Maximizing Total Upload in Latency-Sensitive P2P Applications

John R. Douceur
Microsoft Research
Redmond WA 98052
johndo@microsoft.com

Jacob R. Lorch
Microsoft Research
Redmond WA 98052
lorch@microsoft.com

Thomas Moscibroda
Microsoft Research
Redmond WA 98052
moscitho@microsoft.com

ABSTRACT

Motivated by an application in distributed gaming, we define and study the *latency-constrained total upload maximization problem*. In this problem, a peer-to-peer overlay network is modeled as a complete graph and each node v_i has an upload bandwidth capacity c_i and a set of receivers $R(i)$. Each sender-receiver pair (v_i, v_j), where $v_j \in R(i)$, is a *request* that should be satisfied, i.e., v_i should send a data packet to each $v_j \in R(i)$. The goal is to find a set of at most n multicast-trees T_i of depth at most 2, such that each node can be part of multiple trees, all capacity constraints are met, and the number of satisfied requests is maximized. In this paper, we prove that the problem is NP-complete, and we present an algorithm with approximation ratio $1 - 2/\sqrt{c_{\min}}$, where $c_{\min}$ is the minimum upload capacity. Finally, we also study the impact of *network coding* on the quality and approximability of the solution.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Routing and layout*
C.2.2 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*

General Terms

Algorithms, Theory

Keywords

Peer-to-peer gaming, multicast, upload maximization

1. INTRODUCTION

Real-time *distributed gaming* is a large and fast-growing business and massively multi-player games in which physically separated players participate and play over the Internet are becoming increasingly popular. The key requirement in devising a network infrastructure for such games is to ensure that every player, at any time receives real-time updates from every other player that is currently in its *focus*, say, on its computer screen. While satisfying this requirement is challenging even in small-scale distributed gaming applications based on client/server architectures [8], it is even more so when considering peer-to-peer architectures.

SPAA'07, June 9–11, 2007, San Diego, California, USA.

The motivation for studying peer-to-peer-based solutions for distributed gaming applications is scalability. The approach taken by platforms such as PS3 or XBOX Live is to use the player with highest bandwidth as a centralized server to which peers permanently send their data, and which then forwards this data to every interested receiver. The problem with this approach is that the upload bandwidth used at the server may grow in the order of $O(n^2)$. By studying a peer-to-peer-based approach, the hope is to alleviate this scalability problem by leveraging the available upload bandwidth of *all* participating peers, not just the centralized server. Utilization of the *total upload bandwidth* is the crucial bottleneck in distributed peer-to-peer games because in residential areas, broadband access is typically asymmetric with high download rates, but only small upload rates. Thus, the higher the upload bandwidth utilization, the more accurately the game play can be rendered.

In the absence of widespread use of IP-layer multicast [14], a frequently proposed remedy against upload bandwidth limitations has been application-layer multicast. Unfortunately, existing multicast schemes (used for streaming, for example) do not meet the very *tight latency constraints* found in real-time distributed gaming applications [7, 24, 25]. In particular, with a maximum tolerable delay of 100ms between any two players, the number of hops on the path between sender and receiver can be no larger than 2 or 3. Hence, the log-scale latencies typically found in structured P2P overlays [27, 28] are not sufficient. In our study, we restrict our attention to multicast trees of depth 2, as these trees have the advantage of virtually overhead-free routing.

Besides tight latency constraints and the necessity for high outbound bandwidth utilization, two more aspects characterize our problem setting. First, with a targeted size of up to 1000 players per game [1], distributed games are small in scale in comparison to other distributed applications. For this reason, our system architecture uses a central server whose purpose is to compute and disseminate information about the multicast structure to all peers in the game. This does not thwart scalability, because the actual sending of data to receivers is performed by the peers and hence, the server's upload requirement grows only as $O(n)$. Secondly, in contrast to many other problem settings in networking and peer-to-peer computing, efficient *worst-case behavior* is a necessity in distributed gaming applications. Partly, this is because stalling a game in execution is unacceptable and partly because worst-case scenarios are actually quite likely to occur in practice. As an example, a person holding the flag in a battlefield may simultaneously be in the focus of a large number of other players, and thus needs to send its updates to all of them.

In this paper, we study the problem of maximizing the cumulative upload bandwidth in latency-constrained peer-to-peer networks. In particular, we formally define and study the latency-

constrained *maximum upload bandwidth problem* (MUBP), which seeks to maximize the total number of satisfied *requests*. A request is a pair of nodes v_i and v_j, where the receiver v_j requires update information from sender v_i. The goal is to find a set of multicast trees (one rooted at every node) of depth at most 2 that maximizes the total number of satisfied requests. Every node can participate in multiple trees, but its total upload bandwidth in all trees must not exceed its upload capacity.

Clearly, the total number of satisfied requests is at most C, where C denotes the sum of upload bandwidths of all nodes. In certain settings where bandwidths and requirements are unevenly distributed, however, only a small portion of all requests can be satisfied even though the cumulative upload bandwidth C itself would be sufficient. Moreover, we prove that MUBP is NP-hard and as our main contribution, we give a polynomial-time algorithm that approximates the problem to within a factor of $1 - 2/\sqrt{c_{\min}}$, where $c_{\min}$ is the minimum upload capacity of any peer. We then show how, theoretically, *network coding* can be used to improve both the quality of the achievable solution as well as its approximation.

Finally, notice that while our immediate motivation for this work stems from distributed gaming, the problem of maximizing total outbound bandwidth is of interest beyond this specific application. The success of BitTorrent, for instance, is based on the fact that upload capacity is shared among numerous peers.

The remainder of the paper is organized as follows. After discussing related work in Section 2, Section 3 formalizes the maximum outbound bandwidth problem and defines the terminology used throughout the paper. Section 4 gives some intuition about the algorithmic challenges of the problem. The paper's main technical contribution, the worst-case efficient approximation algorithm, is presented in Section 5. Section 6 studies the impact of network coding, before the paper is concluded in Section 7.

2. RELATED WORK

In the systems and networking community, there has recently been a number of works addressing problems in the context of distributed gaming. Apparently the first peer-to-peer, serverless multiplayer game is MiMaze, which uses IP multicast [18]. A complete peer-to-peer based architecture for massively multi-player games was proposed in [17]. Also recently, the works of [9, 10] propose infrastructure components for supporting online multi-player games. In view of the limited deployment of IP layer multicast, numerous application layer multicast schemes have been proposed, e.g. [5, 13, 20]. Being typically tailored for applications such as streaming, existing multicast schemes do not address the particularly tight latency-constraints faced in distributed gaming applications. Multicast support for distributed multi-player gaming applications was studied in [22].

All of the above works provide heuristic solutions to the problem at hand and are concerned primarily with architectural problems. In contrast, there exists little applicable *algorithmic* work. The rich literature on *(multi-commodity) network flows* includes studies with latency bounds. In this setting, an *L-length bounded $s-t$-flow* is specified by a collection of $s-t$-paths $\mathcal{P} = (P_1, \ldots, P_k)$ and corresponding flow values $f_1, \ldots, f_k$, in which no path $P_i \in \mathcal{P}$ is longer than L. Baier [3] gives an extensive survey of what is known for length-bounded flows, and more recent results have been obtained for instance in [4]. Unfortunately, (multicommodity) network flows do not adequately model *multicast problems*, because the flow conservation condition does not apply to multicast problems: In multicast applications, intermediate relay nodes may send received data to several receivers, thus reducing the required bandwidth at the original sender.

In graphs, the problem of constructing an efficient multicast tree is often modeled as finding an appropriate Steiner tree [26]. Somewhat closer to our needs are Steiner trees with bounded depth that have also been studied extensively in the literature, e.g. [21, 23]. Constructing an efficient multicast tree under various constraints with regard to latency has been readily studied, e.g. [6, 12, 19, 29]. None of these works, however, is directly applicable in our setting since instead of a single tree, we need to construct n partially overlapping multicast trees, one rooted at each node.

3. PROBLEM STATEMENT

The peer-to-peer network consists of a set $V = \{v_1, \ldots, v_n\}$, of n peers that are mutually connected. Each node $v_i \in V$ has a limited upload bandwidth capacity c_i. For each node v_i, the (possibly empty) receiver set $R(i)$ contains the set of nodes that are interested in receiving updates from v_i. A sender v_i sends the *same data* to every receiver in $R(i)$, but the data sent by two different senders is independent of each other. Each pair (v_i, v_j) with $v_j \in R(i)$ is called a *request* that should be satisfied by v_i. A request is satisfied if v_i sends one unit of data to v_j either directly or via at most one intermediate relay node. [1] The *requirement* $r_i = |R(i)|$ of node v_i is the size of its receiver set. Notice that requirements r_i and capacities c_i are measured in equal units, so-called *upload-units* measured in bit/s. Hence, when not using any relay nodes, an upload capacity of c_i is sufficient to satisfy exactly c_i requests.

Let $c_{\min} := \min_{v_i \in V} c_i$ denote the minimum upload capacity of any node in the system. In distributed game like Quake II, for instance, the minimum maintainable upload capacity of a node with 128kb/s upload links would be roughly $c_{\min} \approx 5$ [11]. The sum of all requirements and unit upload capacities is denoted by $R := \sum_{v_i \in V} r_i$ and $C := \sum_{v_i \in V} c_i$, respectively.

In order to account for the rigid latency-constraints, we consider only multicast trees of depth 2. In particular, we say that a request (v_i, v_j) is *satisfied* if either v_i uploads its data directly to v_j, or if there exists a path via at most one relay node v_k, on which v_i sends to v_j. Node v_i satisfies $x \leq |R(i)|$ of its requests if it can thus send data to x of its receivers in $R(i)$. Each node may participate in multiple multicast trees. Unless its receiver set $R(i)$ is empty, it is the root node in one multicast tree. Additionally, it may serve as a relay node in possibly several other trees, and in some trees, it may be a leaf receiving data. In total, however, the number of data units uploaded by node v_i must not exceed c_i. Each sending operation (either as a sender to a relay node or as a relay node to a receiver) costs one unit of upload bandwidth. That is, a relay node v_k can forward data from v_i to multiple receivers $v_j \in R(i)$ as long as its capacity constraint is not violated.

For $i \neq j$, let $I_i^j \in \{0, 1\}$ denote whether node v_i sends its data to node v_j. If $I_i^j = 1$, one upload unit at v_i is used to transmit data to v_j. If $v_j \in R(i)$, this means that the request (v_i, v_j) is satisfied and additionally, each such node v_j may serve as a relay node for v_i. Further, let $U_{ij} \in \mathbb{N}$ be the number of upload units that such a node v_j uses in its role to relay data for node v_i. That is, U_{ij} is the number of receivers in $R(i)$ to which v_j relays data for v_i. Notice that $U_{ij} > 0$ only if $I_i^j = 1$ and conversely, $I_i^j = 0$ implies that $U_{ij} = 0$. We can formalize the *Latency-Constrained Upload Bandwidth Maximization Problem* (MUBP) as follows.

DEFINITION 3.1 (MUBP). *Consider a set of nodes V where each node $v_i \in V$ has a set of receivers $R(i)$ and upload capacity c_i. Find at most $|V|$ multicast trees (one rooted at each node) of*

[1] Our results can be generalized to the scenario in which the update packets of different nodes have different sizes.

depth at most 2, such that $\sum_{v_j \in V} I_i^j + \sum_{v_j \in V} U_{ji} \leq c_i$ *for every node* $v_i \in V$*, and such that the number of satisfied requests* (v_i, v_j) *with* $v_j \in R(i)$ *is maximized. A request* (v_i, v_j) *is satisfied if* v_j *is contained in* v_i*'s multicast tree.*

Let T_{OPT} and T_{ALG} denote the number of requests satisfied in an optimal solution and by some algorithm ALG, respectively. Algorithm ALG achieves an *approximation ratio* of α if for every instance of the problem, it holds that $T_{ALG} \geq \alpha \cdot T_{OPT}$.

4. COMPLEXITY AND INTUITION

MUBP can be shown to be NP-complete by a simple reduction to the 3-*partition problem*.

THEOREM 4.1. *The decision version of the latency-constrained upload bandwidth maximization problem is* NP*-complete.*

PROOF. The problem is in NP since given a solution, its total upload volume can be easily verified. We show NP-hardness by reducing the well-known 3-*partition problem* to it [16]. In this problem we are given a set A of $3m$ items $A = \{1, \ldots, 3m\}$ with associated sizes $a_1, \ldots, a_{3m} \in \mathbb{N}$, with $B/4 < a_i < B/2$, for each i, and $\sum_{i=1}^{3m} a_i = mB$, and we must decide whether A can be partitioned into m disjoint sets $I_1, \ldots I_m$ such that $\sum_{i \in I_j} a_i = B$, for $j = 1, \ldots, m$. Note that due to the bounds for the item sizes a_i, all sets I_j must have cardinality 3.

Given an instance of the 3-partition problem, construct an instance of the upload maximization problem with $B + 4m$ nodes $v_1, \ldots, v_{B+4m}$ as follows. For $1 \leq j \leq B$, let $c_j = 0$ and $R(j) = \emptyset$; for $B + 1 \leq j \leq B + 3m$, let $c_j = a_{j-B}$ and $R(j) = \emptyset$; and finally, for $B + 3m + 1 \leq j \leq B + 4m$, let $c_j = 3$ and $R(j) = \{v_1, \ldots, v_B\}$. That is, for each item a_i, there is a node v_{B+i} with upload capacity a_i. Additionally, there are m nodes $v_{3m+1}, \ldots v_{4m}$, each having upload capacity 3 and requirement B. Because each of these nodes can use at most 3 relay nodes, there is a solution with total upload bandwidth mB if and only if the set A of items has a 3-partition. □

The proof of Theorem 4.1 does not convey the full picture of the problem's complexity, as it only captures the difficulty of selecting a proper subset of relay nodes. Another substantial algorithmic challenge is that every upload unit may be used in two ways: to upload to a receiver (either as a relay or directly) or, as the root of the tree to send to a relay. The main challenge is thus to decide how much of its upload bandwidth each node should allocate for serving as a relay or send directly, and how much it should dedicate to its own multicast tree.

As an example, consider a network with $n + 2$ nodes $v_1, \ldots, v_n$ and two special nodes w_1 and w_2. Nodes w_a and w_b have requirements $r_a = r_b = n - \sqrt{n}$ (with receiver set $\{v_{\sqrt{n}+1}, \ldots, v_n\}$). The capacities of these two special nodes is $c_a := 1$ and $c_b := n$, respectively. The nodes $v_1, \ldots, v_{\sqrt{n}}$ have capacity $c_1, \ldots, c_{\sqrt{n}} := \sqrt{n}$. Finally, all nodes $c_1, \ldots, c_n$ have an empty receiver set.

In this example, it is possible to satisfy all requests, i.e., $T_{OPT} = R = 2(n - \sqrt{n})$. In this optimal solution, w_b sends its data via $v_1, \ldots, v_{\sqrt{n}}$ even though it would be capable of serving its own requirements. This is in order to to free its upload resources as much as possible for w_a. On the other hand, every algorithm in which peers greedily dedicate their upload bandwidth to their own multicast tree (and allocate only the spare capacity to other trees) achieves an upload of at most n, as w_a can satisfy at most $\sqrt{n}$ of its requests using relay nodes.

The example shows that no algorithm that allows nodes to greedily serve their own requests before relaying for other nodes can achieve an approximation ratio better than $1/2 + \epsilon$. Furthermore, since every upload unit may be used in two different roles, intuitively, an approximation ratio of $1/2$ seems "natural". In the following section, we show how to surpass this bound.

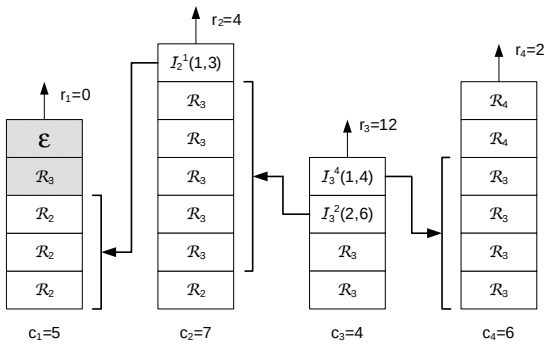

Figure 1: The total upload achieved by these four nodes is 17. The shaded upload unit $\mathcal{R}_3$ at v_1 is not referenced and hence wasted. The total upload could be increased e.g. by setting $U_2[1] = \mathcal{R}_3$, $U_1[4] = \mathcal{R}_2$, and adjusting the indirection unit accordingly.

5. WORST-CASE EFFICIENT ALGORITHM

In this section, we present the main technical contribution of this paper: a worst-case efficient approximation algorithm for the MUBP problem. The algorithm is presented in Algorithm 1 and consists of several subroutines. On a high level, the algorithm proceeds by greedily assigning different *roles* to upload units of different peers. The algorithm may subsequently change the role assigned to an upload unit, but at any time, each unit is in exactly one role. The final role assignment determines the multicast trees.

Let the kth upload unit of a player $v_i \in V$ be written as $U_i[k]$. $U_i[k_1, k_2] := (U_i[k_1], U_i[k_1 + 1], \ldots, U_i[k_2])$ denotes a contiguous range of upload units at v_i. At any time during the course of the algorithm, each upload unit can be assigned to one of four different *roles*.

- Initially, all upload units are *unspecified*, $U_i[k] = \mathcal{Q}$. Once a unit is assigned a role other than $\mathcal{Q}$, it can never become unspecified again.
- An upload unit $U_i[k] = \mathcal{R}_j$ at v_i is used for sending data to an intended receiver $v_\ell \in R(j)$ (see Figure 1). If $i \neq j$, this means that v_i is serving as a relay in v_j's multicast tree, and if $i = j$, then v_i is uploading directly to one of its receivers.
- An *indirection unit* $U_i[k] = \mathcal{I}_i^j[k_1, k_2]$ means that upload unit $U_i[k]$ at v_i is used for sending data to v_j, which then sends this data to $k_2 - k_1 + 1$ receivers $v_\ell \in R(i)$ using its upload units $k_1, \ldots, k_2$, i.e., $U_j[k] = \mathcal{R}_i$ for all $k_1 \leq k \leq k_2$. The *fan-out* $S = k_2 - k_1 + 1$ of an indirection unit $U_i[k] = \mathcal{I}_i^j[k_1, k_2]$ is the number of upload units it references.
- Finally, it will be convenient to let the algorithm assign the *empty* role $U_i[k] = \mathcal{E}$ to certain upload units. If at the end of the algorithm, an upload unit is still in role $\mathcal{E}$, it is not used in any multicast tree.

For notational convenience, we write only $\mathcal{I}_i^j$ if the exact upload units referenced are not of interest, and $\mathcal{I}$ or $\mathcal{R}$ to indicate that a specific upload unit is used in this role. During the course of the algorithm, every peer has a *dynamic capacity* $\hat{c}_i$, which (roughly) captures the number of currently free upload units in v_i. Similarly, the notation $\hat{r}_i$ describes a node's *residual requirement*, i.e., how many of its requests have not yet been allocated. Initially, $\hat{c}_i = c_i$

and $\hat{r}_i = r_i$. While $\hat{c}_i$ and $\hat{r}_i$ will be decreased during the course of the algorithm, c_i and r_i remain unchanged. For a node $v_i \in V$, Q_i denotes the set of upload units assigned to role $\mathcal{Q}$, $Q_i = \{U_i[k] | U_i[k] = \mathcal{Q}\}$. Sets E_i, R_i, and I_i are defined analogously for the other roles.

Finally, it is important to observe the following. Our algorithm does not specify to which receiver $v_\ell \in R(j)$ a specific unit $\mathcal{R}_j$ is used for uploading. For each relay node, the algorithm merely states *how many* units are designated for a certain multicast tree. Notice that this is sufficient because it is irrelevant which relay node v_j of v_i ultimately sends to which receiver in $R(i)$. For an example, let $R(i) = \{w_1, \ldots, w_{10}\}$ and assume that the algorithm allocates upload units as follows: v_i uses 2 upload units to send directly to its receivers. Additionally, it has 2 relay nodes (both of which are not in $R(i)$), each relaying v_i's data to 4 receivers in $R(i)$. Converting this assignment into a valid multicast tree is trivial. For instance, v_i directly uploads to w_1 and w_2, whereas its two relay nodes send to $w_3, \ldots, w_6$ and $w_7, \ldots, w_{10}$, respectively. Because the actual assignment of relays to receivers in each multicast tree is thus irrelevant (as long as it is consistent), our algorithm merely specifies how many upload units each node allocates to each multicast tree.

5.1 Algorithm

The main algorithmic difficulty is to determine the amount of upload bandwidth each node should allocate to each multicast tree. The algorithm addresses this challenge by first accommodating the nodes that have high requirement, but only little capacity (and hence can use only few relay nodes). Specifically, it proceeds in a greedy fashion and allocates the bandwidth requirements (i.e., forms multicast trees) of nodes $v_i \in V$ in non-increasing order of the ratio r_i/c_i, which we call the "clumsiness ratio". An exception is made for nodes with small requirement, $r_i \leq X$, for $X = \sqrt{c_{\min} + 2} - 1$ as specified in the algorithm. These nodes are dealt with in a second phase (Lines 29-31).

Consider the iteration in which some node v_i's multicast tree is built (starting from Line 5). Assume that there are $m \leq n$ nodes with positive *dynamic capacity* $\hat{c}_i > 0$. The algorithm maintains a list of nodes $v_{L(1)}, \ldots, v_{L(m)}$ in non-decreasing order of $\hat{c}_i$. It then selects at most $c_i + 1$ *consecutive nodes*, $v_{L(a)}, \ldots, v_{L(b)}$ that serve as relay nodes in v_i's multicast tree. In order to determine the indexes a and b, the algorithm selects the smallest index a such that the combined dynamic capacity of the c_i+1 *consecutive nodes*, $v_{L(a)}, \ldots, v_{L(b)}$, $b \leq a + c_i$, suffices for satisfying all r_i requests of v_i. In case a is 1—i.e., the $c_i + 1$ nodes with smallest non-zero dynamic capacity can satisfy all of v_i's requirement—b is the minimal index so that $v_{L(a)}, \ldots, v_{L(b)}$ has enough combined dynamic capacity to serve all of v_i's requirement (Lines 9-14 of the algorithm). In case no window of consecutive relays suffices, the algorithm forms a multicast tree using the $c_i + 1$ relay nodes with highest dynamic capacity.

Notice that this allocation initially leads to an infeasible solution in which some nodes v_i use $c_i + 1$ (instead of at most c_i) relay nodes. This is infeasible because a node with capacity c_i can send its data to at most c_i different relay nodes. In the algorithm, however, these nodes are temporarily assigned an indirection unit $\mathcal{I}_i$ in a so-called "overflow unit" $U_i[c_i + 1]$. The algorithm's final phase (Lines 33-40) turns this initial scheme into a feasible solution without losing too much upload bandwidth.

The main problem with the aforementioned greedy allocation procedure is that at the time v_i's tree is considered, many of its upload units may already be used for previously built multicast trees (i.e., set to roles $\mathcal{R}_j$, for $j \neq i$). In order to give such a node v_i the opportunity to build its own multicast tree, the algorithm allows upload units $U_i[k] = \mathcal{R}_j$ to be replaced (or overwritten) by role $U_i[k] = \mathcal{I}_i$. While increasing the capacity of v_i's multicast tree, this decreases the total upload of node v_j, to whose multicast tree this upload unit $U_i[k] = \mathcal{R}_j$ was originally assigned. In other words, each such overwrite operation decreases the fan-out of the indirection unit allocated at v_j and thus reduces its effectiveness. Our algorithm maintains the invariant that upload units $\mathcal{R}_j$ are assigned to v_i only if the indirection unit's fan-out is at least X, i.e., only if v_i allocates at least X upload units to v_j's tree. The goal is to ensure multicast trees with high fan-out so that every upload unit $\mathcal{I}_i^{L(t)}$ invested for indirection leads to a large number of upload units $\mathcal{R}_i$.

In more detail, the allocation of upload units to roles takes place in the $\text{allocate}(v_i, v_{L(t)}, S)$ subroutine. Upload units of nodes $v_{L(t)}$, $a \leq t \leq b$, are filled up by setting each of the s available units (see Line 4) to $\mathcal{R}_i$ (or alternatively $\mathcal{E}$ if $s < X$). As we prove,

Algorithm 1 Algorithm - Main Procedure

Input: Upload capacities c_i, requirements r_i
Output: A feasible upload scheme $\mathcal{S}'$

1: Define $X := \sqrt{c_{\min} + 2} - 1$;
2: Initially, $U_i[k] = \mathcal{Q}$ for all $v_i \in V, 1 \leq k \leq c_i$;
3: For each node, set $\hat{c}_i = c_i$ and $\hat{r}_i = r_i$;
4: Label nodes $v_1, \ldots, v_n$ such that $\frac{r_1}{c_1} \geq \frac{r_2}{c_2} \geq \ldots \geq \frac{r_n}{c_n}$;
5: **for each** v_i with $r_i \geq X$ in this order **do**
6: Let m be the number of nodes with $\hat{c}_i > 0$
7: Let $L[1], \ldots, L[m]$ be a sorted list of nodes in non-decreasing order $0 < \hat{c}_{L[1]} \leq, \ldots, \leq \hat{c}_{L[m]}$;
8: $a := 1$; $b = 1$;
9: **while** $\sum_{p=1}^{b} \hat{c}_{L(p)} < r_i$ **and** $b < c_i + 1$ **and** $b < m$ **do**
10: $b := b + 1$;
11: **od**
12: **while** $\sum_{p=a}^{b} \hat{c}_{L(p)} < r_i$ **and** $b < m$ **do**
13: $a := a + 1$; $b := b + 1$;
14: **od**
15: **for** $t := a$ **to** $b - 2$ **do**
16: $\text{allocate}(v_i, v_{L(t)}, \hat{c}_{L(t)})$;
17: **end for**
18: **if** $\hat{r}_i - \hat{c}_{L(b-1)} \geq X$ **then**
19: $\text{allocate}(v_i, v_{L(b-1)}, \hat{c}_{L(b-1)})$;
20: $\text{allocate}(v_i, v_{L(b)}, \hat{r}_i)$;
21: **else**
22: $\text{allocate}(v_i, v_{L(b-1)}, \hat{r}_i - X)$;
23: for all $U_{L(b-1)}[k] = \mathcal{Q}$ do $U_{L(b-1)}[k] := \mathcal{E}$;
24: $\hat{c}_{L(b-1)} := 0$;
25: $\text{allocate}(v_i, v_{L(b)}, X)$;
26: **end if**
27: **end for**
28: {* Try to satisfy requirements of nodes with $r_i < X$*}
29: **for each** v_i with $r_i < X$ **do**
30: Set at most r_i units $U_i[g] = \mathcal{E} \cup \mathcal{Q}$ to $U_i[g] := \mathcal{R}_i$;
31: **end for**
32: {* Make solution feasible–remove overflow units*}
33: **for each** node v_i with $U_i[c_i + 1] \neq \mathcal{Q}$ **do**
34: $k :=$ **nextAloc**(v_i);
35: **if** $k = c_i + 1$ **then**
36: choose $U_i[k] = \mathcal{I}_i^j[k_1, k_2]$ with minimal fan-out from $1 \leq k \leq c_i + 1$
37: for all $k_1 \leq k \leq k_2$ do $U_j[k] := \mathcal{E}$;
38: **end if**
39: $U_i[k] := U_i[c_i + 1]$;
40: **end for**

Algorithm 2 Subroutine - $\text{allocate}(v_i, v_{L(t)}, S)$

1: $s := 0$; $\Delta c_{L(t)} := c_{L(t)} - \hat{c}_{L(t)}$;
2: $\hat{c}_{L(t)} := \hat{c}_{L(t)} - S$; $\hat{r}_i := \hat{r}_i - S$;
3: **for** $y := \Delta c_{L(t)} + 1 \ldots \Delta c_{L(t)} + S$ **do**
4: **if** $U_{L(t)}[y] \neq \mathcal{I}^*_{L(t)}$ **then** $s := s + 1$;
5: **end for**
6: **if** $s \geq X$ **then**
7: $U_{L(t)}[y] := \mathcal{R}_i$, for every $y \in \{\Delta c_{L(t)}+1, \ldots, \Delta c_{L(t)}+s\}$
8: $\text{next} :=$ **nextAloc**(v_i);
9: $U_i[\text{next}] := \mathcal{I}_i^{L(t)}[\Delta c_{L(t)} + 1, \Delta c_{L(t)} + s]$;
10: **reassign**(v_i, next);
11: **end if**
12: **else**
13: **for** $y := \Delta c_{L(t)} + 1 \ldots \Delta c_{L(t)} + S$ **do**
14: **if** $U_{L(t)}[y] = \mathcal{Q}$ **then** $U_{L(t)}[y] := \mathcal{E}$;
15: **end for**
16: **end if**

Algorithm 3 Subroutine - $\text{reassign}(v_i, \text{next})$

1: **if** there is a node $v_h \in V$ with $U_h[g] = \mathcal{I}_h^i[c, d]$ and $d = \text{next}$ **then**
2: $U_h[g] := \mathcal{I}_h^i[c, d-1]$;
3: **if** $d - c < X$ **then**
4: $U_h[g] := \mathcal{R}_h$;
5: For all $0 \leq g' \leq c_i$, set $U_i[g'] = \mathcal{R}_h$ to role $U_i[g'] := \mathcal{E}$;
6: **end if**
7: **end if**

Algorithm 4 Subroutine - $\text{nextAloc}(v_i)$

1: **if** $Q_i \neq \emptyset$ **then**
2: **return** highest k with $U_i[k] = \mathcal{Q}$;
3: **else if** $E_i \neq \emptyset$ **then**
4: **return** highest k with $U_i[k] = \mathcal{E}$;
5: **else if** $R_i \neq \emptyset$ **then**
6: **return** highest k with $U_i[k] = \mathcal{R}_*$;
7: **else return** $c_i + 1$; {* Return *overflow unit* *}
8: **end if**

this is done in such a way that existing indirection units at $v_{L(t)}$ are never overwritten. The dynamic capacity of $v_{L(t)}$ is set to 0 in Line 2 of the subroutine, and the residual requirement $\hat{r}_i$ of v_i is reduced accordingly. Note that—as defined in Lines 18-26—the last two relays, $v_{L(b-1)}$ and $v_{L(b)}$, are treated in a subtly different way. By doing so, the algorithm guarantees that the number of units $\mathcal{R}_i$ allocated to $v_{L(b)}$ (Lines 20 or 25) is at least X. For this reason, in Lines 22-24, only $\hat{r}_i - X$ upload units of $v_{L(b-1)}$ are set to $\mathcal{R}_i$, whereas the remaining at most $X - 1$ unspecified units are set to $\mathcal{E}$.

Allocating upload units $\mathcal{R}_i$ to a relay v_j makes sense only when referencing these units using an indirection unit $\mathcal{I}_i^j$ at v_i. As defined in Lines 6-11 of the $\text{allocate}(v_i, v_{L(t)}, S)$ subroutine, upload units of $v_{L(t)}$ are assigned to $\mathcal{R}_i$ only if there are at least X of them. As mentioned before, the algorithm allows to overwrite upload units $\mathcal{R}_i$ by indirection units $\mathcal{I}_{L(t)}$ in the subsequent iteration in which the multicast tree of $v_{L(t)}$ is constructed. This reduces $v_{L(t)}$'s contribution to v_i's multicast tree by one unit. Thus, the overwriting diminishes the indirection's fan-out and reduces its effectiveness. In order to avoid cycles of overwritings of upload units $\mathcal{R}$ by indirection units $\mathcal{I}$, which themselves point to indirection units, etc..., the algorithm employs the $\text{reassign}(v_i, \text{next})$ subroutine. This subroutine keeps track of the fan-outs of indirection units and it overwrites any unit $\mathcal{I}_i^j$ whose fan-out decreases below X (see Figure 2 for an example). In this case v_i's requirements are no longer sufficiently satisfied by relay v_j and instead, $\mathcal{I}_i^j$ is set to a *direct* upload $\mathcal{R}_i$ in Line 4. That is, v_j is no longer a relay in v_i's multicast tree.

Another important question is *which* upload units of v_i should be used for its multicast tree (and thus be replaced by role $\mathcal{I}_i^{L(t)}$ in Line 9). The $\text{nextAloc}(v_i)$ subroutine first returns any upload unit in roles $\mathcal{Q}$ or $\mathcal{E}$. If no such unit exists, it starts overwriting upload units in role $\mathcal{R}_i$ in decreasing order of k. Finally, as mentioned above, in case all of v_i's upload units are in role $\mathcal{I}$, the algorithm resorts to the creation of a temporarily infeasible solution by allocating the new indirection unit to an "overflow" unit $U_i[c_i + 1]$.

5.2 Analysis

The first lemma of the analysis provides an upper bound of the optimal solution. The remainder of the proof then unfolds in a series of lemmas that collectively derive a lower bound on how many referenced upload units $\mathcal{R}$ are allocated to the different multicast trees in total. For the upper bound, consider the nodes to be numbered in non-increasing order of their capacity, i.e., $c_1 \leq \ldots \leq c_n$. Let the maximal *spread* of a set of nodes $W \subseteq V$ be defined as $\gamma(W) := \sum_{i \in W} c_i$. The first lemma bounds the optimum by giving an bound on the achievable total upload.

LEMMA 5.1. *The maximum number of requirements that can be satisfied by an optimal upload scheme is at most*

$$T_{OPT} \;\leq\; \left(1 + \frac{1}{c_{\min}}\right) \cdot \min_{W \subseteq V} \left\{ \sum_{i=0}^{\gamma(W)-1} c_{n-i} + \sum_{i \in V \setminus W} r_i \right\}.$$

PROOF. Let $W \subseteq V$ be an arbitrary subset of nodes and consider all multicast trees of nodes in W. The total number of relay nodes that can be used in all these trees is at most $\gamma(W)$. Therefore, the total number of upload units that can be utilized by nodes in W cannot exceed the complete upload capacity of nodes $v_{n-\gamma(W)+1}, \ldots, v_n$ plus their own capacity (first and second term). In addition, the optimum cannot do better than satisfying all requirements of nodes $V \setminus W$ completely (third term). Hence,

$$T_{OPT} \;\leq\; \min_{W \subseteq V} \left\{ \sum_{i=0}^{\gamma(W)-1} c_{n-i} + \sum_{i \in W} c_i + \sum_{i \in V \setminus W} r_i \right\}. \quad (1)$$

Because each node has a capacity of at least $c_{\min}$, it holds that $\sum_{i=0}^{\gamma(W)-1} c_{n-i} \geq c_{\min} \cdot \sum_{i \in W} c_i$. Solving this inequality for $\sum_{i \in W} c_i$ and plugging in the resulting bound into (1) yields the lemma. □

For the subsequent proofs, we call an upload unit in role $U_j[\ell] = \mathcal{R}_i$ at a node r_j, $i \neq j$, *referenced* if there exists an indirection unit $U_i[k] = \mathcal{I}_i^j[k_1, k_2]$ such that $k_1 \leq \ell \leq k_2$. Units that are directly routed to a receiver, i.e., $U_i[\ell] = \mathcal{R}_i$, are also called referenced. With this definition, the number of *referenced upload units* in role $\mathcal{R}_i$ at the end of the algorithm corresponds to the total number of satisfied requests T_{ALG}. The following lemma and its proof show that the algorithm avoids non-referenced upload units.

LEMMA 5.2. *At the end of every call of the* allocate*-subroutine, all upload units in role $\mathcal{R}$ are* referenced*.*

PROOF. Consider a node v_i. Requirements that are routed to a receiver directly (in an upload unit $U_i[k] = \mathcal{R}_i$) are referenced by definition. Consider an upload unit $U_j[k] = \mathcal{R}_i$ at some node $v_j \neq v_i$. The only place in the algorithm where this unit can be set

to $\mathcal{R}_i$ is Line 7 of the $\mathrm{allocate}(v_i, v_{L(t)}, S)$ subroutine. In Line 9, a corresponding indirection unit $\mathcal{I}_i^{L(t)}[k_1, k_2]$ with $k_1 \le k \le k_2$ is set up at v_i. It therefore remains to show that $\mathcal{I}_i^{L(t)}[k_1, k_2]$ stays as long as $U_{L(t)}[k] = \mathcal{R}_i$.

We first show that in Line 7 of the $\mathrm{allocate}$ subroutine an indirection unit $\mathcal{I}_i^{L(t)}[k_1, k_2]$ can never be overwritten. The reason is that the $\mathrm{nextAloc}(v_i)$ subroutine assigns indirection units to upload units with largest k first, and because only upload units $U_{\Delta c_{L(t)}+1,\ldots,\Delta c_{L(t)}+s}$ for the s previously determined in Lines 3 and 4 are overwritten, this cannot be an indirection unit.

This leaves the $\mathrm{reassignment}$ subroutine as the only place where an indirection unit $\mathcal{I}_i^{L(t)}[k_1, k_2]$ could potentially be overwritten or adjusted. In Line 2, the upper bound k_2 is decreased by one only if the corresponding upload unit $U_{L(t)}[k_2] = \mathcal{R}_i$ was overwritten (because the $\mathrm{nextAloc}(v_i)$ subroutine always returns the upload unit $U_{L(t)}[k] = \mathcal{R}$ with the largest index), and hence no longer exists. If the indirection unit $\mathcal{I}_i^{L(t)}[k_1, k_2]$ is overwritten in Line 4 of the $\mathrm{reassignment}$ subroutine, all upload units with $U_{L(t)}[k] = \mathcal{R}_i$ are overwritten and set to $\mathcal{E}$ in Line 5. Finally, in the last part of Algorithm 1, if an indirection unit is overwritten in Line 36, then all upload units at the relay are set to $\mathcal{E}$. Hence, all upload units in role $\mathcal{R}$ are referenced throughout the algorithm. □

The next lemma relates each node's dynamic capacity $\hat{c}_i$ to the number of its upload units to which a specified role is assigned.

LEMMA 5.3. *Throughout the algorithm and for every node $v_i \in V$, it holds that $|E_i \cup I_i \cup R_i| \ge c_i - \hat{c}_i$.*

PROOF. We prove by induction the stronger claim that for every $v_i \in V$, every upload unit $U_i[1, c_i - \hat{c}_i]$ is assigned to a specified role: $\mathcal{R}$, $\mathcal{E}$, or $\mathcal{I}$. At the outset of the algorithm, the induction hypothesis holds because $|E_i \cup I_i \cup R_i| = c_i - \hat{c}_i = 0$. We now consider the two places in the algorithm where $\hat{c}_i$ is reduced (and hence $c_i - \hat{c}_i$ increased). In Line 24, $\hat{c}_{L(b-1)} := 0$, but in the previous line, *all* unspecified upload units $U_{L(b-1)}[k] = \mathcal{Q}$ are set to $\mathcal{E}$. It follows that $|Q_i| = 0$ and hence, the induction holds.

In Line 2 of the $\mathrm{allocate}(v_i, v_{L(t)}, S)$ subroutine, $\hat{c}_{L(t)}$ is diminished by S. Let $\Delta c_{L(t)} = c_{L(t)} - \hat{c}_{L(t)}$ at the beginning of the subroutine. By induction hypothesis, all upload units $U_{L(t)}[1, \Delta c_{L(t)}]$ are assigned to specified roles initially. Hence, we need to show that upload units $U_{L(t)}[\Delta c_{L(t)} + 1, \Delta c_{L(t)} + S]$ are specified at the end of the subroutine. In Line 7, upload units $U_{L(t)}[\Delta c_{L(t)} + 1, \Delta c_{L(t)} + s]$ are set to role $\mathcal{R}$. It remains to consider upload units $U_{L(t)}[\Delta c_{L(t)} + s + 1, \Delta c_{L(t)} + S]$. In Line 4, the variable s is increased if an upload unit $U_{L(t)}[\Delta c_{L(t)} + 1, \Delta c_{L(t)} + S]$ is in a state other than $\mathcal{I}$. Since, indirection units are assigned in decreasing order of index by the $\mathrm{nextAloc}(v_i)$ subroutine, it follows that if $s < S$, all upload units $U_{L(t)}[\Delta c_{L(t)} + s + 1, \Delta c_{L(t)} + S]$ are in role $\mathcal{I}$. Finally, in Line 14, all unspecified upload units $U_{L(t)}[\Delta c_{L(t)} + 1, \Delta c_{L(t)} + S]$ are set to $\mathcal{E}$. □

The next two lemmas relate the total number of specified upload units to the optimal solution. Recall that nodes $v_1, \ldots, v_n$ are labeled in the order of their clumsiness ratio. Let $V_q \subseteq V$ be the first q nodes $v_1, \ldots, v_q$ with $r_i > X$ whose multicast trees are constructed by the algorithm. Further, define v_z, $1 \le z \le n$, to be the first node for which $b - a < c_z$, that is, v_z is the first node whose requirement is allocated without the overflow-unit.

LEMMA 5.4. *After the first $z-1$ iterations of the main* for*-loop of Algorithm 1, it holds that*

$$\sum_{v_i \in V_{z-1}} (c_i - \hat{c}_i) \ge \min_{W \subseteq V_{z-1}} \left\{ \sum_{i=0}^{\gamma(W)-1} c_{n-i} + \sum_{i \in V_{z-1} \setminus W} r_i \right\}. \quad (2)$$

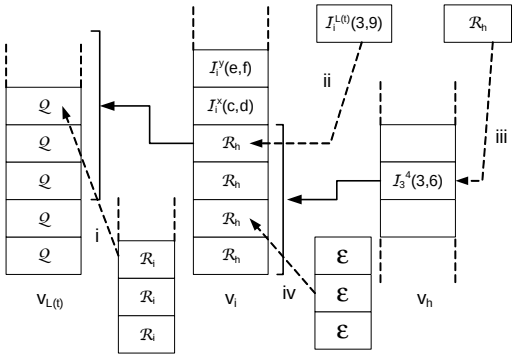

Figure 2: The figure depicts a reassignment. When constructing its multicast tree, v_i assigns upload units in role $\mathcal{R}_i$ to node $v_{L(t)}$ (i) and sets one of its units to $\mathcal{I}_i^{L(t)}$ (ii). If the number of upload units $\mathcal{R}_h$ at v_i drops below X , the $\mathrm{reassign}(v_i, \mathrm{next})$ subroutine replaces the indirection unit at v_h (iii) and sets upload units $\mathcal{R}_h$ at v_i to $\mathcal{E}$ (iv).

PROOF. The requirement of every node v_q, $1 \le q \le z-1$ is allocated to exactly $c_q + 1$ different relay nodes in its multicast tree (potentially including itself). In the first $j \le c_q - 1$ of these relay nodes, the $\mathrm{allocate}(v_q, v_{L(t)}, S)$ subroutine is called with parameter $S = \hat{c}_{L(t)}$ (Line 16) and hence, in Line 2 of the subroutine, the dynamic capacity is reduced to $\hat{c}_{L(t)} := 0$. Similarly, $\hat{c}_{L(b-1)} := 0$ in Lines 19 or 24. Finally, the dynamic capacity $\hat{c}_{L(b)}$ of the relay node referenced in the overflow-unit is reduced by at least $\min\{X, \hat{c}_{L(b)}\}$, because if the remaining requirement $\hat{r}_q$ after allocation to the first c_q relay nodes is less than X, the *else*-branch of Line 21 is executed, i.e., the number of $\mathcal{R}$-units assigned to $v_{L(b)}$ is rounded up to X. Notice that v_q therefore sets the dynamic capacity of at least c_q *consecutive nodes* in the list L to 0. The final $c_q + 1$st node may be left with positive dynamic capacity.

As $\hat{r}_q$ decreases by the same amount as the dynamic capacity $\hat{c}_{L(t)}$ in Line 2, total reduction of $\sum_{v_i \in V} \hat{c}_i$ during the main-loop iteration of node v_q is at least $r_q - \hat{r}'_q$, where $\hat{r}'_q$ is v_q's residual requirement at the end of the loop-iteration. Hence,the total decrease of residual requirements $\sum_{v_i \in V} \hat{r}_i$ is less or equal to the total decrease of dynamic capacity $\sum_{v_i \in V} \hat{c}_i$.

Assume that after constructing the multicast tree of v_{z-1}, the χ nodes with the highest (original) capacities c_i are all fully used up, i.e., $\hat{c}_j = 0$ for all $n - \chi + 1 \le j \le n$. I.e., $v_{n-\chi}$ is the highest-capacity node whose dynamic capacity is strictly positive after v_{z-1}'s loop iteration. Since in all iterations $q < z$ the dynamic capacity of *consecutive* nodes $v_{L(a)}, \ldots, v_{L(a+c_i+1)}$ is reduced, the following holds: Every node v_q that uses a relay node v_y for $y \le n - \chi$ has satisfied its *entire requirements*, i.e., $\hat{r}'_q = 0$. Let $Q_1 \subseteq V_{z-1}$ be the set of these nodes with a node v_y in their multicast tree. Each node in Q_1 has satisfied all its request and hence, the total reduction of $\sum_{v_i \in V} \hat{c}_i$ (and hence, the increase of the left-hand side of (2)) caused by nodes in Q_1 is at least $\sum_{v_q \in Q_1} r_q$.

Next, consider nodes $Q_2 = V_{z-1} \setminus Q_1$ that use only relay nodes $v_{n-\chi+1}, \ldots, v_n$. Because every such node $v_q \in Q_2$ sets the dynamic capacity of at least c_q consecutive relays to 0, it holds that the number χ of nodes with highest capacity c_j that have been set to $\hat{c}_j = 0$ during the first $z-1$ main-loop iterations is at least $\chi \ge \gamma(Q_2)$. Hence, the total reduction of $\sum_{v_i \in V} \hat{c}_i$ caused by nodes in Q_2 is at least $\sum_{j=0}^{\gamma(Q_2)-1} c_{n-j}$. Combining Q_1 and Q_2, it holds that the total reduction of dynamic capacity and thus, the left-hand side of (2) is at least$\sum_{v_q \in V_Q} (c_q - \hat{c}_q) \ge \sum_{v_q \in Q_1} r_q + \sum_{j=0}^{\gamma(Q_2)-1} c_{n-j}$, which proves the claim. □

For the following, we need one more definition. Define T^*_{OPT} to be the maximum number of satisfied requests in an optimal solution if nodes with $r_i < X$ are not considered. It holds that $T^*_{OPT} \leq T_{OPT}$, with equality if there are no nodes with $r_i < X$.

LEMMA 5.5. *In Line 28 of the algorithm, it holds that*

$$\sum_{v_i \in V} |E_i \cup I_i \cup R_i| \;\geq\; \left(\frac{c_{\min}}{c_{\min}+1}\right) \cdot T^*_{OPT}.$$

PROOF. Lemma 5.4 in combination with Lemma 5.3 proves the lower bound on $\sum_{v_i \in V_{z-1}} |E_i \cup R_i \cup I_i|$ for the first $z-1$ iterations of the algorithm's main *for*-loop. We now show that for every subsequent iteration $q \geq z$,

$$\sum_{v_i \in V_q} (c_i - \hat{c}_i) \;\geq\; \min_{W \subseteq V_q} \left\{ \sum_{i=0}^{\gamma(W)-1} c_{n-i} + \sum_{i \in V_q \setminus W} r_i \right\} \qquad (3)$$

continues to hold. The proof is then concluded by plugging in the bound on the optimal solution in Lemma 5.1.

Recall that v_z is the first node for which $b - a < c_z$. There are two possible reasons for this. First, it could be that $\sum_{p=1}^{b} \hat{c}_{L(p)} \geq r_z$ for $b \leq c_z$, i.e., the entire requirement of v_z can be satisfied using the $b \leq c_z$ relay nodes with smallest positive dynamic capacity. Second, it could be that $\sum_{p=1}^{c_z} \hat{c}_{L(p)} < r_z$, but there are only c_z or less relay nodes with non-zero dynamic capacity left.

We start with the second case and consider the iteration of node v_q, $q \geq z$. In this case, the algorithm guarantees that either, r_q is fully satisfied (in which case, $c_q - \hat{c}_q$ as well as the right-hand side of (3) increases by exactly r_q) or the dynamic capacity of every node $v \in V$ is 0 after the iteration. In this case, both sides of (3) sum up to exactly $\sum_{v \in V} c_v$.

Now, assume that in the iteration of node v_z, the first case applies, i.e., v_z's requirement r_z can be fully satisfied using the $b \leq c_z$ relays $v_{L(1)}, \ldots, v_{L(b)}$. At the beginning of the iteration, the residual requirement $\hat{r}_z$ is r_z. For each of the $b \leq c_z$ relays, $\hat{r}_z$ is reduced by at most $\hat{c}_{L(j)}$, where $\hat{c}_{L(j)}$ denotes the relay's dynamic capacity before the allocation of v_z. As r_z reaches 0, node $v_{L(b)}$ must have had a dynamic capacity of at least

$$\hat{c}_{L(b)} \;\geq\; r_z/c_z \;\geq\; r_q/c_q, \qquad (4)$$

for each node v_q with $q > z$ before this iteration of the loop. The first inequality is due to the ordering of nodes in the list L according to their dynamic capacity. The second inequality holds because multicast trees are constructed in order of the nodes' "clumsiness ratio" r_i/c_i in Lines 4 and 5. That is, if $q > z$, then $r_z/c_z \geq r_q/c_q$.

Consider the situation at the beginning of the next iteration, when v_{z+1} is considered. After satisfying the requirements of v_z, the dynamic capacities of $v_{L(1)}, \ldots, v_{L(b-1)}$ are set to 0 and the dynamic capacity $\hat{c}_{L(b)}$ decreases. Consequently, node $v_{L(b)}$ in v_z's iteration becomes the new node $v_{L(1)}$ in v_{z+1}'s iteration. Also, all nodes $v_{L(j)}$ for $j > b$ had a higher dynamic capacity $\hat{c}_{L(j)}$ than $\hat{c}_{L(b)}$. Hence, due to Inequality (4), it holds that for each of these nodes $\hat{c}_{L(j)} \geq r_z/c_z \geq r_q/c_q$ for $q \geq z$. In the new iteration (after the reordering of L), it therefore holds that each node $v_{L(2)}, v_{L(3)}, \ldots$ has a dynamic capacity of at least r_q/c_q, which implies that $\sum_{j=2}^{c_q+1} \hat{c}_{L(j)} \geq r_q$. This proves that as long as there are enough non-empty relay nodes, the entire requirement of v_q is satisfied and allocated to at most c_q relays. As the total decrease of dynamic capacity is r_q, both sides of (3) increase equally. Finally, in Lines 29-31, only the lemma's left-hand side may increase. □

As pointed out earlier, the initial solution obtained by the algorithm may be infeasible as some nodes may be assigned an overflow-unit to store one additional indirection unit. The following lemma states that the transformation into a feasible solution does not significantly reduce the overall upload bandwidth.

LEMMA 5.6. *Let I_i, R_i, and I'_i, R'_i denote the corresponding sets before and after Lines 33-40 of the algorithm, respectively. It holds that for each $v_i \in V$,*

$$|R'_i \cup I'_i| \;\geq\; \left(1 - \frac{1}{c_{\min}+1}\right) \cdot |R_i \cup I_i|.$$

PROOF. In Lines 33-40, Algorithm 1 considers all nodes v_i with specified overflow-unit, i.e., nodes that are sending their data via $c_i + 1$ different relays. For each such node, if there exists any upload unit in role $\mathcal{E}$ or $\mathcal{Q}$, the indirection unit $U_i[c_i+1]$ is moved to this unit. In this case, $|R'_i \cup I'_i| = |R_i \cup I_i|$. If no such units exist, an upload unit in role $\mathcal{R}$ is overwritten with $U_i[c_i+1]$. Because in this case there are at least $c_{\min}$ specified units in v_i and $|E_i| = 0$, $|R_i \cup I_i|$ is reduced by at most a factor of $1/(c_{\min}+1)$.

Finally, if v_i contains only units $\mathcal{I}$, the algorithm replaces the indirection unit with *smallest fan-out* (Lines 36, 39). This replacement leaves v_i's multicast tree with at most c_i relays, and the number of satisfied requests of v_i is reduced by at most a fraction of $1/(c_i+1) \leq 1/(c_{\min}+1)$. □

The following lemma describes the ratio between indirection units and upload units. Essentially, the theorem follows directly from the construction of the algorithm.

LEMMA 5.7. *Throughout the algorithm, it holds that*

$$\sum_{v_i \in V} |I_i| \;\leq\; \frac{1}{X} \sum_{v_i \in V} |R_i|.$$

PROOF. We show that every indirection unit $\mathcal{I}$ can be mapped to a *distinct* set of at least X referenced upload units in role $\mathcal{R}$, i.e., the fan-out of every indirection unit is at least X. The only place in the algorithm where a new indirection unit $\mathcal{I}_i^{L(t)}[k_1, k_2]$ is established is in Line 9 of the $\mathrm{allocate}(v_i, v_{L(t)}, S)$ subroutine. Before this, however, all upload units $U_{L(t)}[\Delta c_{L(t)}+1], \ldots, U_{L(t)}[\Delta c_{L(t)}+s]]$ are set to $\mathcal{R}_i$ (Line 7). As Line 6 implies $s \geq X$, the initial fan-out of $\mathcal{I}_i^{L(t)}[k_1, k_2]$ is at least X.

We now show that the fan-out of every existing indirection unit remains at least X. The only places where upload units $\mathcal{R}_i$ could potentially be overwritten are Lines 7 and 9 of the $\mathrm{allocate}$ subroutine, and in Line 5 of the $\mathrm{reassign}$ subroutine. In Line 7, overwriting is impossible as for each $v_i \in V$ the dynamic capacity $\hat{c}_i$ is reduced whenever upload units $U_i[k]$ are set to role $\mathcal{R}$ or $\mathcal{E}$. Hence, there is no $U_i[k] = \mathcal{R}$ for $k > c_{L(t)} - \hat{c}_{L(t)} = \Delta c_{L(t)}$.

Next, consider Line 9 of the $\mathrm{allocate}$ subroutine. Here, an upload unit $\mathcal{R}_i$ at v_j to which $\mathcal{I}_i^j[k_1, k_2]$ was pointing may be replaced by a new indirection unit $\mathcal{I}_j$. In this case, $\mathrm{reassign}(v_i, \mathrm{next})$ is invoked, which updates the fan-out to the new value $\mathcal{I}_i^j[k_1, k_2 - 1]$ in Line 2. If the reduced fan-out $k_2 - k_1$ is still at least X, the claim continues to hold and nothing happens. If the reduced fan-out drops below X, reassignment occurs (Lines 3-6): The indirection unit $\mathcal{I}_i^j[k_1, k_2 - 1]$ is replaced by a direct upload unit $\mathcal{R}_i$ (see Figure 2). That is, an existing indirection unit is removed as soon as its fan-out drops below X. □

Algorithm 1 may assign the role $\mathcal{E}$ to certain upload units, while still decreasing these nodes' dynamic capacity. Unless these units are subsequently overwritten with roles $\mathcal{R}$ or $\mathcal{I}$, they are wasted. The following two lemmas state that the algorithm does not assign too many empty slots to any node.

LEMMA 5.8. *At Line 28 of the algorithm, it holds for every node $v_i \in V$ that $|E_i| > 0 \Rightarrow |Q_i| = 0$.*

PROOF. An upload unit $U_i[k]$ can be set to $\mathcal{E}$ in Line 14 of the $\mathrm{allocate}(v_j, v_i, S)$ subroutine, in Line 5 of the $\mathrm{reassign}(v_i, \mathrm{next})$ subroutine, or in Line 23 of the main algorithm. We show that in any case, an upload unit is set to $\mathcal{E}$ only if $|Q_i| = 0$.

First consider Line 5 of $\mathrm{reassign}(v_i, \mathrm{next})$. By definition of $\mathrm{nextAloc}(v_i)$, a newly allocated $\mathcal{I}_i$ is always written in any available upload unit in roles $\mathcal{Q}$ or $\mathcal{E}$ first. The *if*-statement in Line 3 of the $\mathrm{reassign}(v_i, \mathrm{next})$ subroutine implies that Line 5 is only executed if the fan-out of an indirection unit dropped below X. That is, a newly allocated $\mathcal{I}$ role must have replaced an $U_i[k] = \mathcal{R}_h$ unit for some v_h, from which it follows that all upload units at v_i are in role either $\mathcal{I}$ or $\mathcal{R}$. From this, we derive that $|Q_i| = 0$.

Now, consider Lines 13-15 of the $\mathrm{allocate}(v_j, v_i, S)$ subroutine. We distinguish three cases, depending on whether the selected relay node v_i is one of the earlier relay nodes, $v_i = v_{L(t)}$, $t \in \{a, \ldots, b-2\}$, the second to last relay node, $v_i = v_{L(b-1)}$, or the last one, $v_i = v_{L(b)}$, respectively.

In the first case, it holds that $S = \hat{c}_i$ and hence, after the subroutine, all upload units of v_i are specified and $\hat{c}_i := 0$. In other words, $|I_i \cup R_i \cup E_i| = c_i$ and consequently, $|Q_i| = 0$. As for the second to last relay node $v_i = v_{L(b-1)}$, we distinguish two cases. If this node is allocated in Line 19, then the entire node is filled up and hence, with the same argument as above, $|Q_i| = 0$. If this node is allocated in Line 22, all its upload units are also set to a role other than $\mathcal{Q}$ in subsequent Line 23. Hence, $|Q_i| = 0$ in this case, too. Finally, consider the last relay node $v_{L(b)}$. By the definition of Lines 20 and 25, at least X upload units are allocated. Hence, if all these upload units can indeed be set to role $\mathcal{R}_j$, then $s \geq X$ and therefore, Lines 13-15 of $\mathrm{allocate}(v_j, v_i, S)$ are not executed, i.e., $|E_i| = 0$. If fewer than X upload units at $v_{L(b)}$ can be set to $\mathcal{R}_j$, then, because the $\mathrm{nextAloc}(v_i)$ subroutine overwrites $\mathcal{Q}$ roles in decreasing order, it must also hold that $|Q_i| = 0$. The reason is that at least one intended upload unit for the $\mathcal{R}_j$ role is already occupied with an $\mathcal{I}$ role. In either case, the lemma holds. □

LEMMA 5.9. *At Line 28 of the algorithm, it holds for every $v_i \in V$ that $|E_i| < X$.*

PROOF. The proof is by induction. At the outset of the algorithm, it holds that $U_i[1, c_i] = \mathcal{Q}$. As pointed out, there are three places before Line 28 where $\mathcal{E}$ may be assigned to upload units.

Consider Line 14 of the $\mathrm{allocate}(v_i, v_{L(t)}, S)$ subroutine. Assume that after assigning role $\mathcal{E}$ in Lines 13-15, there were $Y \geq X$ upload units with role $\mathcal{E}$ at v_i. As the *else*-branch of the *if*-statement has been executed, we know that $s < X$. Because s expresses how many of the S upload units are not currently used as indirection units (Line 4), the number of newly assigned $\mathcal{E}$ roles in Line 14 is upper bounded by $s < X$. This, implies that there must have been $Y - s > 0$ upload units in role $\mathcal{E}$ already *before* the subroutine call. Because of $|E_i| > 0$, it follows from Lemma 5.8 that $|Q_i| = 0$ and hence, $|R_i| + |I_i| + |E_i| \geq c_i$. In Line 14, neither units in role $\mathcal{I}$ nor $\mathcal{R}$ are set to $\mathcal{E}$, i.e., $|E_i|$ does not increase and the induction hypothesis continues to hold.

Next, consider Line 5 of the $\mathrm{reassign}(v_i, \mathrm{next})$ subroutine. As already shown in the proof of Lemma 5.8, when Line 5 is executed, an upload unit newly set to role $\mathcal{I}$ must have replaced an $U_i[k] = \mathcal{R}_h$ unit for some v_h, and hence, all upload units at v_i are in role either $\mathcal{I}$ or $\mathcal{R}$. Furthermore, the maximum number of upload units set to role $\mathcal{R}_h$ in Line 5 is less than X by definition, because the *if*-statement in Line 3 evaluates to true only if there are less than X remaining referenced $\mathcal{R}_h$ units at v_i (see Figure 2). That is, if the claim holds before a reassignment, it continues to hold afterwards. □

THEOREM 5.10. *Algorithm 1 computes a feasible solution to the MUBP problem and achieves an approximation ratio α, where*

$$\alpha \geq 1 - \frac{4\sqrt{c_{\min}+2} - 4}{c_{\min}+1} + \frac{2\sqrt{c_{\min}+2} - 3}{c_{\min}} \geq 1 - \frac{2}{\sqrt{c_{\min}}}.$$

PROOF. We denote by I_i'', E_i'', and R_i'' the corresponding sets after Line 28, by I_i', E_i', and R_i' after Line 32, and by I_i, E_i, and R_i at the end of the algorithm. We start the proof by placing a lower bound on $\Lambda := \sum_{v_i \in V} |R_i' \cup I_i'|$ in terms of T_{OPT}. By applying the relationship between the total number of upload units in roles $\mathcal{R}$ and $\mathcal{I}$, this bound can be transformed into a bound on the number of satisfied requests. Finally, we compute the optimal value for X.

Partition the set of nodes V into two sets $V_<$ and $V_\geq$ containing the nodes with requirements $r_i < X$ and $r_i \geq X$, respectively. Lemma 5.5 implies that after constructing the multicast tree of all nodes in $V_\geq$ (Line 28), the total number of allocated specified upload units is $\sum_{v_i \in V} |E_i'' \cup R_i'' \cup I_i''| \geq (\frac{c_{\min}}{c_{\min}+1}) \cdot T^*_{OPT}$.

In Lines 29-31, each node $v_i \in V_<$ may replace up to r_i upload units in roles $\mathcal{E}$ or $\mathcal{Q}$ to $\mathcal{R}_i$ on its own machine, thereby satisfying its own requests. If there are fewer than r_i units in roles $\mathcal{E} \cup \mathcal{Q}$ available, however, the remainder of v_i's requirement is not satisfied. Let $\check{r}_i \leq r_i$ denote the number of requests of a node $v_i \in V_<$ that are *not* satisfied. Using this notation, we can express Λ as

$$\Lambda = \sum_{v_i \in V} |E_i'' \cup R_i'' \cup I_i''| - \sum_{v_i \in V} |E_i''| + \sum_{v_i \in V_<} r_i - \sum_{v_i \in V_<} \check{r}_i.$$

This can be rewritten as $\Lambda = \Lambda_\geq + \Lambda_<$, where

$$\Lambda_\geq = \sum_{v_i \in V_\geq} (|E_i'' \cup R_i'' \cup I_i''| - |E_i''|)$$
$$\Lambda_< = \sum_{v_i \in V_<} (|E_i'' \cup R_i'' \cup I_i''| - |E_i''| + r_i - \check{r}_i).$$

In the sequel, we bound $\check{r}_i$ and $|E_i''|$. First, consider a node $v_i \in V$. We know from Lemma 5.9 that $|E_i''| < X$. Moreover, Lemma 5.8 shows that $|E_i''|$ is larger than 0 only if $|Q_i''| = 0$ and consequently, only if $|R_i'' \cup I_i'' \cup E_i''| = c_i$. Hence, for every $v_i \in V$, $|E_i''|$ can be expressed as $|E_i''| \leq (\frac{X-1}{c_i}) \cdot |R_i'' \cup I_i'' \cup E_i''|$, and therefore

$$\Lambda_\geq \geq \left(1 - \frac{X-1}{c_i}\right) \cdot \sum_{v_i \in V_\geq} |R_i'' \cup I_i'' \cup E_i''|. \qquad (5)$$

While this bound is sufficiently good for all nodes $v_i \in V_\geq$, we now derive a stronger bound for $\Lambda_<$.

Let $\Lambda_<^i = |E_i'' \cup R_i'' \cup I_i''| - |E_i''| + r_i - \check{r}_i$. We bound $\Lambda_<$ by considering each $\Lambda_<^i$ individually; distinguishing the following three cases:

1) $|\mathbf{E_i'}| > 0$: It holds for each node $v_i \in V_<$ that if $|E_i'| > 0$, then $\check{r}_i = 0$. This is true because if $|E_i'| > 0$ and $\check{r}_i > 0$, v_i would have set these empty upload units to $\mathcal{R}_i$, thereby reducing $\check{r}_i$ in Lines 29-31. Hence, $\Lambda_<^i = |E_i'' \cup R_i'' \cup I_i''| - |E_i''| + r_i$, and when plugging in the bound for $|E_i''|$, $\Lambda_<^i \geq (1 - \frac{X-1}{c_i})|E_i'' \cup R_i'' \cup I_i''| + r_i$.

2) $|\mathbf{E_i'}| = 0$ and $|\mathbf{E_i''}| > 0$: Because $|E_i''| > 0$ implies $|Q_i''| = 0$, it holds that $\check{r}_i + |E_i''| = r_i$ for every node $v_i \in V_<$ with $|E_i'| = 0$. Plugging in this equation yields $\Lambda_<^i = |E_i'' \cup R_i'' \cup I_i''| - r_i + \check{r}_i + r_i - \check{r}_i = c_i$. Because r_i is at most $X - 1$ by definition of $V_<$, and due to $|E_i'' \cup R_i'' \cup I_i''| = c_i$, it holds that

$$\Lambda_<^i = c_i \geq \left(1 - \frac{X-1}{c_i}\right)(|E_i'' \cup R_i'' \cup I_i''| + r_i).$$

3) $|\mathbf{E_i'}| = 0$ and $|\mathbf{E_i''}| = 0$: In this case, $\Lambda_<^i = |E_i'' \cup R_i'' \cup I_i''| + r_i - \check{r}_i$. Because $\check{r}_i$ is by definition at most $X - 1$, it follows that $\Lambda_<^i \geq \left(1 - \frac{X-1}{c_i}\right)(|E_i'' \cup R_i'' \cup I_i''| + r_i)$.

Each node $v_i \in V_<$ is covered by exactly one of the cases and we therefore obtain

$$\Lambda_< \;\geq\; \left(1 - \frac{X-1}{c_i}\right) \cdot \sum_{v_i \in V_<} \left(|R_i'' \cup I_i'' \cup E_i''| + r_i\right)$$

In total, this yields

$$\begin{aligned}\Lambda \;&\geq\; \left(1 - \frac{X-1}{c_{\min}}\right) \Big(\sum_{v_i \in V} |E_i'' \cup R_i'' \cup I_i''| + \sum_{v_i \in V_<} r_i \Big) \\ &\geq\; \left(1 - \frac{X-1}{c_{\min}}\right) \Big(\frac{c_{\min}}{c_{\min}+1} \cdot T^*_{OPT} + \sum_{v_i \in V_<} r_i \Big) \\ &\geq\; \left(1 - \frac{X-1}{c_{\min}}\right) \left(\frac{c_{\min}}{c_{\min}+1} \right) T_{OPT}.\end{aligned}$$

The second inequality is due to Lemma 5.5 and the last inequality follows from the observation that the optimal solution cannot be better than satisfying *all* requirements of nodes in $V_<$ in addition to T^*_{OPT}, and therefore $T^*_{OPT} + \sum_{v_i \in V_<} r_i \geq T_{OPT}$. Plugging in the result of Lemma 5.6 yields

$$\sum_{v_i \in V} |R_i \cup I_i| \geq \left(1 - \frac{1}{c_{\min}}\right) \Lambda \geq \left(1 - \frac{X-1}{c_{\min}}\right) \left(\frac{c_{\min}-1}{c_{\min}+1} \right) T_{OPT}.$$

It follows by Lemma 5.7 that $\sum_{v_i \in V} |I_i| \leq \frac{1}{X} \sum_{v_i \in V} |R_i|$ and consequently, $\sum_{v_i \in V} |R_i| \geq \beta \cdot \sum_{v_i \in V} |R_i \cup I_i|$ for $\beta = \frac{X}{X+1} = 1 - \frac{1}{X+1}$. Using this bound, we obtain

$$\sum_{v_i \in V} |R_i| \geq \left(1 - \frac{1}{X+1}\right) \left(1 - \frac{X-1}{c_{\min}}\right) \left(\frac{c_{\min}-1}{c_{\min}+1} \right) T_{OPT}.$$

Since by Lemma 5.2, all upload units in role $\mathcal{R}$ are referenced, it follows that $T_{ALG} \geq \alpha T_{OPT}$ for $\alpha := (1 - \frac{1}{X+1})(1 - \frac{X-1}{c_{\min}})(\frac{c_{\min}-1}{c_{\min}+1})$. The value X that maximizes this approximation ratio is determined by $\frac{\delta \alpha}{\delta X} \stackrel{\Delta}{=} 0$, which yields $X = \sqrt{c_{\min}+2} - 1$, the value used in the algorithm. The proof is concluded by plugging this value into the expression for α. □

Notice that the exact bound on the approximation ratio α is significantly higher than $1 - 2/\sqrt{c_{\min}}$ for small values of $c_{\min}$. For $c_{\min} = 5$, for instance, $1 - 2/\sqrt{c_{\min}} \approx 0.1$, whereas the exact bound is approximately 0.36.

6. THE IMPACT OF NETWORK CODING

The original problem statement of the MUBP problem assumes that all information destined for a specific receiver is sent to this receiver on a specific single path. Alternatively, there has recently been a growing (theoretical and practical) interest in network coding techniques that allow mixing of information at intermediate nodes [2]. In our problem setting, it is conceivable that a sender *splits* its data, sends each piece to a different relay node in its multicast tree, and has the receiver combine these pieces to recover the original data packet. This section briefly points out theoretical properties and practical limitations of this simple *network coding*.

The MUBP problem stated in Section 3 can be formulated as the following integer linear program (ILP). The variable d_{ik} is 1 if v_i directly sends a unit of data to v_k. Similarly, the binary variable f_{ijk} indicates whether node v_i sends data to node v_k via relay v_j. Finally, s_{ik} is 1 if v_i sends exactly one unit of data to its receiver $v_k \in R(i)$.

$$\begin{aligned}\max \quad & \sum_{v_i \in V} \sum_{v_k \in R(i)} s_{ik} \\ & d_{ik} + \sum_{v_j \in V} f_{ijk} \;\geq\; s_{ik},\ \forall v_i \in V, v_k \in R(i) \\ & \sum_{v_k \in V} d_{ik} + \sum_{v_j \in V} \sum_{v_k \in V} f_{jik} \;\leq\; c_i,\ \forall v_i \in V \\ & f_{ijk} - d_{ij} \;\leq\; 0,\ \ \forall v_i, v_j, v_k \in V \\ & d_{ik}, f_{ijk}, s_{ik} \;\in\; \{0,1\},\ \ \forall v_i, v_j, v_k \in V\end{aligned}$$

The first constraint describes that a successful upload requires the entire data unit to be sent from v_i to $v_k \in R(i)$. The second condition states that the total bandwidth used by v_i in all multicast trees must not exceed c_i. The third constraint captures the fact that v_j can serve as a relay for v_i only if v_i sends data to v_j.

If all integrality constraints are relaxed to $0 \leq d_{ik}, f_{ijk}, s_{ik} \leq 1$, we obtain a *fractional MUBP*, whose optimum can be computed in polynomial time by solving the resulting linear program. However, this fractional problem does not correspond to *MUBP with network coding*, because whereas the fractional problem allows data to be uploaded fractionally, in the MUBP problem, an upload can be considered successful only if it is received in its entirety: $s_{ik} = 1$. Hence, in the MUBP with network coding, only the integrality constraints of d_{ik} and f_{ijk} are relaxed. Unlike the fractional problem, MUBP with network coding may be NP-hard.

Splitting and reuniting "flows" in an arbitrarily fine-grained manner allows for solutions in which the total available upload capacity C is highly utilized. Recall that the total number of requests is R.

THEOREM 6.1. *Let $\chi := C/R$ be the ratio describing the total available spare capacity in the network. There is an algorithm ALG such that, if $\chi \geq \frac{c_{\min}}{c_{\min}-1}$, then $T_{ALG} = R$. Otherwise, $T_{ALG} \geq (1 - 1/c_{\min})C - 1$.*

PROOF. The bound can be achieved if each node serves as a relay node for every other node. Specifically, ALG consider the nodes $v \in V$ in order of non-decreasing requirement $r_1 \leq \ldots \leq r_n$. When node v_i is considered, send c_j/C of data from v_i to each node $v_j \in V$. (Note that a node also serves as its own relay.) For each receiver $v_k \in R(i)$, every such relay v_j allocates an upload bandwidth of c_j/C. Because node v_i needs to send updates to r_i many receivers, v_j allocates a total upload bandwidth of $(c_j r_i)/C$ for serving v_i.

First, consider the case $\chi \geq \frac{c_{\min}}{c_{\min}-1}$. The total bandwidth b_i allocated by a node v_i is at most

$$b_i = \sum_{v_j \in V} \frac{c_j}{C} + \sum_{v_j \in V} \frac{c_i r_j}{C} = 1 + \frac{c_i}{\chi} \;\leq\; c_i,$$

that is, the resulting upload scheme is feasible. Also, all requests are successfully uploaded because $\sum_{v_i \in V} \sum_{v_j \in V} (c_j r_i)/C = R$.

In case $\chi < \frac{c_{\min}}{c_{\min}-1}$, the resulting schedule may be infeasible as nodes become overloaded. At each node v_i, one upload unit is used for sending v_i's data to its relay nodes, and the remaining $c_i - 1 \geq c_{\min} - 1$ units are used for sending other nodes' data to their receivers. Hence, the total number of requests satisfied is at least $(1 - 1/c_{\min})C - 1$. The -1 comes from the fact that one request may only be partially satisfied (the request that is allocated when nodes become overloaded). □

As the optimal solution T_{OPT} is clearly bounded from above by R and C, the algorithm discussed in the proof improves the approximation ratio of Algorithm 1 for the network coding problem.

COROLLARY 6.2. *The algorithm described in the proof of Theorem 6.1 achieves an approximation ratio of $1 - O(1/c_{\min})$ for* MUBP with network coding.

Theorem 6.1 implies that for $C \geq c_{\min}/(c_{\min}-1) \cdot R$, all requirements are successfully uploaded. Theoretically, network coding could thus be used to find a virtually optimal bandwidth allocation using only *two hops* between sender and receiver. This implies that there are multicast trees in which the latency constraints are satisfied at essentially no additional cost. In contrast, the achievable upload bandwidth is significantly smaller without network coding, and it may not be possible to upload all requirement for any $C < n/c_{\min} \cdot R$.

THEOREM 6.3. *There are instances in which—without network coding—any upload scheme can satisfy only* $(c_{\min}+1)\cdot\frac{C}{n}$ *requests.*

PROOF. Consider an instance with one node v having requirement $r_v = R$ and all $r_i = 0$ for all other nodes. Every node has upload capacity $c_{\min}$. The best achievable upload U of any scheme in this instance is $U = c_{\min}^2 + c_{\min} = (\frac{C}{n})(c_{\min}+1)$. □

Unfortunately, the astonishing power of network coding is to some extent theoretical gimmick. In practical scenarios, application layer data packets (as small as they may be) are wrapped in UDP or TCP with headers of at least 40 bytes. This per-packet overhead limits the practicability of network coding schemes as splitting data into small flows results in a significant net-increase of network traffic.

7. CONCLUSION

The latency-sensitive upload bandwidth maximization problem is of critical importance in distributed peer-to-peer-based multiplayer games. The higher the utilization of the cumulative upload capacity of all peers, the more accurately the players can view and play the game. In this paper, we have formalized the problem as a combinatorial maximization problem and presented a computationally efficient algorithm whose total upload is within a small fraction of the optimum even in worst-case scenarios.

Focusing on the core algorithmic ideas, out solution abstracts away various practically important aspects of the problem, including NAT-ed nodes [15], which may thwart our assumption of having a complete graph overlay. And, although it may come at the cost of more complicated routing, it would be interesting to quantify the achievable gain when relaxing the restriction to depth 2 multicast trees. On the theory side, our paper leaves open the exact approximability of the problem. It would be interesting to either devise a PTAS for the problem, or rule out its existence.

8. ACKNOWLEDGEMENTS

We would like to thank the anonymous SPAA reviewers for their valuable comments.

9. REFERENCES

[1] http://planetside.station.sony.com.

[2] R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung. Network Information Flow. *IEEE Transactions on Information Theory*, 2000.

[3] G. Baier. Flows with Path Restrictions. PhD Thesis, TU Berlin, 2003.

[4] G. Baier, T. Erlebach, A. Hall, E. Köhler, H. Schilling, and M. Skutella. Length-Bounded Cuts and Flows. In *Proc. 33rd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 679–690, 2006.

[5] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable Application Layer Multicast. In *Proc. ACM SIGCOMM*, 2002.

[6] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller. Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications. In *Proc. 23th IEEE Infocom*, 2003.

[7] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool. The Effects of Loss and Latency on User Performance in Unreal Tournament 2003. In *Proc. 3rd Workshop on Network and System Support for Games (NETGAMES)*, pages 144–151, 2004.

[8] Y. W. Bernier. Latency Compensating Methods in Client/Server In-Game Protocol Design and Optimization. In *Game Developers Conference*, 2001.

[9] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting Scalable Multi-Attribute Range Queries. In *Proc. ACM SIGCOMM*, 2004.

[10] A. R. Bharambe, V. N. Padmanabhan, and S. Seshan. Supporting Spectators in Online Multiplayer Games. In *Proc. of HotNets*, 2004.

[11] M. S. Borella. Source Models of Network Game Traffic. *Computer Communications*, 23(4), 2000.

[12] E. Brosh and Y. Shavitt. Approximation and Heuristic Algorithms for Minimum Delay Application-Layer Multicast Trees. In *Proc. 24th IEEE Infocom*, 2004.

[13] Y.-H. Chu, S. G. Rao, S. Seshan, and H. Zhang. A Case for End System Multicast. *IEEE Journal on Selected Areas in Communications*, 20(8), 2002.

[14] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment Issues for the IP Multicast Service and Architecture. *IEEE Networks*, 14(1), 2000.

[15] B. Ford, P. Srisuresh, and D. Kegel. Peer-to-Peer Communication Across Network Address Translators. In *Proc. 2005 USENIX Annual Technical Conference*, 2005.

[16] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[17] C. GauthierDickey, D. Zappala, V. Lo, and J. Marr. Low Latency and Cheat-Proof Event Ordering for Peer-to-Peer Games. In *Proc. 14th NOSSDAV*, 2004.

[18] L. Gautier and C. Diot. Design and Evaluation of MiMaze, a Multi-Player Game on the Internet. In *Proc. IEEE International Conference on Multimedia Computing and Systems*, 1998.

[19] D. A. Helder and S. Jamin. End-Host Multicast Communication Using Switch-Trees Protocols. In *Proc. 2nd ACM/IEEE Symposium on Cluster Computing and the Grid*, 2002.

[20] J. Janotti, D. K. Gifford, D. K. Johnson, K. L. Kaashoek, and J. R. O'Toole. Overcast: Reliable Multicasting with an Overlay Network. In *Proc. 4th Symposium on Operating Systems Design and Implementation (OSDI)*, 2000.

[21] M. Karpinski, I. I. Mandoiu, A. Olshevsky, and A. Zelikovsky. Improved Approximation Algorithms for the Quality of Service Multicast Tree Problem. *Algorithmica*, 42(2), 2005.

[22] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peet-to-Peer Support for Massively Multiplayer Games. In *Proc. 24th IEEE Infocom*, 2004.

[23] G. Kortsarz and D. Peleg. Approximating the Weight of Shallow Steiner Trees. *Discrete Applied Mathematics*, 93(2-3), 1999.

[24] J. Pang, F. Uyeda, and J. R. Lorch. Scaling Peer-to-Peer Games in Low-Bandwidth Environments. In *Proc. 6th Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, 2007.

[25] P. Quax, P. Monsieurs, W. Lamotte, D. D. Vleeschauwer, and N. Degrande. Objective and Subjective Evaluation of the Influence of Small Amounts of Delay and Jitter on a Recent First Person Shooter Game. In *Proc. 3rd Workshop on Network and System Support for Games (NETGAMES)*, pages 152–156, 2004.

[26] G. Robins and A. Zelikovski. Improved Steiner Tree Approximation in Graphs. In *Proc. 11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2000.

[27] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Proc. 18th IFIP/ACM Int. Conference on Distributed Systems Platforms*.

[28] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proc. ACM SIGCOMM*, 2001.

[29] J. Vogel, J. Widmer, D. Farin, M. Mauve, and W. Effelsberg. Priority-Based Distribution Trees for Application-Level Multicast. In *Proc. 2nd Workshop on Network and System Support for Games (NETGAMES)*, 2003.