/2$ rounds. As C messages must cross each edge e_j and only one such message can cross per rounds, there are some $C/2$ multicast trees whose message cannot cross e_j before $C/2$ rounds have passed; let S'_j be these "slow" trees for edge e_j and let

$$\bar{\mathcal{S}}' := (S'_{2i-1} \cup S'_{2i})_{i=1}^{2^{D-2}}$$

be a partition of the labels corresponding to these slow edges into sets of size C.

The tuple $\bar{S}'$ belongs to $I(\mathcal{S})$ and so, as discussed above, the fixed schedule must first send at least one message to a root in $M_{\bar{\mathcal{S}}'}$ and then solve $M_{\bar{\mathcal{S}}'}$. By definition of $\bar{S}'$, no messages arrive at roots of trees in $M_{\bar{\mathcal{S}}'}$ until at least $\frac{C}{2}$ rounds have passed. On the other hand, by the inductive hypothesis, the latter sub-instance takes at least $\frac{C(D-1)}{2}$ additional rounds. Thus, the schedule must use at least $\frac{C(D-1)}{2} + \frac{C}{2} = \frac{CD}{2}$ rounds. □

Combining Lemmas 3.2 and 3.3 immediately yields Theorem 1.1.

4 Existence of $O(C + D + \log^2 n)$-Length Schedules

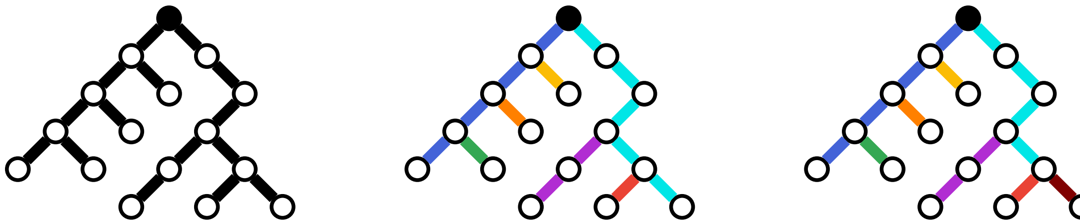
Here we demonstrate that length $O(C+D+\log^2 n)$ simultaneous multicast schedules always exist. For this result we rely on heavy path decompositions, first introduced by Sleator and Tarjan [38].

Defintion 4.1 (Heavy path decomposition [38])**.** *A* heavy path decomposition *of a rooted tree T is obtained as follows. First, each non-leaf node selects one* heavy *edge, which is an edge to a child with the greatest number of descendants (breaking ties arbitrarily). Other edges are termed* light. *We consider inclusion-wise maximal paths consisting of heavy edges, and for each highest node v of such a path p, we add to the path p the edge from v to its ancestor (if any). The obtained paths form the heavy path decomposition.*

It is easy to see that this is indeed a decomposition of the tree; that is, that each edge belongs to exactly one path in the heavy path decomposition. Moreover, each root-to-leaf path intersects at most $\log_2 n$ heavy paths, as each such path can have at most $\log_2 n$ light edges because the number of nodes in a subtree decreases by at least a factor of two every time one traverses down a light edge. This will allow us to decompose the trees into "short paths" such that each root-to-leaf path intersects few short paths. Specifically, we define a refinement of this decomposition in a top-down fashion, by breaking up each heavy path into short paths of length at most $\log_2 n$; that is, starting from the top of a heavy path of length l, we cut it into $\lceil l/\log n \rceil$ short paths. See Figure 6. Both the decomposition and its refinement exist, and are even computable deterministically in linear time.

As each root-to-leaf path intersects at most $\log_2 n$ heavy paths, this refined decomposition has each root-to-leaf path intersect at most $\frac{D}{\log_2 n} + \log_2 n$ short paths. We will refer to such a decomposition as a $(\log n, \log n)$-short (path) decomposition. We use this particular name as we generalize this notion further in Appendix B to (l,k)-decompositions for any integers k and l. This refined path decomposition together with some additional random delays will allow us to reduce the task of simultaneous multicast to that of $O\left(\frac{C}{\log n} + \frac{D}{\log n} + \log n\right)$ many simultaneous unicast instances with congestion and dilation $O(\log n)$, from which we obtain the following result. We illustrate the schedules in this result in Figure 7.

Theorem 1.2. *Each simultaneous multicast instance with congestion C and dilation D in an n-node network admits a schedule of length at most $O(C + D + \log^2 n)$.*



(a) Multicast tree T_i. (b) Heavy-light decomposition. (c) Cut heavy paths into short paths.

Figure 6: Our decomposition for multicast tree T_i. Each heavy path in Figure 6b and short path in Figure 6c drawn in different colors. Notice the far right path is cut into two short paths since its length is $5 > \log_2 n = 4$.

Proof. We prove this result by means of the probabilistic method. First, we consider a $(\log n, \log n)$ decomposition of each multicast tree. For each short path p in the $(\log n, \log n)$-decomposition of a tree, we say p is at *level* j if there are exactly $j-1$ other short paths between p's root and the tree's root. That is, if we were to schedule a particular tree by forwarding along all paths of level $j = 1, 2, \ldots$ during $R = O(\log n)$ rounds, the path of level j would be scheduled in rounds $((j-1)\cdot R, j \cdot R]$, which we refer to as the j-th *frame*. Our goal will be to schedule the sets of short paths with limited congestion in parallel, using simultaneous unicast schedules guaranteed by Leighton et al. [27].

In order to break up simultaneous multicast to multiple simultaneous unicasts, we shift the levels of each tree T_i by a random offset X_{T_i} chosen uniformly in $[C/\log n]$. Now a short path of level j in tree T_i will be scheduled during frame $j + X_{T_i}$. Since each edge e has congestion C, the expected number of paths of different trees that use e during any given frame is at most $O(\log n)$. So, by standard Chernoff concentration inequalities, the congestion of each edge during any frame is at most $O(\log n)$ w.h.p. Therefore, applying a union bound over all edges and time frames, we find that w.h.p., all edges have congestion at most $O(\log n)$ for all (shifted) frames $j = 1, 2, \ldots, \frac{C}{\log n} + \frac{D}{\log n} + \log n$ (recall that each root-to-leaf path intersects at most $\frac{D}{\log n} + \log n$ paths of length at most $\log n$). In particular, there exist random delays such that each time frame consists of a simultaneous unicast instance with congestion $C' = O(\log n)$ and dilation $D' = O(\log n)$. Therefore, by Leighton et al. [27, 28], there exists a schedule of length $O(C' + D') = O(\log n)$ for these time frames' simultaneous unicasts. Combining these schedules one time frame after another, we obtain a schedule of length

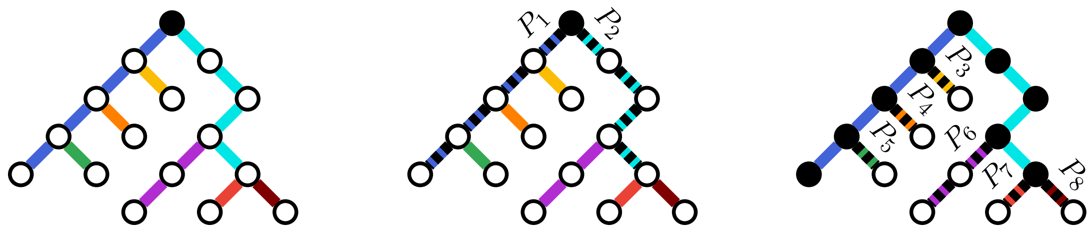
$$\left(\frac{C+D}{\log n} + \log n\right) \cdot O(\log n) = O(C + D + \log^2 n). \qquad \square$$

The above proof can be made algorithmic, deterministic, and even allows for efficient distributed algorithms. See Appendix B for details.

5 Additive $\Omega(\log n)$ Necessary

In this section we use our $\Omega(CD)$ lower bound to show that any simultaneous multicast bound of the form $O(C + D) + f(n)$ must have $f(n) = \Omega(\log n)$. That is, we show the following.

Theorem 1.3. *Suppose there is a function f such that for any simultaneous multicast instance with congestion C and dilation D, there is a schedule delivering all packets in $O(C + D) + f(n)$ steps. Then $f(n) = \Omega(\log n)$.*



(a) Wait for T_i's random delay. (b) Run LMR on P_1, P_2. (c) Run LMR on $P_3, \ldots, P_8$.

Figure 7: Our multicast schedule on T_i using the decomposition from Figure 6. Nodes with m_i colored in black. Short unicast paths are dashed in black. "LMR" is the schedule given by Leighton et al. [27].

This result demonstrates the near optimality of the length $O(C + D + \log^2 n)$ schedules we gave in the previous section. For the remainder of this section we let G be our lower bound graph as defined in Section 3.1.

Recall that our lower bound graph $G_{\mathcal{S}}$ is constructed with respect to a particular C and D. A simple induction on D and standard approximations allows us to bound the number of nodes in our lower bound graph; see Appendix A for a proof.

Lemma 5.1. $|V(G_{\mathcal{S}})| \leq 2^{C(2^D)+D}$.

This lemma, along with Theorem 1.1, allows us to conclude Theorem 1.3.

Proof of Theorem 1.3. Assume for the sake of contradiction that every simultaneous multicast instance admitted a schedule of length $\alpha(C + D) + f(n)$ for constant α and $f(n) = o(\log n)$.

Let $D = 4\alpha$ and let $C = \frac{\log n}{2^{4\alpha+1}}$ for n to be fixed later. Consider the simultaneous multicast instance given by $M_{\mathcal{S}}$ as defined in Section 3. Notice that $C2^{D+1} = \frac{\log n}{2^{4\alpha+1}} 2^{4\alpha+1} = \log n$. Thus, by Lemma 5.1 we know that $|V(G_{\mathcal{S}})| \leq 2^{C(2^D)+D} \leq 2^{C2^{D+1}} \leq n$; by adding dummy isolated nodes we can make our instance have exactly n nodes.

Furthermore, by Theorem 1.1 we have that the optimal schedule of this simultaneous multicast instance with congestion C, dilation D and n nodes has length at least

$$L := \frac{CD}{2} = \frac{\alpha}{2^{4\alpha}} \log n.$$

But, by our assumption for contradiction we have that this instance admits a schedule of length at most

$$U := \alpha(C + D) + f(n) = \frac{\alpha}{2^{4\alpha+1}} \log n + 4\alpha^2 + f(n).$$

We then have a contradiction because $U < L$. In particular for n sufficiently large,

$$U - L = -\frac{\alpha}{2^{4\alpha+1}} \log n + 4\alpha^2 + o(\log n) < 0. \qquad \square$$

6 Future Directions

We conclude our paper with future directions for work in the scheduling of simultaneous multicasts. Of course, one can try and tighten the polylogarithmic additive terms in our results. More interestingly, one could extend the simultaneous multicast setting in ways similar to how the simultaneous unicast scheduling work of Leighton et al. [27, 28] has been extended.

We give two notable examples. First, one could study what sort of approximation algorithms are possible if one is permitted to choose the trees over which multicast is performed as was done in the simultaneous unicast setting [5, 26, 39]. Roughly speaking, this corresponds to a depth-bounded version of the multicast congestion problem [7, 25, 42]. We point out that choices of trees with optimal congestion + dilation (or nearly-optimal, up to constant multiplicative and additive polylogarithmic terms) combined with our algorithm to output length $O(C + D + \log^2 n)$-length schedules would imply near-optimal simultaneous multicasts for this setting. Second, we note that our schedules have logarithmic-sized edge queues. That is, messages may have to wait up to $\Theta(\log n)$ rounds before being sent over an edge. This is not due to our use of the schedules of Leighton et al. [27], whose queue sizes are constant, but rather due to $\Theta(\log n)$ messages arriving to a node by the end of simultaneous unicast frames used in our schedules. An interesting open question is whether there exist efficient simultaneous multicast schedules which minimize both time and edges' queue sizes.

A Deferred (Lower Bound) Content of Section 3

Lemma 3.1. *Let $\mathcal{S} = (S_i)_i^{2^{D-1}}$ be a partition of $C \cdot 2^{D-1}$ distinct labels as above. Then each label in $\bigcup_i S_i$ induces a rooted tree in $G_{\mathcal{S}}$.*

Proof. We prove this by induction on D. Let T_χ be the graph induced by label χ and let r_χ be χ's root. We will prove the slightly stronger claim that T_χ is a tree containing r_χ and for any two labels $\chi \neq \chi'$ we have $r_\chi \in T_{\chi'}$ only if $r_\chi = r_{\chi'}$. Call this latter property $\circledast$.

As a base case suppose that $D = 1$. We then have that $\mathcal{S} = (S_1)$ is a single set of labels and so by definition of $M_{\mathcal{S}}$ we have that our graph will consist of a single edge (r, v) where r is the root for every label and (r, v) is labeled by every label in S_1. Since each label induces the edge (r, v) where r is the root for this label, clearly every label induces a rooted tree containing its root. Moreover, $\circledast$ holds since every label has the same root.

As an inductive hypothesis suppose that for any $D' < D$ and $\mathcal{S}'$ of size $C \cdot 2^{D'-1}$, we have that every label in $M_{\mathcal{S}'}$ induces a tree containing the label's root and all induced trees in $M_{\mathcal{S}'}$ satisfy $\circledast$. Thus, our inductive hypothesis tells us that every label in $M_{\mathcal{S}'}$ for $\mathcal{S}' \in I(\mathcal{S})$ induces a tree containing the label's root where all labels satisfy $\circledast$.

We will first verify that each T_χ is a tree containing r_χ. Clearly, since the edge leaving r_χ is labeled χ, we have that $r_\chi \in T\chi$. Let $\mathcal{T}'_\chi$ be all trees induced by label χ in $M'_{\mathcal{S}}$ for $\mathcal{S}' \in I(\mathcal{S})$. $M_{\mathcal{S}}$ is created by taking the disjoint union of $G_{\mathcal{S}'}$ for $\mathcal{S}' \in I(\mathcal{S})$, identifying several roots of $M_{\mathcal{S}'}$ trees and then adding new roots and edges. Notice that for any χ this identifying of nodes as the same nodes does not cause any cycles in a T_χ since by $\circledast$ we identify exactly one node from each tree in $\mathcal{T}'_\chi$ with another node. Next, to see that $\circledast$ still holds notice that if a node is designated a root in $M_{\mathcal{S}}$ then it is incident to a single edge and is a root for every label this edge was assigned. □

Lemma 3.2. *$M_{\mathcal{S}}$ has congestion C and dilation D.*

Proof. As each edge receives C labels in our construction, each of which corresponds to a multicast tree, clearly the congestion is C. For the dilation, we prove by induction on D. As a base case notice that if D is 1 then $|\mathcal{S}|$ is 1 and so G consists of a single edge used by all all trees, giving a dilation of 1. Suppose that for $D' < D$ we have that the dilation of $M_{\mathcal{S}'}$ is D' where $|\mathcal{S}'| = C \cdot 2^{D-1}$. The claim follows by simply noticing that each tree in $M_{\mathcal{S}}$ extends the root of every tree in $M_{\mathcal{S}'}$ by 1 edge. □

Lemma 5.1. $|V(G_{\mathcal{S}})| \leq 2^{C(2^D)+D}$.

Proof. Clearly, to upper bound the total number of vertices it suffices to upper bound the total number of edges introduced. Thus, we will count the number of edges introduced at each level of our recursion.

Fix C. Define $m_D := |E(G_{\mathcal{S}})|$. We claim by induction on D that $m_D \leq 2^{C(2^D)+D}$. As a base case notice that when $D = 1$ we have $m_1 = 1 \leq 2^{C(2^D)+D}$. For our inductive step consider $G_{\mathcal{S}}$. $G_{\mathcal{S}}$ is constructed by introducing 2^{D-1} edges and unioning together $G_{\mathcal{S}'}$ for $\mathcal{S}' \in I(\mathcal{S})$ of which there are $\binom{C}{C/2}^{2^{D-1}}$, each of which have m_{D-1} edges. Thus, we have

$$\begin{aligned}
m_D &= 2^{D-1} + \binom{C}{C/2}^{2^{D-1}} m_{D-1} \\
&\leq 2^{D-1} + 2^{C2^{D-1}} 2^{C2^{D-1}+D-1} && \left(\text{By } \binom{C}{C/2} \leq 2^C \text{ and inductive hypothesis}\right) \\
&\leq 2^{D-1} + 2^{2C2^D+D-1} \\
&\leq 2^{2C2^D+D} && \left(\text{By } 2^{D-1} \leq 2^{2C2^D+D-1}\right) \qquad \square
\end{aligned}$$

B Algorithmic Results

In this section we present centralized and distributed algorithms for the computation of simultaneous multicast schedules of length $O(C + D + \log^2 n)$, as guaranteed to exist by Theorem 1.2.

B.1 Centralized Algorithm

It is easy to see that the probabilistic method proof in Theorem 1.2 yields a randomized algorithm which succeeds with high probability. Moreover, by standard limited independence methods [37], one can make this algorithm deterministic.

Theorem B.1. *There exists a deterministic, centralized algorithm which, given a simultaneous multicast instance, outputs a schedule of length $O(C + D + \log^2 n)$ in time polynomial in $|\mathcal{T}|$ and n.*

Proof. Let us begin by explaining why the proof of Theorem 1.2 immediately yields a polynomial-time randomized algorithm which succeeds with high probability. Recall that the schedules in Theorem 1.2 were produced by taking a $(\log n, \log n)$-short decomposition of each T_i, delaying each T_i by $X_{T_i} \sim [C/\log n]$ and then concatenating together unicast schedules given by Leighton et al. [27]. As noted in Section 4, a $(\log n, \log n)$-short decomposition can be computed deterministically in polynomial (in fact, linear) time. Clearly, drawing a random delay from $[C/\log n]$ for each T_i is also doable in polynomial time by a randomized algorithm. Lastly, by Leighton et al. [28], the schedules of Leighton et al. [27] can be computed deterministically in polynomial time. By Theorem 1.2 the resulting schedule is of the appropriate length.

Let us now explain how this algorithm can be made deterministic. Let $n' = n + |\mathcal{T}|$. The only randomization used in the above algorithm is the random delays drawn from $[C/\log n]$. As with most proofs that show concentration by Chernoff bounds, it is easy to see that each X_{T_i} need only be $\frac{1}{\text{poly} n'}$-approximate, $O(\log n')$-wise independent for the above algorithm to succeed with high probability in n'. Recalling that one can generate polynomially-many binary $\frac{1}{\text{poly} n'}$-approximate $O(\log n')$-wise independent random variables with only $O(\log n')$ random bits, our deterministic algorithm can simply brute force over all possible assignments to these $O(\log n')$ bits, and check if each resulting schedule is of the appropriate length. The result is a deterministic algorithm which is polynomial-time in $|\mathcal{T}|$ and n and outputs a schedule of the appropriate length. For more background on limited independence, see Schmidt et al. [37]. $\square$

B.2 Distributed Algorithm

In this section we give our distributed simultaneous multicast algorithm in the CONGEST model.

In the classic CONGEST model of distributed communication [33], a network is modeled as an undirected simple n-node graph $G = (V, E)$. Communication is conducted over discrete, synchronous rounds. During each round each node can send an $O(\log n)$-bit message along each of its incident edges. Every node has an arbitrary and unique ID of $O(\log n)$ bits, first only known to itself (this is the KT_0 model of Awerbuch et al. [3]).

In the CONGEST model in a simultaneous multicast instance, each node initially knows a unique ID associated with each tree T_i to which it belongs, as well as which of its incident edges occur in which trees. We think of m_i in this setting as being an $O(\log n)$-bit message, which is therefore transmittable along an edge in a single round. As in the centralized version of the problem, initially only r_i knows m_i.

Our main result for this section is as follows.

Theorem B.2. *For any constant $\epsilon > 0$, there exists a* CONGEST *algorithm which given access to shared randomness solves simultaneous multicast in time*

$$O\left((C + D) \cdot \left(1 + \frac{\log \min\{C, D\}}{\log \log n}\right) + \log^{2+\epsilon} n\right).$$

with high probability. If nodes also know their depth in each tree, then there exists another CONGEST *algorithm which solves simultaneous multicast in $O(C + D + \log^{2+\epsilon})$ time.*

As noted above, simultaneous multicast has proven to be a crucial subroutine in many recent algorithms in CONGEST for fundamental problems like MST, shortest path and approximate min cut. Therefore, improving simultaneous multicast in the CONGEST model is an important step towards obtaining better algorithms for many of these fundamental problems. Furthermore, in the above applications of simultaneous multicast the parameters C and D are polylogarithmic in n provided the input graph has certain structure such as being planar [15–18]. If C and D are sufficiently large polylogarithmic terms, i.e., $\max\{C, D\} = \Omega(\log^{2+\epsilon} n)$, then, assuming nodes know their heights, our distributed algorithm gives an optimal $O(C + D)$ time distributed algorithm. Thus, we view our distributed algorithm as an important step towards obtaining better algorithms for many distributed problems, including MST, shortest path and approximate minimum cut.

Before, proceeding, let us discuss the preprocessing assumptions in Theorem B.2. Our distributed algorithms assume nodes have access to shared randomness or to their height in each of their incident multicast trees. Both of these assumptions can be dispensed with provided nodes are allowed to do some preprocessing: see Ghaffari [12] for how to share randomness and note that nodes can compute their heights by a single simultaneous multicast computation where we could, for e.g. use the aforementioned $O(C + D \log n)$ length schedules. If this preprocessing is performed only once and many simultaneous multicasts are performed, its cost amortizes away. Furthermore, the assumption of shared randomness is a common assumption for distributed algorithms—see for e.g. Ghaffari [12] where it was assumed for multiple broadcasts and Ghaffari [13] where it was assumed for simultaneous scheduling of general distributed programs. Thus, provided nodes share randomness we have that after a preprocessing step equivalent to the current state of the art distributed simultaneous multicast algorithm, subsequent simultaneous multicasts can be performed in time $O(C+D+\log^{2+\epsilon} n)$, which as discussed earlier, is essentially as close as one can get to a bound of $O(C+D)$.

B.2.1 Intuition and Overview

Before moving on, we will provide an intuition for and an overview of our results. As mentioned earlier, Ostrovsky and Rabani [30] provided a distributed algorithm for simultaneous unicast using $O(C + D + \log^{1+\epsilon} n)$ rounds. Since our centralized algorithm has shown that simultaneous multicast can be reduced to simultaneous unicast by way of a $(\log n, \log n)$-short decomposition, the focus of our distributed algorithm is the efficient distributed computation of a $(\log n, \log n)$-short decomposition.

The challenge of computing such a decomposition in a distributed manner is that it seems as hard as solving simultaneous multicast. In particular, computing a heavy path decomposition requires that every node in a T_i aggregate information from all of its children. It is not hard to see that performing such a "convergecast" at every node can be seen as performing a multicast on every T_i in reverse. Even worse, the message size sent by nodes to their parents in such a convergecast to compute a heavy path decomposition must consist of $\log_2 n$ bits to count the size of their sub-tree; i.e. sending just one such message fully uses the bandwidth of a CONGEST link in one round. Thus, it seems that if we want to solve simultaneous multicast by using a $(\log n, \log n)$-short decomposition, then we must circularly solve a simultaneous convergecast—i.e. simultaneous multicast in reverse—in which large messages must be sent.

However, we show that, in fact, one can compute what is essentially a $(\log n, \log n)$-short decomposition more efficiently than one can solve simultaneous multicast. In particular, we show how to efficiently compute what we call a $(\log^{1+\epsilon} n, \log n)$-short decomposition. We demonstrate that a $(\log^{1+\epsilon} n, \log n)$-short decomposition for every T_i can be efficiently computed in a distributed fashion by using a "rank-decomposition" rather than a heavy path decomposition. Computing a rank-decomposition will require nodes to send exponentially fewer bits to their parents than computing a heavy path decomposition. By exploiting this exponential decrease in the total number of bits that must be passed, we are able to efficiently pack rank information into sent messages and compute a $(\log^{1+\epsilon} n, \log n)$-short decomposition. We are then able to translate our centralized algorithm to the distributed setting by making use of the distributed simultaneous unicast algorithms of Ostrovsky and Rabani [30].

B.2.2 $(\log^{1+\epsilon} n, \log n)$-Short Decomposition Using a Rank Decomposition

The decomposition which we compute will not be exactly identical to those of our centralized algorithms and so we generalize these decompositions as follows.

Defintion B.3. *For any integers k and ℓ, we say a path decomposition of a tree of depth D is* (ℓ, k)-short *if each root-to-leaf path in the tree intersects at most $D/\ell + k$ paths of the decomposition.*

We note that some trees do not admit an $(O(\log n), k)$-short path decomposition with $k = \omega(\log_2 n)$—for example, it is easy to see that a complete binary tree on n nodes admits no such decomposition.

We now define our rank-based decompositions as follows; to our knowledge the notion of rank we use here first appeared in the union find data structure [40].

Defintion B.4 (Rank-based path decomposition)**.** *A* rank-based path decomposition *of a rooted tree T is obtained in a bottom-up fashion, as follows. Each leaf v has rank zero; i.e., $rank(v) = 0$. Each internal node v with children set $child(v)$ has rank*

$$rank(v) = \begin{cases} \max_{u \in child(v)}\{rank(u)\} & |\arg\max_{u \in child(v)}\{rank(u)\}| = 1 \\ \max_{u \in child(v)}\{rank(u)\} + 1 & else. \end{cases}$$

Each non-leaf node selects one preferred *edge, which is an edge to a child of highest rank (breaking ties arbitrarily). We consider inclusion-wise maximal paths consisting of preferred edges, and for each highest* *node v of such a path p, we add to the path p the edge from v to its ancestor (if any). The obtained paths form the rank-based path decomposition.*

As with heavy path decompositions, the above is clearly a path decomposition of the tree. Moreover, for this decomposition, too, each root-to-leaf path intersects at most $\log_2 n$ paths of the decomposition, due to the following simple observation, which follows by induction on the nodes' heights.

Observation B.5. *Each node v of rank i has at least 2^i descendants.*

Another consequence of Observation B.5 is that no node has rank greater than $\log_2 n$. This in particular implies that nodes can send their rank information using only $\log \log n$ bits. This will prove useful when trying to compute these rank-based decomposition, by relying on random offsetting and appropriate bit-packing.

Lemma B.6. *For any $\epsilon \geq 0$, there exists a* CONGEST *algorithm which, given a simultaneous multicast instance and shared randomness, computes a $(\log^{1+\epsilon} n, \log n)$-short path decomposition of the multicast trees with high probability in time*

$$O\left((C+D)\cdot\left(1+\frac{\log\min\{C,D\}}{\log\log n}\right)\right).$$

Proof. We will first aggregate information needed to compute ranks, from which we compute a rank-based path decomposition. We then refine this decomposition to obtain the required $(\log^{1+\epsilon} n, \log n)$-short path decomposition.

Our algorithm proceeds in time frames of $O\big(1+\frac{\log\min\{C,D\}}{\log\log n}\big)$ rounds each (to be specified below). During each time frame some nodes send messages in some of their trees to their parents in those tree, as follows. First, each tree T has its leaves begin transmitting at some time frame X_T, where X_T is a random integer in the range $[C]$. Internal nodes transmit once they have received messages from all of their children. Whenever node u transmits to its parent v in a tree T during some time frame, u sends its rank in this tree. In addition, u also it transmits some additional information which allows v to map this rank information to the appropriate tree, as follows. If $C \leq D$, then u transmits the of tree T among the (at most C) trees that contain edge (u, v), using only $O(\log C)$ bits. Otherwise, u also transmits the height of u in T, denoted by $h_T(u)$. To see how the latter information allows v to determine the ID of T, we note that a node v receives a message from its child u in tree T in time frame $X_T + h_T(u)$, where $h_T(u)$ is the height of u in T. Therefore, as v knows X_T and receives $h_T(u)$, then if a single tree transmits along (u, v) during that time frame, v knows precisely which tree this is. To avoid ambiguity due to several trees $T_1, T_2, \ldots$ having the same value of $X_T + h_T(u)$, the node u sends its messages of $(rank_T(u), h_T(u))$ sorted by the IDs of T. Recalling that u transmits to its parent v in tree T during time frame $X_T + h_T(u)$, and that X_T is chosen uniformly in $[C]$, we find that each edge has messages sent up it by at most one tree in expectation at any given time frame. Moreover, by standard concentration inequalities and union bound, there are at most $O\big(\frac{\log n}{\log\log n}\big)$ trees that use any given edge during during any time frame, w.h.p. Therefore, as we can send $O(\log n)$ bits along any edge in one round, the $(\log\log n + \log\min\{C,D\})$-sized messages of all trees using this edge during that time frame can be sent (w.h.p.) in time frames of length $O\big(1+\frac{\log D}{\log\log n}\big)$ rounds. As we use $O\big(\frac{C}{\log^{1+\epsilon} n}+\frac{D}{\log^{1+\epsilon} n}+\log n\big)$ many time frames, this rank aggregation step takes $O\left((C+D)\cdot\left(1+\frac{\log\min\{C,D\}}{\log\log n}\right)\right)$ rounds.

To obtain a rank-based path decomposition from this rank information, we spend a further $O(C)$ rounds after the ranks are computed, as follows. For each tree T, each node v in T waits X_T rounds, after which it notifies each of its children u in T whether the edges (u, v) is v's preferred edge in T. As before, w.h.p., each edge (u, v) has only $O\left(\frac{\log n}{\log\log n}\right)$ trees T for which this (single-bit of) information needs to be sent, which can be performed in a single round (with the bits for each tree sorted by the ID of the tree). Therefore, these C rounds suffice to compute a rank-based decomposition. The overall claimed running time follows.

Finally, we compute the desired $(\log^{1+\epsilon} n, \log n)$-short decomposition by refining the rank-based decomposition in a top-down manner, as follows. As before, we take time frames of length $O\left(1 + \frac{\log\min\{C,D\}}{\log\log n}\right)$. This time, we have nodes transmit information downwards to their children, in reverse order relatively to when they received information from their children (this allows the children to determine to what tree each message corresponds). Specifically, nodes send information corresponding to the length of a short path (i.e., of length at most $\log^{1+\epsilon} n$), which will form part of the $(\log^{1+\epsilon} n, \log n)$-short decomposition. In addition, nodes send the relevant tree T's ID, or their depth in T. As with our bottom-up subroutine, either message allows to determine T (if we also sort the messages by the tree's IDs). The information corresponding to the length of a short path is zero if this edge is not preferred, or more generally if it is the first edge of a path of length $\log^{1+\epsilon} n$ in our more $(\log^{1+\epsilon} n, \log n)$-short path decomposition. When a node v in tree T receives such length information ℓ, it sends $\ell + 1 \mod \log^{1+\epsilon} n$ to its child in T. This information can be encoded using $O(\log\log n)$ bits. Consequently, as for the computation of ranks, the relevant $O\big(\frac{\log n}{\log\log n}\big)$ messages sent along any edge at any time frame (w.h.p.) can be sent during a time frame of $O\big(\frac{\log n}{\log\log n}\big) \cdot (\log\log n + \log\min\{C, D\}) = O\big(1 + \frac{\log\min\{C,D\}}{\log\log n}\big)$ rounds. Therefore, as here too we use $O\big(\frac{C}{\log^{1+\epsilon} n} + \frac{D}{\log^{1+\epsilon} n} + \log n\big)$ many time frames, the desired $(\log^{1+\epsilon} n, \log n)$-short path decomposition is computed in the claimed $O\left((C + D) \cdot \left(1 + \frac{\log\min\{C,D\}}{\log\log n}\right)\right)$ rounds. □

B.2.3 Using Our $(\log^{1+\epsilon} n, \log n)$-Short Decompositions to Solve Simultaneous Multicast

Leveraging the distributed algorithm of Ostrovsky and Rabani [30] together with our $(\log^{1+\epsilon}, \log n)$-short path decomposition of all trees, we immediately obtain a distributed schedule with similar running time to that of the algorithm in Lemma B.6. Partitioning the trees further into subtrees of polylogarithmic depth, we even obtain a near $O(C + D)$ bound, provided nodes know their height in each tree. Concluding, we have the following theorem which gives the properties of our distributed algorithm.

Theorem B.2. *For any constant $\epsilon > 0$, there exists a* CONGEST *algorithm which given access to shared randomness solves simultaneous multicast in time*

$$O\left((C + D) \cdot \left(1 + \frac{\log\min\{C, D\}}{\log\log n}\right) + \log^{2+\epsilon} n\right).$$

with high probability. If nodes also know their depth in each tree, then there exists another CONGEST *algorithm which solves simultaneous multicast in $O(C + D + \log^{2+\epsilon})$ time.*

Proof. The first bound follows rather directly from Lemma B.6 and our remark following of Section 4, whereby any $(\log^{1+\epsilon} n, \log n)$-short decomposition of the trees of a simultaneous multicast instance allows us to compute by a local-control algorithm (specifically, the algorithm of Ostrovsky and Rabani [30]), a schedule of length $C + D + \log^{2+\epsilon} n$. Since the time to compute the short path decomposition takes $O\left((C + D) \cdot \left(1 + \frac{\log\min\{C,D\}}{\log\log n}\right)\right)$, the bound follows.

Now, suppose all nodes know their depth in each multicast tree they belong to. We perform the multicasts of time frames of length $O(\log^{2+\epsilon} n)$, as follows. For each tree, we divide the tree into subtrees, and in particular consider a partition of each tree into ranges of $L \triangleq \log^{2+\epsilon} n$ consecutive levels. Note that there are at most $D/L+1$ such levels per tree. Each tree T will choose a random delay of X_T time frames, chosen uniformly in $[\lceil C/L \rceil]$. Once the time frame X_T arrives, we will transmit down the first L levels of the tree. In the next time frame we transmit this information down the following L levels, and so on and so forth. All such transmissions during a time frame are an instance of simultaneous multicast, but what are its congestion and dilation? The dilation here is trivially $D' = L$, by choice of levels. As for the congestion, since each edge belongs to C trees and each tree delays the its transmissions by $X_T \sim_R [\lceil C/L \rceil]$ time frames, the congestion for each edge during any time frame is $O(L)$ in expectation and w.h.p. (since $L = \log^{2+\epsilon} n) = \Omega(\log n)$). Therefore, by the first bound of this theorem, and since $\log D' = O(\log\log n)$, each such time frame's simultaneous multicast instance can be scheduled in time

$$O\left((C'+D')\cdot\left(1+\frac{\log\min\{C',D'\}}{\log\log n}\right)+\log^{2+\epsilon} n\right) = O(\log^{2+\epsilon} n) = O(L).$$

Our algorithm runs for $\lceil C/L \rceil + D/L + 1$ time frames (also accounting for the random delays). This simultaneous multicast algorithm therefore takes time at most

$$\left(\frac{C}{L}+\frac{D}{L}+2\right)\cdot O(L) = O(C+D+L) = O(C+D+\log^{2+\epsilon} n). \qquad \square$$

References

[1] Noga Alon and Joel H Spencer. *The probabilistic method.* John Wiley & Sons, 2016.

[2] Friedhelm Meyer auf der Heide and Berthold Vöcking. Shortest-path routing in arbitrary networks. *Journal of Algorithms*, 31(1):105–131, 1999.

[3] Baruch Awerbuch, Oded Goldreich, Ronen Vainish, and David Peleg. A trade-off between information and communication in broadcast protocols. *Journal of the ACM (JACM)*, 37(2):238–256, 1990.

[4] Amotz Bar-Noy, Sudipto Guha, Joseph Naor, and Baruch Schieber. Message multicasting in heterogeneous networks. *SIAM Journal on Computing (SICOMP)*, 30(2):347–358, 2000.

[5] Dimitris Bertsimas and David Gamarnik. Asymptotically optimal algorithms for job shop scheduling and packet routing. *Journal of Algorithms*, 33(2):296–318, 1999.

[6] Costas Busch, Malik Magdon-Ismail, Marios Mavronicolas, and Paul Spirakis. Direct routing: Algorithms and complexity. In *Proceedings of the 12th Annual European Symposium on Algorithms (ESA)*, pages 134–145, 2004.

[7] Robert Carr and Santosh Vempala. Randomized metarounding. *Random Structures & Algorithms*, 20 (3):343–352, 2002.

[8] Yang Chu, Sanjay Rao, Srinivasan Seshan, and Hui Zhang. Enabling conferencing applications on the internet using an overlay muilticast architecture. *ACM SIGCOMM computer communication review*, 31(4):55–67, 2001.

[9] Stephen E Deering. Host extensions for IP multicasting. 1988.

[10] Michal Dory and Mohsen Ghaffari. Improved distributed approximations for minimum-weight two-edge-connected spanning subgraph. In *Proceedings of the 38th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 521–530, 2019.

[11] Michael Elkin and Guy Kortsarz. Sublogarithmic approximation for telephone multicast: path out of jungle. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 76–85, 2003.

[12] Mohsen Ghaffari. Distributed broadcast revisited: Towards universal optimality. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 638–649. Springer, 2015.

[13] Mohsen Ghaffari. Near-optimal scheduling of distributed algorithms. In *Proceedings of the 34th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 3–12, 2015.

[14] Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks i: Planar embedding. In *Proceedings of the 35th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 29–38, 2016.

[15] Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks II: Low-congestion shortcuts, mst, and min-cut. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 202–219, 2016.

[16] Mohsen Ghaffari and Jason Li. New distributed algorithms in almost mixing time via transformations from parallel algorithms. In *Proceedings of the 32ndInternational Symposium on Distributed Computing (DISC)*, pages 31:1–31:16, 2018.

[17] Bernhard Haeupler, Taisuke Izumi, and Goran Zuzic. Low-congestion shortcuts without embedding. In *Proceedings of the 35th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 451–460, 2016.

[18] Bernhard Haeupler, Taisuke Izumi, and Goran Zuzic. Near-optimal low-congestion shortcuts on bounded parameter graphs. In *Proceedings of the 30thInternational Symposium on Distributed Computing (DISC)*, pages 158–172, 2016.

[19] Bernhard Haeupler, D Ellis Hershkowitz, and David Wajc. Round-and message-optimal distributed graph algorithms. In *Proceedings of the 37th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 119–128, 2018.

[20] Bernhard Haeupler, Jason Li, and Goran Zuzic. Minor excluded network families admit fast distributed algorithms. In *Proceedings of the 37th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 465–474, 2018.

[21] Stephan Holzer and Roger Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *Proceedings of the 31st ACM Symposium on Principles of Distributed Computing (PODC)*, pages 355–364, 2012.

[22] Jennifer Iglesias, Rajmohan Rajaraman, R Ravi, and Ravi Sundaram. Rumors across radio, wireless, telephone. In *35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2015.

[23] Jennifer Iglesias, Rajmohan Rajaraman, R Ravi, and Ravi Sundaram. Plane gossip: Approximating rumor spread in planar graphs. In *Proceedings of the 13th Latin American Theoretical Informatics Symposium (LATIN)*, pages 611–624, 2018.

[24] John Jannotti, David K Gifford, Kirk L Johnson, M Frans Kaashoek, et al. Overcast: reliable multicasting with on overlay network. In *Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation-Volume 4*, page 14, 2000.

[25] K Jansen and H Zhang. An approximation algorithm for the multicast congestion problem via minimum steiner trees. In *Proceedings of the 3rd International Workshop on Approximation and Randomized Algorithms in Communication Networks*, pages 77–90, 2002.

[26] Ronald Koch, Britta Peis, Martin Skutella, and Andreas Wiese. Real-time message routing and scheduling. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 217–230. 2009.

[27] Thomson Leighton, Bruce M Maggs, and Satish B Rao. Packet routing and job-shop scheduling in O(congestion+ dilation) steps. *Combinatorica*, 14(2):167–186, 1994.

[28] Tom Leighton, Bruce Maggs, and Andrea W Richa. Fast algorithms for finding o (congestion+ dilation) packet routing schedules. *Combinatorica*, 19(3):375–401, 1999.

[29] Christoph Lenzen and David Peleg. Efficient distributed source detection with limited bandwidth. In *Proceedings of the 32nd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 375–382, 2013.

[30] Rafail Ostrovsky and Yuval Rabani. Universal $O(congestion+dilation+\log^{1+\epsilon} n)$ local control packet switching algorithms. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, volume 29, pages 644–653, 1997.

[31] Britta Peis and Andreas Wiese. Universal packet routing with arbitrary bandwidths and transit times. In *Proceedings of the 13th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 362–375, 2011.

[32] Britta Peis, Martin Skutella, and Andreas Wiese. Packet routing: Complexity and algorithms. In *Proceedings of the 7th Workshop on Approximation and Online Algorithms (WAOA)*, pages 217–228, 2009.

[33] David Peleg. *Distributed computing*, volume 5. 2000.

[34] Yuval Rabani and Éva Tardos. Distributed packet switching in arbitrary networks. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC)*, volume 96, pages 366–375, 1996.

[35] Thomas Rothvoß. A simpler proof for $O(Congestion + Dilation)$ packet routing. In *Proceedings of the 16th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 336–348, 2013.

[36] Christian Scheideler. *Universal routing strategies for interconnection networks*, volume 1390. Springer, 2006.

[37] Jeanette P Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff–hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995.

[38] Daniel D Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.

[39] Aravind Srinivasan and Chung-Piaw Teo. A constant-factor approximation algorithm for packet routing and balancing local vs. global criteria. *SIAM Journal on Computing (SICOMP)*, 30(6):2051–2068, 2001.

[40] Robert E Tarjan and Jan Van Leeuwen. Worst-case analysis of set union algorithms. *Journal of the ACM (JACM)*, 31(2):245–281, 1984.

[41] Donald M. Topkis. Concurrent broadcast for information dissemination. *IEEE Transactions on Software Engineering*, (10):1107–1112, 1985.

[42] Santosh Vempala and Berthold Vöcking. Approximating multicast congestion. In *Proceedings of the 10th Annual International Symposium on Algorithms and Computation (ISAAC)*, pages 367–372, 1999.