Distributed Algorithms for Content Caching in Mobile Backhaul Networks

Valentino Pacifici and Slađana Jošilo and György Dán
ACCESS Linnaeus Center, School of Electrical Engineering
KTH, Royal Institute of Technology, Stockholm, Sweden
E-mail: {pacifici, josilo, gyuri}@kth.se

***Abstract*—The growing popularity of mobile multimedia content and the increase of wireless access bitrates are straining backhaul capacity in mobile networks. A cost-effective solution to reduce the strain, enabled by emerging all-IP 4G and 5G mobile backhaul architectures, could be in-network caching of popular content during times of peak demand. In this paper we formulate the problem of content caching in a mobile backhaul as a binary integer programming problem, and we propose a 2-approximation algorithm for the problem. The 2-approximation requires full information about the network topology and the link costs, as well as about the content demands at the different caches, we thus propose two distributed algorithms that are based on limited information on the content demands. We show that the distributed algorithms terminate in a finite number of steps, and we provide analytical results on their approximation ratios. We use simulations to evaluate the proposed algorithms in terms of the achieved approximation ratio and computational complexity on realistic mobile backhaul topologies.**

I. Introduction

The penetration of high speed mobile access technologies, such as HSDPA and LTE, together with the proliferation of powerful handheld devices has stimulated a rapid increase of user demand for mobile multimedia content in recent years. The traffic growth is predicted to continue in coming years, with an estimated 10-fold increase in mobile data traffic in 5 years and an increasing peak-to-average traffic ratio, and puts significant strain on mobile backhaul capacity.

Recent measurement studies of mobile data traffic indicate that caching could be an effective means of decreasing the mobile backhaul bandwidth requirements: caching could reduce the bandwidth demand by up to 95% during peak hours and could at the same time reduce content delivery time by a factor of three [1]. At the same time, mobile traffic is dominated by downloads; up to 75% of daily traffic load comes from download traffic, and the demand shows significant diurnal fluctuations with low loads during early morning hours [2].

While tunelling imposed by previous 3GPP standards made backhaul in-network caching technically challenging, allowing only caches at the network edge, in emerging all-IP mobile backhaul architectures the caches could be co-located with every switch and could implement cooperative caching policies throughout the backhaul. Since fairly accurate content popularity predictions can be obtained for Web and video content [3], [4], the most popular contents could be downloaded into the caches of the mobile backhaul in the early morning hours when the load is relatively low, thereby alleviating the traffic demand during peak hours.

Given predicted content popularities, a fundamental problem of in-network caching in a mobile backhaul is to find efficient content placement algorithms that take into consideration the characteristics of mobile backhaul topologies and of mobile data traffic. The algorithms should achieve close to optimal bandwidth cost savings and should have low computational complexity. Furthermore, they should require as little information as possible, e.g., about content popularities and network topology, in order to allow fully distributed operation and scaling to large topologies with small communication overhead. While previous works proposed centralized and distributed content placement algorithms for two-level hierarchical topologies [5], general topologies with an ultrametric [6], and topologies in a metric space [7], efficient distributed algorithms based on limited topological information have received little attention.

In this paper we formulate the problem of content placement in a mobile backhaul based on predicted demands as a 0-1 integer programming problem. We show that a 2-approximation to the problem can be obtained using a distributed greedy algorithm when global information is available, and propose two computationally simple distributed algorithms that do not require global information. We evaluate the algorithms through extensive simulations on various network topologies. Our results show that information about object demands at descendants is not sufficient for achieving good performance, but the proposed h-Push Down algorithm achieves consistently good performance based on a limited amount of information about object placements.

The rest of the paper is organized as follows. Section II describes the system model and provides the problem formulation. Section III describes the 2-approximation algorithm based on global information, and Section IV describes the distributed algorithms based on limited information. Section V shows performance results based on simulations. Section VI discusses related work and Section VII concludes the paper.

II. System Model and Problem Formulation

We consider a typical mobile backhaul, and model its active topology by a symmetric acyclic directed graph $\mathcal{G}(\mathcal{N}, E)$, where the vertices $\mathcal{N}$ are routers that connect cell sites and may

The work was partly funded by SSF through the Modane project and by the Swedish Research Council through project 621-2014-6035.

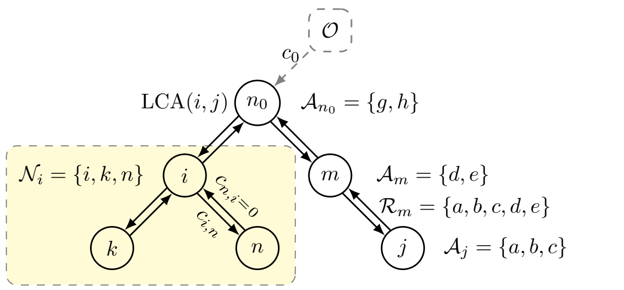

Fig. 1: Example backhaul topology with nodes in three levels, showing commonly used notation.

aggregate traffic from other routers (and thus cell cites), and for every connected pair of nodes $i, j \in \mathcal{N}$ there exist edges $(i,j) \in E$ and $(j,i) \in E$. Observe that since $\mathcal{G}$ is connected and acyclic, $\mathcal{G}$ is a tree. We denote by $\mathcal{L}$ the set of leaf nodes in $\mathcal{G}$, by $\mathcal{I}$ the set of internal nodes and by n_0 the root node, i.e., $\mathcal{N} = \mathcal{L} \cup \mathcal{I} \cup n_0$. We denote the unique simple path from node i to node j by $P_{i,j} = \big((i,v_1),(v_1,v_2),...,(v_{|P_{i,j}|-1},j)\big)$, and we denote by $|P_{i,j}|$ the number of edges in path $P_{i,j}$. Observe that $|P_{i,j}| = |P_{j,i}|$. We define the level $l(i)$ of node $i \in \mathcal{N}$ in the tree $\mathcal{G}$ as the number of edges from node i to the tree's root node n_0 in the unique simple path from i to n_0, i.e., $l(i) = |P_{i,n_0}|$. We denote the children of node $i \in \mathcal{N}$ by $\mathcal{C}(i) \triangleq \{j|(i,j) \in E \wedge l(j) > l(i)\}$ and the parent of node i by $\mathcal{P}(i)$, where $\mathcal{P}(i) \in \mathcal{N}$ such that $i \in \mathcal{C}(\mathcal{P}(i))$. We denote by $\mathcal{P}^l(i)$ the l^{th}-ancestor of node i, e.g., $\mathcal{P}^2(i) = \mathcal{P}(\mathcal{P}(i))$. By definition $\mathcal{P}^0(i) = i$. We refer to an edge (i,j) as the downlink direction if $j \in \mathcal{C}(i)$ and as the uplink direction if $i \in \mathcal{C}(j)$.

We say that two nodes are siblings if they have the same parent, and define the sibling set $\mathcal{S}(i) \triangleq \{j|\mathcal{P}(j) = \mathcal{P}(i) \wedge i \neq j\}$. We denote the descendants of node i by $\mathcal{D}(i) \triangleq \{j|l(j) > l(i) \wedge \mathrm{LCA}(i,j) = i\}$, where $\mathrm{LCA}(i,j)$ denotes the lowest common ancestor of nodes i and j, furthermore we use the notation $\mathcal{G}_i(\mathcal{N}_i, E_i)$ for the subgraph induced by $\mathcal{N}_i = \{i\} \cup \mathcal{D}(i)$ rooted in i.

A. Objects, Demand and Storage

We denote the set of objects requested by mobile nodes by $\mathcal{O}$. We follow common practice and consider that every object has unit size [8], [9], which is a reasonable simplification if content is divisible into unit-sized chunks. We denote the average request rate (demand) predicted for the peak hours for object $o \in \mathcal{O}$ at the cell site connected to node i by w_i^o.

Every node $i \in \mathcal{N}$ is equipped with a cache, and we denote the size of the cache at node i by K_i. We denote the set of objects stored in the cache at node i by $\mathcal{A}_i \subset \mathcal{O}$, $|\mathcal{A}_i| \leq K_i$. We use the shorthand notation $\mathcal{A}_V \triangleq (\mathcal{A}_j)_{j \in V}$, where $V \subseteq \mathcal{N}$, and $\mathcal{A}_{-i} \triangleq (\mathcal{A}_j)_{j \in \mathcal{N}\setminus\{i\}}$. We denote by $\mathfrak{A}_i$ the set of object placements that satisfy the storage capacity constraint at node i, i.e. $\mathfrak{A}_i = \{\mathcal{A}_i \in 2^{\mathcal{O}} : |\mathcal{A}_i| \leq K_i\}$, where $2^{\mathcal{O}}$ is the powerset of $\mathcal{O}$. Finally, we denote the set of objects stored at node i and at its descendants by $\mathcal{R}_i(\mathcal{A}) = \mathcal{A}_i \bigcup_{j \in \mathcal{D}(i)} \mathcal{R}_j(\mathcal{A})$. Figure 1 shows an example topology with a maximum level of 2, illustrating some of the commonly used notation.

B. Cost model

We denote the unit cost of using edge (i,j) by $c_{i,j}$. Since during peak hours most of the traffic in a mobile backhaul is flowing downlink (serving users' requests for content) [1], [2], we consider that uplink edges have zero unit cost, i.e., $c_{i,\mathcal{P}(i)} = 0$. Without loss of generality, the cost of downlink edges is $c_{\mathcal{P}(i),i} > 0$. We consider that edge costs are additive, i.e., if a request for object o arrives at node i and is served from node j then the unit cost is $d_{i,j} = \sum_{(v,w) \in P_{j,i}} c_{v,w}$. We call $d_{i,j}$ the *distance* from node j to node i. Note that the terms $c_{v,w}$ are zero if they correspond to an uplink, i.e., if $w = \mathcal{P}(v)$. Furthermore, observe that in general $d_{j,i} \neq d_{i,j}$, thus distance is not symmetric (hence it is a hemimetric).

A request for object o generated by a mobile user connected to the cell site at node $i \in \mathcal{N}$ is served locally if $o \in \mathcal{A}_i$. Otherwise, if node i has a descendant $j \in \mathcal{D}(i)$ for which $o \in \mathcal{A}_j$, the node forwards the request to the closest such descendant. Otherwise, node i forwards the request to its parent $\mathcal{P}(i)$, which follows the same algorithm for serving the request. If an object o is not stored in any node (i.e., $o \notin \mathcal{R}_{n_0}$) then it needs to be retrieved through the Backbone via the root node n_0 at a unit cost of c_0.

Given a placement $\mathcal{A} = (\mathcal{A}_j)_{j \in \mathcal{N}}$ we can define the unit cost to serve a request for object o at node i as

$$d_i(o,\mathcal{A}) = \begin{cases} \min\limits_{\{j \in \mathcal{N} | o \in \mathcal{A}_j\}} d_{i,j} & \text{if } o \in \mathcal{R}_{n_0} \\ d_{i,n_0} + c_0 & \text{if } o \notin \mathcal{R}_{n_0}, \end{cases}$$

which together with the demand w_i^o determines the cost incurred by node i as

$$C_i(\mathcal{A}) = \sum_{o \in \mathcal{O}} C_i^o(\mathcal{A}) = \sum_{o \in \mathcal{O}} w_i^o d_i(o,\mathcal{A}). \quad (1)$$

Finally, we define the total cost $C(\mathcal{A}) = \sum_{i \in \mathcal{N}} C_i(\mathcal{A})$.

C. Problem formulation

Motivated by minimizing the congestion in the mobile backhaul during peak hours, our objective is to find a placement that minimizes the total cost $C(\mathcal{A})$. We refer to this as the mobile backhaul content placement problem (MBCP), which can be formulated as finding $\bar{\mathcal{A}} = \arg\min_{\mathcal{A} \in \times_{i \in \mathcal{N}} \mathfrak{A}_i} C(\mathcal{A})$.

It is easy to see that the MBCP problem can be formulated as the following $0-1$ integer linear program

$$\min \sum_{i \in \mathcal{N}} \sum_{o \in \mathcal{O}} w_i^o \Big(\sum_{j \in \mathcal{N}, j \neq i} d_{i,j} x_{i,j,o} + (d_{i,n_0} + c_0) x_{i,-1,o} \Big) \quad \text{s.t.}$$

$$\textstyle\sum_{o \in \mathcal{O}} x_{i,o} \leq K_i, \quad \forall i \in \mathcal{N}$$

$$x_{i,j,o} \leq x_{j,o}, \quad \forall i,j \in \mathcal{N}, o \in \mathcal{O}$$

$$\textstyle\sum_{j \in \mathcal{N}} x_{i,j,o} + x_{i,-1,o} \geq 1, \quad \forall i \in \mathcal{N}, o \in \mathcal{O}$$

$$x_{i,o}, x_{i,j,o}, x_{i,-1,o} \in \{0,1\},$$

where $x_{i,o}$ indicates whether object o is in the storage of node i (i.e. $x_{i,o} = 1 \Leftrightarrow o \in \mathcal{A}_i$), $x_{i,j,o}$ indicates whether a request for object o at node i is served from node j, and $x_{i,-1,o}$ indicates whether object o is retrieved from the Backbone, i.e., the *level* of the Backbone is indicated with -1.

It can be shown that for 4 or more nodes the constraint matrix is not totally unimodular and solving the MBCP would be computationally infeasible already for moderate sized instances of the problem. We are thus interested in finding computationally feasible, scalable distributed algorithms to approximate the solution.

III. Distributed 2-Approximation Algorithm Based On Global Information

In what follows we show that if global information is available about the object demands and placements at every node of the network, then it is possible to obtain a 2-approximation to the optimal solution using the *Depth First Greedy* (*DFG*) algorithm. The *DFG* algorithm is based on a depth-first traversal of the graph $\mathcal{G}$, i.e., an ordering $i_1, \ldots, i_{|\mathcal{N}|}$ of the vertices in $\mathcal{N}$, and can be executed by the nodes in an iterative (distributed) manner. The algorithm starts with an empty allocation ($\mathcal{A}_i = \emptyset$); at iteration $1 \leq k \leq |\mathcal{N}|$ node i_k populates its cache with K_{i_k} objects, one at a time, that provide the highest global cost saving. The *DFG* algorithm is shown in Figure 2.

DFG Algorithm

```
INPUT: DF Traversal (i_1, ..., i_|N|)
k ← 1
A_i ← ∅, for all i ∈ N
for k = 1 ... |N| do
    while |A_{i_k}| < K_{i_k} do
        o* ← arg max_{o∈O} (C(A_{-i_k}, A_{i_k}) − C(A_{-i_k}, A_{i_k} ∪ {o}))
        A_{i_k} ← A_{i_k} ∪ {o*}
    end while
end for
```

Fig. 2: Pseudo-code of the *DFG* algorithm

Theorem 1. *The* DFG *algorithm is a 2-approximation algorithm for the MBCP problem in terms of cost saving, i.e.,* $\frac{C(\emptyset)-C(\bar{\mathcal{A}})}{C(\emptyset)-C(\mathcal{A}^{\mathrm{DFG}})} \leq 2$.

Before we prove the theorem we introduce some definitions and previous results.

Definition 1. Let E be a finite set and let $\mathcal{F}$ be a collection of subsets of E. The pair (E,$\mathcal{F}$) is a *partition matroid* if $E = \bigcup_{i=1}^{k} E_i$ is the disjoint union of k sets, $l_1, ..., l_k$ are positive integers and $\mathcal{F} = \{F | F = \bigcup_{i=1}^{k} F_i, F_i \subseteq E_i, |F_i| \leq l_i, i = 1, ..., k\}$.

Definition 2. Let E be a finite set, and $f : 2^E \to \mathbb{R}$ a real valued function on subsets of E. Then f is submodular if for every $A, B \in E$ we have

$$f(A \cap B) + f(A \cup B) \leq f(A) + f(B).$$

Let us now recall a fundamental result about the maximization of submodular functions over partition matroids.

Lemma 1. *[10] Let $\mathcal{F}$ be a partition matroid over a set E, and $f : \mathcal{F} \to \mathbb{R}$ be a non-decreasing submodular function with* $f(\emptyset) = 0$. *Then the* DFG *algorithm achieves a 2-approximation of* $\max_{F \in \mathcal{F}} f(F)$.

In what follows we show that MBCP can be formulated as the maximization of a non-decreasing submodular function over a partition matroid. Let us define for every object $o \in \mathcal{O}$ one fictitious object (o, i) per node $i \in \mathcal{N}$, i.e., $(o, i) \in \mathcal{O} \times \mathcal{N}$. The set of fictitious objects that can be assigned to node i is then $\mathcal{E}_i = \{(o, i) | o \in \mathcal{O}\}$ and we define the set $\mathcal{E} = \bigcup_{i \in \mathcal{N}} \mathcal{E}_i$. We denote by $\mathfrak{A}$ the family of subsets of $\mathcal{E}$, defined as $\mathfrak{A} = \times_{i \in \mathcal{N}} \mathfrak{A}_i$, where $\mathfrak{A}_i \subseteq \mathcal{E}_i$, $|\mathfrak{A}_i| \leq K_i$ is the set of object placements that satisfy the storage capacity constraint at node i, as defined in Section II-A.

Proposition 2. *The pair* $(\mathcal{E}, \mathfrak{A})$ *is a partition matroid.*

Proof. Consider an allocation $\mathcal{A} \in \mathfrak{A}$ and a fictitious object $(o, i) \in \mathcal{A}_i$. If we remove (o, i) from $\mathcal{A}_i$, i.e. $\mathcal{A}'_i = \mathcal{A}_i \setminus \{(o, i)\}$, then $\mathcal{A}'_i \subseteq \mathcal{E}_i$ will still hold as well as $\mathcal{A}_j \subseteq \mathcal{E}_j$, for $j \in \mathcal{N} \setminus \{i\}$, which implies that $(\mathcal{E}, \mathfrak{A})$ is an independence system.

Consider now two allocations $\mathcal{A}, \mathcal{A}' \in \mathfrak{A}$. If $|\mathcal{A}| < |\mathcal{A}'|$ then $\exists \mathcal{E}_i$ such that $|\mathcal{A}' \cap \mathcal{E}_i| > |\mathcal{A} \cap \mathcal{E}_i|$, which implies that there is a node $i \in \mathcal{N}$ with at least one free space in its cache, i.e. $|\mathcal{A}_i| < K_i$. Therefore, there is an $(o, i) \in (\mathcal{A}' \setminus \mathcal{A}) \cap \mathcal{E}_i$ such that $\mathcal{A} \cup \{(o, i)\} \in \mathfrak{A}$. ☐

Proof of Theorem 1. We prove the theorem by showing that the function $\bar{C}(\mathcal{A}) = -C(\mathcal{A})$ is a nondecreasing submodular function on $\mathcal{E}$. Let us define the change of the global cost after inserting an object o in the cache of node i as $\Delta C(\mathcal{A}) = \bar{C}(\mathcal{A} \cup \{(o, i)\}) - \bar{C}(\mathcal{A})$, where $\mathcal{A} \in \mathfrak{A}$ and $\exists i \in \mathcal{N}$ for which $|\mathcal{A}_i| < K_i$. We show that $\bar{C}(\mathcal{A} \cup (o, i)\}) - \bar{C}(\mathcal{A}) \geq \bar{C}(\mathcal{A}' \cup (o, i)\}) - \bar{C}(\mathcal{A}')$ for all $\mathcal{A} \subseteq \mathcal{A}' \in \mathfrak{A}$ and $(o, i) \in \mathcal{E}_i \setminus \mathcal{A}'_i$. We now distinguish between two cases. If $\exists j$ such that $(o, j) \in \mathcal{A}'_j \setminus \mathcal{A}_j$ then the difference $\Delta C(\mathcal{A})$ is

$$\begin{aligned}\Delta C(\mathcal{A}) = \; & c_0 \sum_{k \in \{\mathcal{N} | \,\mathrm{LCA}(k,i) = n_0\}} w_k^o + (c_0 + d_{i,n_0}) \sum_{k \in \mathcal{N}_i} w_k^o + \\ & + \sum_{t=1}^{l(i)-1} (c_0 + d_{\mathcal{P}^t(i),n_0}) \sum_{k \in \{\mathcal{N}_{\mathcal{P}^t(i)} \setminus \mathcal{N}_{\mathcal{P}^{t-1}(i)}\}} w_k^o,\end{aligned}$$

and the difference $\Delta C(\mathcal{A}')$ is

$$\begin{aligned}\Delta C(\mathcal{A}') = \; & (c_0 + d_{i,\mathrm{LCA}(j,i)}) \sum_{k \in \mathcal{N}_i} w_k^o + \\ & + \sum_{t=1}^{l(i)-l(\mathrm{LCA}(j,i))-1} (c_0 + d_{\mathcal{P}^t(i),\mathrm{LCA}(j,i)}) \sum_{k \in \{\mathcal{N}_{\mathcal{P}^t(i)} \setminus \mathcal{N}_{\mathcal{P}^{t-1}(i)}\}} w_k^o.\end{aligned}$$

Since $l(\mathrm{LCA}(j, i)) \geq 0$, it holds that $\Delta C(\mathcal{A}) > \Delta C(\mathcal{A}')$. Otherwise, if $\exists j$ such that $(o, j) \in \mathcal{A}_j$ or if $\nexists j$ such that $(o, j) \in \mathcal{A}'_j$ then $\Delta C(\mathcal{A}) = \Delta C(\mathcal{A}')$. The result then follows by applying Lemma 1 to $C(\emptyset) - C(\mathcal{A})$. ☐

Observe that the approximation ratio is bounded for arbitrary traversals of the graph. Nonetheless, a pre-order depth-first traversal allows for a distributed implementation of *DFG* with a communication overhead of $\sum_{k=1}^{|\mathcal{N}|} (|\mathcal{N}| - k) K_{i_k}$.

It is important to note that *DFG* differs from the distributed global greedy (*DGG*) algorithm used in [5], [11]. *DGG* chooses in every iteration the fictitious item (i, o) that maximizes the cost saving, and thus has computational complexity $O(|\mathcal{N}|^2 \max_i K_i |\mathcal{O}| \log(|\mathcal{N}||\mathcal{O}|))$. In contrast, *DFG* populates the caches of the nodes one-by-one, and thus has computational complexity $O(|\mathcal{N}| \max_{i_k} K_{i_k} |\mathcal{O}| \log(|\mathcal{O}|))$. Unfortunately, *DFG* requires global information at every node of the network, which may cause significant communication overhead. We therefore turn to distributed approximation algorithms based on limited information.

IV. Distributed Algorithms under Limited Information

In what follows we propose two distributed algorithms that do not need global information about the demands and the network topology.

A. Local Greedy Swapping (LGS) Algorithm

The first algorithm, called Local Greedy Swapping (*LGS*), allows nodes to swap objects with their parents based on the aggregate demands and the object placements in their *descendants only*. Denoting the placement at node i at iteration k by $\mathcal{A}_i(k)$, the *LGS* algorithm starts with an arbitrary initial object placement $\mathcal{A}(0) = (\mathcal{A}_i(0))_{i\in\mathcal{N}}$ in which each node $i \in \mathcal{N}$ stores K_i objects. At iteration k the algorithm computes the set of beneficial swaps $T(\mathcal{A}(k)) \subset \mathcal{N} \times \mathcal{O}^2$. A triplet $(i, o, p) \in T(\mathcal{A}(k))$ corresponds to that node i can swap object $p \in \mathcal{A}_i(k)$ with object $o \in \mathcal{A}_{\mathcal{P}(i)}(k)$ at its parent node $\mathcal{P}(i)$. For $i = n_0$, i.e., $(n_0, o, p) \in T(\mathcal{A}(k))$ the root node n_0 can evict object p and can fetch object o through the Backbone. The set of implemented swaps $S(\mathcal{A}(k)) \subseteq T(\mathcal{A}(k)$ is then chosen to increase the local cost saving greedily.

To define the set of beneficial swaps $T(\mathcal{A})$, let us introduce the function $I(i, o, p)$ to indicate whether the aggregate demand at node i and its descendants $\mathcal{D}(i)$ is higher for object o than for object p,

$$I(i,o,p) = \begin{cases} 1, & \text{if} \sum_{j\in\mathcal{N}_i}(w_j^o - w_j^p) > 0 \\ 0, & \text{otherwise.} \end{cases} \qquad (2)$$

Given a placement $\mathcal{A}$, node i might be interested in swapping object $p \in \mathcal{A}_i$ with object $o \in \mathcal{A}_{\mathcal{P}(i)}$ at its parent if $I(i, o, p) = 1$ or if p is available in the cache of node i's descendants $\mathcal{D}(i)$, i.e., $p \in \mathcal{R}_i \setminus \mathcal{A}_i$, as in this case node i can retrieve object p at no cost even if $p \notin \mathcal{A}_i$. We use this observation to define the set of node-object triplets that would be beneficial for swapping at placement $\mathcal{A}$,

$$\begin{aligned} T(\mathcal{A}) = \{&(i,o,p) | i \in \mathcal{N}, o \in \mathcal{A}_{\mathcal{P}(i)} \setminus \mathcal{R}_i, p \in \mathcal{A}_i, \\ &\big((p \in \mathcal{R}_i \setminus \mathcal{A}_i) \vee (p \notin \mathcal{R}_i \setminus \mathcal{A}_i \wedge I(i,o,p) = 1)\big)\}. \end{aligned}$$

The algorithm terminates at iteration k if the set $T(\mathcal{A}(k))$ is empty. The pseudo-code of *LGS* is shown in Fig. 3.

To complete the definition of the algorithm, we now describe a greedy algorithm to choose the set $S(\mathcal{A}(k)) \subseteq T(\mathcal{A}(k))$ at iteration k. Given $T(\mathcal{A}(k))$, we choose a node i_k with a child that would like to swap (i.e., $\exists j \in \mathcal{C}(i_k)$ and $(j, o, p) \in T(\mathcal{A}(k))$). Given i_k we select the best swap (j_k, o_k, p_k) of its children, i.e., the one that maximizes the local cost saving in the subtree $\mathcal{N}_{i_k}$ (swap with parent), and we then allow every child node $j \in \mathcal{C}(i_k)$ to insert into its cache objects $o \in \mathcal{A}_{i_k}(k) \cup \{p_k\}$, if doing so would increase the local cost saving (copy from parent). The algorithm is shown in Algorithm 1.

LGS Algorithm

$k \leftarrow 0$
while $|T(\mathcal{A}(k))| > 0$ **do**
  $\mathcal{A}(k+1) \leftarrow \mathcal{A}(k)$
  for each $(i, o, p) \in S(\mathcal{A}(k))$ **do**
    $\mathcal{A}_i(k+1) \leftarrow (\mathcal{A}_i(k) \cup \{o\} \setminus \{p\})$
    if $p \notin \mathcal{A}_{\mathcal{P}(i)}(k)$ **then**
      $\mathcal{A}_{\mathcal{P}(i)}(k+1) \leftarrow (\mathcal{A}_{\mathcal{P}(i)}(k) \cup \{p\} \setminus \{o\})$
    end if
  end for
  $k \leftarrow k+1$
end while

Fig. 3: Pseudo-code of the *LGS* algorithm

Lemma 2. *The global cost C decreases strictly at every swap.*

Proof. Consider $(i, o, p) \in S(\mathcal{A}(k))$ at iteration k. For every node $j \in \mathcal{N} \setminus \mathcal{N}_i$ it holds $d_{j,i} = d_{j,\mathcal{P}(i)} + c_{i,\mathcal{P}(i)} = d_{j,\mathcal{P}(i)}$, hence $d_j(o, \mathcal{A}(k+1)) = d_j(o, \mathcal{A}(k))$ and $d_j(p, \mathcal{A}(k+1)) = d_j(p, \mathcal{A}(k))$. Consequently, $C_j(\mathcal{A}(k+1)) = C_j(\mathcal{A}(k))$ for all $j \in \mathcal{N} \setminus \mathcal{N}_i$.
Consider now node $j \in \mathcal{N}_i$. Since $S(\mathcal{A}(k)) \subseteq T(\mathcal{A}(k))$, it follows that $o \notin \mathcal{R}_i(k)$ and $o \in \mathcal{A}_{\mathcal{P}(i)}(k)$. Hence $d_j(o, \mathcal{A}(k)) = d_{j,i} + c_{\mathcal{P}(i),i}$, $d_j(o, \mathcal{A}(k+1)) = d_{j,i}$, and the difference in the cost $\Delta C(k+1)$ before and after the swap is

$$\begin{aligned} \Delta C(k+1) =& \sum_{j\in\mathcal{N}_i} \left[C_j(\mathcal{A}(k+1)) - C_j(\mathcal{A}(k))\right] \\ =& \sum_{j\in\mathcal{N}_i} \big[w_j^o d_{j,i} - w_j^o(d_{j,i} + c_{\mathcal{P}(i),i}) + w_j^p d_j(p, \mathcal{A}(k+1)) \\ & - w_j^p d_j(p, \mathcal{A}(k))\big] = \sum_{j\in\mathcal{N}_i} \big[-w_j^o c_{\mathcal{P}(i),i} + w_j^p \, \big(d_j(p, \mathcal{A}(k+1)) \\ & - d_j(p, \mathcal{A}(k))\big)\big]\,. \end{aligned}$$

Similarly, $S(\mathcal{A}(k)) \subseteq T(\mathcal{A}(k))$ implies that $p \in \mathcal{A}_i(k)$, hence $d_j(p, \mathcal{A}(k)) \leq d_{j,i}$. We now distinguish between two cases. If $d_j(p, \mathcal{A}(k)) < d_{j,i}$, then $d_j(p, \mathcal{A}(k+1)) = d_j(p, \mathcal{A}(k))$, which implies that $\Delta C(k+1) < 0$. Otherwise, if $d_j(p, \mathcal{A}(k)) = d_{j,i}$, then $d_j(p, \mathcal{A}(k+1)) = d_{j,i} + c_{\mathcal{P}(i),i}$. Since $I(i, o, p) = 1$, then $\Delta C(k+1) = c_{\mathcal{P}(i),i} \sum_{j\in\mathcal{N}_i}(w_j^p - w_j^o) < 0$. This proves the lemma. □

We can use this result to show that the algorithm terminates after a finite number of iterations.

Theorem 3. *The* LGS *algorithm terminates after a finite number of iterations.*

Proof. Consider iteration k of the *LGS* algorithm. Call $s(\mathcal{A})$ the object placement that results from applying swap $s = (j, o, p)$ to placement $\mathcal{A}$. It follows from the proof of Lemma 2 that for any swap $s = (j, o, p) \in S(\mathcal{A}(k))$ and every node $l \in \mathcal{N} \setminus \mathcal{N}_j$, it holds $\{\mathcal{R}_j(\mathcal{A}(k)) \cup \mathcal{A}_{i_k}(k)\} = \{\mathcal{R}_j(s(\mathcal{A}(k))) \cup s(\mathcal{A}_{i_k}(k))\}$ and hence $C_l(s(\mathcal{A}(k))) = C_l(\mathcal{A}(k))$. Since for every $j, l \in \mathcal{C}(i_k), j \neq l$ it holds $l \notin \mathcal{N}_j$, we can consider each node $j \in \mathcal{C}(i_k)$ separately.
Consider swap $s = (j, o, p) \in S(\mathcal{A}(k))$. It follows from (3) that either $p \in \mathcal{R}_j \setminus \mathcal{A}_j(k)$ or $I(j, o, p) = 1$. Therefore, from the proof of Lemma 2, it follows that $C_l(s(\mathcal{A}(k))) \leq C_l(\mathcal{A}(k))$ for

Algorithm 1 $S(\mathcal{A}(k)) = \text{populateS}(\mathcal{A}(k), i_k)$

1: Select the best swapping opportunity at the children of i_k,

$$(j_k, o_k, p_k) \quad \leftarrow \underset{\{(j,o,p)\in T(\mathcal{A}(k))|j\in\mathcal{C}(i_k)\}}{\arg\max} \sum_{n\in\mathcal{N}_j} c_{i,j}(w_n^o - w_n^p)$$

$$S(\mathcal{A}(k)) \quad \leftarrow \quad (j_k, o_k, p_k)$$

2: Further decrease the cost function through allowing nodes in $\mathcal{C}(i_k)$ to insert objects available at $\{\mathcal{A}_{i_k}(k) \cup \{p_k\}\}$.

$\mathrm{PE}_j \leftarrow (\mathcal{A}_{i_k}(k) \cup \{p_k\}) \cap \mathcal{A}_j(k)$
$\mathrm{PO}_j \leftarrow (\mathcal{A}_{i_k}(k) \cup \{p_k\}) \setminus \mathcal{R}_j(k)$
while $\exists(j, o, p)$ s.t. $o \in \mathrm{PO}_j$ and $p \in \mathrm{PE}_j$ and

$$(p \in \mathcal{R}_j \setminus \mathcal{A}_j(k)) \vee (p \notin \{\mathcal{R}_j \setminus \mathcal{A}_j(k)\} \wedge I(j, o, p) = 1)) \tag{3}$$

do
 $S(\mathcal{A}(k)) \leftarrow S(\mathcal{A}(k)) \cup \{(j, o, p)\}$
 $\mathrm{PE}_j \leftarrow \mathrm{PE}_j \setminus \{p\}$
 $\mathrm{PO}_j \leftarrow \mathrm{PO}_j \setminus \{o\}$
end while

all $l \in \mathcal{N}_j$. In particular, for swap $s_k = (j_k, o_k, p_k) \in T(\mathcal{A}(k))$, it holds that $I(j_k, o_k, p_k) = 1$, which implies $C_{j_k}(s_k(\mathcal{A}(k))) < C_{j_k}(\mathcal{A}(k))$.
Since $\times_{i\in\mathcal{N}}\mathfrak{A}_i$ is a finite set, $C(\mathcal{A}(k))$ can not decrease indefinitely and the *LGS* algorithm terminates after a finite number of iterations. ☐

Besides guaranteed to converge starting from an arbitrary initial placement, a nice property of *LGS* is that if started from an optimal placement, the algorithm is stable in the sense that it does not make any changes, as we show next.

Corollary 1. *An optimal content placement $\bar{A}$ is stable under the* LGS *algorithm.*

Proof. From Lemma 2 and Theorem 3 it follows that $C(\mathcal{A}(k+1)) < C(\mathcal{A}(k))$ for any swap $s \in S(\mathcal{A}(k))$. By definition $\nexists \mathcal{A}' \in \times_{i\in\mathcal{N}}\mathfrak{A}_i$ s.t. $C(\mathcal{A}') < C(\bar{\mathcal{A}})$, hence the result. ☐

For simplicity, we restricted ourselves to a single i_k per iteration when defining $S(\mathcal{A}(k))$, but the above results hold for any set of nodes that are not each others' descendants, hence the algorithm can be executed in parallel.

B. *h-Push Down Algorithm*

In the *LGS* algorithm, every node i swaps objects based on the information about the object placement and the aggregate demand for objects at its descendants $\mathcal{D}(i)$. In the following we provide a distributed algorithm that allows node i to leverage additional information on placements and on aggregate demands for objects. In the *h-Push Down* algorithm, every node i has information about the placement $\mathcal{A}_{\mathcal{N}_j}$ and about the object demands w_k^o, $k \in \mathcal{N}_j$, for every ancestor j that lies within its information horizon h, i.e., for $j = \mathcal{P}^l(i)$ for $0 \le l \le h$.

The algorithm starts with an object placement $(\mathcal{A}_i(0))_{i\in\mathcal{N}}$ in which each node $i \in \mathcal{N}$ stores K_i objects that have the highest aggregated demands in the subnetwork $\mathcal{N}_i$ and that are not available in the cache of node i's descendants $\mathcal{D}(i)$.

Algorithm 2 $\mathcal{A}' = \text{PushDown}(i, \mathcal{A})$

$t \leftarrow 0$
$\mathcal{A}^0 \leftarrow \mathcal{A}$
do
 $n \leftarrow \mathcal{P}^t(i)$
 $o^t \leftarrow \arg\min_{o\in\mathcal{A}^t_{\mathcal{P}(n)}} C(\mathcal{A}^t_n \cup \{o\}, \mathcal{A}^t_{-n})$
 $\mathcal{A}^{t+1}_n \leftarrow \mathcal{A}^t_n \cup \{o^t\}$
 $\mathcal{A}^{t+1}_{\mathcal{P}(n)} \leftarrow \mathcal{A}^t_{\mathcal{P}(n)} \setminus \{o^t\}$
 $t \leftarrow t + 1$
while $n \neq n_0$
return $\mathcal{A}'$

An iteration of the algorithm consists of two steps. The first step is an eviction operation at some node i. The second step is a *PushDown* move, a sequence of placement updates such that at each update one object $o \in \mathcal{A}_{\mathcal{P}^l(i)}$ is moved from $\mathcal{P}^l(i)$ to $\mathcal{P}^{l-1}(i)$, for $l = 1, 2, ..., k$, where $\mathcal{P}^{k-1}(i) = n_0$. In the last update of the *PushDown* move, i.e., $l = k$, one object is retrieved through the Backbone and stored at the root node $\mathcal{P}^{k-1}(i) = n_0$. The pseudo-code of the *PushDown* move of the *h-Push Down* algorithm is shown in Algorithm 2.

Central to the algorithm is the LCA of node i and the node from which node i would retrieve object o in the placement $(\varnothing, \mathcal{A}_{-i})$, i.e., if it had no objects cached,

$$P_i^o(\mathcal{A}_{-i}) \triangleq \mathrm{LCA}\Big(i, \underset{\{j\in\mathcal{N}\setminus\{i\}|o\in\mathcal{A}_j\}}{\arg\min} d_{i,j}\Big). \tag{4}$$

Similarly, we define $P_i^o(\mathcal{A})$ for placement $\mathcal{A}$, i.e., $P_i^o(\mathcal{A}) = i$ if $o \in \mathcal{A}_i$, otherwise $P_i^o(\mathcal{A}) = P_i^o(\mathcal{A}_{-i})$.

The following lemma shows an important property of the *PushDown* move.

Lemma 3. *A move $\mathcal{A}' = \text{PushDown}(i, \mathcal{A})$ always decreases the global cost by*

$$\Delta C_{\mathrm{PD}}(i, \mathcal{A}) \triangleq C(\mathcal{A}) - C(\mathcal{A}') = \sum_{t=0}^{l(i)} c_{\mathcal{P}^{t+1}(i),\mathcal{P}^t(i)} \sum_{j\in T(t)} w_j^{o^t},$$

where $T(t) = \{j \in \mathcal{N}_{\mathcal{P}^t(i)} | P_j^{o^t}(\mathcal{A}) = \mathcal{P}^{t+1}(i)\}$.

Proof. Consider iteration t of move $\mathcal{A}' = \text{PushDown}(i, \mathcal{A})$. Since $c_{n,\mathcal{P}(n)}$=0, for all $j\in\mathcal{N}\setminus\mathcal{N}_n$ it holds that $d_j(o^t, \mathcal{A}^t)$= $d_j(o^t, \mathcal{A}^{t+1})$. For nodes $j \in \mathcal{N}_n$ we need to distinguish between two cases. If $P_j^{o^t}(\mathcal{A}^t) \neq \mathcal{P}(n)$, then $P_j^{o^{t+1}}(\mathcal{A})$=$P_j^{o^t}(\mathcal{A}^t)$ and $d_j(o^t, \mathcal{A}^t)$=$d_j(o^t, \mathcal{A}^{t+1})$. It follows that, if $j \notin T(t)$, then $C_j^o(\mathcal{A}^t) - C_j^o(\mathcal{A}^{t+1}) = 0$. Otherwise, $P_j^{o^t}(\mathcal{A}^t)$=$\mathcal{P}(n)$ implies $P_j^{o^t}(\mathcal{A}^{t+1})$=$n$, and hence $C_j^o(\mathcal{A}^t) - C_j^o(\mathcal{A}^{t+1}) = w_j^{o^t} c_{\mathcal{P}(n),n}$. By summing over all the $l(i)$ iterations of the *PushDown* move, we prove the lemma. ☐

In the *h-Push Down* algorithm, a node i can only initiate a move, and therefore evict one object o, if o is cached at node i's descendants or if $P_i^o(\mathcal{A}_{-i})$ lies within node i's information horizon, i.e., $P_i^o(\mathcal{A}_{-i}) = \mathcal{P}^l(i)$ for some $0 < l \le h$. We use $\mathrm{Z}_i(\mathcal{A})$ to denote the set of objects that are candidate for eviction at node i under placement $\mathcal{A}$, i.e.,

$$\mathrm{Z}_i(\mathcal{A}) = \{o \in \mathcal{A}_i | P_i^o(-\mathcal{A}) \in \textstyle\bigcup_{l=0}^h \mathcal{P}^l(i) \vee o \in \bigcup_{j\in\mathcal{D}(i)} \mathcal{A}_j\}.$$

We use $\Delta C_{\mathrm{EV}}(i, o, \mathcal{A}) \triangleq C(\mathcal{A}) - C(\mathcal{A}_i \setminus \{o\}, \mathcal{A}_{-i})$ to denote *h-Push Down Algorithm*

$k \leftarrow 0$
$\mathrm{Z}^0 \leftarrow \{i \in \mathcal{N} \text{ such that } |\mathrm{Z}_i(\mathcal{A}(0))| > 0\}$
$\mathcal{A} \leftarrow \mathcal{A}(0)$
while $|\mathrm{Z}^k| > 0$ **do**
  Pick $i_k \in \mathrm{Z}^k$
  Compute the least cost eviction
$$o^k \leftarrow \arg\min_{o \in \mathrm{Z}_{i_k}} |\Delta C_{\mathrm{EV}}(i_k, o, \mathcal{A})|$$
  Compute $\Delta C^h_{\mathrm{PD}}(i_k, \mathcal{A})$ as
$$\Delta C^h_{\mathrm{PD}}(i_k, \mathcal{A}) = \sum_{t=0}^{\min(h, l(i_k))} c_{\mathcal{P}^{t+1}(i_k), \mathcal{P}^t(i_k)} \sum_{j \in T(t)} w_j^{o^t},$$
  if $\Delta C^h_{\mathrm{PD}}(i_k, \mathcal{A}) + \Delta C_{\mathrm{EV}}(i_k, o^k, \mathcal{A}) > 0$ **then**
    $\mathcal{A}(k+1) \leftarrow \mathrm{PushDown}(i, (\mathcal{A}_{i_k} \setminus \{o^k\}, \mathcal{A}_{-i_k}))$
    $k \leftarrow k+1$
    $\mathcal{A} \leftarrow \mathcal{A}(k)$
    $\mathrm{Z}^k \leftarrow \{i \in \mathcal{N} \text{ such that } |\mathrm{Z}_i(\mathcal{A}(k))| > 0\}$
  else
    $\mathrm{Z}^k \leftarrow \mathrm{Z}^k \setminus \{i_k\}$
  end if
end while

Fig. 4: Pseudo code of the *h-Push Down* algorithm.

the change in the global cost caused by the eviction of object o at node i. Observe that $\Delta C_{\mathrm{EV}}(i, o, \mathcal{A}) \leq 0$.

The pseudo-code of the *h-Push Down* algorithm is shown in Figure 4. We start with showing that the algorithm terminates in a finite number of iterations.

Theorem 4. *The h-Push Down algorithm terminates after a finite number of iterations.*

Proof. We prove the theorem by showing that the global cost $C(\mathcal{A})$ decreases at every iteration of the h-Push Down algorithm. From Lemma 3 it follows that

$$\Delta C_{\mathrm{PD}}(i_k, (\mathcal{A}(k)_{i_k} \setminus \{o^k\}, \mathcal{A}(k)_{-i_k})) \geq \Delta C^h_{\mathrm{PD}}(i_k, \mathcal{A}(k)). \quad (5)$$

By definition, the variation of the global cost at iteration k can be written as the sum of the variation due to the eviction and the variation due to *PushDown* move, i.e., $\Delta C_{\mathrm{EV}}(i_k, o^k, \mathcal{A}(k)) + \Delta C_{\mathrm{PD}}(i_k, (\mathcal{A}(k)_{i_k} \setminus \{o^k\}, \mathcal{A}(k)_{-i_k})) = C(\mathcal{A}(k)) - C(\mathcal{A}(k+1))$. The proof of the theorem follows from (5). ☐

Furthermore, similar to *LGS*, the algorithm does not make any changes to an optimal placement, as shown next.

Corollary 2. *The optimal content placement $\bar{\mathcal{A}}$ is stable with respect to the h-Push Down algorithm.*

Proof. The proof is analogous to the proof of Corollary 1. ☐

Observe that the computation of $\Delta C^h_{\mathrm{PD}}(i_k, \mathcal{A}(k))$ depends only on the object demands and the placements at the nodes in the set $\mathcal{N}_{\mathcal{P}^h(i_k)}$. Furthermore, in order to compute $\Delta C_{\mathrm{EV}}(i_k, o^k, \mathcal{A}(k))$, node i_k only requires information about placements and demands in the subnetwork $\mathcal{N}_{P_i^{o^k}(\mathcal{A}_{-i}(k))}$, which lies within node i_k's information horizon h.

V. Numerical Results

We use simulations to evaluate the approximation ratio and the convergence rate of the proposed algorithms. To generate backhaul topologies, we use the *Manhattan* model, in which $|\mathcal{N}|$ nodes are randomly placed on a $|\mathcal{N}| \times |\mathcal{N}|$ grid. Given the node placement, we build a weighted complete graph by setting the weight on edge (i, j) equal to the Euclidean distance between nodes i and j, computed based on their coordinates. We then run *Kruskal*'s algorithm [12] on the resulting weighted complete graph to compute a minimum spanning tree to obtain the topology $\mathcal{G}$. We consider two different cost models. In the *distance* cost model the edge costs $c_{\mathcal{P}(i),i}$ are equal to the weights used for generating the tree. In the *descendants* cost model the edge costs $c_{\mathcal{P}(i),i}$ are proportional to the size of the subtree $\mathcal{N}_i$, as larger subnetworks likely lead to higher peak loads and less available bandwidth on the links serving them.

The object demands w_i^o follow Zipf's law. For the ranking of the object demands at the nodes we consider two models. In the case of *homogeneous demands*, the object demands have the same rank at all nodes. In the case of *heterogeneous demands*, every demand w_i^o for object o at node i is ranked as in the case of *homogeneous demands* with 0.5 probability. With 0.5 probability, the rank of w_i^o is picked uniformly at random. The results shown are the averages of 500 simulations, and the error bars show 95% confidence intervals.

As a baseline for comparison, we use a selfish distributed algorithm called *Distributed Local-Greedy* (DLG), which is based on global information about the object demands and placements at every node of the network. Following the *DLG* algorithm, starting from a randomly chosen allocation, at iteration k node i_k optimizes its placement of objects $\mathcal{A}_{i_k}(k)$ so as to minimize the cost for serving the requests from the local cell site, given the placement of objects $\mathcal{A}_{-i_k}(k)$ at the other nodes in the network [13], [14], [15]. As there is no guarantee that the *DLG* algorithm terminates [15], we run it for $|\mathcal{N}|$ iterations and we set $i_k = k$. Note that although *DLG* is seemingly similar to *DFG*, *DFG* minimizes the global cost based on global information, while *DLG* minimizes the local cost based on global information, hence it is algorithmically simpler.

A. Performance of distributed algorithms

In order to compare the performance of the proposed algorithms, as well as to evaluate the tightness of the analytical results, we computed the optimal placement $\bar{\mathcal{A}}$ and the cost-approximation ratio $C(\mathcal{A})/C(\bar{\mathcal{A}})$ for each algorithm. To make the computation of the optimal placement feasible, we considered a relatively small scenario with $|\mathcal{N}| = 20$, $|\mathcal{O}| = 100$ and $K_i = 2$ for all $i \in \mathcal{N}$. Figure 5 shows the cost-approximation ratio as a function of the Zipf exponent of the object demand distribution for *LGS*, *DLG*, *DFG* and for the *h-Push Down* algorithm with global information, i.e., for $h = \max_{i \in \mathcal{N}} l(i)$, for the *descendants* cost model.

The most salient feature of the figure is that the approximation ratio of the *LGS* algorithm increases exponentially with the Zipf exponent at a fairly high rate. The reason for the poor performance in the case of homogeneous demands is

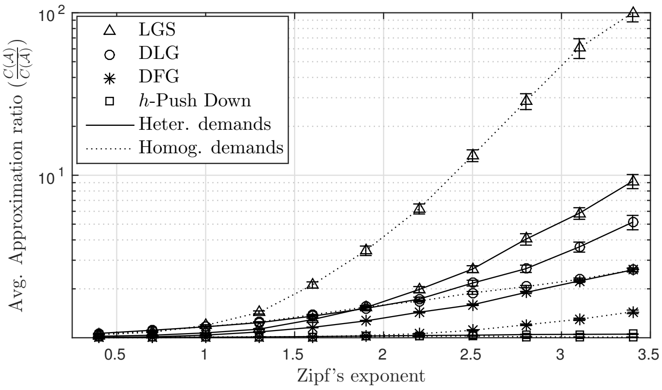

Fig. 5: Average approximation ratio vs. Zipf exponent for the *LGS*, *DLG*, *DFG*, and *h-Push Down* algorithms. *Heterogeneous* and *homogeneous demands*, $|\mathcal{O}| = 100$, $|\mathcal{N}| = 20$, $K_i = 2$.

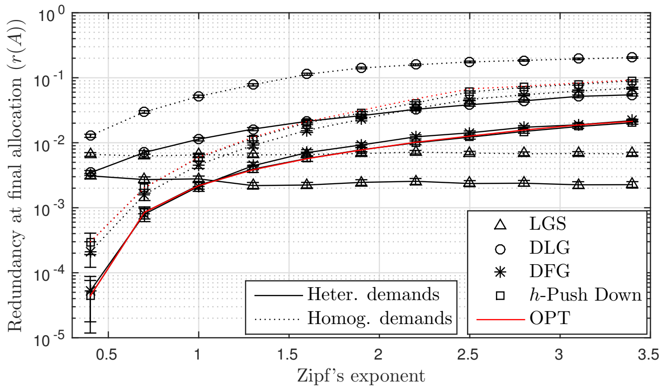

Fig. 6: Redundancy $r(\mathcal{A})$ vs. Zipf exponent for *LGS*, *DLG*, *DFG*, and *h-Push Down* and for the optimal placement. *Heterogeneous* and *homogeneous demands*, $|\mathcal{O}| = 100$, $|\mathcal{N}| = 20$, $K_i = 2$.

that the *LGS* algorithm populates the set $S(\mathcal{A}(k))$ exclusively based on the rankings of the object demands and not based on their values. As the Zipf exponent increases, the demand of the most popular content increases and the optimal solution might differ significantly from the allocation reached by the *LGS* algorithm. In order to validate this hypothesis, we computed the redundancy of a placement $\mathcal{A}$ using the index

$$r(\mathcal{A}) = \frac{\sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N} \setminus \{i\}} \left(1 - \frac{\min(K_i, K_j) - |\mathcal{A}_i \cap \mathcal{A}_j|}{\min(K_i, K_j)}\right)}{|\mathcal{N}|(|\mathcal{N}| - 1)}. \quad (6)$$

Intuitively, $r(\mathcal{A})$ is the average ratio of objects common between all pairs of placements $\mathcal{A}_i$ and $\mathcal{A}_j$. In Figure 6 we plot the average $r(\mathcal{A})$ index of the final placements reached by the algorithms, for the same scenario as Figure 5. The figure confirms that as the Zipf exponent increases, the *LGS* algorithm fails to introduce redundancy, which explains its poor performance.

Comparing the performance of *h-Push Down* to that of *DFG* we observe that *h-Push Down* (with global information) performs better than *DFG*, which is also reflected by the redundancy index, which is very close to the optimal (cf. Fig. 5). Finally, it is noteworthy that the *DLG* algorithm, which corresponds to selfish local optimization, fails to achieve performance close to the optimal, despite the availability of global information.

In order to evaluate the performance of the algorithms for larger scenarios, in the following we use the *DLG* algorithm as a baseline for comparison, as it is prohibitive to compute the optimal placement. Recall that the *DLG* algorithm optimizes the placement of objects in order to minimize the local cost, which would make it a reasonable simple choice in absence of more elaborate distributed algorithms.

To capture the performance of the algorithms relative to *DLG* we define the performance gain of an algorithm as the ratio between the cost of the placement reached by the *DLG* algorithm and the cost of the placement reached by the algorithm. It follows from (1) that the performance gain is also a measure of the increased hit rate achieved by the algorithm relative to *DLG*. Figure 7 shows the performance gain for the *LGS*, *DFG* and *h-Push Down* (for two values of the information horizon h) algorithms, as a function of the number of nodes for $K_i = 20$. The results are shown for *heterogeneous demands* using a Zipf exponent of 1, for the two cost models. We observe that the performance gain for the *DFG* and the *h-Push Down* algorithms increases with the number of nodes. Furthermore, the figure shows that *h-Push Down* outperforms *DFG* (i.e., it is close to optimal) for both values of the horizon h. The figure also shows that *LGS* performs just slightly better than *DLG*, with a decreasing gain as the network size increases.

Figure 8 shows the number of iterations needed to compute the final object placement corresponding to the results shown in Figure 7. Recall that the *DFG* algorithm starts with an empty allocation and terminates in $\sum_{i \in \mathcal{N}} K_i$ iterations, and can thus be used a baseline in terms of convergence. The results show that *LGS* performs worst, while *h-Push Down* for $h = 4$ requires almost an order of magnitude less iterations to terminate than *DFG*.

Figure 9 shows the performance gain as a function of the cache sizes for $|\mathcal{N}| = 50$. The figure shows that for higher cache sizes the performance gain of the *DFG* and *h-Push Down* algorithms over the *DLG* algorithm increases faster than exponentially. In the case of global information, the *h-Push Down* algorithm outperforms the *DFG* algorithm, while in the case of non-global information, i.e., for $h = 4$, it achieves performance close to the *DFG* algorithm. Furthermore, the performance gap between the *h-Push Down* algorithm with global and non-global information increases for higher cache sizes. The figure also confirms that the *LGS* and *DLG* algorithms achieve a comparable total cost.

B. Impact of the information horizon (h)

Finally, we evaluate the impact of the information horizon h on the performance of h-Push Down. We define the performance gain $\mathrm{PG}^h(\mathcal{A})$ for horizon h as the ratio between the cost of the placement $\mathcal{A}^1$ reached by the h-Push Down algorithm with $h = 1$ and the cost of the placement $\mathcal{A}^h$ reached with horizon h, i.e. $\mathrm{PG}^h(\mathcal{A}) = \frac{C(\mathcal{A}^1)}{C(\mathcal{A}^h)}$.

Figures 10 and 11 show the performance gain $\mathrm{PG}^h(\mathcal{A})$ and the number of iterations, respectively, for the *h-Push*

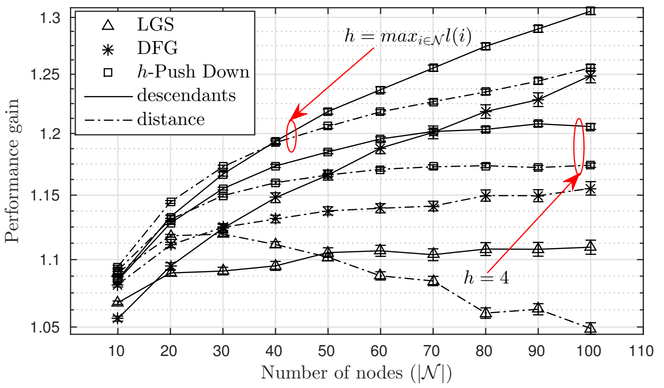

Fig. 7: Performance gain vs number of nodes $|\mathcal{N}|$ for the *h-Push Down*, *LGS* and *DFG* algorithms on the Manhattan graph with *descendants* and *distance* cost model, $|\mathcal{O}| = 5000$, $K_i = 20$.

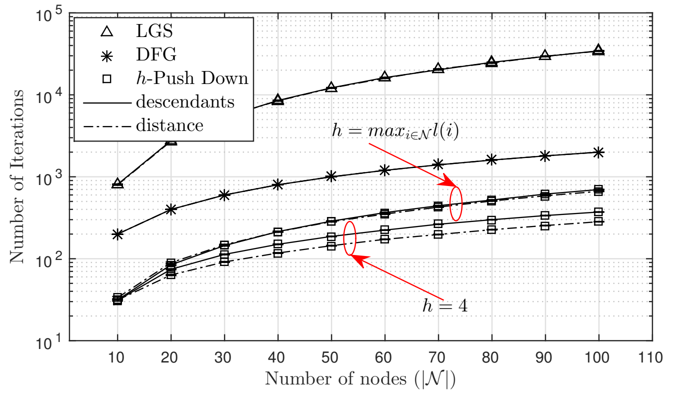

Fig. 8: Number of iterations vs number of nodes $|\mathcal{N}|$ for the *h-Push Down*, *LGS* and *DFG* algorithms on the Manhattan graph with *descendants* and *distance* cost model, $|\mathcal{O}| = 5000$, $K_i = 20$.

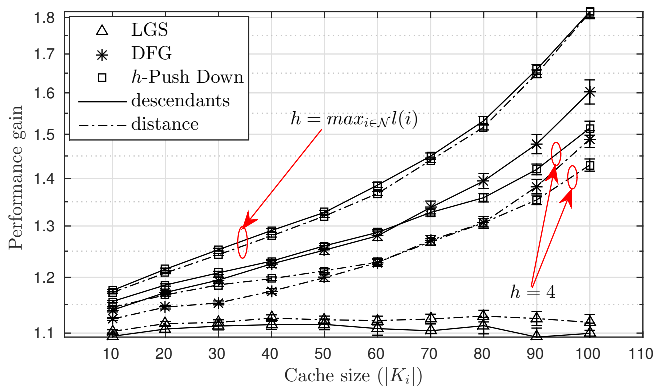

Fig. 9: Performance gain vs. cache size K_i for *h-Push Down*, *LGS* and *DFG* on the Manhattan graph with *descendants* and *distance* cost model. Results for $|\mathcal{O}| = 5000$, $|\mathcal{N}| = 50$.

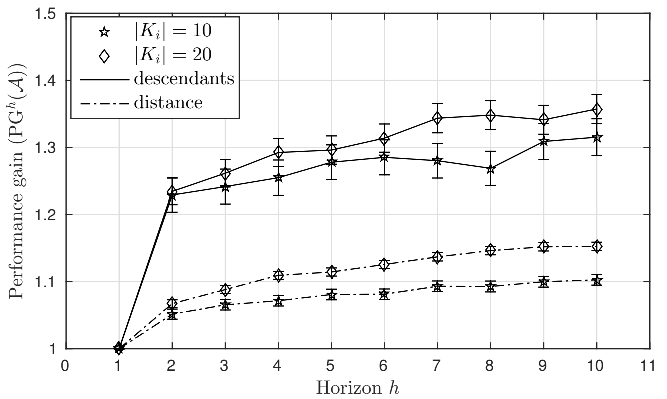

Fig. 10: Performance gain vs. horizon h for cache sizes $K_i \in \{10, 20\}$ on the Manhattan graph with *descendants* and *distance* cost models. Results for $|\mathcal{O}| = 5000$ and $|\mathcal{N}| = 100$.

Down algorithm as a function of the information horizon h for $|\mathcal{N}| = 100$ and two different cache sizes K_i. We plot the performance gain $\mathrm{PG}^h(\mathcal{A})$ for the same cost and object demands models as in Figures 7 and 9. We observe that the performance gain increases with a decreasing marginal gain in h, making the algorithm perform fairly well with limited available information (low h). Furthermore, the same observation holds for the convergence time, hence a moderate value of h provides a good trade-off between performance and convergence time. Figure 10 also shows that as the horizon h increases, the performance gain increases more in the case of the *descendants* cost model than in the case of the *distance* cost model. The reason is that as the horizon h increases, the nodes have access to the cost of edges between nodes at lower levels of the tree (i.e., closer to the root), which in the case of the *descendants* cost model are the edges with highest cost, and thus they have a higher impact on the total cost.

VI. Related Work

Closest to ours are recent works on content placement in networks [16], [5], [7]. The authors in [16] provide an algorithm for computing the optimal placement in a hierarchical network by reducing the content placement problem to a minimum-cost flow problem. Motivated by the computational complexity of the problem, they design a distributed amortizing algorithm that achieves a constant factor approximation. The model considered in [16] is based on the ultrametric cost model introduced in [6], which differs from our model on the assumption of symmetric costs between nodes. The authors in [5] give insights in the structure of the optimal placement in a regular two level hierarchical network, and they develop a greedy distributed 2-approximation algorithm. The authors in [11] consider a hybrid network with in-network caching and they propose a $(1 - 1/e)$-approximation greedy algorithm. A more generic cost model was considered in [7], where the authors develop a 10-approximation algorithm by rounding the optimal solution of the LP-relaxation of the problem. [17] proposed a set of centralized, polynomial time algorithms with approximation guarantees, for the joint problem of request routing and content replication under strict bandwidth constraints at the storage sites. In contrast to [16], [5], [11], [7], [17], in our work we developed two distributed algorithms for computing a content placement based on limited information on the content demands and on the network topology, that can be used to solve large problem instances with prohibitive space complexity.

Related to ours are recent works on game theoretical analyses of distributed selfish replication on graphs [13], [18], [19], [20], [21], [14], [15], as they can serve as a basis for distributed

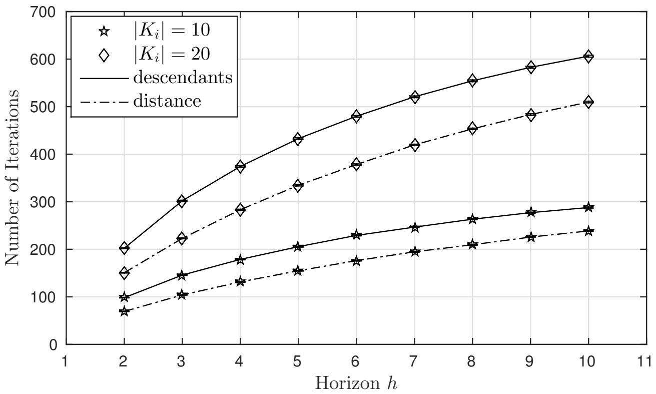

Fig. 11: Number of iterations vs horizon h for two values of cache sizes $K_i \in \{10, 20\}$ on the Manhattan graph with *descendants* and *distance* cost models. Results for $|\mathcal{O}| = 5000$ and $|\mathcal{N}| = 100$.

content placement algorithms. Equilibrium existence when the access costs are homogeneous and nodes form a complete graph were provided in [13], and results on the approximation ratio (referred to as the price of anarchy) were provided in [18], [19] for homogeneous costs and a complete graph. Non-complete graphs were considered in [20], [21], [14], and results on the approximation ratio of a distributed greedy algorithm were given for the case of unit storage capacity and an infinite number of objects in [20]. [21] considered a variant of the problem where nodes can replicate a fraction of objects, and showed the existence of equilibria, while convergence results were provided for the integer problem in [14] in the case of homogeneous neighbor costs. The case of heterogeneous neighbor costs, for which the non-convergence of distributed greedy replication was shown in [15] is a generalization of our model, and thus the negative result provided in [15] may not apply to our case. Different from these works, in this paper we consider caches managed by a single entity, and thus we consider the minimization of the total cost as opposed to the selfish minimization of the cost of the individual nodes. Our objective of minimizing the total cost also sets this work apart from recent work on cache networks in the context of content centric networks, e.g., [22].

VII. Conclusion

We considered the problem of minimizing the bandwidth demand in a mobile backhaul through cooperative caching, and formulated it as a 0-1 integer linear program. We proposed a 2-approximation distributed algorithm that is based on global information. Furthermore, we proposed a low complexity distributed algorithm based on information about object demands at descendants, and an algorithm with an adjustable level of available information. We proved convergence and stability of the algorithms. We used extensive simulations to evaluate the performance of the proposed algorithms. Our results show that information about object demands at descendants is insufficient for good cooperative caching performance, but the proposed h-Push Down algorithm achieves consistently good performance despite limited information availability, consistently better than greedy optimization based on global information.

References

[1] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino, "Scalable Mobile Backhauling via Information-Centric Networking," in *Proc. of IEEE International Workshop on Local and Metropolitan Area Networks (LANMAN)*, 2015, pp. 1–6.

[2] U. Paul, A. P. Subramanian, M. M. Buddhikot, and S. R. Das, "Understanding traffic dynamics in cellular data networks," in *Proc. of IEEE INFOCOM*, 2011, pp. 882–890.

[3] H. Pinto, J. M. Almeida, and M. A. Gonçalves, "Using early view patterns to predict the popularity of youtube videos," in *Proc. of ACM Intl. Conf. on Web Search and Data Mining (WSDM)*, 2013, pp. 365–374.

[4] A. Tatar, M. D. de Amorim, S. Fdida, and P. Antoniadis, "A survey on predicting the popularity of web content," *Journal of Internet Services and Applications*, vol. 5, no. 1, p. 8, 2014.

[5] S. Borst, V. Gupta, and A. Walid, "Distributed Caching Algorithms for Content Distribution Networks," in *Proc. of IEEE INFOCOM*, 2010, pp. 1478–1486.

[6] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web," *Proc. of ACM Symposium on Theory of Computing (STOC)*, pp. 654–663, 1997.

[7] I. D. Baev and R. Rajaraman, "Approximation algorithms for data placement in arbitrary networks," in *Proc. of ACM SODA*, 2001.

[8] E. J. Rosensweig, J. Kurose, and D. Towsley, "Approximate Models for General Cache Networks," in *Proc. of IEEE INFOCOM*, 2010, pp. 1–9.

[9] C. Fricker, P. Robert, and J. Roberts, "A versatile and accurate approximation for LRU cache performance," in *Proc. of the 24th International Teletraffic Congress (ITC)*, 2012, pp. 1–8.

[10] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey, "An Analysis of Approximations for Maximizing Submodular Set Functions - II," *Mathematical Programming Study*, vol. 8, pp. 73–78, 1978.

[11] M. Dehghan, A. Seetharam, B. Jiang, T. He, T. Salonidis, J. Kurose, D. Towsley, and R. Sitaraman, "On the Complexity of Optimal Routing and Content Caching in Heterogeneous Networks," in *Proc. of IEEE INFOCOM*, 2015, pp. 936–944.

[12] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. 48–48, 1956.

[13] N. Laoutaris, O. Telelis, V. Zissimopoulos, and I. Stavrakakis, "Distributed selfish replication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 12, pp. 1401–1413, 2006.

[14] V. Pacifici and G. Dán, "Convergence in Player-Specific Graphical Resource Allocation Games," *IEEE J. Sel. Areas Commun.*, vol. 30, no. 11, pp. 2190–2199, 2012.

[15] ——, "Distributed Algorithms for Content Allocation in Interconnected Content Distribution Networks," in *Proc. of IEEE INFOCOM*, 2015, pp. 2362–2370.

[16] M. R. Korupolu and M. Dahlin, "Coordinated placement and replacement for large-scale distributed caches," *IEEE Trans. Knowl. Data Eng.*, vol. 14, no. 6, pp. 1317–1329, 2002.

[17] K. Poularakis, G. Iosifidis, and L. Tassiulas, "Approximation Algorithms for Mobile Data Caching in Small Cell Networks," *IEEE Transactions on Communications*, vol. 62, no. 10, pp. 3665–3677, 2014.

[18] G. Pollatos, O. Telelis, and V. Zissimopoulos, "On the social cost of distributed selfish content replication," in *Proc. of IFIP Networking*, 2008, pp. 195–206.

[19] E. Jaho, M. Karaliopoulos, and I. Stavrakakis, "Social similarity favors cooperation: the distributed content replication case," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 3, pp. 601–613, 2013.

[20] B.-G. Chun, K. Chaudhuri, H. Wee, M. Barreno, C. H. Papadimitriou, and J. Kubiatowicz, "Selfish caching in distributed systems: a game-theoretic analysis," in *Proc. of ACM symposium on Principles of Distributed Computing (PODC)*, 2004, pp. 21–30.

[21] G. Dán, "Cache-to-cache: Could ISPs cooperate to decrease peer-to-peer content distribution costs?" *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 9, pp. 1469–1482, 2011.

[22] N. Laoutaris, H. Che, and I. Stavrakakis, "The LCD Interconnection of LRU Caches and its Analysis," *Performance Evaluation*, vol. 63, no. 7, pp. 609–634, 2006.