Sending Secrets Swiftly: Approximation Algorithms for Generalized Multicast Problems

Afshin Nikzad[1] and R. Ravi[2]

[1] MS&E Department, Stanford University, USA
nikzad@stanford.edu
[2] Tepper School of Business, Carnegie Mellon University, USA
ravi@cmu.edu

Abstract. We consider natural generalizations of the minimum broadcast time problem under the telephone model, where a rumor from a root node must be sent via phone calls to the whole graph in the minimum number of rounds; the telephone model implies that the set of edges involved in communicating in a round form a matching. The extensions we consider involve generalizing the number of calls that a vertex may participate in (the capacitated version), allowing conference calls (the hyperedge version) as well as a new multicommodity version we introduce where the rumors are no longer from a single node but from different sources and intended for specific destinations (the multicommodity version). Based on the ideas from [6,7], we present a very simple greedy algorithm for the basic multicast problem with logarithmic performance guarantee and adapt it to the extensions to design typically polylogarithmic approximation algorithms. For the multi-commodity version, we give the first approximation algorithm with performance ratio $2^{O\left(\log\log k\sqrt{\log k}\right)}$ for k source-sink pairs. We provide nearly matching lower bounds for the hypercasting problem. For the multicommodity multicasting problem, we present improved guarantees for other variants involving asymmetric capacities, small number of terminals and with larger additive guarantees.

Keywords: approximation algorithms, graph algorithms, b-matching, LP rounding.

1 Introduction and Motivation

Rumor spreading in networks has been an area of much study involving the gamut from finding the minimum possible number of messages to spread gossip around the network [23,2,12] to finding graphs with minimum number of edges that are able to spread rumors in the minimum possible time in the network [9]. An important NP-hard formulation asks to find a scheme that spreads a rumor from a single root node to all other nodes under the popular "telephone" model where every node can participate in a telephone call with at most one other neighbor in each round, and the goal is to minimize the number of rounds. This is the minimum broadcast time problem for which there has been active work in designing approximation algorithms [14,20,10,7]. We study generalizations of this problem that involve (i) sending the message to only a subset of receivers (multicasting), (ii) capacity constraints on the number of calls in which a node can participate (capacitated cases), (iii) allowing conference calls modeled by hyperedges (hypercasting problem), and (iv) multiple sources of rumors with different sets that are the targets for the different rumors (the multicommodity case). Our paper initiates work on the capacitated, hypercasting and multicommodity extensions of the "rapid rumor ramification" problem [20], bringing it to next range of generalizations of "sending secrets swiftly".

Problem Definition

Definition 1. *In the (minimum time)* **Multicast** *problem, we are given an undirected graph $G(V, E)$ which represents a telephone network on V, where two adjacent nodes can place a telephone call to each other. We are given a source vertex r and a set of terminals $R \subseteq V$. The source vertex has a message and it wants to inform all the terminals of the message. To do this, the vertices of the graph can communicate in rounds: In each round, we pick a matching of G and arrange a bidirected phone call between each vertex in the matching and its matched pair. If any of the two vertices knows the message before the phone call, the other one will also know it afterwards. The goal is to deliver the message to all the terminals in the minimum number of rounds.*

When $R = V$, the Multicast problem is known as the (minimum time) *broadcast* problem which is one of the most basic and well-studied problems in this setting. Applications of this problem arise commonly in multicasting in networks [21], keeping the information consistent across copies of replicated databases by broadcasting from the changed copy to the others [16,17], as well as in finding schemes that ensure that maximum information delay in problems modeled by vector clocks [15] is minimized.

In this paper, we investigate the Multicast Problem in the following three more general settings: i. Allowing multiple source vertices, i.e. the multi-commodity setting rather than the single-source setting. ii. Allowing conference calls (with possibly more than two participants) rather than just having phone calls. iii. Imposing capacities for the vertices, i.e. allowing a vertex to be in a number of phone calls in each round, which can not exceed its capacity. In all of these generalizations, the objective function remains minimizing the total number of rounds used in the solution.

The first generalization is having messages with arbitrary source and destination vertices, i.e. unlike the Multicast Problem, the messages do not need to share the same source vertex.

Definition 2. *In the* **Multicommodity Multicast Problem** *(MM), a graph $G(V, E)$ is given along with a set of pairs of nodes $P = \{(s_i, t_i) | 1 \leq i \leq k\}$, known as demand pairs. Each vertex s_i has a message m_i which needs be delivered to t_i. The vertices communicate similar to the Multicast problem, i.e. during a phone* *call, each vertex can pass (a copy of) all of the messages that it has to the other vertex.*

The second aspect in our model is having conference calls rather than having only phone calls, i.e. calls involving (possibly) more that two, rather than only two, persons. So, instead of a simple graph G, we are given a hypergraph the edges of which represent the potential conference calls. We call this problem the *Hypercast Problem*; we define this problem formally and study it extensively in Section D.

Finally, we bring in the notion of *capacity of a vertex* to our model for the Multicast Problem, and allow a vertex to be in possibly more than a single call in each round; the maximum number of phone calls that a vertex can have in each round is called the *capacity* of the vertex. The *Capacitated Multicast Problem* is formally defined and studied in Section C.

In this paper, we develop a unified solution framework that can incorporate each of the Hypercast, Multi-commodity Multicast, and Capacitated Multicast aspects, leading to the first approximation algorithms for any combination of these extensions such as the Capacitated Hypercast Problem, or Multi-commodity Hypercast Problem. In the rest of this paper, we let $n = |V|$, $k = |R|$ and OPT be the optimal number of rounds needed to solve the given Multicast instance, unless it is specified otherwise.

2 Related Work

Finding optimal broadcast schedules for trees was one of the first theoretical problems in this setting and was solved using Dynamic Programming [18]. For general graphs, Kortsarz et. al. developed an additive approximation algorithm which uses at most $c \cdot OPT + O(\sqrt{n})$ rounds for some constant c. Later, Ravi [20] provided a $O(\frac{\log^2 n}{\log\log n})$-approximation for the same problem using the result of Raghavan [19] for randomized rounding of an LP formulation for the concurrent multicommodity flow problem.

Guha et.al. [10] improved the approximation factor for Multicasting in general graphs to $O(\log k)$ where k is the number of terminals. To the best of our knowledge, the best approximation factor for the Multicast problem is $O(\frac{\log k}{\log\log k})$ [7]. Both of [7,10] present a recursive algorithm which reduces the total number of uninformed terminals in each step of the recursion, while using $O(OPT)$ number of rounds in that step. In [10], they reduce the number of uninformed terminals by a constant factor in each step and so they obtain a $O(\log k)$-approximation, but in [7], the number of uninformed terminals is reduced by a factor of OPT which gives a $O(\frac{\log k}{\log\log k})$-approximation due to the fact that $OPT = \Omega(\log k)$.

3 Our Results

Our main contribution is developing a framework to design approximation algorithms for various generalizations of the Multicast Problem. A summary of the key results in this paper is listed below. For the complete list of our results, see Tables 1 and 2 in Section 7.

1. In Section 5, we give a *very simple* $O(\log k)$-approximation for the Multicast Problem, which is based on the ideas of [6,7]. With a slight modification of this algorithm in Section C, we adapt it to solve the Capacitated Multicast Problem.
2. In Section 6, based on our simple algorithm for the Multicast Problem, we design a $2^{O\left(\log\log k\sqrt{\log k}\right)}$-approximation for the Multi-commodity Multicast Problem (note that the approximation factor is, while being super-polylogarithmic, still smaller than any constant root of k).
3. In Section D, we develop our simple algorithm further and obtain a $O(\log k \cdot \log n \cdot D)$-approximation for the Hypercast Problem, where D is the maximum size of a hyperedge. Also, our hardness results for the Hypercast Problem show that the dependence on D is unavoidable (see Section I).
4. In Sections G and H, we explore the Multi-commodity Multicast problem further and design two polylogarithmic approximation algorithms which carry additional additive factors of $\sqrt{n}$ and $\Delta(G)$ (maximum degree).

While our algorithms for Multicast and Multi-commodity Multicast problems are purely combinatorial, the algorithm for Hypercast involves solving a Linear Program and randomized rounding of the LP solution.

Our results are not limited to this since our framework can handle more general cases of the Multicast Problem, e.g. Capacitated Multi-commodity Hypercast Problem; all of them are summarized in Section 7.

4 Preliminaries

4.1 The Multicast Problem

In the context of any of the problems discussed in Section 1, *sending vertex u to vertex v* means sending the information of u to v in a (potentially specified) number of rounds. Given that $P = \{(s_i, t_i) | 1 \leq i \leq k\}$ is the set of demand pairs in any single-source or multi-commodity instance, let $S = \{s_1, \dots, s_k\}$ be the set of sources, $T = \{t_1, \dots, t_k\}$ be the set of sinks, and $R = S \cup T$ be the set of *terminals*. Also, let $\mathcal{I}(G, P)$ denote the Multicast (Hypercast) instance defined by P on the graph (Hypergraph) G. We also use $\mathcal{I}$ when both G, P are clearly known from the context.

4.2 Schedules

A *schedule* is a sequence of rounds. The length of a schedule $\mathcal{S}$ is denoted by $|\mathcal{S}|$ and is the number of rounds it contains. A schedule for a (single-source or multi-commodity) Multicast instance is *non-lazy* if, in any round of it, any two idle and adjacent vertices have identical information in that round.

4.3 Graphs and Matchings

Suppose G is a simple graph and let $n = |V(G)|$. Let $N(v)$ be the set of the neighbors of a vertex v, and for any $S \subseteq V(G)$, let $N(S) = \cup_{v \in S} N(v)$. Denote the degree of the maximum-degree vertex in G by $\Delta(G)$. For any $X \subseteq V(G)$, let $G[X]$ be the induced subgraph of G on X. For any family of subgraphs of G, such as $\mathcal{F}$, let $V(\mathcal{F})$ denote the union of the vertices of the subgraphs in $\mathcal{F}$ and $E(\mathcal{F})$ denote the union of the edges of the subgraphs in $\mathcal{F}$.

The distance between two vertices of G, such as u and v, is represented by $d_G(u,v)$. If there is no path in G between u, v, then let $d_G(u,v) = \infty$. The diameter of G, denoted by $diam(G)$, is $\max\limits_{u,v \in V(G)} d_G(u,v)$. Also, define $diam_P(G) = \max\limits_{(s,t)\in P} d_G(s,t)$, where P is a set of pairs of vertices, e.g. the set of demand pairs.

Given a bipartite graph $H[X, Y]$ with partitions X and Y, a b-matching in H is a subset M of the edges of H such that each vertex of X is incident to exactly one edge of M and each vertex of Y is incident to at most b edges of M.

4.4 Spiders

Spiders are subgraphs that have been useful in designing algorithms for the Multicast and directed Broadcast problems in [10,5]. A *spider* S is a set of (almost) vertex-disjoint paths all starting at the same vertex, e.g. v, and sharing no vertex other than v. Define the *center of* S to be v. Also, let the *degree of* S, denoted by $deg(S)$, be the degree of v in S, and the *length of* S, denoted by $len(S)$, be $\max_{u\in V(S)} d_S(u,v)$.

It's very easy to verify the following lemma stated in [5].

Lemma 1. *Using a non-lazy schedule, the center of a spider S can send (broadcast) a message to the rest of its vertices in $deg(S) + len(S) - 1$ rounds.*

5 The Multicast Problem

In this section, we present a $O(\log k)$-approximation for the Multicast problem, which is obtained by simplifying the $O(\frac{\log k}{\log\log k})$-approximation given in [7]. Our simple algorithm plays an important role in our framework. Later, this algorithm will be developed further to design algorithms for the introduced generalizations of the Multicast problem. For instance, with a slight modification, it turns into a $O(\log k)$-approximation for the Capacitated Multicast problem (see Section C).

5.1 Outline of the Algorithm

Our algorithm accepts a parameter L as a part of the input, which is our guess for the optimal solution of the given multicast instance. Since n is an upper bound on the length of the optimal schedule, we can easily try all the possible values for L from 1 to n and run the algorithm once for each of these values. Our algorithm is guaranteed to return a schedule of length $O(L \cdot \log k)$ assuming that L is the length of the optimal schedule. So, from now on in this section, we think of L as the length of the optimal schedule W.L.O.G.

The algorithm is a recursive algorithm and has 4 phases. In Phase 1 of the algorithm, we reduce the given instance to a smaller instance. In Phase 2, we solve the smaller instance recursively, and finally in Phase 3 and 4, we inform the rest of the vertices which didn't receive the message in Phase 2. We explain each of these phases briefly and after that, we'll see the full description of the algorithm.

Phase 1. This phase starts with finding a family of *vertex disjoint* paths $\mathcal{P}$ each of length at most $4L$ such that the endpoints of the paths belong to R. We find these paths greedily, i.e. we start with $\mathcal{P} = \emptyset$ and using BFS, we search for a new path of length at most $4L$ between the terminals. We continue until we can add no more such paths to $\mathcal{P}$. Pick one endpoint from each path and designate the picked set of vertices to be R'.

Phase 2. Solve the multicast problem for the new set of terminals R' recursively and run the obtained schedule. (So, all the vertices in R' will receive the message by the end of this phase.)

Phase 3. Inform all the vertices belonging to $V(\mathcal{P})$ in $4L$ rounds. This is possible since in Phase 2, we have already informed at least one vertex of each path in $\mathcal{P}$.

Phase 4. For each of the uninformed terminals, namely $v \in R \backslash V(\mathcal{P})$, find a path M_v which connects v to one of the informed vertices (note that the set of the informed vertices is currently $V(\mathcal{P})$). These paths are guaranteed to satisfy the following properties:

1. The length of each path is at most $2L$.
2. Only one vertex on each path belongs to $V(\mathcal{P})$, which is an endpoint of the path.
3. The paths won't share any vertices except possibly in the endpoints belonging to the set $V(\mathcal{P})$. Moreover the degree of any node in $V(\mathcal{P})$ due to these paths is at most L.

In other words, $M = \bigcup_{v \in R \backslash V(\mathcal{P})} M_v$ is a union of vertex disjoint spiders of length at most $2L$ and we will inform the vertices in $R \backslash V(\mathcal{P})$ using these spiders. (see Figure 1)

Before presenting the algorithm formally, we describe Phase 4 in more details.

5.2 The Algorithm: Phase 4

Our goal in Phase 4, assuming that the vertices in $V(\mathcal{P})$ have received the message, is to inform the rest of the terminals in $O(L)$ rounds. The only assumptions we need here are that the choice of $\mathcal{P}$ is maximal in Phase 1 and $V(\mathcal{P})$ is informed.

To inform the rest of the terminals, we find a family of vertex-disjoint spiders such that each of them has a length at most $2L$ and a center belonging to $V(\mathcal{P})$. Moreover, we need the spiders to contain all the terminals in $R \backslash V(\mathcal{P})$. To construct the family of spiders, we start with finding the paths M_v for all $v \in R \backslash V(\mathcal{P})$. In order to find the paths, we construct a bipartite graph

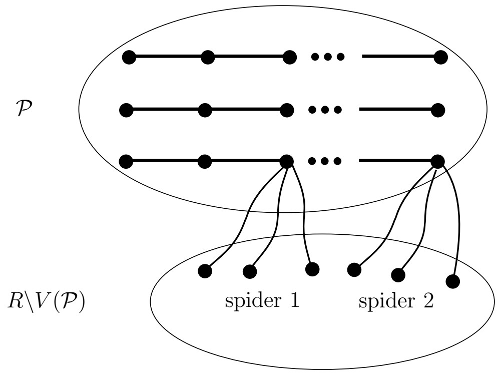

Fig. 1. Informing the vertices of $R\backslash V(\mathcal{P})$ using spiders

$H[R\backslash V(\mathcal{P}), V(\mathcal{P})]$ with edges defined as follows: There is an edge in H between two vertices $x \in R\backslash V(\mathcal{P})$ and $y \in V(\mathcal{P})$ if there is path of length at most $2L$ connecting x to y such that no other vertex of the path belongs to $V(\mathcal{P})$ except y. So, each edge in H is associated with a path in G (if there were many such paths for x, y, then choose one of them arbitrarily). Now, find a b-matching in H for the minimum possible integer b. Then, for all $v \in R\backslash V(\mathcal{P})$, define M_v to be a path in G which is associated with the edge incident to v in the b-matching. Let M be the subgraph of G defined as $M = \bigcup_{v \in R\backslash V(\mathcal{P})} M_v$, then, the following lemma holds for M.

Lemma 2. *M is a union of vertex disjoint spiders each with length at most $2L$ and degree at most L.*

Proof. First, we show that M is a union of vertex disjoint spiders. To prove this, note that for any two vertices $u, v \in R\backslash V(\mathcal{P})$, M_u and M_v can't share any vertex except possibly in the endpoints belonging to $V(\mathcal{P})$, since if they do, it contradicts the maximality of $\mathcal{P}$, i.e. there would have been a path of length at most $4L$ between u and v which could be added to $\mathcal{P}$.

So, M is a union of vertex-disjoint spiders, namely the family $\mathcal{M}$ of spiders, and since the length of each path M_v is at most $2L$, then $len(\mathcal{M}) \leq 2L$. It only remains to prove that $deg(\mathcal{M}) \leq L$. Consider the optimal multicast schedule which uses exactly L rounds. Let E' be the subset of the edges of G which are used in the optimal schedule and G' be the subgraph of G with E' as its edge set. It's easy to verify that G' is a tree of diameter at most $2L$ and maximum degree at most L [20]. Now, for each $v \in R\backslash V(\mathcal{P})$, define M'_v to be the unique path in G' from v to $f(v)$, where $f(v) = \arg\min_{u \in V(\mathcal{P})} d_{G'}(v, u)$. Note that $d_{G'}(v, f(v)) \leq 2L$ since the diameter of G' is at most $2L$, which implies $(v, f(v)) \in E(H)$ because of the existence of M_v.

Next, we prove that $\bigcup_{v \in R \setminus V(\mathcal{P})} (v, f(v))$ is a L-matching in H, which shows the existence of a b-matching in H with $b \leq L$, and that's all we need to show that $deg(\mathcal{M}) \leq L$. To prove the claim, just note that the family of paths $\{M'_v : v \in R \setminus V(\mathcal{P})\}$ in G' are edge-disjoint, which means there can't be more than $\Delta(G')$ of these paths with the same endpoint, implying that $\bigcup_{v \in R \setminus V(\mathcal{P})} (v, f(v))$ is a $\Delta(G')$-matching in H. The fact that $\Delta(G') \leq L$ finishes the proof.

From Lemmas 1 and 2 we conclude the following:

Lemma 3. *Assuming that the vertices in $V(\mathcal{P})$ have received the message, we can find a schedule in polynomial time which informs the rest of the terminals in $3L$ rounds.*

5.3 The Algorithm

The whole algorithm for the Multicast problem is presented more formally below.

```
Algorithm Multicast
Input: A graph $G$ and a set of terminals $R$
1.  $\mathcal{P}, R' \leftarrow \emptyset$
2.  for all $(u,v) \in R \times R$ such that $u \neq v$ (* Phase 1 *)
3.      do Find the shortest path in $G[V(G) \setminus V(\mathcal{P})]$ from $u$ to $v$, namely $Q_{u,v}$.
4.         if the length of $Q_{u,v}$ is not more than $4L$
5.            then $\mathcal{P} \leftarrow \mathcal{P} \cup Q_{u,v}$
6.                 $R' \leftarrow R' \cup \{u\}$
7.  Inform $R'$ recursively by calling Multicast$(G,R')$. (* Phase 2 *)
8.  Inform $V(\mathcal{P})$ using the paths in $\mathcal{P}$ in at most $4L$ rounds. (* Phase 3 *)
9.  Construct the bipartite graph $H[R \setminus V(\mathcal{P}), V(\mathcal{P})]$.(* Beginning of Phase 4 *)
10. Find the smallest integer $b$ such that $H$ has a $b$-matching.
11. Use the family of spiders $\mathcal{M}$ associated with the $b$-matching and inform
    $R \setminus V(\mathcal{P})$ in at most $deg(\mathcal{M}) + len(\mathcal{M}) - 1$ rounds.
```

Theorem 1. *Algorithm Multicast is a $7 \log k$-approximation for the Multicast problem.*

Proof. The proof of correctness is trivial. We just analyze the approximation factor of the algorithm here. Note that in each level of the recursion, the number of terminals are at least halved, which means there will be at most $\log k$ levels. Moreover, in each level we use at most $4L$ rounds in Line 9 and $deg(\mathcal{M}) + len(\mathcal{M}) - 1$ rounds in Line 12. By lemma 3 we have $deg(\mathcal{M}) + len(\mathcal{M}) - 1 \leq 3L$ implying that we use at most $7L$ rounds in each level of the recursion and $7L \cdot \log k$ rounds in total.

6 The Multicommodity Multicast Problem

In this section, we present a $2^{O\left(\log\log|R| \cdot \sqrt{\log|R|}\right)}$-approximation for the Multicommodity Multicast Problem; recall that R is the set of terminals.

6.1 Preleminaries: Multicast Schedules

Any *single-source* Multicast schedule can be represented by a directed tree such that there will be phone calls only on the tree edges. Given a multicast schedule, this tree is defined by choosing, for every vertex other than the source, the unique edge along which a message is conveyed to that vertex for the first time [20]. We state this fact in the proposition below.

Proposition 1. *Any single-source Multicast schedule can be represented by a subgraph of G which is a tree.*

Also, using Proposition 1, we can assume that the output of Algorithm *Multicast* is a tree:

Proposition 2. *W.L.O.G. the output of Algorithm Multicast can be assumed to be a tree, i.e. the set of edges used in the phone calls form a tree.*

6.2 Sparsification

Before describing our algorithm, we need a key lemma related to graph spanners, i.e. sparse subgraphs such that distances between adjacent nodes are preserved within a logarithmic factor in the subgraph. To the best of our knowledge, this result first appeared in [1] as Lemma 3.1. We only state and use a simple corollary of this lemma, for which an independent proof is given also in Section H of this paper.

Corollary 1. *Given a simple n-vertex graph G, we can find a subgraph $s(G)$ of G in polynomial time, such that $|E(s(G))| \leq 2n \log n$ and for each $(u, v) \in E(G)$, we have $d_{s(G)}(u, v) \leq 8 \log n \cdot d_G(u, v)$.*

Algorithm *Sparsify* follows as a consequence of Corollary 1. It takes a simple undirected graph $H(V, P)$ as its input and computes $s(H)$. We will use this subroutine later in our main algorithm.

```
Algorithm Sparsify
Input: A vertex set $V$, and a set of pairs of $V$, called $P$
Output: A subset of $P$
1. Assuming that $H(V, P)$ is a simple undirected graph, use Corollary 1 to
   compute $s(H)$.
2. Output $E(s(H))$.
```

6.3 The Algorithm

In the rest of this section, assume that we want to solve the instance $\mathcal{I}(G, P)$, and $\mathcal{L}$ is an optimal solution for this instance. Similar to Algorithm *Multicast*, our algorithm for the MM Problem accepts a parameter L in the input, which is our guess for the optimal solution.

The algorithm consists of 3 phases. In Phase 1, we (potentially) reduce the number of the demand pairs using Algorithm *Sparsify*, i.e. by calling *Sparsify*(R,P). Assuming that a subset of P, namely $\hat{P}$, is the output, we will see that repeating any feasible solution of $\mathcal{I}(G,\hat{P})$ for $O(\log n)$ times would give a feasible solution for $\mathcal{I}(G,P)$.

In Phase 2, we try to satisfy a large fraction of the demand pairs greedily. If it is done successfully, we repeat, otherwise, we go to Phase 3 and solve an instance with a fewer number of terminals recursively. A key new idea to handle the smaller instance is to use a fictitious multicast scheme to assign the terminal pairs not satisfied in this phase to one of the terminals that are.

Before seeing the formal description of the algorithm, we explain Phases 2 and 3 more precisely.

Phase 2. In this phase, we find a maximal family of vertex-disjoint paths, namely $\mathcal{P}$, where each path in $\mathcal{P}$ has a length at most L and connects s_i to t_i for some pair (s_i,t_i) in P. If $|\mathcal{P}|$ was large enough, then using these paths we satisfy a large fraction of the demand pairs in L rounds, and repeat Phase 2. Otherwise, we go to Phase 3.

More precisely, in the beginning of Phase 2, $\mathcal{P}$ is empty. We sort the pairs in P in some arbitrary order, say (s_i,t_i) for $1 \leq i \leq k$, and visit the pairs in this order. When visiting the i-th pair, we check if there exists a path of length at most L between s_i and t_i in $G[V(G)\backslash V(\mathcal{P})]$. If there was such a path, we add it to $\mathcal{P}$. After visiting all of the k pairs in P, assume $P' = \{(s'_1,t'_1),\ldots,(s'_{k'},t'_{k'}\})$ is the subset of the pairs in P for which we were able to find the path of length at most L. Now, two possible cases can happen based on the size of $\mathcal{P}$. Define $\xi(x) = 2^{\log x-\sqrt{\log x}}$ for all positive x, then, if $|\mathcal{P}| > \xi(|R|)$ (sufficiently large, and hence sufficiently good progress), remove P' from P and repeat Phase 2. Otherwise, go to Phase 3.

Phase 3. In this Phase, we construct a smaller instance on the set of terminals $R' = \{s'_1,\ldots,s'_{k'}\}$, solve it recursively, and then provide a solution for the original problem using the solution of the smaller instance, as in our original framework for multicasting. Intuitively, we send each vertex in $R\backslash R'$ to a vertex in R' in a small number of rounds. By doing so, we create a new instance of the MM problem, i.e. the old instance induced on the set R'. We solve this new instance recursively, and convert its solution to a solution for the old instance. Note that since R' is "small", the recursive problem is much smaller than the original one and hence we have made progress, but at the expense of having to route all the remaining demands of the recursively picked demand terminals. Formally, we perform the following steps in Phase 3:

1. Construct a function $f: R \to R'$ and a schedule $\mathcal{S}$ such that $\mathcal{S}$ sends v to $f(v)$ for all $v \in R$, and also $|\mathcal{S}| \leq L$. Run the schedule $\mathcal{S}$ and send v to $f(v)$ for all $v \in R$.
2. Construct a new instance of the MM problem on G with the set of terminals R' and the set of demand pairs $P' = \{(f(u),f(v)) : \forall(u,v) \in P\}$. Solve this instance recursively and run the obtained schedule.

3. Run the schedule $\mathcal{S}$ in the reverse order to send $f(v)$ to v simultaneously for all $v \in R$.

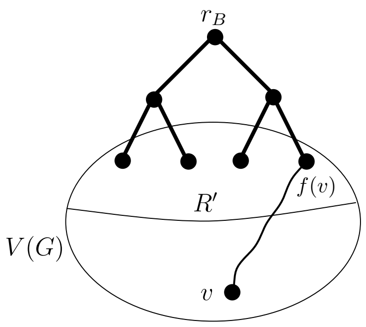

Fig. 2. The Multicast instance obtained from adding a dummy binary tree to the graph

Constructing $\mathcal{S}$ and f for Phase 3. To complete the above description for Phase 3, we need to find the schedule $\mathcal{S}$ and the function f. To do so, we construct and solve an *auxiliary Multicast instance*, the solution of which gives $\mathcal{S}$ and f.

In the beginning of Phase 3, construct an arbitrary binary tree B of height $O(\log |R'|)$ rooted at a dummy vertex r_B, such that the only common vertices between B and G are the leaves of B, which coincide with the set R' (see Figure 2).

Using Algorithm *Multicast*, solve the Multicast instance with the root r_B and the set of terminals $R \backslash R'$. Let the schedule $\mathcal{S}$ be the solution to this instance, which uses the tree $T_{\mathcal{S}}$ (recall that by Proposition 2, the solution provided by Algorithm *Multicast* is a tree). Then, define f as follows: If $v \in R'$, let $f(v) = v$, otherwise, consider the unique path in $T_{\mathcal{S}}$ from v to r_B. Define $f(v)$ to be the closest vertex to r_B which is on this path and belongs to $V(G)$ (see Figure 2).

Defining f and $\mathcal{S}$ completes the description of Phase 3. Now, we present our algorithm more formally below.

Algorithm *MM*
Input: A graph G and a set of pairs P

```
1.  if $P = \{\}$ then return.
2.  $S \leftarrow \{s | \exists t : (s,t) \in P\}$
3.  $T \leftarrow \{t | \exists s : (s,t) \in P\}$
4.  $R \leftarrow S \cup T$
5.  if $|P| > 2|R|.\log|R|$ (* Phase 1 *)
6.     then $P = Sparsify(R,P)$
7.             for 1 to $8\log|R|$
8.                 do $MM(G,P)$
9.             return.
10. $\mathcal{P}, X \leftarrow \emptyset$ (* Phase 2 *)
11. for $i \leftarrow 1$ to $|P|$
12.      do Find the shortest path in $G[V(G)\backslash V(\mathcal{P})]$ from $s_i$ to $t_i$, namely $Q_i$.
13.         if the length of $Q_i$ is not more than $L$
```

```
14.             then $\mathcal{P} \leftarrow \mathcal{P} \cup \{Q_i\}$
15.                  $X \leftarrow X \cup \{(s_i, t_i)\}$
16. if $|\mathcal{P}| \geq \xi(|R|)$
17.     then For all $(s_i, t_i) \in X$ simultaneously, send $s_i$ to $t_i$ through the path
             $Q_i$.
18.          $P \leftarrow P \backslash X$
19.          Go to Line 10.
20.     else Construct and solve the auxiliary Multicast instance to obtain $\mathcal{S}$
             and $f$. (* Phase 3 *)
21.          Run the schedule $\mathcal{S}$ in the reverse order to send $v$ to $f(v)$ for all
             $v \in R$.
22.          $Y \leftarrow \{(f(u), f(v)) | (u, v) \in P\}$
23.          $MM(G, Y)$
24.          Run the schedule $\mathcal{S}$ to send $f(v)$ to $v$ for all $v \in R$
```

Before analyzing the approximation ratio of the algorithm, we give the following lemma to bound the length of schedule $\mathcal{S}$ in terms of L and m, where $m = |R|$.

Lemma 4. $|\mathcal{S}| \leq 21 \log m \cdot L + 14 \log^2 m$.

Proof. Note that $\mathcal{S}$ is found by Algorithm *Multicast* as a solution to the auxiliary Multicast instance, and so by Theorem 1, $|\mathcal{S}| \leq 7 \log m \cdot L_{aux}$, where L_{aux} is the length of the optimal schedule for the auxiliary instance. Consequently, to prove the claimed bound, it's enough to prove that $L_{aux} \leq 3L + 2 \log m$. So, we show there exists a feasible schedule of length $3L + 2 \log m$ for the auxiliary instance. This schedule has 4 steps: 1. Use the dummy binary tree and send its root, i.e. r_B, to all of its leaves, i.e. the set R'. 2. Use the path Q_i and inform $V(Q_i)$ simultaneously for all $Q_i \in \mathcal{P}$. 3. Run the schedule $\mathcal{L}$. 4. Run the schedule $\mathcal{L}$ in the reverse order.

The suggested schedule has a length at most $3L + 2 \log m$ since Steps 1,2,3,4 have a length at most $2 \log m, L, L, L$ respectively. In the rest of the proof, we show the feasibility of this schedule. Note that after Step 1, the set of vertices R' is informed since they are the leaves of the binary tree. After Step 2, the set $V(\mathcal{P})$ is informed since each path $Q_i \in \mathcal{P}$ has an endpoint $s_i \in R'$. For the sake of contradiction, assume there is a terminal $r \in R \backslash R'$ which is not informed after Step 4. This terminal has to be in at least one demand pair, namely the pair (s, t). The schedule $\mathcal{L}$ sends s to t via a path in G, namely Q. The path Q must share at least a vertex with $\mathcal{P}$, due to the maximality of $\mathcal{P}$. This fact, and the fact that $V(\mathcal{P})$ is informed by the end of Step 2, imply that t should be informed after Step 3, and s should be informed after Step 4. Which is a contradiction with r not being informed by the end of Step 4.

Theorem 2. *Algorithm MM is a* $2^{O\left(\log\log|R| \cdot \sqrt{\log|R|}\right)}$*-approximation for the Multi-commodity Multicast Problem.*

Proof. Let $T(m)$ denote the approximation factor of our algorithm. By induction on m, we prove that $T(m) \leq 2^{\epsilon \log\log m . \sqrt{\log m}}$, for any fixed $\epsilon > 6$. To do so, we provide an upperbound for $T(m)$ as follows:

$$T(m) \leq 8 \log m \cdot \left(\frac{2m \log m}{\xi(m)} \cdot L + 7 \log m \cdot (6L + 4 \log m) + T(\xi(m)) \cdot (43 \log m \cdot L + 28 \log^2 m) \right) \times \frac{1}{L}$$

Before analyzing this recurrence relation, we show that its right-hand side gives a valid upperbound on $T(m)$: The last coefficient in the right-hand side, i.e. $\frac{1}{L}$, is due to the definition of approximation ratio of an algorithm. The first coefficient, i.e. $8 \log m$, stands for Line 7 of the algorithm, as a result of the (potential) sparsification. It remains to analyze the middle coefficient.

The summand $\frac{2m \log m}{\xi(m)} \cdot L$ is an upperbound on the number of rounds used in Phase 2 of the algorithm. The two other summands bound the number of rounds used in Phase 3 of the algorithm. We justify the latter fact separately as follows.

The summand $7 \log m \cdot (6L + 4 \log m)$ is an upperbound on the number of rounds used in Lines 21 and 24 overall. Since only the schedule $\mathcal{S}$ is run in these lines, we equivalently show that $|\mathcal{S}| \leq 7 \log m \cdot (3L + 2 \log m)$, which is done in Lemma 4.

Finally, we verify that the summand $T(\xi(m)) \cdot (43 \log m \cdot L + 28 \log^2 m)$ is an upperbound on the number of rounds used in Line 23 of the algorithm: Let L_Y be the optimal number of rounds needed to solve $\mathcal{I}(G, Y)$. By the induction hypothesis, the number of rounds used in Line 23 is at most $T(\xi(m)) \cdot L_Y$. So, it's enough to show that $L_Y \leq 43 \log m \cdot L + 28 \log^2 m$. We do this by giving a feasible solution of length at most $43 \log m \cdot L + 28 \log^2 m$ for $\mathcal{I}(G, Y)$: Run the schedule $\mathcal{S}$, then run $\mathcal{L}$, and finally run the schedule $\mathcal{S}$ in the reverse order. The claimed upperbound for the length of this schedule simply follows from Lemma 4.

It only remains to analyze the recursive formula $T(m)$ and prove the claimed bound for it; we omit this part here, the complete proof appears in Section B.

7 Conclusion

Our main contribution is developing a unified recursive framework that we use to design approximation algorithms for various extensions of the Multicast Problem. In particular, we consider three generalizations: i. allowing conference calls involving possibly more than two participants; ii. allowing multiple source and destination vertices; and iii. allowing capacities for vertices .

A comprehensive summary of all our results can be seen in Tables 1 and 2. A descriptive summary of the tables also appears in Section A. For the cells in the tables which are marked with [$*$], the proofs have been omitted in this paper, since they are very similar to the proofs that appear in the paper and can be derived from them with slight modifications.

Designing a poly-logarithmic approximation algorithm for the multicommodity multicast problem is the most important remaining open problem from our work.

Table 1. This table summarizes our approximation ratios and additive approximations for the Multicast Problem

Multicast	*Single-source*	*Multi-commodity*
Non-capacitated	$O(\log k/\log\log k)$ [7] $\Omega(3-\epsilon)$ [5]	$2^{O(\log\log k\cdot\sqrt{\log k})}$ $O\left(\log n\cdot OPT+\sqrt{n}\log^2 n\right)$ $O\left(\frac{\log^3 n}{\log\log n}\cdot(OPT+\Delta(G))\right)$
Capacitated	$O(\log k)$	$2^{O(\log\log k\cdot\sqrt{\log k})}$ [*]

Table 2. This table summarizes our hardness results and approximation ratios for the Hypercast Problem

Hypercast	*Single-source*	*Multi-commodity*
Non-capacitated	$O(\log k\cdot\log n\cdot D)$ $\Omega(D^{1/3})$	$2^{O(\sqrt{\log k}(\log\log n+\log D))}$ [*]
Capacitated	$O(\log k\cdot\log n\cdot D)$ [*]	$2^{O(\sqrt{\log k}(\log\log n+\log D))}$ [*]

Acknowledgments. We would like to thank Takuro Fukunaga for discussions in the early stages of this work. We also thank Subhash Khot for pointing out the hardness result of [11] on finding large independent sets and Guy Kortsarz for his comments about the choice of the presentation.

References

1. Awerbuch, B., Kutten, S., Peleg, D.: On buffer-economical store-and-forward deadlock prevention. IEEE Transactions on Communication 42, 2934–2937 (1994)
2. Baker, B., Shostak, R.: Gossips and telephones. Discrete Math 2, 191–193 (1972)
3. Censor-Hillel, K., Haeupler, B., Kelner, J., Maymounkov, P.: Global Computation in a Poorly Connected World: Fast Rumor Spreading No Dependence on Conductance. In: STOC: ACM Symposium on Theory of Computing (2012)
4. Dvork, T.: Chromatic Index of Hypergraphs and Shannons Theorem. European Journal of Combinatorics 21(5), 585–591 (2000)
5. Elkin, M., Kortsarz, M.G.: Combinatorial logarithmic approximation algorithm for directed telephone broadcast problem. In: Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing, STOC 2002 (2002)
6. Elkin, M., Kortsarz, G.: An approximation algorithm for the directed telephone multicast problem. Algorithmica 45(4), 569–583 (2006)
7. Elkin, M., Kortsarz, G.: Sublogarithmic approximation for telephone multicast. In: SODA 2003 Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 76–85 (2003)
8. Feige, U., Kilian, J.: Zero knowledge and the chromatic number. J. Comput. System Sci. 57, 187–199 (1998)
9. Grigni, M., Peleg, D.: Tight bounds on minimum broadcast networks. SIAM J. Discrete Math. 4, 207–222 (1991)

10. Guha, S., Bar-noy, A., Naor, J., Schieber, B.: Multicasting in heterogeneous networks. In: STOC 1998 Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, pp. 448–453 (1998)
11. Guruswami, V., Sinop, A.K.: The complexity of finding independent sets in bounded degree (hyper)graphs of low chromatic number. In: SODA 2011 Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1615–1626 (2011)
12. Hajnal, A., Milner, E.C., Szemeredi, E.: A cure for the telephone disease. Canad Math. Bull 15, 447–450 (1976)
13. Haeupler, B.: Simple, Fast, and Deterministic Gossip and Rumor Spreading. In: SODA: ACM-SIAM Symposium on Discrete Algorithms (2013)
14. Kortsarz, G., Peleg, D.: Approximation algorithms for minimum time broadcast. SIAM Journal on Discrete Methods 8, 401–427 (1995)
15. Kossinets, G., Kleinberg, J., Watts, D.: The structure of information pathways in a social communication network. In: Proceedings of the 14th SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 435–443 (2008)
16. Leighton, F.T., Lewin, D.M.: Global Hosting System, US Patent 6108703 (Issued August 22, 2000)
17. Onus, M., Richa, A.W.: Minimum maximum-degree publish-subscribe overlay network design. IEEE/ACM Transactions on Networking, TON (2011)
18. Proskurowski, A.: Minimum broadcast trees. IEEE Trans. Comput. C-30, 363 (1981)
19. Raghavan, P.: Probabilistic construction of deterministic algorithms: Approximating packing integer programs. In: 27th Annual Symposium on Foundations of Computer Science (FOCS 1986), pp. 10–18 (1986)
20. Ravi, R.: Rapid rumor ramification: approximating the minimum broadcast time. In: 35th Annual Symposium on Foundations of Computer Science, FOCS 1994 (1994)
21. Scheuermann, P., Wu, G.: Heuristic Algorithms for Broadcasting in Point-to-Point Computer Networks. IEEE Transactions on Computers 33(9), 804–811 (1984)
22. Schrijver, A.: Combinatorial optimization: Polyhedra and Eficiency, ch. 21. Springer (2003)
23. Tijdeman, R.: On a Telephone Problem. Nieuw Arch. Wisk. 19, 188–192 (1971)

A Roadmap for the Appendix

The main results that appear in the Appendix are listed below.

1. In Section C, we present a very simple $O(\log k)$-approximation for the Capacitated Multicast Problem.
2. In Section D, we develop our simple algorithm further and obtain a $O(\log k \cdot \log n \cdot D)$-approximation for the Hypercast Problem, where D is the maximum size of a hyperedge. The dependence on D is natural due to our strong hardness results for the Hypercast Problem (see Section I): we prove that for any constant $\epsilon > 0$, the Hypercast Problem is $\Omega(n^{1-\epsilon})$-hard. But this hardness result, as our algorithm also suggests, only holds for large values of $D \geq \Omega(n^{1-\epsilon})$. The more natural and interesting case though, is when D is small, for which we provide a hardness ratio of $\Omega(D^{1/3})$ under Khot's 2-to-1 conjecture.

3. By modifying our algorithm for the Multi-commodity Multicast problem, we obtain two other algorithms for the same problem in Appendices G and H, which respectively produce solutions of length at most $O\left(\log n \cdot OPT + \sqrt{n}\log^2 n\right)$ and $O\left(\frac{\log^3 n}{\log\log n} \cdot (OPT + \Delta(G))\right)$.
4. In Section H we show an (approximate) equivalence between the Multi-commodity Multicast problem and the following Minimum Poise Subgraph problem: find a subgraph H of G in which the *Poise* of H [20], namely, the maximum pairwise distance in H between all pairs (s_i, t_i) plus the maximum degree in H, is minimized. We prove that any α-approximation algorithm for either of these problems gives an $O\left(\alpha \cdot \frac{\log^3 n}{\log\log n}\right)$-approximation for the other one.

Below, we briefly discuss these results an their connection to each other.

The algorithm for the Capacitated Multicast Problem is presented in Section C. As it can be seen in the tables, the approximation ratio for the capacitated problems almost always matches the ratio for the associated non-capacitated version, which means our framework can handle capacities very well.

The algorithm for the Hypercast Problem appears in Section D. In the Hypercast Problem, note that despite the strong $\Omega(n^{1-\epsilon})$ hardness result, which holds only for the less interesting case of $D \geq \Omega(n^{1-\epsilon})$, we develop a $O(\log k \cdot \log n \cdot D)$-approximation. It is not hard to see that this result is tight up to logarithmic factors if $D \geq n^{\epsilon}$ for some constant $\epsilon > 0$ (the proof is very similar to the proof for the $\Omega(n^{1-\epsilon})$-hardness). For the general case when we have no restrictions on D, we can also prove an $\Omega(D^{1/3})$-hardness under Khot's 2-to-1 conjecture. All these hardness results appear in the Section I.

Although there is a significant gap between the existing hardness ratio and the approximation ratio provided for the Multicommodity Multicast Problem, we are able to tighten this gap when $OPT \geq \Omega(\sqrt{n})$. For this case, we have a $O(\log^2 n)$-approximation as a consequence of an alternate algorithm that we present in Section G, which uses at most $O(\log n \cdot OPT + \sqrt{n}\log^2 n)$ rounds.

On the way to obtaining this additive approximation, we prove results for the special case when the instance has a small number of terminals in Section F. For given demand pairs P, this variant finds a schedule of length $O(|P| + D)$ where D denotes $diam_P(G)$.

To derive the result for small number of terminals, we develop and employ an extension of our framework for asymmetric capacities in Section E. Note that our results on this model also apply for the GOSSIP models recently studied in the literature [3,13] to give *relative* approximation algorithms in contrast to the more absolute guarantees provided in these papers in terms of the size of the graph or its diameter.

We shed more light on the Multicommodity Multicast problem by showing that it is (approximately) equivalent to a Minimum Poise Subgraph Problem, i.e. the problem of finding a subgraph H of the graph G which minimizes $\Delta(H) + diam_P(H)$. More precisely, we show that any α-approximation algorithm for either of these problems gives an $O\left(\alpha \cdot \frac{\log^3 n}{\log\log n}\right)$-approximation for the other one. This equivalence is formally proved in Section H. As a consequence of this equivalence, we obtain an algorithm which guarantees to produce a schedule of length at most $O\left(\frac{\log^3 n}{\log\log n} \cdot (OPT + \Delta(G))\right)$. Note that this gives a poly-logarithmic approximation in the instances when $OPT = \Omega(\Delta(G))$.

B Complete Proof of Theorem 2

Proof (of Theorem 2). Let $T(m)$ denote the approximation factor of our algorithm. By induction on m, we prove that $T(m) \leq 2^{\epsilon \log\log m . \sqrt{\log m}}$, for any fixed $\epsilon > 6$. To do so, we provide an upperbound for $T(m)$ as follows:

$$T(m) \leq 8\log m \cdot \left(\frac{2m\log m}{\xi(m)} \cdot L + 7\log m \cdot (6L + 4\log m) + T(\xi(m)) \cdot (43\log m \cdot L + 28\log^2 m)\right) \times \frac{1}{L}$$

Before analyzing this recurrence relation, we show that its right-hand side gives a valid upperbound on $T(m)$: The last coefficient in the right-hand side, i.e. $\frac{1}{L}$, is due to the definition of approximation ratio of an algorithm. The first coefficient, i.e. $8\log m$, stands for Line 7 of the algorithm, as a result of the (potential) sparsification. It remains to analyze the middle coefficient.

The summand $\frac{2m\log m}{\xi(m)} \cdot L$ is an upperbound on the number of rounds used in Phase 2 of the algorithm. The two other summands bound the number of rounds used in Phase 3 of the algorithm. We justify the latter fact separately as follows.

The summand $7\log m \cdot (6L + 4\log m)$ is an upperbound on the number of rounds used in Lines 21 and 24 overall. Since only the schedule $\mathcal{S}$ is run in these lines, we equivalently show that $|\mathcal{S}| \leq 7\log m \cdot (3L + 2\log m)$, which is done in Lemma 4.

Finally, we verify that the summand $T(\xi(m)) \cdot (43\log m \cdot L + 28\log^2 m)$ is an upperbound on the number of rounds used in Line 23 of the algorithm: Let L_Y be the optimal number of rounds needed to solve $\mathcal{I}(G, Y)$. By the induction hypothesis, the number of rounds used in Line 23 is at most $T(\xi(m)) \cdot L_Y$. So, it's enough to show that $L_Y \leq 43\log m \cdot L + 28\log^2 m$. We do this by giving a feasible solution of length at most $43\log m \cdot L + 28\log^2 m$ for $\mathcal{I}(G, Y)$: Run the schedule $\mathcal{S}$, then run $\mathcal{L}$, and finally run the schedule $\mathcal{S}$ in the reverse order. The claimed upperbound for the length of this schedule simply follows from Lemma 4.

Now, we prove the claimed bound for $T(m)$. First, we simplify the above recurrence relation and write a slightly weaker version of it:

$$T(m) \leq 16\log^3 m \cdot \frac{m}{\xi(m)} + 1128\log^3 m \cdot T(\xi(m)) \tag{1}$$

Recall that $\xi(m) = 2^{\log m - \sqrt{\log m}}$. Use the induction hypothesis to bound the right-hand side of (1) by

$$\leq 16\log^3 m . (2^{\sqrt{\log m}} + 2^{7 + \epsilon \log\log \xi(m) . \sqrt{\log \xi(m)}})$$

$$\leq 2^{12 + 3\log\log m + \epsilon \log\log \xi(m) \cdot \sqrt{\log \xi(m)}}$$

where in the last inequality, we have used the fact that $\sqrt{\log m} \leq 7+\epsilon \log\log \xi(m). \sqrt{\log \xi(m)}$ for all $m \geq 1$. So, the proof is complete if we show that

$$12 + 3 \log\log m + \epsilon \log\log \xi(m) \cdot \sqrt{\log \xi(m)} \leq \epsilon \log\log m. \sqrt{\log m}$$

Observe that:

$$12+3 \log\log m+\epsilon \log\log \xi(m)\cdot \sqrt{\log \xi(m)} \leq \epsilon \log\log m\cdot \left(\frac{12}{\epsilon \log\log m} + \frac{3}{\epsilon} + \sqrt{\log \xi(m)} \right) \tag{2}$$

Since we have $\sqrt{\log m - \sqrt{\log m}} \leq \sqrt{\log m} - 0.5$ for all $m \geq 4$, then we can bound the right-hand side of (2) by

$$\leq \epsilon \log\log m \cdot \left(\frac{12}{\epsilon \log\log m} + \frac{3}{\epsilon} + \sqrt{\log m} - 0.5 \right) \tag{3}$$

And since for any fixed $\epsilon > 6$, there exists a fixed positive integer m_ϵ such that $\frac{12}{\epsilon \log\log m} + \frac{3}{\epsilon} < 0.5$ for all $m \geq m_\epsilon$, then we can bound (3) by $\epsilon \log\log m \cdot \sqrt{\log m}$, which finishes the proof.

C The Capacitated Multicast Problem

In this section, we bring in the notion of *capacity of a vertex* to our model for the Multicast Problem, and allow a vertex to be in possibly more than a single call in each round; the maximum number of phone calls that a vertex can have in each round is called the *capacity* of the vertex.

Definition 3. *In the* **Capacitated Multicast Problem** *(CM) we are given an instance of the Multicast problem along with an integer c_v for each vertex $v \in V$ as its capacity. The only difference with the Multicast Problem is that here, each vertex can be in up to c_v phone calls in a round, i.e. in each round, we can pick a subgraph H of G such that the degree of each vertex v in H is at most c_v, and arrange phone calls between the endpoints of all the edges in H. Note that we can assume w.l.o.g. that the capacities c_v are all at least one since we can delete nodes that have zero capacity from the problem.*

In this section, we present an $O(\log k)$-approximation for the Capacitated Multicast Problem.

C.1 Preliminaries

First, we need to define a new notion of b-matchings due to the presence of capacities. Given that $\boldsymbol{c}$ is a capacity vector for the vertices of Y, i.e. a $|Y|$-dimensional vector of positive integers such that $\boldsymbol{c}_y$ denotes the capacity of the vertex $y \in Y$, we define a $\boldsymbol{c}$-matching in H to be a subset M of the edges of H such that each vertex of X is incident to exactly one edge of M and each vertex $y \in Y$ is incident to at most $\boldsymbol{c}_y$ edges of M.

We also need to refine the definition of the degree of a spider. When we have capacities c_v on the vertices, we can state a lemma similar to Lemma 1. Define the relative degree of a spider S, denoted by $rdeg(S)$, to be $\lceil \frac{deg(S)}{c_v} \rceil$ where v is the center of the spider. Then we have:

Lemma 5. *Using a non-lazy schedule, the center of a spider S can send (broadcast) a message to the rest of its vertices in $rdeg(S) + len(S) - 1$ rounds.*

C.2 The Algorithm

Our algorithm is an adaptation of Algorithm *Multicast* and the only difference between our algorithm and Algorithm *Multicast* is in Phase 4. Before proceeding to more details about Phase 4, first verify that Phases 1-3 are still valid and can be executed with the presence of capacities. Particularly in Phase 3, all we do is using matchings for sending the message through the paths in $\mathcal{P}$, and this is possible since all the capacities are at least 1.

Our goal in Phase 4, assuming that the vertices in $V(\mathcal{P})$ have received the message, is to inform the rest of the terminals in $O(L)$ rounds.

To do so, we find a family of vertex-disjoint spiders such that each of them has a length at most $2L$ and a center belonging to $V(\mathcal{P})$, moreover, we need the spiders to contain all the terminals in $R \backslash V(\mathcal{P})$. Assuming that $\mathcal{S}$ is such a family of spiders, by Lemma 5 we can inform the set $R \backslash V(\mathcal{P})$ in at most $rdeg(\mathcal{S}) + len(\mathcal{S}) - 1$ rounds, where $rdeg(\mathcal{S}) = \max_{S \in \mathcal{S}} rdeg(S)$. We show we can find a family $\mathcal{S}$ with $rdeg(\mathcal{S}) \leq L$ and $len(\mathcal{S}) \leq 2L$, which implies the set $R \backslash V(\mathcal{P})$ can be informed in at most $3L - 1$ rounds in Phase 4. The other parts of the analysis will be identical to the analysis of Algorithm *Multicast*. So, in the rest of this section, we only show how to find the desired family of spiders.

Construct the bipartite graph $H[R \backslash V(\mathcal{P}), V(\mathcal{P})]$ similar as before, but instead of finding the smallest integer b such that H has a b-matching, find the smallest b such that H has a $(b \cdot \boldsymbol{c})$-matching, where $\boldsymbol{c}$ is the capacity vector for the vertices in $V(\mathcal{P})$. Then, using the $(b \cdot \boldsymbol{c})$-matching, construct the family of spiders $\mathcal{M}$ identical to the way we construct them in subsection 5.2. Following the proof of Lemma 2, it can be seen that $len(\mathcal{M}) \leq 2L$, and it only remains to show that $rdeg(\mathcal{M}) \leq L$.

Consider the optimal multicast schedule which uses exactly L rounds. Let E' be the subset of the edges of G which are used in the optimal schedule and G' be the subgraph of G with E' as its edge set. Note that G' is not necessarily a tree as in the previous section; however, the maximum degree of any node v in E' is $c_v \cdot L$ by the optimality of the schedule. Now, for each $v \in R \backslash V(\mathcal{P})$, define M'_v to be an arbitrary shortest path in G' from v to $f(v)$, where $f(v) = \arg\min_{u \in V(\mathcal{P})} d_{G'}(v, u)$.

We prove that $\displaystyle\bigcup_{v \in R \backslash V(\mathcal{P})} (v, f(v))$ is an $(L \cdot \boldsymbol{c})$-matching in H, which shows the existence of a $(b \cdot \boldsymbol{c})$-matching in H with $b \leq L$, which implies $rdeg(\mathcal{M}) \leq L$. To prove the claim, just note that the family of paths $\{M'_v : v \in R \backslash V(\mathcal{P})\}$ in G' are edge-disjoint (otherwise, it contradicts the maximality of $\mathcal{P}$). This means for any

$v \in V(\mathcal{P})$, there are no more than $c_v \cdot L$ of these paths with the same endpoint v, since otherwise, the degree of v in G' is more than $c_v \cdot L$, a contradiction. Consequently, $\bigcup_{v \in R \setminus V(\mathcal{P})} (v, f(v))$ is an $(L \cdot \boldsymbol{c})$-matching in H. This proves the claim.

D The Hypercast Problem

In this section, we study the problem with conference calls, i.e. calls involving (possibly) more that two, rather than only two, persons. We formally define the Hypercast problem as follows.

Definition 4. *In the* **Hypercast** *problem, we are given a hypergraph $G(V, E)$ where there can be a conference call between two or more nodes if G has a hyperedge containing* exactly *these nodes. Similar to the Multicast problem, we are also given a source vertex r and a set of terminals R. Our goal is to deliver a message from the source vertex r to all the terminals in the minimum number of rounds. To do this, the vertices of the graph can communicate in rounds: In each round, we pick a matching of G, i.e. a set of vertex-disjoint edges, and for each edge of the matching, we arrange a conference call containing all the vertices in that edge. If any of the vertices in the conference call knows the message, the others will also know it after the call.*

In this section, we present an $O(\log k \cdot \log n \cdot D)$-approximation for the hypercast problem, where D denotes the maximum size of an edge in the hypergraph G. Unlike the algorithm for Multicast Problem which was purely combinatorial, this algorithm needs to set up a Linear Program for designing some parts of the multicast schedule.

The dependence on D in the approximation factor is natural due to our strong hardness results for the Hypercast Problem (see Section I): we prove that for any constant $\epsilon > 0$, the Hypercast Problem is $\Omega(n^{1-\epsilon})$-hard. But this hardness result, as our algorithm also suggests, only holds for large values of $D \geq \Omega(n^{1-\epsilon})$. The more natural and interesting case though, is when D is small, for which we provide a hardness ratio of $\Omega(D^{1/3})$ under Khot's 2-to-1 conjecture.

D.1 Preliminaries: Spiders and Hypergraphs

The set of vertices and (hyper)edges of a hypergraph G are respectively denoted by $V(G)$ and $E(G)$. We say an edge e intersects a subset of vertices $S \subseteq V(G)$ (or another edge e') if e contains at least one vertex of S (one vertex of e'). With abuse of notation, we denote this by $e \cap S \neq \emptyset$ (respectively $e \cap e' \neq \emptyset$). A subset $M \subseteq E(G)$ is called a matching if any two edges in M have an empty intersection.

A path P is a sequence of edges such as $e_0, \ldots, e_m$ where $e_i \cap e_j \neq \emptyset$ iff $|i - j| \leq 1$. The length of P is denoted by $len(P)$ and is equal to m. Having the definition of the length of a path, the notions of connectivity, distance and diameter in a hypergraph are trivially adapted from simple graphs.

A spider S is a family of paths $P_1, \ldots, P_k$ such that the first edge of all of them is the same, and moreover, they will be a family of vertex-disjoint paths if their first edge is deleted. The set of vertices in the first edge (or simply, the first edge, when it's clear from the context) is called the center of S. By adapting the definition of length of a spider in simple graphs, we define $len(S) = \max_i len(P_i)$.

D.2 Outline of the Algorithm

Our algorithm is similar to Algorithm *Multicast* presented in Section 5; it is a recursive algorithm with 4 phases. In Phase 1 of the algorithm, we reduce the given instance to a smaller instance. In Phase 2, we solve the smaller instance recursively and as the result, inform a subset of the terminals. And finally in Phases 3 and 4, we inform the rest of the vertices that didn't receive the message in Phase 2. First we explain each of these phases briefly, and then present the full description of the algorithm and its analysis.

Phase 1. This phase starts with finding a family of vertex disjoint paths $\mathcal{P}$ each of length (number of the hyperedges) at most $4L$ such that the first and last edge of each path intersects R. Similar to Algorithm *Multicast*, we find these paths greedily, i.e. we start with $\mathcal{P} = \emptyset$ and using any shortest path algorithm for hypergraphs, we search for a new path in $G[V(G) \backslash V(\mathcal{P})]$ which connects two of the terminals. Moreover, length of the path must be at most $4L$. We continue until we can add no more such paths to $\mathcal{P}$. Then, pick an arbitrary terminal from the first edge of each path and let the obtained set of vertices be R'.

Phase 2. Solve the multicast problem for the new set of terminals R' recursively and run the obtained schedule. (So, all the vertices in R' will receive the message by the end of this phase.)

Phase 3. Inform all the vertices belonging to $V(\mathcal{P})$ in $4L$ rounds. This is possible since in Phase 2, we have already informed at least one vertex of each path in $\mathcal{P}$.

Phase 4. For each of the uninformed terminals, namely $v \in R \backslash V(\mathcal{P})$, find a path M_v which connects v to one of the informed vertices (note that the set of the informed vertices is currently $V(\mathcal{P})$). These paths will be guaranteed to satisfy the following properties:

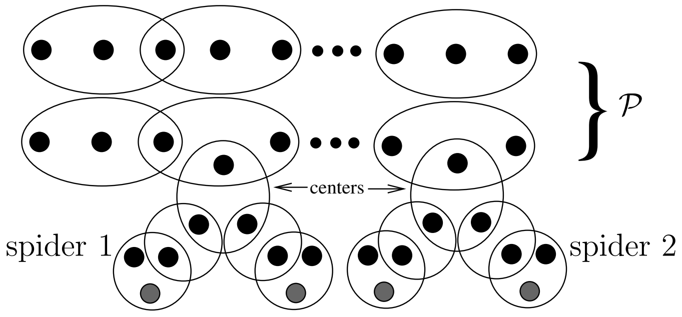

Fig. 3. The gray vertices are in $R \backslash V(\mathcal{P})$. Despite this figure, centers of the spiders can possibly intersect.

1. For each $v \in R \backslash V(\mathcal{P})$, the first edge of M_v contains v and the last edge intersects $V(\mathcal{P})$. Moreover, no other edge of M_v intersects $V(\mathcal{P})$.
2. The length of each path is at most $2L$.
3. The paths would be vertex disjoint if the last edge of each path is removed.

In other words, $M = \bigcup_{v \in R \backslash V(\mathcal{P})} M_v$ is a union of spiders of length at most $2L$ and we will inform the vertices in $R \backslash V(\mathcal{P})$ using these spiders (see Figure 3). However, this is not as simple as it is in Section 5, since unlike there, the centers of spiders may intersect. So, bounding the degree and length of each of the spiders is not enough to bound the total number of rounds needed for informing $R \backslash V(\mathcal{P})$. We need the family of spiders to satisfy an additional constraint, which we call z-disjointness. We define this concept and then describe Phase 4 for the Algorithm. The other 3 Phases remain unchanged.

D.3 Disjoint and Fast Spiders

Definition 5. *A family $\mathcal{F}$ of spiders is z-*disjoint *if they satisfy the following conditions.*

- *If 2 edges which belong to 2 different spiders intersect, then both of them are the centers (also called first edges earlier) of spiders.*
- *No vertex is in more than z of the centers.*

The following lemma gives an upperbound on the broadcast time in a z-disjoint family of spiders. Recall that for a family $\mathcal{F}$ of spiders, $deg(\mathcal{F}) = \max_{S \in \mathcal{F}} deg(\mathcal{F})$ and $len(\mathcal{F}) = \max_{S \in \mathcal{F}} len(S)$.

Lemma 6. *We are given a z-disjoint family of spiders $\mathcal{F}$, such that for each spider, at least one vertex in its center is informed. Assuming that D is the maximum size of a hyperedge, we can find a broadcast schedule for $\mathcal{F}$ with a length at most $len(\mathcal{F}) + 2zD$ in polynomial time.*

Proof. Let $\mathcal{H}$ be a hypergraph where $V(\mathcal{H})$ is the set of the vertices which belong to the center of at least one spider in $\mathcal{F}$. Also, let $E(\mathcal{H})$ be the set of the centers of spiders in $\mathcal{F}$, i.e. each of the centers is a hyperedge in $\mathcal{H}$. We claim $\chi'(\mathcal{H}) \leq 2zD$. If we prove this claim and find such a coloring in polynomial time, we can easily convert it to a broadcast schedule for $\mathcal{F}$ with a length at most $len(\mathcal{F}) + 2zD$ as follows: Dedicate one round to each color, e.g. assume that rounds $1, \ldots, \chi'(G)$ are respectively dedicated to the colors $1, \ldots, \chi'(G)$. Then for each color i, pick the subset of the edges with color i as the matching used in round i. Having that each hyperedge in $\mathcal{H}$ contains an informed vertex implies we can inform $V(\mathcal{H})$ in $\chi'(\mathcal{H})$ rounds. Also, we can inform the rest of the vertices in $\mathcal{F}$ in an additional $len(\mathcal{F})$ rounds, due to the disjointness of the spiders legs. This means the length of the obtained broadcast schedule is at most $\chi'(\mathcal{H}) + len(\mathcal{F})$. To complete the proof, we show that $\chi'(\mathcal{H}) \leq 2zD$ and find a coloring with at most $2zD$ colors in polynomial time.

Let the strong degree of a vertex v, denoted by $\bar{d}(v)$, be the summation, over all $u \neq v$, of the number of hyperedges that contain both v and u. Also, let $\bar{\Delta}(\mathcal{H}) = \max_{v \in V(\mathcal{H})} \bar{d}(v)$. It's a well-known fact that $\chi'(\mathcal{H}) \leq 2\bar{\Delta}(\mathcal{H})$, and in fact, we can find such a coloring by a greedy algorithm in polynomial time [4].

Since $\mathcal{H}$ is loop-less, $\bar{d}(v)$ is at most the summation of the size of the hyperedges that contain v. Also, note that $\mathcal{F}$ is z-disjoint, which means v is in at most z hyperedges in $\mathcal{H}$. It implies $\bar{\Delta}(\mathcal{H}) \leq zD$ since the maximum size of a hyperedge is at most D. Consequently, $\chi'(\mathcal{H}) \leq 2zD$.

Recall that our goal in Phase 4 is informing the set of uninformed vertices, $R \backslash V(\mathcal{P})$, using a family $\mathcal{F}$ of spiders. If we can find a z-disjoint family of spiders $\mathcal{F}$ such that both z and $len(\mathcal{F})$ are small, then by Lemma 6 we get a short broadcast schedule for $\mathcal{F}$, and consequently, a short schedule for the original hypercast instance. The properties required for the family $\mathcal{F}$ are formally defined below:

Definition 6. *A family $\mathcal{F}$ of spiders is called* fast *if it satisfies the following properties:*

i. *$len(\mathcal{F}) \leq 2L$.*
ii. *$\mathcal{F}$ is z-disjoint for $z = O(L \log n)$*
iii. *The center of each spider in $\mathcal{F}$ has a non-empty intersection with $V(\mathcal{P})$.*
iv. *Each vertex in $R \backslash V(\mathcal{P})$ belongs to exactly one of the spiders in $\mathcal{F}$.*

Lemma 7. *If we can find a fast family of spiders for any given Hypercast instance, then we can design an $O(\log k \cdot \log n \cdot D)$-approximation for the Hypercast problem.*

Proof. Our recursive algorithm for the Hypercast problem reduces the number of terminals by at least a factor of 2 in each level of the recursion. This is easily observable by the definition of R' in Phase 2 of the Algorithm: from the two terminals connected by a path in $\mathcal{P}$, only one of them is in R'. So, the depth of the recursion is at most $\log k$.

The proof is complete if we show that each level of the recursion takes at most $O(LD \cdot \log n)$ number of rounds. First, see that the cost of the optimal solution does not increase when the number of terminals decreases in the next levels of recursion. So, if we find a fast family $\mathcal{F}$ at each level of the recursion, then by Lemma 6 we spend at most $O(LD \cdot \log n)$ rounds in that level. This means the length of the obtained schedule for the given Hypercast instance would be at most $O(LD \cdot \log k \cdot \log n)$.

So, by Lemma 7, all we need to do in Phase 4 is finding a fast family $\mathcal{F}$ and a broadcast schedule for it. In the rest of this section, we describe an algorithm for finding a fast family of spiders, given that $\mathcal{P}$ is maximal and $V(\mathcal{P})$ is informed.

D.4 Finding a Fast Family of Spiders

The idea for finding the family of spiders is similar to Section 5, except that instead of a bipartite graph, we construct a tripartite graph $H[A, B, C]$, where $A = R \backslash V(\mathcal{P})$, B has a vertex for every hyperedge which intersects both $V(\mathcal{P})$ and $V(G) \backslash V(\mathcal{P})$, and $C = V(G)$. Note that B is in fact the set of potential centers for the family of spiders $\mathcal{F}$.

The edges of H, which lie only between the partitions A, B and B, C, are defined as follows: There is an edge between the vertices $v \in A$ and $e \in B$ iff there is a path in $G[V(G) \backslash V(\mathcal{P})]$ of length at most $2L$, such that its first edge contains v and its last edge intersects with the hyperedge e. Also, there is an edge between the vertices $e \in B$ and $u \in C$ iff the hyperedge e contains the vertex u.

The induced bipartite graph between the partitions A, B, denoted by $H[A, B]$ is used in a way similar to Section 5.2, i.e. any $|A|$-matching in $H[A, B]$, e.g. M, represents a family of spiders, and more precisely, any connected component in M, represents a spider. Observe that any connected component of M forms a star. The vertex e in the center of the star, represents the center of the spider, and the edges of the star, represent the legs of the spider, i.e. the paths which are attached to the center of the spider. Recall that each edge of $H[A, B]$ is associated with a path in G. Any two legs, whether or not belonging to the same spider, are vertex-disjoint, due to the maximality of $\mathcal{P}$.

The family of spiders represented by M, namely $\mathcal{F}$, is z-disjoint for $z = |B|$. It is easy to verify that $\mathcal{F}$ is z-disjoint if each vertex in C is adjacent to at most z of the centers of the stars in M.

For $\mathcal{F}$ to be a fast family, we only need to choose M in such a way that $\mathcal{F}$ becomes z-disjoint for $z = O(L \log n)$, and this is where we use partition C of H. To be more formal, for any subset $S \subseteq B$, define the weight of S, denoted by $w(S)$, to be $\max_{v \in C} |S \cap N(v)|$. Now, finding a fast family $\mathcal{F}$ can be reformulated as finding a subset $S \subseteq B$, such that $N(S) = A$ and $w(S) \in O(L \log n)$. If we find such a subset S, then we can easily find a fast family by choosing any $|A|$-matching in $H[A, S]$. The family of spiders associated with the $|A|$-matching would be $w(S)$-disjoint, and consequently, fast.

All that remains to do is finding such a subset S. First, we show that there exists such S, and then we present an algorithm for finding it in Section D.5.

Lemma 8. *There exists a subset $S \subseteq B$ with $N(S) = A$ and $w(S) \leq L$.*

Proof. Let $\mathcal{L}$ be an optimal schedule for the given Hypercast instance. Note that $|\mathcal{L}| = L$. Also, let J be a hypergraph where $V(J) = V(G)$ and $E(J)$ is the set of the hyperedges used in $\mathcal{L}$. Delete the isolated vertices of J. Then clearly, J is connected and $diam(J) \leq 2L$. So, for each vertex $v \in R\backslash V(\mathcal{P})$, J should contain a path of length at most $2L$ connecting v to a vertex in $V(\mathcal{P})$. If there are many such paths, then select one of them arbitrarily and define $g(v)$ to be the first edge on this path (closest to v) which intersects both $V(\mathcal{P})$ and $R\backslash V(\mathcal{P})$.

By the definition of partition B, it contains a vertex for each hyperedge $g(v)$, which we denote by $g'(v)$. We prove that $S = \bigcup_{v \in R\backslash V(\mathcal{P})} g'(v)$ satisfies the requirements of the lemma: For all $v \in R\backslash V(\mathcal{P})$, since there is an edge between v and $g(v)$ in $H[A, B]$, then $N(S) = A$. Also, since each vertex $u \in V(G)$ is in at most L of the hyperedges of J, then $|N(u) \cap S| \leq L$ for all $u \in C$, which means $w(S) \leq L$.

D.5 An LP-Rounding Approach for Finding Fast Spiders

Given a tripartite graph $H[A, B, C]$, we want to find a subset $S \subseteq B$ which minimizes $w(S)$ and satisfies $N(S) = A$. Here we present a $8 \ln n$-approximation for this problem. This result, along with Lemma 8 implies that we can find a subset S such that $w(S) \leq 8L \ln n$ and $N(S) = A$.

To solve the above optimization problem, first we solve the fractional version of it by writing a linear program and then, using randomized rounding, we round the fractional solution of the linear program and obtain an integral one. To write this LP, a variable x_i is associated with each vertex $i \in B$.

$$\begin{aligned} \min\ & z \\ \text{s.t.}\ & \sum_{i \in N(u)} x_i \geq 1,\ \forall u \in A \ (1) \\ & z \geq \sum_{j \in N(v)} x_j,\ \forall v \in C \ (2) \\ & 0 \leq x_i \leq 1, \quad \forall i \in B \ \ (3) \end{aligned}$$

Note that there is a one-to-one correspondence between the *integral* solutions of this LP and the subsets S such that $N(S) = A$. To see this, let x be an integral solution, then the set $S = \{i : x_i = 1\}$ would be a feasible solution for the original problem, because constraint (1) is enforcing $N(S) = A$. The reverse direction can also be verified easily: if $N(S) = A$ for a subset S, then $x = \mathbb{1}_S$, i.e. the characteristic vector of S, is a feasible integral solution for the LP. Also, note that constraint (2) is enforcing $z = w(S)$. However, we can't solve the integer program in polynomial time, and so, we relax the program by adding constraint (3) and use the optimal solution of the obtained LP as a lower bound on the optimal integral solution.

Let $\bar{x}$ be the optimal solution (assignment of values to the variables) for this LP. Also, let the optimal objective values for the integer program and the (fractional) LP be denoted by OPT, OPT_f respectively. We round $\bar{x}$ to obtain a feasible integral solution $\mathring{x}$ with objective value at most $8 \ln n \cdot \max\{1, OPT_f\}$. This gives an $(8 \ln n)$-approximation since $OPT_f \leq OPT$ and $1 \leq OPT$. The rounding procedure is described below.

The Rounding Procedure. For each $j \in B$, flip a coin $2 \ln n$ times independently where the coin comes up heads with probability $\bar{x}_j$. If heads were observed at least once, then pick j to be in S. More formally, we create $2 \ln n$ independent random variables $x_{j,i}$ such that $x_{j,i} = 1$ with probability $\bar{x}_j$ and $x_{j,i} = 0$ with probability $1 - \bar{x}_j$. We pick j to be in S iff $\sum_{i=1}^{2 \ln n} x_{j,i} \geq 1$.

Lemma 9. *The rounding procedure outputs a set S such that with probability at least $1 - \frac{2}{n}$ both of the following conditions are satisfied: $N(S) = A$, and $w(S) \leq 8 \ln n \cdot \max\{1, OPT_f\}$.*

Proof. First we prove that the $N(S) = A$ with high probability. For any $u \in A$, we compute the probability that u is not covered, i.e. none of the neighbors of u are in S:

$$\Pr[u \text{ is not covered}] = \prod_{j \in N(u)} (1 - \bar{x}_j)^{2 \ln n} \leq \prod_{j \in N(u)} e^{-\bar{x}_j 2 \ln n} \leq \frac{1}{n^2}$$

An application of the union bound over the nodes implies that the probability of one of the elements of A not being covered, i.e. $N(S) \neq A$, is at most $\frac{1}{n}$.

For the second condition, we claim that $w(S) \leq 8 \ln n \cdot \max\{1, OPT_f\}$ with high probability. To prove this, we fix a vertex $v \in C$ and compute an upper bound on the probability of having $|N(v) \cap S| > 8 \ln n \cdot \max\{1, OPT_f\}$. Then, we prove the main claim using a union bound over all v.

More formally, for any $v \in C$ we prove that

$$\Pr\left[\sum_{j \in N(v)} \mathring{x}_j > 2\alpha \ln n \cdot \max\{1, \bar{z}\}\right] \leq \frac{1}{n^2} \tag{4}$$

where $\alpha = 4$ is a fixed constant and $\bar{z} = OPT_f$. First, observe that

$$\begin{aligned}\Pr\left[\sum_{j \in N(v)} \mathring{x}_j > 2\alpha \ln n \cdot \bar{z}\right] &\leq \Pr\left[\sum_{j \in N(v)} \mathring{x}_j > 2\alpha \ln n \cdot \sum_{j \in N(v)} \bar{x}_j\right] \\ &\leq \Pr\left[\sum_{j \in N(v)} \sum_{i=1}^{2 \ln n} x_{j,i} > \alpha \sum_{j \in N(v)} \sum_{i=1}^{2 \ln n} \mathbb{E}[x_{j,i}]\right] \leq \left(\frac{e^{\alpha-1}}{\alpha^\alpha}\right)^\mu\end{aligned} \tag{5}$$

where $\mu = \sum_{j \in N(v)} \sum_{i=1}^{2 \ln n} \mathbb{E}[x_{j,i}]$. Note that (5) is a direct consequence of the Chernoff bound. Now we consider two cases: $\mu \geq 2 \ln n$ and $\mu < 2 \ln n$.

In the first case, having $\mu \geq 2\ln n$ implies $\left(\frac{e^{\alpha-1}}{\alpha^{\alpha}}\right)^{\mu} \leq \frac{1}{n^2}$, and this proves (4). On the other hand, if $\mu < 2\ln n$ then we have

$$\Pr\left[\sum_{j\in N(v)} \mathring{x}_j > 2\alpha\ln n\right] \leq \Pr\left[\sum_{j\in N(v)}\sum_{i=1}^{2\ln n} x_{j,i} > \frac{2\alpha\ln n}{\mu}\cdot\mu\right] \leq \left(\frac{e^{\beta-1}}{\beta^{\beta}}\right)^{\mu} \quad (6)$$

where $\beta = \frac{2\alpha\ln n}{\mu}$. Now, since $\mu < 2\ln n$ we have

$$\left(\frac{e^{\beta-1}}{\beta^{\beta}}\right)^{\mu} \leq \left(\frac{e}{\alpha}\right)^{2\alpha\ln n} \leq \frac{1}{n^2}$$

This fact, and (6) together imply the correctness of (4) in the second case as well. By a union bound over all $v \in C$ we have

$$\Pr\left[w(S) > 2\alpha\ln n\cdot\max\{1,\bar{z}\}\right] \leq \sum_{v\in C}\Pr\left[\sum_{j\in N(v)} \mathring{x}_j > 2\alpha\ln n\cdot\max\{1,\bar{z}\}\right] \leq \frac{1}{n}$$

which is due to (4). Consequently, the events $w(S) > 8\ln n\cdot\max\{1, OPT_f\}$ and $N(S) \neq A$ each happen with probability at most $\frac{1}{n}$. This proves the lemma.

E Asymmetric Multicast Problem

In this section, we look at a variation of the Multicast Problem where the phone-calls are not bidirected. They are directed in the sense that only one vertex will be the sender, and the other one will be the receiver, i.e. a vertex u can call another vertex v and send (any number of) messages to v, but then v can not send any messages to u in the same phonecall. This variation is formally defined below:

Definition 7. *In the Asymmetric Multicast Problem, each vertex v has an out-capacity c_v^- as well as an in-capacity c_v^+, which are respectively the number of the vertices that v can send messages to, and receive messages from, in a single round. The objective is identical to the objective of the Multicast Problem.*

An asymmetric variation of the Multi-commodity Multicast Problem can also be defined in a natural way, which we call the *Asymmetric Multi-commodity Multicast* Problem (AMM). It is worth mentioning that a natural extension of Algorithm *MM* can solve AMM as long as all the capacities are non-zero. We do not state the algorithm, but the idea of the extension is very similar to the idea we used for designing the algorithm for the Capacitated Multicast Problem.

Below, we prove a lemma for comparing the lengths of the optimum schedules for an AMM instance and its corresponding MM instance. This Lemma will be used later in Section F.

Lemma 10. *Assume we are given an AMM instance $\mathcal{I}_A$ with the set of demand pairs P such that $c_v^- = c_v^+ = 1$ for all $v \in V(G)$. Define $\mathcal{I}(P, G)$ to be the corresponding MM instance. Also, let L_A, L respectively denote the length of optimal schedules for $\mathcal{I}_A, \mathcal{I}$. Then, we have $\frac{L}{3} \leq L_A \leq 2L$. Moreover, any schedule of length l for $\mathcal{I}$ can be converted to a schedule of length $2l$ for $\mathcal{I}_A$ in polynomial time, and also, any schedule of length l_A for $\mathcal{I}_A$ can be converted to a schedule of length $3l_A$ for $\mathcal{I}$ in polynomial time.*

Proof. First, observe that any schedule $\mathcal{S}$ for $\mathcal{I}$ can be easily turned into a schedule $\mathcal{S}_A$ for $\mathcal{I}_A$ as follows. For each round in $\mathcal{I}$, we have exactly two rounds in $\mathcal{I}_A$: In each of these two rounds, we use the same matching which is used in $\mathcal{I}$, e.g. the matching M, except that the edges of M will be used in different directions in each of these two rounds. Consequently, $L_A \leq 2L$.

To prove $\frac{L}{3} \leq L_A$, we show that given any feasible schedule $\mathcal{S}_A$ for the instance $\mathcal{I}_A$, we can construct a schedule $\mathcal{S}$ for the instance $\mathcal{I}$ such that $|\mathcal{S}| \leq 3|\mathcal{S}_A|$. This time, for each round in $\mathcal{S}_A$ we will have 3 rounds in $\mathcal{S}$.

To see this, first fix any arbitrary round in $\mathcal{S}_A$, and let M be the subset of the (directed) edges used in that round. Note that M is a union of paths and cycles, due to the fact that all the in-capacities and out-capacities are 1. Consequently, we can decompose the edges of M into 3 matchings, M_1, M_2, M_3, and use each of these matchings in one of the 3 rounds in $\mathcal{S}$. Clearly, $|\mathcal{S}| \leq 3|\mathcal{S}_A|$, and also, all of the demand pairs that were satisfied in $\mathcal{S}_A$ will also be satisfied in the obtained schedule $\mathcal{S}$.

F An Algorithm for Small Numbers of Terminals

Given a Multi-commodity Multicast instance, we present an algorithm which finds a schedule of length $O(|P| + D)$ where D denotes $diam_P(G)$. Instead of solving this problem directly, we solve the Asymmetric MM instance where $c_v^+ = c_v^- = 1$ for all $v \in V(G)$, and then by Lemma 10, we can convert this schedule to a schedule for the original MM instance. So from now on, we think of the given instance as the Asymmetric MM instance described above.

Before presenting our algorithm, we need a few definitions. Let $m(u, v)$ denote the message that vertex u wants to send to v, given that $(u, v) \in P$. Also, let $M = \{m(u, v) | (u, v) \in P\}$ be the set of all of the messages.

F.1 The Algorithm

Our algorithm has two Phases. In Phase 1, we find a path for each message through which the message is sent. In Phase 2, a non-lazy schedule is used to send all the messages through the paths that were found in Phase 1. A non-lazy schedule in this context, is a schedule in which the messages are greedily sent to the next vertex on their paths, i.e. a message does not wait on a vertex if it can be sent to the next vertex on its path without violating the capacity constraints.

Phase 1. For each message $m(s,t)$ we find an s,t-shortest path. Moreover, these paths are found in such a way that the intersection of any two of them is a path itself (possibly empty). To find such a family of paths, we introduce the notion of *lightness* and choose the lightest path among all the s,t-shortest paths.

Definition 8. *Let $\prec$ be an arbitrary precedence relation defined on $E(G)$. For any path Q, let $\bar{Q}$ be the sequence of the edges of Q sorted in a decreasing order with respect to $\prec$. A path Q is lighter than Q' if $\bar{Q}$ is lexicographically (dictionary ordering) smaller than $\bar{Q}'$. We abuse the notation and denote this by $Q \prec Q'$. Also, we extend this definition in the natural way to compare any two subsets of the edges (not necessarily forming a path).*

Definition 9. *For any two vertices s,t, define the* best *s,t-path to be the lightest path among all the shortest paths between s and t.*

All we do in Phase 1, is finding the best s,t-path for each pair $(s,t) \in P$. We finish the description of this Phase by showing how to find these paths in polynomial time.

Lemma 11. *For any two vertices s,t, the best s,t-path can be found in polynomial time using Dijkstra's Algorithm.*

Proof. The proof has the same analysis as the analysis of Dijkstra's Algorithm. Define the weight of the i-th edge in the precedence relation to be $1+2^{-i-|E(G)|}$ and run Dijkstra's Algorithm to find the shortest (s,t)-path with respect to these weights, the obtained path would be the best (s,t)-path in G.

Phase 2. In Phase 2, we use an arbitrary non-lazy schedule which for all $(s,t) \in P$, sends $m(s,t)$ from s to t through the best s,t-path that was found in Phase 1.

The Algorithm is now completed by describing Phase 2. The schedule produced in Phase 2 is clearly feasible. In the analysis of the algorithm, we bound the length of this schedule by $2|P| + D$. But before moving to the analysis, we further exploit the structure of best paths.

F.2 On the Structure of Best Paths

Proposition 3. *Assume $A, B, C, D \subseteq E(G)$ such that $A \prec B$ and $C \prec D$. Moreover, we have $A \nsubseteq C, C \nsubseteq A, B \nsubseteq D$ and $D \nsubseteq B$. Then $(A \cup C) \prec (B \cup D)$.*

Lemma 12. *For any 4 (not necessarily disjoint) vertices s, s', t, t', the intersection of the best s,t-path and the best s',t'-path is a path itself (possibly empty).*

Proof. Let the best s,t-path and the best s',t'-path be respectively denoted by $P_{s,t}$ and $P_{s',t'}$. For the sake of contradiction, assume the intersection of $P_{s,t}$ and $P_{s',t'}$ is not a path, then, it is easy to verity that there exist two vertices a, b satisfying the following properties:

1. $a, b \in V(P_{s,t}) \cap V(P_{s',t'})$.
2. If we denote the path connecting a, b in $P_{s,t}$ by Q, and denote the path connecting a, b in $P_{s',t'}$ by Q', then Q and Q' are of the same length, $E(Q), E(Q')$ are non-empty, and $E(Q) \nsubseteq E(Q')$ and $E(Q') \nsubseteq E(Q)$.

Note that Q, Q' must be of the same length due to the fact that $P_{s,t}, P_{s',t'}$ are shortest paths. Let $\hat{P}_{s,t}$ be the path from s to t which is similar to $P_{s,t}$ except that it uses Q' instead of Q for connecting a and b. Similarly, Let $\hat{P}_{s',t'}$ be the path from s' to t' which is similar to $P_{s',t'}$ except that it uses Q instead of Q' for connecting a and b. Observe that $\hat{P}_{s,t}$ and $\hat{P}_{s',t'}$ can not have repeated vertices, again due to the fact that $P_{s,t}, P_{s',t'}$ are shortest paths.

We claim that $P_{s,t} \prec \hat{P}_{s,t}$ and $P_{s',t'} \prec \hat{P}_{s',t'}$. To verify this claim, first note that $E(P_{s,t}) \neq E(\hat{P}_{s,t})$ and $E(P_{s',t'}) \neq E(\hat{P}_{s',t'})$, due to the fact that $E(Q) \nsubseteq E(Q')$ and $E(Q') \nsubseteq E(Q)$. Also, see that $\hat{P}_{s,t}$ and $\hat{P}_{s',t'}$ are both shortest, but not the lightest shortest paths. This implies $E(P_{s,t}) \prec E(\hat{P}_{s,t})$ and $E(P_{s',t'}) \prec E(\hat{P}_{s',t'})$. Now, since the conditions stated in Proposition 3 apply, we can use this proposition and imply

$$(E(P_{s,t}) \cup E(P_{s',t'})) \prec (E(\hat{P}_{s,t}) \cup E(\hat{P}_{s',t'}))$$

But this is a contradiction, since we have

$$(E(P_{s,t}) \cup E(P_{s',t'})) = (E(\hat{P}_{s,t}) \cup E(\hat{P}_{s',t'}))$$

by the definition of $\hat{P}_{s,t}$ and $\hat{P}_{s',t'}$.

Analysis of the Algorithm. We say a message m is *out-waiting* for another message m' in round t, if the following conditions hold in that round:

i. m, m' have not reached their destinations and currently, they are on the same vertex u.
ii. The next vertices that m, m' should be sent to, are respectively v, v', and we have $v \neq v'$.
iii. In round t, we send m' to the next vertex by a phone-call from u to v'.

Similarly, we say a message m is *in-waiting* for another message m' in round t, if the following conditions hold in that round:

i. m, m' have not reached their destinations and currently, they are on two different vertices, u, u' respectively.
ii. The vertex v is the next vertex that m, m' should be sent to.
iii. In round t, we send m' to the vertex v by a phone-call from u' to v.

The *in-delay* of a message $m \in M$ in a given Multicast schedule Π is defined as the number of rounds in which m has been in-waiting for at least one other message, and it is denoted by $\xi^+_\Pi(m)$, or simply $\xi^+(m)$, whenever Π is clear from the context. Similarly, the *out-delay* of a message m in a given Multicast schedule Π is defined as the number of rounds in which m has been out-waiting for at least one other message, and is denoted by $\xi_{\Pi}^{-}(m)$. The *delay* of a message m, denoted by $\xi_{\Pi}(m)$, is simply equal to $\xi_{\Pi}^{+}(m) + \xi_{\Pi}^{-}(m)$.

The length of any non-lazy schedule Π is then clearly at most $\max_{(s,t)\in P} d_G(s,t) + \xi(m(s,t))$, or simply, $D + \xi$ where ξ denotes $\max_{(s,t)\in P} \xi(s,t)$.

To bound the length of the non-lazy schedule obtained in Phase 2 by $2|P|+D$, its enough to show that $\xi \leq 2|P|$.

Lemma 13. *For any message m we have $\xi(m) \leq 2|P|$.*

Proof. We prove the claim by showing that $\xi^{+}(m), \xi^{-}(m) \leq |P|$. To do so, we show that a message m would not be out-waiting for any other message m' in more than one round of the non-lazy schedule, which implies $\xi^{-}(m) \leq |P|$. Similarly, we show that m would not be in-waiting for another message m' in more than one round of the schedule, and imply that $\xi^{+}(m) \leq |P|$.

Assume m is out-waiting for m', and let Q, Q' respectively be the best paths associated with m, m'. Since m is out-waiting for m', then they are currently on the same vertex, e.g. u, but the vertices right after u on Q, Q' are two different vertices, e.g. respectively v, v'. Lemma 12 implies the uniqueness of u. In other words, it says that for any fixed m, m', there is no more than one 5-tuple (m, m', u, v, v') satisfying the properties above. Consequently, m would out-wait for m' in at most one round, which means $\xi^{-}(m) \leq |P|$.

Now, assume m is in-waiting for m', and let Q, Q' respectively be the best paths associated with m, m'. Since m is in-waiting for m', then they are currently on different vertices, e.g. u, u', but the vertices right after u, u' on Q, Q' are the same, e.g. the vertex v. Again, Lemma 12 implies the uniqueness of the 5-tuple (m, m', u, u', v). Consequently, m would in-wait for m' in at most one round, which means $\xi^{+}(m) \leq |P|$.

Theorem 3. *The length of the non-lazy schedule is at most $2|P| + D$.*

Proof. Recall that length of the schedule is bounded by $\xi + D$. By Lemma 13 we have $\xi \leq 2|P|$, which implies length of the schedule is at most $2|P| + D$.

G An Additive Approximation Algorithm

In this section, we provide an algorithm for the MM Problem which uses no more than $O(\log n \cdot OPT + \sqrt{n}\log^2 n)$ rounds. Before presenting the algorithm, we need the following definition.

Definition 10. *During the running time of a broadcast schedule, a pair of vertices (u, v) is called* active *if either (u, v) or (v, u) is an unsatisfied demand pair, i.e. the message from the source node is not yet received by the sink node. A subset of vertices X is called active, if there exist some active pair (u, v) such that $u \in X$ and $v \notin X$. In particular, a terminal u is called active if there is some vertex v such that (u, v) is active.*

Our algorithm has 4 phases. In Phase 1, we select a set C of at most $\sqrt{n}$ vertices from G, which we call the set of *centers*. In Phase 2 we find a schedule $\mathcal{S}$ with length $O(\sqrt{n})$ which sends each vertex $v \in R \backslash C$ to one of the centers, namely $f(v)$. Overall, our goal in Phases 1 and 2 is to send all the messages to the set of centers. In Phase 3, we solve another MM instance defined on G, in which the set of demand pairs is $P' = \{(f(u), f(v)) | (u, v) \in P\}$. In Phase 4, we simply run the schedule $\mathcal{S}$ in the reverse order to complete the information received by every $f(v)$ to the corresponding v. We describe each of these phases in more details, and then formally present the algorithm.

Phase 1. The goal in this phase is finding and running a schedule which we call $\mathcal{S}_0$. First, find two families of vertex-disjoint connected subgraphs of G, namely $\mathcal{T}$ and $\mathcal{T}'$. The subgraphs in $\mathcal{T}, \mathcal{T}'$ are called the *components* of $\mathcal{T}, \mathcal{T}'$, or simply components, whenever it's clear from the context. These components will be chosen in such a way that we can broadcast within each of them in at most $\sqrt{n}$ rounds, which also means this can be done simultaneously for all of them, since they're vertex disjoint.

To construct $\mathcal{T}$, repeatedly find vertex-disjoint trees of size $\sqrt{n}$: Order the vertices of G arbitrarily and visit the vertices in that order. When visiting a vertex v, run DFS to find a tree of size $\sqrt{n}$ rooted at v, which does share any vertices with the trees (of size $\sqrt{n}$) that are found previously. After visiting all the vertices of G, assume $\mathcal{T} = \{T_1, \ldots, T_k\}$ is the family of the trees we have found and $\mathcal{C} = \{r_1, \ldots, r_k\}$ is the set of their roots where r_i is the root of T_i.

Find an arbitrary spanning tree in each component of $G[V(G) \backslash V(\mathcal{T})]$, and let the obtained set of trees be $\mathcal{T}' = \{T'_1, \ldots, T'_{k'}\}$. Clearly, any tree in $\mathcal{T}'$ has less than $\sqrt{n}$ vertices since otherwise, another tree could have been added to $\mathcal{T}$. So, there exist a broadcast schedule of length at most $\sqrt{n}$ to transmit messages from the root to all nodes in each component in $\mathcal{T}, \mathcal{T}'$. By running this schedule twice, first in the reverse order to the root and then regularly from the root, we can inform all the vertices in a component about the information of the rest of the vertices in that component. Moreover, this can be done simultaneously for all of the components since they are vertex disjoint. Let $\mathcal{S}_0$ be the final schedule accomplishing this. Run the schedule $\mathcal{S}_0$ in Phase 1.

Phase 2. This Phase is a preparation for Phase 3, where we solve a new MM instance defined on the set $\mathcal{C}$ as the set of new terminals. In Phase 2, we construct this instance by sending each active vertex $v \notin \mathcal{C}$ to some vertex in $\mathcal{C}$, namely $f(v)$. Note that this has already been done for all $v \in V(\mathcal{T})$ in Phase 1, i.e. we sent each vertex $v \in V(\mathcal{T})$ to one of the centers.

To accomplish this for all $v \notin V(\mathcal{T})$ as well, we find a schedule $\mathcal{S}$ which sends a vertex from each active components in $\mathcal{T}'$ to another arbitrary vertex in $V(\mathcal{T})$. To find $\mathcal{S}$, we find a b-matching in the bipartite graph $G[V(\mathcal{T}), V(\mathcal{T}')]$ in polynomial time, which saturates exactly one vertex from each active component T'_i, and minimizes b. Recall that b-matchings in bipartite graphs are like regular matchings, except that the degrees on one side of the partition, i.e. $V(\mathcal{T})$ in here, can be as large as b.

Before giving an algorithm for finding the b-matching, we show how to use it to construct $\mathcal{S}$. Assuming that a vertex $v \in V(\mathcal{T}')$ is matched to $m(v)$ in our b-matching, the schedule $\mathcal{S}$ sends a message from each vertex v to $m(v)$. This clearly can be done in b rounds by decomposing the edges of the b-matching into at most b disjoint matchings, so we have $|\mathcal{S}| \leq b$.

All we do in Phase 2, is run the schedule $\mathcal{S}$ and then $\mathcal{S}_0$. By doing so, we send each active vertex $v \in V(\mathcal{T}')$ to one of the centers, i.e. the vertex $f(v)$. To see why, note that in Phase 1, $\mathcal{S}_0$ already sent all the information from all the nodes in $V(\mathcal{T}')$ to all other nodes in the same component of $\mathcal{T}'$, and hence in particular to the node v in the component which has an edge of the b-matching incident on it. In this phase, $\mathcal{S}$ sends all the messages from v to its matched vertex $m(v)$ in $V(\mathcal{T})$, and $\mathcal{S}_0$ sends all the information from $m(v)$ in $V(\mathcal{T})$ to one of the centers.

Finally, it remains to find the optimal b-matching in polynomial time.

Lemma 14. *We can find the minimum integer b along with a b-matching in $G[V(\mathcal{T}), V(\mathcal{T}')]$ in polynomial time which saturates one vertex from each active component in $V(\mathcal{T}')$.*

Proof. Shrink each active component in $V(\mathcal{T}')$ and replace it with a single node. Also, remove all of the inactive components from the graph. Now, it's enough to find a minimum b-matching which saturates all of the shrunk nodes. This is doable by the standard algorithms for finding minimum b-matchings in bipartite graphs, e.g. see [22]. Note that since every shrunk node has at least one neighbor in $V(\mathcal{T})$ since the message from an active vertex in this node must transmit the message to its made outside the shrunk node, the existence of such a b-matching is guaranteed.

Phase 3. In this Phase, we reduce the original MM instance to a new instance with a smaller number of terminals. Naturally, since v is sent to $f(v)$ for every active terminal v, we want to reduce this instance to a new instance $\mathcal{I}'$ defined on the new set of terminals $R' = \{f(v) | v \in R\}$ along with the new set of demand pairs $P' = \{(f(u), f(v)) | (u, v) \in P\}$.

To find a schedule for $\mathcal{I}'$, we can use Theorem 3, which guarantees a schedule of length $O(|P'| + diam_{P'}(G))$. But note that $|P'|$ can be as large as $\Omega(n)$, which makes this solution inefficient. To overcome this issue, we use an idea similar to Phase 1 of Algorithm *MM*. We can sparsify the demand pairs P' using Algorithm *Sparsify*, and obtain a subset $\hat{P'} \subseteq P'$, which satisfies the properties below:

1. $|\hat{P'}| \leq 2|R'| \log |R'|$
2. Assuming that $\hat{\mathcal{S}}'$ is a feasible schedule for $\mathcal{I}(G, \hat{P'})$, then repeating $\hat{\mathcal{S}}'$ for $8 \log |R'|$ times makes a feasible schedule for $\mathcal{I}(G, P')$.

The process of finding the sparsified subset of demand pairs, $\hat{P'}$, is identical to Algorithm *MM*. To summarize, all we need to do in Phase 3 is:

1. Construct the new set of demand pairs $P' = \{(f(u), f(v)) | (u, v) \in P\}$.
2. Find the sparsified subset of demand pairs $\hat{P'}$.

3. By applying Theorem 3 on $\hat{P}'$, find a schedule $\hat{\mathcal{S}}'$ for $\mathcal{I}(G, \hat{P}')$.
4. Run the schedule $\hat{\mathcal{S}}'$ for $8 \log |R'|$ times to solve the instance $\mathcal{I}(G, P')$.

Phase 4. At this point, we know that for any $(u, v) \in P$, vertex $f(v)$ has received the message p_u, i.e. the message from vertex u. So, all we need to do to complete the solution is to inform v of what $f(v)$ knows. Note that by running Phase 1 and Phase 2, exactly the reverse happened, i.e. $f(v)$ was informed of what v knew. Consequently, all we need to do in Phase 4, is the same as we did in Phase 1 and Phase 2, but in the reverse order.

Theorem 4. *The given algorithm produces a schedule of length* $O(\log n \cdot OPT + \sqrt{n} \log^2 n)$.

Proof. Let $\mathcal{S}^*$ be the optimal schedule, so we have $|\mathcal{S}^*| = OPT$. We provide an upper bound, in terms of OPT, on the number of rounds used in each of the four phases. In Phase 1, we only run the schedule $\mathcal{S}_0$, for which we showed $|\mathcal{S}_0| \leq 2\sqrt{n}$ in the description of the phase.

In Phase 2, we run the schedule $\mathcal{S}$ and then $\mathcal{S}_0$, where we have $|\mathcal{S}| \leq b$. Recall that b was the minimum integer for which there exists a b-matching saturating exactly one vertex from each active component T'_i. Here we prove $b \leq OPT$ which implies that $|\mathcal{S}| \leq OPT$. To see this, we should consider the optimal multicast schedule, $\mathcal{S}^*$.

Let $E^* \subseteq E(G)$ be the subset of the edges used in $\mathcal{S}^*$. For any active component $T'_i \in \mathcal{T}'$, there must exist an edge $(x_i, y_i) \in E^*$ such that $x_i \in V(\mathcal{T}'_i)$ and $y_i \in V(\mathcal{T})$. This holds since otherwise, in the schedule $\mathcal{S}^*$, the active component T'_i would be disconnected from the rest of the vertices, contradicting the feasibility of $\mathcal{S}^*$. Now, let

$$M = \{(x_i, y_i) \mid T'_i \in \mathcal{T}',\ T'_i \text{ is active}\}$$

Clearly, M defines a b-matching in $G[V(\mathcal{T}'), V(\mathcal{T})]$, but for what value of b? In other words, what is the maximum number of edges in M which share the same endpoint in $V(\mathcal{T})$? To answer this question, note that $M \subseteq E^*$, and no vertex in E^* has a degree more than OPT, due to the optimality of $\mathcal{S}^*$. This implies $b \leq OPT$, and so, $|\mathcal{S}| \leq OPT$. Recall that we run the schedule $\mathcal{S}$ and then $\mathcal{S}_0$ in Phase 2, which means we use at most $OPT + 2\sqrt{n}$ rounds in this phase.

In Phase 3, we run the schedule $\hat{S}'$ for $8 \log |R'|$ times, so, the number of rounds used in this phase is at most $8 \log |R'| \cdot |\hat{S}'|$, which we are going to compute in terms of OPT. By applying Theorem 3 on $\hat{P}'$, we get

$$\begin{aligned} |\hat{\mathcal{S}}'| &\leq O(|\hat{P}'| + diam_{\hat{P}'}(G)) \\ &\leq O(2|R'| \log |R'| + diam_{\hat{P}'}(G)) && (7) \\ &\leq O(\sqrt{n} \log n + 4\sqrt{n} + OPT) && (8) \end{aligned}$$

where (7) is due to the fact that $|\hat{P}'| \leq 2|R'|\log|R'|$ as a result of the sparsification, and (8) holds since $diam_{\hat{P}'}(G) \leq 4\sqrt{n} + OPT$. To see why, note that $d_G(v, f(v)) \leq 2\sqrt{n}$ for all v, which implies

$$\begin{aligned} d_G(f(u), f(v)) &\leq d_G(f(u), u) + d_G(u, v) + d_G(v, f(v)) \\ &\leq 4\sqrt{n} + d_G(u, v) \\ &\leq 4\sqrt{n} + OPT \end{aligned}$$

Recall that the number of rounds used in Phase 3 is at most $8\log|R'| \cdot |\hat{S}'|$. If we plug in (8) we get

$$8\log|R'| \cdot |\hat{S}'| \leq O(\log n \cdot (\sqrt{n}\log n + OPT))$$

Consequently, we use at most $O(\log n \cdot OPT + \sqrt{n}\log^2 n)$ rounds in Phase 3.

In Phase 4, we clearly use as many rounds as we use in Phase 1 and Phase 2 overall, i.e. $4\sqrt{n} + OPT$. By summing the upper bounds obtained on the length of each phase, it implies that the total length of the produced schedule is $O(\log n \cdot OPT + \sqrt{n}\log^2 n)$.

H Equivalence with the Minimum Poise Subgraph Problem

The main result of this section is an (approximate) equivalence between the Multicommodity Multicast Problem and the *Minimum Poise Subgraph* Problem.

Definition 11. *In the Minimum Poise Subgraph Problem (MP), we are given a connected graph G along with a subset of pairs of its vertices, P. The goal is to find a subgraph H of G which minimizes $\Delta(H) + diam_P(H)$.*

We denote the quantity $\Delta(H) + diam_P(H)$ as the *Poise* of the subgraph H generalizing from the corresponding version for the broadcast problem in [20]. Recall that if two vertices u, v are not connected in H, then by convention, $d_H(u, v) = \infty$. So the subgraphs H which do not connect a pair $(u, v) \in P$ will be automatically excluded from the solution domain.

Assume we are given an MM instance, $\mathcal{I}(G, P)$, defined on the graph G with the set of demand pairs P. Also, let $\mathcal{J}$ denote the associated MP instance, i.e. the MP instance defined on the graph G with the subset of pairs P.

Theorem 5. *Given a feasible schedule for $\mathcal{I}$, namely $\mathcal{S}$, in polynomial time we can obtain a feasible solution for $\mathcal{J}$ with poise at most $O(|\mathcal{S}|)$. In the other direction, given a feasible solution for $\mathcal{J}$ with poise x, in polynomial time we can obtain a feasible schedule for $\mathcal{I}$ with length at most $O(x \cdot \frac{\log^3 n}{\log\log n})$.*

We need the following Lemma for the proof of Theorem 5.

Lemma 15. *There exist a collection of* $\log|P|$ *subgraphs of G, namely $\mathcal{F}$, satisfying the following properties.*

i. *Each element of $\mathcal{F}$ is a forest,*
ii. *For each pair in P, e.g. (s_i, t_i), there exist a forest in $\mathcal{F}$, which connects both s_i and t_i, and*
iii. *For each $F \in \mathcal{F}$ we have $diam(F) \leq diam_P(G) \cdot 4 \log |P|$ and $\Delta(F) \leq OPT$ where OPT is the length of the optimum schedule for $\mathcal{I}(G, P)$.*

Proof. Let $\mathcal{P} = \{P_1, \ldots, P_k\}$ be the set of paths in the optimal schedule for $\mathcal{I}(G, P)$, where P_i is the path through which the message from s_i is sent to t_i. For any $Q \in \mathcal{P}$, let $N(Q)$ denote the subset of the paths in $\mathcal{P}$ which have at least one vertex in common with Q. Also, for any subset $\mathcal{S}$ of $\mathcal{P}$, let $N(\mathcal{S}) = \{N(Q) | Q \in \mathcal{S}\} \backslash \mathcal{S}$.

Now, we use the following algorithm to find the first forest in $\mathcal{F}$. Later, we find the subsequent forests using the same algorithm.

Algorithm *Extract Forest*
Input: A graph G and a a family of paths $\mathcal{P}$
Output: A subgraph of G, which is a forest

```
1.  $\mathcal{H}, \mathcal{X} \leftarrow \phi$ and $\mathcal{P}' \leftarrow \mathcal{P}$
2.  repeat
3.      Let $Q \in \mathcal{P}'$
4.      $\mathcal{X} \leftarrow \mathcal{X} \cup \{Q\}$
5.      while $|N(\mathcal{X})| \geq |\mathcal{X}|$
6.          do $\mathcal{X} \leftarrow \mathcal{X} \cup N(\mathcal{X})$
7.      $\mathcal{H} \leftarrow \mathcal{H} \cup \mathcal{X}$.
8.      $\mathcal{P}' \leftarrow \mathcal{P}' \backslash (\mathcal{X} \cup N(\mathcal{X}))$
9.  until $\mathcal{P}' = \phi$
10. Let $H$ be a subgraph of $G$, with $V(H) = V(\mathcal{H})$ and $E(H) = E(\mathcal{H})$
11. Using BFS, find a spanning tree with an arbitrary root in each connected
    component of $H$
12. Output the collection of all trees found in the previous step
```

Let the forest F be the output of Algorithm *Extract Forest*. First, we bound the diameter of each connected component of F by $2 \log |\mathcal{P}| \cdot OPT$. See that each connected component of H, namely X, is made of a family of paths, namely $\mathcal{X}$. This family is formed by a consecutive execution of line 6 of the Algorithm *Extract Forest*. So, we can decompose $\mathcal{X}$ into a number of families $\mathcal{X}_1, \ldots, \mathcal{X}_j$, where for each i, $\mathcal{X}_i$ is the family of paths that was added to $\mathcal{X}$ in some iteration of line 6. W.L.O.G we can assume that $\mathcal{X}_a$ is added sooner than $\mathcal{X}_b$ iff $a < b$. Observe that after each iteration of line 6, the size of $\mathcal{X}$ is at least doubled, which implies $j \leq \log |\mathcal{P}|$.

Now, let u, v be two arbitrary vertices of X, such that $u \in V(\mathcal{X}_a)$ and $v \in V(\mathcal{X}_b)$ for some a, b. To prove our bound on the diameter, we show that $d_X(u, v) \leq 2 \log |\mathcal{P}| \cdot OPT$, as follows: Let w be an arbitrary vertex in $V(\mathcal{X}_1)$. Note that there is path in X between u and w with length at most $a \cdot OPT$. Similarly, there is a path between v and w with length at most $b \cdot OPT$. This implies $d_X(u, v) \leq (a + b) \cdot OPT$. Moreover, having $a, b \leq \log |\mathcal{P}|$ implies $d_X(u, v) \leq 2 \log |\mathcal{P}| \cdot OPT$. So, a BFS tree in X with an arbitrary root vertex, has a diameter at most $4 \log |\mathcal{P}| \cdot OPT$.

After we found the first forest, we update $\mathcal{P}$ by removing $\mathcal{H}$ from it. Then, by running the above algorithm on the updated $\mathcal{P}$, we find the second forest. We continue this process until $\mathcal{P}$ becomes empty. To bound the number of forests, we show that after extracting each forest, the size of $\mathcal{P}$ is reduced by at least a factor of 2. This will finish the proof, since it just means $\mathcal{P}$ becomes empty after finding at most $\log|\mathcal{P}|$ forests.

To prove the claim, simply see that in line 8 of the Algorithm *Extract Forest*, whenever we remove a subset of paths from $\mathcal{P}'$, we also add a subset of paths with at least the same size to $\mathcal{H}$, in line 7. It means $|\mathcal{H}| \geq \frac{|\mathcal{P}|}{2}$. Consequently, the size of the updated $\mathcal{P}$, i.e. $\mathcal{P}\backslash\mathcal{H}$, is at most $|\mathcal{P}|/2$. This finishes the proof.

Corollary 2. *Given a simple n-vertex graph G, we can find a subgraph $s(G)$ of G in polynomial time, such that $|E(s(G))| \leq 2n\log n$ and for each $(u,v) \in E(G)$, we have $d_{s(G)}(u,v) \leq 8\log n$.*

Proof. We define an MM instance and then, we prove our claim by applying Lemma 15 on it. Let $\mathcal{I}(G,P)$ be an MM instance where $P = \{(u,v)|(u,v) \in E(G)\}$. By applying Lemma 15 on $\mathcal{I}$, we obtain a family of forests, namely $\mathcal{F}$, which is satisfying the properties (ii) and (iii) of Lemma 15. This means that for every $(u,v) \in P$, there exists a forest in $\mathcal{F}$, namely F, such that $d_F(u,v) \leq 4\log|P|$. So, if we define $E(s(G))$ to be $E(\mathcal{F})$, then, $diam(s(G)) \leq 4\log|P| \leq 8\log n$, and moreover, $|E(s(G))| \leq (\log|P|).n \leq 2n\log n$, which finishes the proof.

Note that Algorithm *Sparsify* which was used in Section 6 is derived in the proof of the above corollary.

Proof (of Theorem 5). First we prove the easy direction. Given a schedule $\mathcal{S}$, we find a subgraph with poise at most $2|\mathcal{S}|$. Let $E^* \subseteq E(G)$ be the subset of edges used in $\mathcal{S}$. Let H be the subgraph of G with $E(H) = E^*$. Clearly, $\Delta(H) \leq |\mathcal{S}|$ and $diam_P(H) \leq |\mathcal{S}|$. This proves the claim.

Now, given a subgraph H with poise x, we construct a schedule $\mathcal{S}$ with $|\mathcal{S}| \leq O\left(x \cdot \frac{\log^3 n}{\log\log n}\right)$. Similar to Lemma 15, we show the existence a collection of $\log|P|$ subgraphs of H, namely $\mathcal{F}$, satisfying the following properties: *i.* Each element of $\mathcal{F}$ is a forest, *ii.* For each pair in P, e.g. (s_i,t_i), there exist a forest in $\mathcal{F}$, e.g. F, which connects s_i and t_i, and *iii.* For each $F \in \mathcal{F}$ we have $diam(F) \leq diam_P(H) \cdot 4\log|P|$.

Proof of this claim is very similar to the proof in Lemma 15. We can obtain the family $\mathcal{F}$ by running Algorithm *Extract Forest* iteratively on the family of paths $\mathcal{P}^*$, which is defined as follows: The family $\mathcal{P}^*$ contains an arbitrary shortest path in H from u to v for all $(u,v) \in P$. It is easy to verify that the output will satisfy the required properties for $\mathcal{F}$, we do not repeat the proof here.

Given the family $\mathcal{F}$, we construct the schedule $\mathcal{S}$ as follows: Order the elements of $\mathcal{F}$ arbitrarily. Then, for each forest F in that order, broadcast within all components of F simultaneously. Since for each $(s_i,t_i) \in P$, there exists a forest in $\mathcal{F}$ which connects s_i and t_i, the schedule is clearly feasible. To provide an upper bound on $|\mathcal{S}|$, we show that we need at most $O(x \cdot \frac{\log^2 n}{\log\log n})$ rounds for each forest F. Then, given that $|\mathcal{F}| = O(\log n)$, we would have $|\mathcal{S}| \leq O(x \log^3 n)$.

To finish the proof, given a forest F, we prove that broadcasting within each component of F can be done in $O(x \cdot \frac{\log^2 n}{\log\log n})$ rounds. Note that this can be done simultaneously for all components since they are clearly vertex-disjoint. Ravi [20] has shown that the minimum broadcast time for a tree T is at most $\frac{\log n}{\log\log n} \cdot \mathit{poise}(T)$. Using dynamic programming, he also provides an Algorithm to find such a broadcast schedule in polynomial time. It just remains to see that each connected component in F is a tree, by the properties of $\mathcal{F}$, and has a poise at most $\Delta(H) + \mathit{diam}_P(G) \cdot 4 \log |P| = O(x \log n)$.

Corollary 3 (of Theorem 5). *Given an MM instance $\mathcal{I}(G, P)$, in polynomial time we can find a feasible schedule for it with length at most $O\left(\frac{\log^3 n}{\log\log n} \cdot (OPT + \Delta(G))\right)$.*

Proof. In polynomial time, we can find a subgraph H of G with poise at most $OPT + \Delta(G)$, i.e. a subgraph H satisfying $\Delta(H) + \mathit{diam}_P(H) \leq OPT + \Delta(G)$. To find such a subgraph, we just need to define H as the union of the (s, t)-shortest paths for all $(s, t) \in P$. Then, clearly we would have $\Delta(H) \leq \Delta(G)$ and $\mathit{diam}_P(H) \leq OPT$, which together imply the desired bound on the poise of H.

After we found H, all we need to do is applying Theorem 5 on H and obtain a schedule of length $O\left(\frac{\log^3 n}{\log\log n} \cdot (\Delta(H) + \mathit{diam}_P(H))\right)$, which is at most $O\left(\frac{\log^3 n}{\log\log n} \cdot (\Delta(G) + OPT)\right)$.

I Hardness

Definition 12. *In the* Hyperedge-Coloring *problem (HC), we are asked to color the edges of a given hypergraph H with the minimum number of colors such that no two intersecting edges have the same color.*

Lemma 16. *Assuming ZPP $\neq$ NP, and for any positive $\epsilon < 1$, there are no $O(m^{1-\epsilon})$-approximations for the HC problem, where m is the number of the edges of the hypergraph.*

Proof. We reduce the problem to the vertex-coloring problem. It is known that the vertex-coloring problem is not approximable within a factor of $O(|V(G)|^{1-\epsilon})$ unless ZPP = NP [8]. Given a graph G as the input for the vertex-coloring problem, we construct a hypergraph $\mathcal{H}$ such that $V(\mathcal{H}) = E(G)$, i.e. there is a vertex $\bar{e}$ in $\mathcal{H}$ for each edge $e \in E(G)$. Also, for each vertex $v \in V(G)$, there is a hyperedge $\bar{v}$ in $\mathcal{H}$, where $\bar{v}$ contains all the vertices $\bar{e}$ such that v is one of the endpoints of e in G, i.e. $\bar{v}$ contains the set of vertices $\{\bar{e} | \exists u : e = (u, v)\}$.

It's easy to verify that $\chi'(\mathcal{H}) = \chi(G)$: Observe that any valid vertex-coloring for G gives a valid edge-coloring for $\mathcal{H}$, and vice versa. This can be done by assigning the same colors to the vertex $v \in V(G)$ and the edge $\bar{v} \in E(\mathcal{H})$. Now, see that the existence of a $O(m^{1-\epsilon})$-approximation algorithm for the HC problem, implies we can approximate $\chi'(\mathcal{H})$ within a factor of $O(|E(\mathcal{H})|^{1-\epsilon})$, and this means we can approximate $\chi(G)$ within a factor of $O(|V(G)|^{1-\epsilon})$ due to the fact that $|E(\mathcal{H})| = |V(G)|$. Contradiction.

Theorem 6. *Assuming $P \neq NP$, and for any positive $\epsilon < 1$, there are no $O(\max\{n, m\}^{1-\epsilon})$-approximations for the Hypercast problem, where n, m are respectively the number of the vertices and edges of the hypergraph G given in the Hypecast instance.*

Proof. To prove the claim, given an instance of the HC problem with the hypergraph H, we reduce it to an instance of the Hypercast problem as follows. Let G be a hypergraph which contains a copy of all the edges and vertices of H. We will modify G step by step to obtain the desired Hypercast instance.

First, for each hyperedge $e \in E(H)$, insert two distinct new vertices e^- and e^+ in it. Then, add another new vertex r^- to G, and also, add a hyperedge which contains r^- and all the vertices e^-. Let r^- be the source node in our Hypercast instance and $R = \{e^+ | e \in E(H)\}$ be the set of terminals. We claim that any L-coloring of the hypergraph H corresponds to a solution of length $L+1$ for the Hypercast instance, and vice versa.

Assume we are given an L-coloring for H, then, we construct a schedule of length $L+1$ as follows: In the first round of the schedule, use the only edge containing r^- to inform all the vertices e^-. Then, use one round for each color-class of the L-coloring, i.e. sort the colors in an arbitrary order and dedicate a round to each color with respect to that ordering. Use each edge of H in the round dedicated to its color. The obtained schedule has length $L+1$ and is obviously feasible since each terminal $e^+ \in R$ is informed by e^- when the edge containing them is used in the round dedicated to its color.

To show the other direction, given a schedule of length $L+1$ for the Hypercast instance, we find an L-coloring for H. We can assume the first round is dedicated to the only edge containing r^- W.L.O.G. and that is because this edge intersects with all the other edges and also is the only edge containing the source node. Since each edge $e \in E(G)$ contains a distinct terminal, then e should be used in at least one of the rounds $2, \ldots, L+1$. Pick one of the rounds in which e is used arbitrarily and let it be the color of e. Clearly, the number of used colors is L, and moreover, no two intersecting edges have the same color due to the feasibility of the given schedule.

The above argument implies that any α-approximation for the Hypercast problem gives an $O(\alpha)$-approximation for the HC problem. Since there are no $O(|V(H)|^{1-\epsilon})$-approximations for the HC problem, and since $n, m \in O(|V(H)|)$, then there are no $O(\max\{n, m\}^{1-\epsilon})$-approximations for the Hypercast problem.

Under Khot's 2-to-1 conjecture, we also provide a harness result in terms of D for the Hypercast Problem. To prove this result, we need the following Theorem of [11]:

Theorem 3.1. from [11] Assuming 2-to-1 conjecture, the following holds: Given any integer $k \geq 7$ and $\Delta \geq \Delta_0(k)$, it is NP-hard (under randomized reductions) to decide whether an unweighted graph G with maximum degree Δ is k-colorable or has largest independent set size at most $O(n/\Delta^{1-c/k-1})$

Proposition 4. *Under Khot's 2-to-1 conjecture, it is NP-hard to color a 7-colorable graph with $O(\Delta^{1/3})$ colors.*

Theorem 7. *Assuming $P \neq NP$, there are no $O(D^{1/3})$-approximations for the Hypercast problem, where D is the maximum size of a hyperedge in the hypergraph G.*

Proof. The proof is similar to the proof of Theorem 6. We use the exact same reduction to the vertex coloring problem, and then, instead of using the $\Omega(|V(G)|^{1-\epsilon})$ hardness ratio of [8], we use Proposition 4.

For contradiction, assume we are given an algorithm for the Hypercast problem, namely Algorithm $\mathcal{A}$, which has an approximation ratio of $O(D^{1/3})$. Using this algorithm, we will design an algorithm which, for any Δ, colors any 7-colorable graph with $O(\Delta^{1/3})$ colors. This will be a contradiction with Proposition 4.

Assuming that we are given a 7-colorable graph G as the input of the vertex coloring problem, we reduce this instance to an instance of the Hypercast Problem. This is done in exactly the similar way as it was done in Lemma 16 followed by Theorem 6. Note that the size of the hyperedges in obtained Hypercast instance would be bounded by Δ, i.e. $D = \Delta$.

By the properties of our reduction, since $\chi(G) \leq 7$, then the optimal schedule for the produced Hypercast instance has length $O(1)$, and so, Algorithm $\mathcal{A}$ generates a schedule of length at most $O(\Delta^{1/3})$. Due to our reduction, this schedule can be converted into a coloring of vertices for the graph G which uses no more than $O(\Delta^{1/3})$ colors. This is a contradiction with Proposition 4.