Growing Half-Balls: Minimizing Storage and Communication Costs in CDNs[⋆]

Reuven Bar-Yehuda[1], Erez Kantor[1], Shay Kutten[1⋆⋆], and Dror Rawitz[2]

[1] Technion, Haifa 32000, Israel.
reuven@cs.technion.ac.il, erez.kantor@gmail.com, kutten@ie.technion.ac.il
[2] Tel-Aviv University, Tel-Aviv 69978, Israel.
rawitz@eng.tau.ac.il

Abstract. The Dynamic Content Distribution problem addresses the trade-off between storage and delivery costs in modern virtual Content Delivery Networks (CDNs). That is, a video file can be stored in multiple places so that the request of each user is served from a location that is near by to the user. This minimizes the delivery costs, but is associated with a storage cost. This problem is NP-hard even in grid networks. In this paper, we present a constant factor approximation algorithm for grid networks. We also present an $O(\log \delta)$-competitive algorithm, where δ is the normalized diameter of the network, for general networks with general metrics. We show a matching lower bound by using a reduction from online undirected Steiner tree. Our algorithms use a rather intuitive approach that has an elegant representation in geometric terms.

1 Introduction

This paper reports research that is been performed as a part of Net-HD [1], an Israeli consortium of Industry and academia, designing a Content Delivery Network (CDN) over the Internet, for delivering video on demand. The idea is to save bandwidth, by storing movies' copies in various nodes in the network. The project is in a rather advanced stage, and a prototype has already been demonstrated. Some of the results reported here were used in its design.

In Net-HD, each CDN is supposed to be controlled by an Internet Service Provider. Thus, the network structure is known to the CDN operator. Still, to avoid load on servers, the technology is of Peer to Peer. This means that a request for a movie may arrive at any node, v. Then, the CDN selects any other node u (not necessarily a central server) which currently has a copy of that movie, and instructs u send a copy to u. Serving from a near by node saves the cost of delivering from a remote server. On the other hand, this incurs a storage cost at u. To save on storage cost, the system may decide to delete copies sometimes. On the other hand, when the demand grows in some localities, the system may decide to add copies. The problem is to optimize the trade-off between the above two costs for each sequence of requests.

[⋆] Supported in part by the Net-HD MAGNET Consortium.
[⋆⋆] Supported in part by the ISF and by the Technion Gordon Center.

The decisions in Net-HD, and in many other CDNs, are made by a central location that knows the topology of the network as explained above. It also knows the locations of the copies. This is feasible since the control messages ordering u to deliver a copy to v consumes negligible bandwidth, compared to the amount consumed by the copy of the movie. Again, this approach is rather common. In Akamai, for example, the HTML of a page can be served from a remote location since it consumes a little bandwidth. However, hyper links to video in that HTML are changed such that the videos are served from locations that are close to the requesting client.

CDNs, especially for video, have been growing rapidly, becoming a major application of the Internet. It is estimated that, even back in 2007, YouTube alone consumed as much bandwidth as the entire Internet did in 2000 [11]. YouTube's consumption was also estimated to be 10% of the total bandwidth of the Internet at that time [2]. Bandwidth consumption has grown since 2007 considerably. In May 2011, Youtube passed the barrier of three billion requests per day, three times as much as it had a year earlier. Netflix, that rents movies over the internet, is claimed to account for 20% of peek U.S. bandwidth use [5]. Some other big players (or emerging players) in renting movies online are Amazon, Hulu, Apple, Blockbuster, even Facebook- Warner Bros announced that it will rent movies online via Facebook [19]. We have not mentioned yet Peer to Peer virtual networks such as Bit Torrent and eMule, or download sites such as Rapidshare. Video traffic, not including P2P traffic, was estimated to be 25% of the Internet traffic in October 2010 [3]. Internet video was one-third of all consumer Internet traffic, and was estimated to approach 40 percent of consumer Internet traffic by the end of 2010, not including the amount of video exchanged through P2P file sharing. The sum of all forms of video was estimated to exceed 91% of the global consumer traffic by 2014 [4]. Reducing video traffic, even over the commercial (and legal) CDNs would make a very significant contribution to providing services over the Internet. (In fact, it has been claimed that some Internet Service Providers suffer from the video load so much that they limit the amount of bandwidth they give to YouTube (perform throttling [18]); this is in spite of the fact that YouTube (Google) transports most of this traffic over its own network, and passes it to ISPs relatively close to the users).

We strive to reduce the traffic, but we assume that storage space in network nodes is associated with a cost too. Indeed, storage is rather cheap, so normally, there is no need to evict a copy of one movie from a node when bringing a copy of another movie. However, when the ISP (or the caching provider) sees a fast growth in the demand at some node, it installs additional storage, incurring costs. Alternatively, storage can be rented from various cloud computing providers and from other CDNs even today. There will probably be more options of renting according to location, when demand grows, similar to the case currently with fiber and links. Another kind of cost materializes in the form of slower servers (since an overloaded server must use a slower secondary storage, or, alternatively, may need to postpone some space consuming tasks). Finally, we note that algorithms that minimize storage cost may help to approximate the more traditional caching, where a charge for space is not assumed, but rigid space bounds are. (e.g., paging algorithms).

Our results. We present a constant approximation algorithm for grid networks. We comment that it is rather easy to show that this problem is NP-hard in general graphs (using a reduction from the STEINER TREE problem [16]) and even on grids (using [15]). (However, as shown in [12], it is not possible to translate upper bounds to online undirected spanning tree algorithms to solve the problem addressed here). We present an $O(\log \delta)$-competitive algorithm, where δ is the normalized diameter of the network, for general networks with general metrics. We show a matching lower bound using a reduction from the Steiner tree problem. Our algorithms use a rather intuitive approach that has an elegant representation in geometric terms.

Related work. A somewhat more general problem was presented by Papadimitriu, Ramanathan, and Rangan [20, 22, 21]. The main difference is that they allow different storage costs for different nodes. (Otherwise, the problems can be translated to each other). They proved that problems to be NP-hard for general graphs. (They have shown an even more general problem, where the weights change in time, to be NP- hard even for star networks). They have also connected the problems to the problem of Steiner Trees in semi directed graphs "space-time" graphs; those are the Cartesian products of the network graphs with the set of requests' times; the directed edges are along the time axis. the current paper (as well as [12, 9] mentioned below) use these "space- time" graphs. They gave a 2 (greedy) approximation on trees for the problem where the costs did not change with time. (For the more general problem, with costs that vary with time, they gave a polynomial algorithm for the special case of a line network plus a star at one of its ends, where all the request must at nodes of the star).

A related problem of dynamic servers with costs associated both with delivery and with storage (or "rental") was studied in [12]. There, the main results are for the case that the deletion of a server is associated with a cost. Hence, it is hard to compare the competitiveness results given there and given here. (Their constraint on deletions may force the adversary off line algorithm to work harder, thus allowing for a better competitive factor). Indeed, we present a (rather straightforward) lower bound of $\Omega(\log \delta)$ for the competitive ratio of our problem on a general graph, while their lower bound is $\Omega(\frac{\log\log\delta}{\log\log\log\delta})$. They also show that it is not obvious how to modify any existing Steiner tree algorithm (or, even closer, a Rectilinear Steiner Arborescence algorithm (RSA)) to solve the problem studied here. Specifically, they show that there exists instances where the cost of an RSA solution is higher in the order of magnitude than the cost of a solution to the no- deletion cost problem; the converse is also true.

Our problem, but on a line network, is also closely related to the symmetric version of the Rectilinear Steiner Arborescence (StRSA) problem, for which a PTAS is known to exist [24]. It may or may not be possible to extend this to the grid networks we approximated. However, the problem solved in [24] was for the continuous case, while our motivation dictates addressing the discrete case.

A static problem with a trade-off between bandwidth and storage on tree networks was introduced in [23]. A generalization and a polynomial algorithm to find an optimal solution appeared in [13] (still for the static case).

Online migration and replication were tasks suggested in [10], but without dealing with the storage cost. A known online problem that resembles our problem somewhat is that of file allocation, see [8, 7]. (In databases and systems contexts, see, e.g. [14] for a survey of early work). Like that problem, the one studied here can be viewed as a combination of the two tasks suggested by [10]. The similarity is in the possibility to replicate a file to a near by location to save in the cost of reading it. However, the pressure to reduce the number of replicas is different, yielding different algorithms. There, when writing a file, one needs to write *all* the replicas. Hence, the algorithms there are motivated to reduce the number of replicas that are "far away" from each other. On the other hand, our algorithms are more likely to eliminate replicas *especially* when they are *not* "far away" (since then bringing a copy later, when needed, will be cheap).

2 Preliminaries

Problem definition. Given a graph $G = (V, E)$, construct another graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, intuitively, by "layering" multiple copies of G, one per time unit. Connect each node in each copy to the same node in the next copy. (See Fig. 1.) More formally, the node set $\mathcal{V}$ contains a *node replica* (sometimes called just a *replica*) $v(t)$ of every $v \in V$, for every time step $t \in \mathbb{N}$. Denote $V(t) = \{v(t) : v \in V\}$ and $E(t) = \{(v(t), u(t)) : (u, v) \in E\}$, for every $t \in \mathbb{N}$. It follows that $\mathcal{V} = \bigcup_t V(t)$. On the other hand, the set of edges contains not only $\mathcal{H} \triangleq \bigcup_t E(t)$, but also a set of directed edges $\mathcal{A} \triangleq \{(v(t), v(t+1)) : v \in V, t \in \mathbb{N}\}$. Pictorially, we term the edges in $\mathcal{H}$ (intuitively, edges of a copy of G) *horizontal edges*, and directed edges in $\mathcal{A}$ (connecting different copies of G) *vertical edges*. (For easy distinction, term the latter *arcs*). Following Fig. 1, we say that $V(t)$ (and the nodes in $V(t)$) are *higher* than $V(t+1)$ (and its nodes).

In the Movie Allocation problem, we are given a graph $G = (V, E)$, an *origin* node $v_0 \in V$ (who is said to hold a movie copy initially), and a set of *requests* $\mathcal{R} \subseteq \mathcal{V}$. (See Fig. 1.) We assume that the storage cost is normalized to 1 and we are given a cost function $c : E \to \mathbb{R}_+$. A feasible solution is a subset of edges $\mathcal{F} \subseteq \mathcal{E}$ such that for every $v(t) \in \mathcal{R}$, there exists a path in $\mathcal{F}$ from $v_0(0)$ to $v(t)$. A horizontal edge $(v(t), v'(t)) \in \mathcal{F} \cap \mathcal{H}$ stands for sending a copy of the movie from v to v', or from v' to v, while a vertical (directed) edge $(v(t), v(t+1)) \in \mathcal{F} \cap \mathcal{A}$ stands for keeping the movie in v's cache at time step t. (See Fig. 2.) The cost of a vertical edge is 1, while the cost of an A horizontal edge $(v(t), v'(t)) \in \mathcal{H}$ is $c(v, v')$. In the Movie Allocation problem our goal is to find a minimum cost feasible solution.

Online model. In the online version of Movie Allocation we need to make two types of decisions at each time t, before seeing future requests: (1) At which nodes to cache a copy of the movie for future use (that is, add vertical edges); (2)

From which current (time t) cache to serve the requests (by adding horizontal edges) that arrive at time t. We assume that the online algorithm may replicate the movie for efficient delivery, but at least one copy of the movie must remain in the network at all times. Alternatively, the system (but not the algorithm) can have the option to delete the movie altogether—this decision is then made known to the algorithm. We argue that this is a natural assumption. Normally, content is not destroyed, but is kept somewhere at least for archival purposes, or because somebody will eventually request. (The heavy tail nature of requests for Internet content, means that it is rather typical that "not so popular" content is sometimes requested nevertheless). If this content resides somewhere in the world and can be brought into the system for some payment, then our algorithms can be adapted to handle this case nevertheless.

Anyhow, without this assumption, no competitive algorithm exists. (It is easy to see that an online algorithm cannot eliminate the last copy in any case, so it spends unboundedly more than an off line algorithm who may delete it.) A similar assumption was made in the case of the file allocation problem [8, 7].

Definitions and notation. Denote $n = |V|$. Let $\mathcal{R} = \{v_1(t_1), \ldots, v_N(t_N)\}$, where $t_j \leq t_{j+1}$, for every j. Namely, we assume that the requests are sorted in a non-decreasing order of time step. Define $\mathcal{R}_t = \mathcal{R} \cap V(t)$. When G is the line network, we assume that $V = \{0, \ldots, n-1\}$, where 0 is the origin. We write for two requests, $i_j(t_j), i_k(t_k)$ that $i_j(t_j) \prec i_k(t_k)$ if either $t_j < t_k$ or $t_j = t_k$ and $i_j < i_k$. This total order implies that if $\mathcal{R} \neq \emptyset$ then it has a minimal request.

Given two nodes $v(t)$ and $v'(t')$, let $\mathcal{P}[v(t), v'(t')]$ be the set of edges that comprises some arbitrary shortest path from $v(t)$ to $v'(t')$, and let $d(v(t), v'(t')) \triangleq \sum_{e \in \mathcal{P}[v(t), v'(t')]} c(e)$. This path is unique in some important cases, e.g. if $v' = v$, or if G is a line and $t' = t$.

In our analysis, OPT is the set of edges in some optimal solution whose cost is $c(\text{OPT})$. In the unit costs case, where $c(e) = 1$, for every e, the cost of the optimal solution is denoted by $|\text{OPT}|$.

Half balls. Our algorithms and their analysis are based on the following notion. Intuitively, a *half-ball*, or an *h-ball*, of *radius* $\rho \in \mathbb{N}$ centered at a node $v(t)$ contains every node from which there exists a path of length ρ to $v(t)$. See Fig. 3. We term them *half-balls* to emphasis that there is no path from a node $u(t+1)$ to a node $v(t)$.

Recall that we assume that the cost $c(e)$ of an edge $e \in E$ can be of any (positive) value. For convenience of description, we define h-balls for the unit cost case. This eases the definition since in this case each edge is either complete inside the h-ball, or completely outside of it.

Definition 1 (H-ball). *Given a node* $v(t) \in \mathcal{V}$ *and a* radius $\rho \in \mathbb{N}$, *the* h-ball *of radius* ρ *centered at* $v(t)$ *is defined as* $\mathcal{V}[v(t), \rho] = \{v'(t') : d(v'(t'), v(t)) \leq \rho\}$. *Also,* $\mathcal{E}[v(t), \rho]$ *denotes the set of* edges of the h-ball. *Such is every edge* $(v_1(t_1), v_2(t_2)) \in \mathcal{E}$ *if both its endpoints* $v_1(t_1), v_2(t_2) \in \mathcal{V}[v(t), \rho]$.

A simple property of the h-balls implies a bound on the optimum:

Algorithm 1 : **Store** $(v_j(t_j))$

```
1: Let $u$ be the closest node with a copy of the movie at time $t_j$, and let $\rho_j = d(u, v_j)/4$.
2: Satisfy request $v_j(t_j)$ via $u(t_j)$.
3: Hold movie until $t_j + 2\rho_j$.                              $\triangleright$ $v_j$ keeps its copy
4: Hold movie as long as this is the only copy.
```

Lemma 1. *Let $\mathcal{R} = \{v_1(t_1), \ldots, v_k(t_k)\}$ be a set of requests and let $\rho_1, \ldots, \rho_k$ be a series of radii. If every edge from $\mathcal{E}$ is contained in at most α h-balls from $\mathcal{E}(v_1(t_1), \rho_1), \ldots, \mathcal{E}(v_k(t_k), \rho_k)$, then $|\text{OPT}| \geq \frac{1}{\alpha}\sum_j \rho_j$.*

Proof. Let $v(t) \in \mathcal{R}$ be a request and let k be the length of the shortest path from $v_0(0)$ to $v(t)$. If $\mathcal{F}$ is a feasible solution and $\rho \leq k$, then $|\mathcal{F} \cap \mathcal{E}(v(t), \rho)| \geq \rho$. (To see that, note that $\mathcal{F}$ must contain a path $\mathcal{P}$ from $v_0(0)$ to $v(t)$; now, $|\mathcal{P} \cap \mathcal{E}(v(t), \rho)| \geq \rho$). Let $\mathcal{F}$ be a feasible solution. The above implies that $|\mathcal{F} \cap \mathcal{E}(v_j(t_j), \rho_j)| \geq \rho_j$, for every j. Since each edge is contained in at most α h-balls, we have $|\mathcal{F}| \geq \frac{1}{\alpha}\sum_j |\mathcal{F} \cap \mathcal{E}(v_j(t_j), \rho_j)| \geq \frac{1}{\alpha}\sum_j \rho_j$. □

A similar result can be shown for general costs. Observe that in this case $\mathcal{E}(v_1(t_j), \rho_j)$ may contain edge parts.

3 Online Algorithm for General Networks

Let us now present an $O(\log \delta)$-competitive online algorithm for general networks, and analyze it using h-balls. For ease of exposition we analyze the algorithm for unit costs. A matching lower bound is given in Appendix B.

Algorithm **Store** is rather simple: upon the arrival of a request at a node v at time t, the request is satisfied by obtaining a movie from the closest node u that has a copy of the movie. The movie is saved in v's cache for a period of time that is proportional to the distance between u and and v. If several requests arrive at the same time, we process them one by one in an arbitrary order.

To upper bound the competitive ratio, consider the h-balls $\mathcal{V}[v_1(t_1), \rho_1]$, ..., $\mathcal{V}[v_n(t_n), \rho_n]$ induced by Algorithm **Store**. We bound the number of h-balls that contain the same edge, and then use Lemma 1.

Lemma 2. *If $\mathcal{V}[v_j(t_j), \rho_j] \cap \mathcal{V}[v_k(t_k), \rho_k] \neq \emptyset$, then either $\rho_k \geq 2\rho_j$ or $\rho_k \leq \frac{1}{2}\rho_j$.*

Proof. W.l.o.g., assume that $t_j \leq t_k$. If $t_k > t_j + 2\rho_j$, then $\mathcal{V}[v_j(t_j), \rho_j] \cap \mathcal{V}[v_k(t_k), \rho_k] \neq \emptyset$ only if $\rho_k \geq 2\rho_j$ (see Fig. 4a). If $t_k \leq t_j + 2\rho_j$ and $d(v_j, v_k) \leq 2\rho_j$, we have that $\rho_k \leq d(v_j, v_k)/4 \leq \frac{1}{2}\rho_j$, since the request at $v_k(t_k)$ can be satisfied by receiving a copy from v_j who has a copy of the movie at time t_k (see Fig. 4b). It remains to consider the case where $t_k \leq t_j + 2\rho_j$ and $d(v_j, v_k) > 2\rho_j$. If $\mathcal{V}[v_j(t_j), \rho_j]$ and $\mathcal{V}[v_k(t_k), \rho_k]$ intersect, they must do so at time t_j. It follows that there is a node v such that $d(v_j, v) \leq \rho_j$ and $d(v_k, v) \leq \rho_k$, hence, $d(v_j, v_k) \leq \rho_j + \rho_k$. On the other hand, $4\rho_k \leq d(v_j, v_k)$, since v_j has a copy of the movie at time t_k. This leads to a contradiction, since

$\rho_j + \rho_k \geq \frac{2d(v_j,v_k)}{2} > \frac{2\rho_j+4\rho_k}{2} > \rho_j + \rho_k$. Hence, $\mathcal{V}[v_j(t_j), \rho_j]$ and $\mathcal{V}[v_k(t_k), \rho_k]$ do not intersect in this case. (See Fig. 4c.) $\square$

Theorem 1. *Algorithm **Store** is an $O(\log\delta)$-competitive online algorithm for* Movie Allocation.

Proof. Break the execution of the algorithm into phases. Two consecutive phases are separated by an interval in which some node v_ℓ holds the movie after $t_\ell + 2\rho_\ell$ (Line 4 is invoked). Let t'_ℓ denote the time at which the movie was removed from v_ℓ's cache. Clearly, **Store** is optimal within time interval $[t_\ell, t'_\ell]$, since every solution must hold at least one copy of the movie during $[t_\ell, t'_\ell]$.

Within the phases, Algorithm **Store** pays at most $6\rho_j$ for servicing the jth request and for storing the movie at v_j. On the other hand, Lemma 2 implies that each edge $(v(t), v'(t'))$ is contained in at most $\log\frac{\max_i \rho_i}{\min_i \rho_i}$ h-balls from $\mathcal{E}(v_1(t_1), \rho_1), \ldots, \mathcal{E}(v_n(t_n), \rho_n)$. Thus, by Lemma 1 we know that $\sum_j \rho_j \leq \log\frac{\max_i \rho_i}{\min_i \rho_i} \cdot \text{OPT} \leq \log\delta \cdot \text{OPT}$. Hence, $|\textbf{Store}(\mathcal{R})| \leq \sum_\ell (t'_\ell - t_\ell) + \sum_j 6\rho_j = O(\log\delta)\cdot\text{OPT}$. $\square$

4 Approximation Algorithm for the Grid Network

In this section, we present an $O(1)$-approximation algorithm for Movie Allocation in the grid network. To simplify the exposition, we first describe the algorithm for the special case of unit costs. The general case is discussed later.

Let us now expand the method of h-balls, to obtain a constant factor approximation also in grid networks. When using the ℓ_∞ norm (and the d_∞ distance), an h-ball centered at $v(t)$ with radius ρ is a half-cube or a *box*. See Fig. 5a. We use $\mathcal{V}_\infty[v(t), \rho]$ and $\mathcal{E}_\infty[v(t), \rho]$ to denote nodes and edges of a box of radius ρ centered at $v(t)$. If $d(v'(t'), v(t)) = \rho$, we say that $v'(t')$ is on the *border* of the h-ball. An edge $(v_1(t_1), v_2(t_2)) \in \mathcal{E}_\infty[v(t), \rho]$ is said to be on the border of the h-ball if both $v_1(t_1)$ and $v_2(t_2)$ are on the border of the h-ball. $\mathcal{E}_\infty(v(t), \rho)$ denotes the set of edges in $\mathcal{E}[v(t), \rho]$ that are not border edges. Note that Lemma 1 still holds for this kind of h-balls, since $\mathcal{E}[v(t), \rho] \subseteq \mathcal{E}_\infty[v(t), \rho]$.

Given a set of boxes $\mathcal{V}_\infty[v_1(t_1), \rho_1], \ldots, \mathcal{V}_\infty[v_{j-1}(t_{j-1}), \rho_{j-1}]$ in $\mathcal{G}$, we *grow* a box centered at $v_j(t_j)$, where $t_j \geq \max_{k=1}^{j-1} t_k$, by increasing its radius ρ_j until it collides with another box. We consider two possible collision types depending on the intersection. (Examples are given in Fig. 6).

Border Collision: There exists a collision node on the border of $v_k(t_k)$'s box.
There exists a node $u \in V$ such that $d_\infty(v_j, u) = \rho_j$ and $d_\infty(v_k, u) = \rho_k$.

Non-border Collision: The top of $v_j(t_j)$'s box is strictly contained in the bottom of $v_k(t_k)$'s box.
For every $u \in V$ such that $d_\infty(v_j, u) \leq \rho_j$ we have that $d_\infty(v_k, u) < \rho_k$.

Algorithm **Skeleton** constructs a solution by assembling parts from each box. Note that the process of growing boxes induces a tree structure on the requests, namely, a request j is said to be the child of request k, if ρ_j was determined by

Algorithm 2 : **Skeleton** $(\mathcal{G}, v_0, \mathcal{R})$

```
1: $\mathcal{F} \leftarrow \emptyset$
2: for $j = 1 \rightarrow N$ do
3:     Let $\rho_j$ be the collision radius of $v_j(t_j)$, and let $k$ be $j$'s parent
4:     if $\exists u$ such that $d_\infty(u, v_j) = \rho_j$ and $d_\infty(u, v_j) = \rho_j$ then let $v''_j = u$
        else let $v''_j$ be some node such that $d_\infty(v''_j, v_j) = \rho_j$
5:     $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{S}[v_j(t_j), \rho_j]$ using $v''_j$ as the connector
6: end for
7: return $\mathcal{F}$
```

a collision with the box of request k. (If there is more than one option, pick one arbitrarily.) The following definition is useful in describing parts of a box that are used for connecting the box to its parent box and children boxes. Specifically, the *skeleton* of a box $\mathcal{V}_\infty(v(t), \rho)$ is defined as follows. (See example in Fig. 5b).

$$\begin{aligned}
&\mathcal{S}[v(t), \rho] = \mathcal{P}[v(t), v'(t)] \ \cup \\
\text{bottom:}\quad & \{(u(t), u'(t)) : d_\infty(v, u) = d_\infty(v, u') = \rho\} \ \cup && (1)\\
\text{connector:}\quad & \mathcal{P}[v''(t), v''(t - \rho)] \ \cup && (2)\\
\text{top:}\quad & \{(w(t - \rho), w'(t - \rho)) : d_\infty(v, w) = d_\infty(v, w') = \rho\} && (3)
\end{aligned}$$

where v' is some node such that $d(v, v') = \rho$, and v'' is some node such that $d_\infty(v, v'') = \rho$. Observe that the only vertical part of the skeleton is part (2). We refer to (1) and (3) as the bottom $\underline{\mathcal{S}}[v(t), \rho]$ and the top $\overline{\mathcal{S}}[v(t), \rho]$ of the skeleton, respectively. (Note that the top [respectively, the bottom] of the skeleton is a subset of the top [resp., the bottom] of the box. Node v'' is called the *connector* of the skeleton, since it is used to connect the bottom and the top of the skeleton. Node v' is called the *border gateway*, since it is used to connect the center of the box (node $v(t)$) to the border of the bottom of the skeleton.

Algorithm **Skeleton** is the first phase of the algorithm. It grows a box for each request and then adds the skeleton of the box to an edge multiset $\mathcal{F}$. We say a "multiset" to stress that all skeleton edges of a request are put in $\mathcal{F}$; we do not take advantage of cases where one of these edges already belongs to $\mathcal{F}$ (e.g. when they belong to the skeleton of a higher box). **Skeleton** copes only with border collisions. Later, we show how to transform $\mathcal{F}$ into a solution that copes with non-border collisions as well.

Each request adds $O(1)$ simple straight paths to the solution $\mathcal{F}$. Hence, it is easy to show that the running time of the algorithm is polynomial.

The set of edges returned by Algorithm **Skeleton** may be an infeasible solution. However, it is feasible if no non-border collision occurs during execution. This is proved in the following lemma. The lemma after that, analyzes the competitiveness (in that case).

Lemma 3. *Let $v_j(t_j)$ be the jth request, and let k be j's parent. If ρ_j was determined by a border collision, then there is a path from $v_k(t_k)$ to $v_j(t_j)$ in $\mathcal{F}$.*

Proof. First, observe that $d_\infty(v''_j, v_k) = \rho_k$ if a border collision occurs. Hence, $v_j(t_j)$ can be reached from $v_k(t_k)$ using the following path. First, go from $v_k(t_k)$ to $v'_k(t_k)$ (i.e. go on the skeleton of the higher box to the border of its bottom). Then, go from $v'_k(t_k)$ to $v''_j(t_k)$ (proceed on the bottom part of the higher skeleton to the connector node). From $v''_j(t_k)$, go to $v''_j(t_j)$ (stay on the connector node until the later time of the bottom of the lower box). Proceed (on the bottom part of the skeleton $\mathcal{S}[v_j(t_j), \rho_j)]$) to $v'_j(t_j)$ (the border gateway of the bottom of the lower box). Finally, go from $v'_j(t_j)$ to $v_j(t_j)$. □

Lemma 4. $|\mathcal{F}| \leq 18 \cdot \text{OPT}$.

Proof. $|\mathcal{S}[v_j(t_j), \rho]| \leq 18\rho_j$, since parts (1) and (3) of $v_j(t_j)$'s skeleton contain up to $8\rho_j$ edges each, while the rest contain a total of $2\rho_j$ edges. The lemma follows from Lemma 1 □

Coping with non-border collisions. Let us now show how to modify the above multiset $\mathcal{F}$, such that $\mathcal{F}$ contains a path from every request to its parent, even in the case of non-border collisions. The modification consists of two actions: (1) replacing certain edges of $\mathcal{F}$ by earlier (higher) copies of these edges, and (2) adding $O(|\text{OPT}|)$ new edges.

We describe the action of edge replacement using the notion of *edge ascension.* An edge $(v(t), u(t)) \in \mathcal{H} \bigcap \mathcal{F}$ is said to be ascended if it is replaced by the edge $(v(t'), u(t'))$, where $t' \leq t$ is the maximal time such that $(v(t'), u(t'))$ is located on the bottom of some box $\mathcal{V}_\infty[v_j(t_j), \rho_j]$. Pictorially, the edge is elevated until it reaches the bottom of some box. It may be that an edge is already located on the bottom of some box, in this case, $t' = t$. If no such time t' exists, then we say that the ascension failed and the edge is deleted.

Consider the jth request $v_j(t_j)$. Since $\mathcal{F}$ is a multiset, every edge in the top of $\overline{\mathcal{S}}[v_j(t_j), \rho_j]$ has a copy (in $\mathcal{F}$) that was not used to connect a request to its parent (in Lemma 3). We try to ascend every edge of $\overline{\mathcal{S}}[v_j(t_j), \rho_j]$, for every j, resulting in a revised (but not yet feasible) solution $\mathcal{F}'$.

Observation 1. $|\mathcal{F}'| \leq |\mathcal{F}|$.

Observation 2. *Let $v_j(t_j)$ be a child request of $v_k(t_k)$, whose collision radius was determined by a non-border collision. Then, $\overline{\mathcal{S}}[v_j(t_j), \rho_j] \subseteq \mathcal{F}'$.*

Proof. Since we are dealing with a non-border collision, the edges in $\overline{\mathcal{S}}[v_j(t_j), \rho_j]$ are located on the bottom of $\mathcal{V}_\infty[v_k(t_k), \rho_k]$ and therefore are not moved during the ascension process. □

Now consider a request $v_k(t_k)$ and let $u_1(s_1), \ldots, u_q(s_q)$ be those among its children, whose collision radii were determined by non-border collisions. We describe an edge set $\mathcal{F}_k$ such that $\mathcal{F}' \cup \mathcal{F}_k$ contains a path from $v_k(t_k)$ to $u_j(s_j)$ for every j. Let G_k be the subgraph of G that is induced by node set $V_k = \{v : d_\infty(v_k, v) \leq \rho_k\}$. Define the following weight function on $E_k = E \cap (V_k \times V_k)$:

$$w(x, y) = \begin{cases} 0 & (x(t_k), y(t_k)) \in \mathcal{F}', \\ 1 & \text{otherwise.} \end{cases}$$

Now use Arora's PTAS for the STEINER TREE problem [6] to compute F_k, which is a $(1+\varepsilon)$-approximate Steiner tree that connects $u''_1, \dots, u''_q$ to v_k in G_k, where u''_j is the connector of the skeleton of $u_j(s_j)$. Translating this set of edges of G to a set of edges in $\mathcal{G}$, we define, $\mathcal{F}_k = \{(x(t_k), y(t_k)) : (x, y) \in F_k\}$.

A subset $\mathcal{T} \subseteq \mathcal{E} \setminus \mathcal{F}'$ is called a *completion* of $\mathcal{F}'$ if $\mathcal{F}' \cup \mathcal{T}$ is a feasible solution.

Lemma 5. *$\bigcup_k \mathcal{F}_k$ is a completion of $\mathcal{F}'$.*

Proof. First, border collision children are connected to their parents by $\mathcal{F}'$. Let $u_j(s_j)$ be a non-border collision child of $v_k(t_k)$. $\mathcal{F}_k$ contains a path from $v_k(t_k)$ to $u''_j(t_k)$ and $\mathcal{F}'$ contains a path from $u''_j(t_k)$ to $u_j(s_j)$ that is located on the skeleton of $u_j(s_j)$. The lemma follows since the edges in $\overline{\mathcal{S}}[u_j(s_j), \sigma_j]$ are not ascended (Observation 2). □

It remains to bound the completion's. For that, we show that any completion $\mathcal{T}$ of $\mathcal{F}'$ can be transformed into some $\mathcal{T}'$ that consists of Steiner trees, one in each $E_k(t_k)$, for every k, without increasing its cost. (Unfortunately, it was too difficult to show a similar claim for $\mathcal{F}$, which is the reason we defined $\mathcal{F}'$).

Lemma 6. $|\bigcup_k \mathcal{F}_k| \le (1+\varepsilon)\text{OPT}$.

Proof. We show that there exists an optimal completion of $\mathcal{F}'$, denoted by $\mathcal{T}'$, that consists of Steiner trees, one in $E_k(t_k)$, for every k. Since any feasible solution is also a completion of $\mathcal{F}'$ it follows that

$$|\textstyle\bigcup_k \mathcal{F}_k| \;=\; \sum_k |F_k| \;\le\; \sum_k (1+\varepsilon)\,|T_k| \;\le\; (1+\varepsilon)\,|\mathcal{T}'| \;\le\; (1+\varepsilon)\text{OPT} \;,$$

where T_k is an optimal Steiner tree in G_k.

It remains to show that there exists an optimal completion of $\mathcal{F}'$ that consists of Steiner trees, one in $E_k(t_k)$, for every k. Let $\mathcal{T}$ be an optimal completion of $\mathcal{F}'$. We modify $\mathcal{T}$ and obtain another optimal completion $\mathcal{T}'$ of $\mathcal{F}'$ that consists of Steiner trees in G_k, for all k. We perform the following operations on $\mathcal{T}$:

1. We perform edge ascension on edges in $(\mathcal{T} \cap \mathcal{H}) \setminus \bigcup_j \mathcal{E}_\infty[v_j(t_j), \rho_j]$, namely on all horizontal edges in $\mathcal{T}$ that are not located within some box.
2. For every connected component $\mathcal{C}$ of $\mathcal{T} \cap \mathcal{E}_\infty[v_j(t_j), \rho_j]$, for some j, we do the following. Let $x(\tau)$ be the root of $\mathcal{C}$. If $\tau = t_j - \rho_j$ and $\mathcal{C}$ contains a node at t_j, then add a shortest path from $v_j(t_j)$ to $x(t_j)$.
3. We perform *edge descension* on edges in $(\mathcal{T} \cap \mathcal{H}) \cap \bigcup_j \mathcal{E}_\infty[v_j(t_j), \rho_j]$, namely we move all horizontal edges in $\mathcal{T}$ that are located within some box $\mathcal{E}_\infty[v_j(t_j), \rho_j]$ to time t_j.
4. We remove the edges in $(\mathcal{T} \cap \mathcal{A}) \cap \bigcup_j \mathcal{E}_\infty[v_j(t_j), \rho_j]$.

The resulting set of edges is denoted by $\mathcal{T}'$.

Observe that in Operations 1 and 3, we move edges, and therefore we do not increase the number of edges. Operation 2 adds at most ρ_j edges for every component $\mathcal{C}$ that satisfies the conditions, while Operation 4 deletes at least ρ_j edges for every such component. It follows that $|\mathcal{T}'| \le |\mathcal{T}|$.

It remains to show that $\mathcal{T}'$ is a feasible completion of $\mathcal{F}'$. Consider a request $v_k(t_k)$ and let $u_1(s_1), \ldots, u_q(s_q)$ be those among its children, whose collision radii $\sigma_1, \ldots, \sigma_q$ were determined by non-border collisions. We distinguish between two types of children of $v_k(t_k)$ according to the last node $x_j(\tau_j)$ on the path from $v_0(0)$ to a child $u_j(s_j)$ in $\mathcal{F}' \cup \mathcal{T}$ that satisfies one of the following two conditions:

1. $d_\infty(x_j, v_k) = \rho_k$ and $\tau_j \in [t_k - \rho_k, s_j]$.
2. $d_\infty(x_j, v_k) < \rho_k$ and $\tau_j = t_k - \rho_k$.

Consider a Type 1 child $u_j(s_j)$. Due to Operations 1 and 3, the path from $x_j(\tau_j)$ to $u_j(s_j)$ is moved to time t_k. However, some edges may be missing due to the following reasons:

1. An edge ascension may be blocked by a box that is located beneath the box of $v_k(t_k)$.
2. An edge may be contained in a box located beneath the box of $v_k(t_k)$, and in this case it descends.

However, the missing parts in the path from $x_j(\tau_j)$ to $u_j(s_j)$ at time t_k are replaced by the ascended skeleton edges that are contained in $\mathcal{F}'$. Since $x_j(t_k)$ is located on the bottom part of the skeleton of $v_k(t_k)$, $\mathcal{T}'$ contains a path that connects $v_k(t_k)$ to the top part of the skeleton of $u_j(s_j)$, and therefore to $u_j(s_j)$. An example is given in Fig. 7.

A Type 2 child is handled similarly with one difference. Node $x_j(t_k)$ is not located on the bottom part of the skeleton of $v_k(t_k)$. However, it is connected to $v_k(t_k)$ due to Operation 2. The lemma follows since $\mathcal{T}'$ induces a Steiner tree in G_k, for all k. $\square$

It is not hard to show that the running time of **Skeleton**, as well as of the computation of the completion, is polynomial. Hence, Lemmas 4 and 6 lead us to the main result of this section:

Theorem 2. *There exists a polynomial-time $(19 + \varepsilon)$-approximation algorithm for* Movie Allocation*, for every $\varepsilon > 0$.*

General instances. Consider the more general case where vertical edges cost 1 and horizontal (grid) edges cost d. We show how to modify our algorithm in order to cope with this case. First, we let Algorithm **Skeleton** grow boxes. The problem is that a box may stop growing due to a collision that occurred between time steps or not on grid points. Observe that if $\mathcal{V}_\infty[v_j(t_j), \rho_j]$ collides with a box $\mathcal{V}_\infty[v_k(t_k), \rho_k]$ from below it is always at t_k. Hence we need to take care only of side collisions. This can be done as follows: for every j, the box $\mathcal{V}_\infty[v_j(t_j), \rho_j]$ is replaced by a box whose base corners are each moved to the closest grid point. The height of the box remains the same. Notice that if $\mathcal{V}_\infty[v_j(t_j), \rho_j]$ collided with a box $\mathcal{V}_\infty[v_k(t_k), \rho_k]$, then the borders of the corresponding boxes still intersect after to this process. We add the skeletons after adjusting the boxes.

Observe that cost of skeletons may increase by a factor of 2. Hence, we still have a constant factor approximation.

References

1. Net-HD Consortium. http://www.nethd.org.il
2. Webpronews. http://www.webpronews.com/youtube-comprises-10-of-all-internet-traffic-2007-06 (June 19, 2007)
3. Cisco visual networking index: Usage study. Cisco Visual Networking Index (October 25, 2010), http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/Cisco_VNI_Usage_WP.html
4. Hyperconnectivity and the approaching zettabyte era. Cisco Visual Networking Index (June 2, 2010), http://www.cisco.com/en/US/solutions/ collateral/ns341/ns525/ns537/ns705/ ns827/VNI_Hyperconnectivity_WP.html
5. Abell, J.C.: Netflix instant account for 20 percent of peek U.S. bandwidth use. Wire magazine (October 21, 2010)
6. Arora, S.: Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. Journal of the ACM 45(5), 753–782 (1998)
7. Awerbuch, B., Bartal, Y., Fiat, A.: Competitive distributed file allocation. In: 25th ACM STOC. pp. 164–173 (1993)
8. Bartal, Y., Fiat, A., Rabani, Y.: Competitive algorithms for distributed data management. In: 24th ACM STOC. pp. 39–50 (1992)
9. Berman, P., Coulston, C.: On-line algorrithms for steiner tree problems. In: 29th Annual ACM Symposium on the Theory of Computing. pp. 344–353 (1997)
10. Black, D., Sleator, D.: Competitive algorithms for replication and migration problems. Tech. Rep. CMU-CS-89-201, CMU (1989)
11. Carter, L.: Web could collaps as video demand soars. Daily Telegraph (4/7/2008)
12. Charlikar, M., Halperin, D., Motwani, R.: The dynamic servers problem. In: 9th Annual Symposium on Discrete Algorithms. pp. 410–419 (1998)
13. Cidon, I., Kutten, S., Soffer, R.: Optimal allocation of electronic content. Computer Networks 40(2), 205–218 (2002)
14. Dowdy, L.W., Foster, D.V.: Comparative models of the file assignment problem. ACM Comp. Surveys 14(2), 287–313 (1982)
15. Garey, M.R., Graham, R.L., Johnson, D.S.: The complexity of computing steiner minimal trees. SIAM J. on Applied Math 32(4), 835–859 (1977)
16. Hwang, F.K., Richards, D.S.: Steiner tree problems. Networks 22(1), 55–89 (1992)
17. Imase, M., Waxman, B.M.: Dynamic steiner tree problem. SIAM J. on Discrete Mathematics 4(3), 369–384 (1991)
18. Jones, D.: Proof that telstra is throtteling youtube bandwidth in australia. www.eevblog.com, www.youtube.com/watch?v=9iDRynyBl0c
19. Kopytoff, V.G.: Shifting online, Netflix faces new competition. New York Times (Business Day, Technology) (September 26, 2010)
20. Papadimitriou, C., Ramanathan, S., Rangan, P.: Information caching for delivery of personalized video programs for home entertainment channels. In: IEEE International Conf. on Multimedia Computing and Systems. pp. 214–223 (1994)
21. Papadimitriou, C., Ramanathan, S., Rangan, P.: Optimal information delivery. In: 6th ISAAC. pp. 181–187 (1995)
22. Papadimitriou, C., Ramanathan, S., Rangan, P., Sampathkumar, S.: Multimedia information caching for personalized video-on demand. Computer Communications 18(3), 204–216 (1995)
23. Schaffa, F., Nussbaumer, J.P.: On bandwidth and storage tradeoffs in multimedia distribution networks. In: IEEE INFOCOM. pp. 1020–1026 (1995)
24. Xiuzhen Cheng, B.D., Lu, B.: Polynomial time approximation scheme for symmetric rectilinear steiner arborescence problem. J. Global Optim. 21, 385–396 (2001)

APPENDIX

A Figures

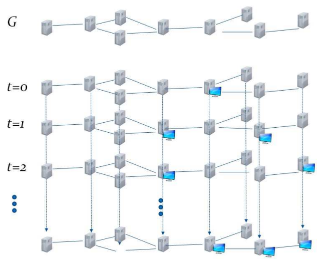


Fig. 1: An example of a network G and its corresponding layered graph $\mathcal{G}$: monitors represent movie requests.

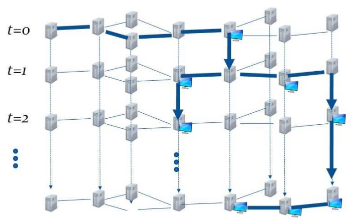


Fig. 2: An example of a feasible solution.

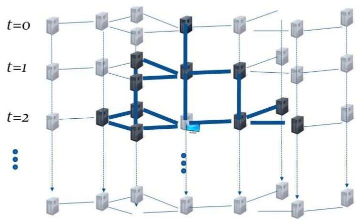
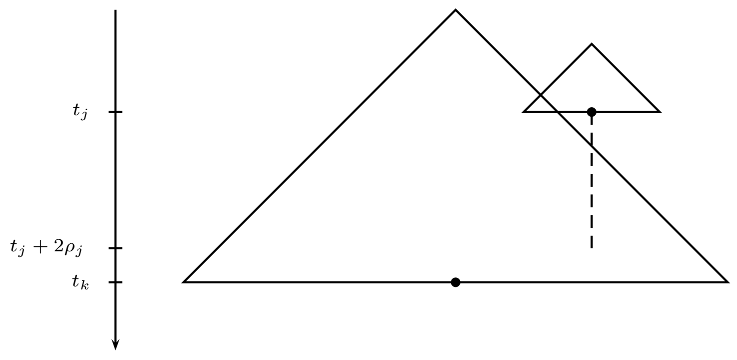
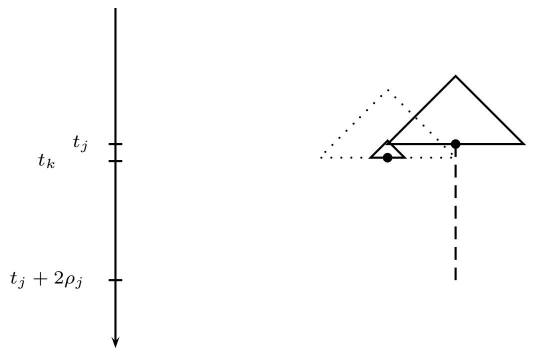


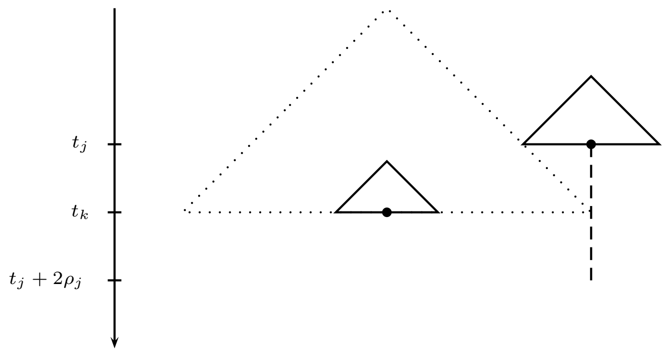
Fig. 3: An example of an h-ball with radius $\rho = 2$.



(a) $t_k \geq t_j + 2\rho_j$.

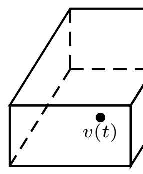


(b) $t_k \leq t_j + 2\rho_j$ and $d(v_j, v_k) \leq 2\rho_j$.

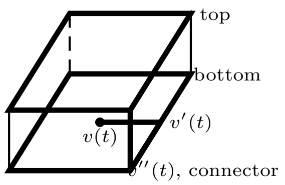


(c) $t_k \leq t_j + 2\rho_j$ and $d(v_j, v_k) > 2\rho_j$.

Fig. 4: Possible intersections of two h-balls $\mathcal{V}[v_j(t_j), \rho_j]$ and $\mathcal{V}[v_k(t_k), \rho_k]$. Solid triangles represent h-balls, dotted triangles represent h-balls with radius $4\rho_j$ or $4\rho_k$, dashed lines represent the time interval in which the movie is saved in v_j's cache.

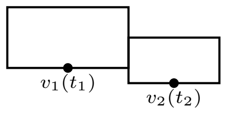


(a) A box centered at $v(t)$.

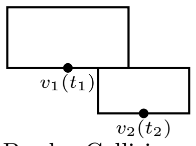


(b) The skeleton of a box centered at $v(t)$.

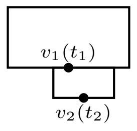
Fig. 5: Example of a box and its skeleton. The box is given on the left, and the box with its skeleton is given on the right. The thick line represent the skeleton.



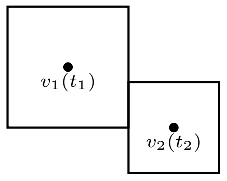
(a) Border Collision: side view.



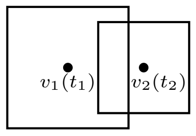
(b) Border Collision: side view.



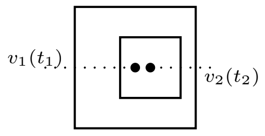
(c) Non-border collision: side view.



(d) Border Collision: top view.



(e) Border Collision: top view.



(f) Non-border Collision: top view.

Fig. 6: Collisions of Boxes: side and top views. Two border collisions are depicted on the left, while a non-border collision is shown on the right.

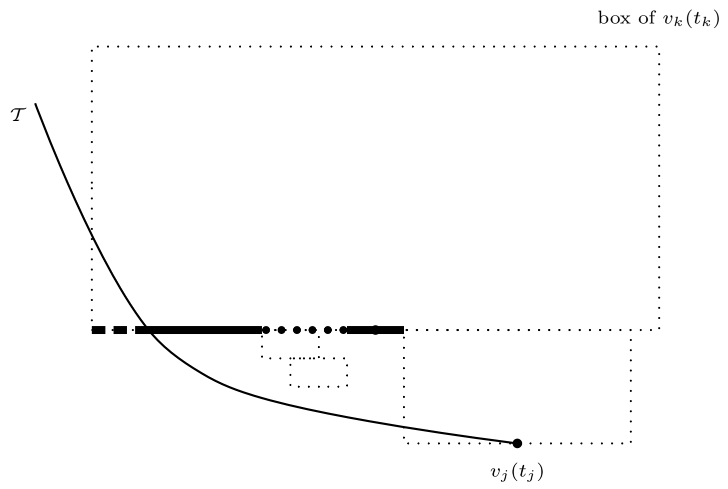


Fig. 7: Four boxes are shown, depicted by thin dotted lines. The given completion $\mathcal{T}$ is represented by a thin solid line. $\mathcal{T}$ will "pay" for the thick horizontal path connecting the skeleton of the box of $v_k(t_k)$ (on the left) to the skeleton of the box of $v_j(t_j)$. The thick horizontal path contains three parts: (1) a thick dotted line representing ascended edges in $\mathcal{F}'$ (the modified solution of algorithm **Skeleton**); (2) a thick dashed line representing descended edges of $\mathcal{T}$ ("paid" for by $\mathcal{T}$); and (3) thick solid lines representing ascended edges of $\mathcal{T}$. Note, that the two smaller boxes prevent $\mathcal{T}$'s edges from being ascended, and thus, prevent $\mathcal{T}$ from paying for the thick dotted part. This thick dotted part is then "paid" by the ascended edges of those same small boxes (ascending the top of their skeletons). Finally, note that the thick path passes on a part of the bottom of the box of $v_k(t_k)$, but not necessarily on the bottom of the skeleton of that box, nor necessarily in $v_k(t_k)$ itself.

B Lower Bound for General Networks

This appendix contains an $\Omega(\log \delta)$ lower bound for online Movie Allocation in general networks. Our construction is based on a reduction from online Steiner Tree and on the following lower bound.

Theorem 3 ([17]). *For every algorithm for online* Steiner Tree *there is an infinite family of unit cost instances for which the competitive ratio is $\Omega(\log N)$. Furthermore, $\log \delta = \Theta(\log N)$ for every instance in this family.*

In the Steiner Tree problem, we are given a graph G and a set $\{\tau_0, \ldots, \tau_{N-1}\}$ of N *terminals*, and the goal is to find a minimum cardinality tree in G that contains the terminals. In the online version of Steiner Tree, terminals arrive one by one. Upon arrival, each terminal τ_j must be connected to the tree containing $\tau_0, \ldots, \tau_{j-1}$ (the terminals that arrived earlier).

Given a graph G and a terminal sequence $\tau_0, \ldots, \tau_{N-1}$ we construct the following instance of Movie Allocation. We construct a graph G' by replacing every edge $e = (u, v)$ in G with a path containing $n^2 - 1$ new nodes and n^2 new edges. Let $u = u'_0, u'_1, \ldots, u'_{n^2} = v$ be the nodes in this path from u to v in G'. The new nodes $u'_1, \ldots, u'_{n^2-1}$ are called the *virtual nodes* between u and v. The new path is termed the path *replacing* edge (u, v). In the following, we use G, OPT, F, etc. when referring to the the Steiner tree problem, and $G', \text{OPT}', \mathcal{F}'$ when referring to the Movie Allocation problem. To complete the definition of the Movie Allocation instance to which we reduce, let the origin be $v_0 = \tau_0$, and the jth request of Movie Allocation is $\tau_j(j)$. Note that the set of requests does not contain replicas of virtual nodes.

Lemma 7. *Consider any set of requests $\mathcal{R} \subseteq \mathcal{V}'$ that does not contain replicas of virtual nodes. Let $\mathcal{F}'$ be a solution in $\mathcal{G}'$ for $\mathcal{R}$. Then the following holds:*

- *There exists a solution $\mathcal{F}''$ such that $|\mathcal{F}''| \leq 2|\mathcal{F}'|$ that never stores at virtual nodes. Furthermore, $\mathcal{F}''$ can be obtained from $\mathcal{F}'$ in an online manner in polynomial time.*
- *If $\mathcal{F}''$ contains some edge (in G') on a replacing path of some edge (u, v) (of G), it contains all of that replacing path.*

Proof. We construct $\mathcal{F}''$ by modifying $\mathcal{F}'$. Consider an edge (u, v) (in G) and let $[t_1, t_2]$ be a maximal time interval in which a copy of the movie is found in at least one cache belonging to one of the virtual nodes between u and v according to $\mathcal{F}'$. W.l.o.g., assume that the movie reached a virtual node through u at t_1, namely that $(u(t), u'_1(t)) \in \mathcal{F}'$. Remove the arcs $(u'_i(t), u'_i(t+1))$, for $t \in [t_1, t_2 - 1]$, and all edges touching virtual nodes in the time interval $[t_1, t_2]$. Instead, add the arcs $(u(t), u(t+1))$, for $t \in [t_1, t_2 - 1]$. Furthermore, if there exist $t \in [t_1, t_2]$ such that $(u'_{n^2-1}(t), v(t)) \in \mathcal{F}'$, then let $t_v = \min_t\{t : (u'_{n^2-1}(t), v(t)) \in \mathcal{F}'\}$. In this case, we also add the arcs $(v(t), v(t+1))$, for $t \in [t_v, t_2 - 1]$. Namely, we keep a copy of the movie at u throughout the time interval $[t_1, t_2]$ and at v during $[t_v, t_2]$. Moreover, if the copy of movie arrived at v at t_v through u'_{n^2-1} we also add the path $\{(u'_{i-1}(t_v), u_i(t_v) : i \in [1, n^2]\}$. Since the movie was saved in one of the virtual nodes throughout $[t_1, t_2]$, the number of added arcs is at most twice the number of removed arcs. Furthermore, the path $\{(u'_{i-1}(t_v), u'_i(t_v) : i \in [1, n^2]\}$ is "funded" by horizontal edges that were removed.

Finally, notice that we may identify an edge (u, v) and t_1, in an online manner, when the edge $(u(t), u'_1(t))$ is added to $\mathcal{F}'$. The time t_v can be identified as the time in which the movie reaches v. The running time is, clearly, polynomial. $\square$

Given an online algorithm ALG for MOVIE ALLOCATION, we design an algorithm for STEINER TREE as follows. First, we modify ALG such that it never stores at virtual nodes if the requests never arrive on virtual nodes. Call the modified algorithm ALG*. Upon the arrival of a terminal τ_j, we execute ALG* on $\tau_j(j)$. Now assume that some edge $e(j)$ (of G') is added to the MOVIE ALLOCATION solution $\mathcal{F}'$. Note that e belongs to the replacing path of exactly one edge e of G. Then e is added to STEINER TREE solution F.

Observation 3. $n^2|F| \leq |\mathcal{F}'|$.

Similarly, let us compare $|\text{OPT}'|$, the size of optimal solution of MOVIE ALLOCATION to $|\text{OPT}|$, the size of the solution of the Steiner tree problem.

Observation 4. $|\text{OPT}'| \leq 2n^2|\text{OPT}|$.

Proof. Let us now generate some solution $\mathcal{F}'$ of the corresponding MOVIE ALLOCATION instance. Let τ_j be a terminal in the input of the STEINER TREE problem. In $\mathcal{F}'$, a copy is delivered to the node replica $\tau_j(0)$ (at time 0), for every j. This costs at most $n^2|F|$ horizontal edges for all the terminals. Moreover, keep the copy at τ_j until after the last request. This costs $(N-1)N$ arcs for all the terminals. Since in the STEINER TREE instance, $N \leq n$,

$$|\mathcal{F}'| \leq n^2|\text{OPT}| + (N-1)n \leq n^2(|\text{OPT}| + 1) ,$$

and we are done since $|\text{OPT}'| \leq |\mathcal{F}'|$. $\square$

Given the lower bound for online STEINER TREE [17], it follows that

Theorem 4. *For every algorithm for online* MOVIE ALLOCATION *there is an infinite family of unit cost instances for which the competitive ratio is* $\Omega(\log N)$. *Furthermore,* $\log \delta = \Theta(\log N)$ *for every instance in this family.*

Proof. By Lemma 7, Observation 3 and Observation 4 any online algorithm ALG for MOVIE ALLOCATION can be used to design an online STEINER TREE using the above mentioned transformation. Since $\delta(G') = n^2 \cdot \delta(G)$, we get the required family of MOVIE ALLOCATION instances by transforming the family of instances from Theorem 3. $\square$