Greedy Caching: A Latency-aware Caching Strategy for Information-centric Networks

[1]Bitan Banerjee, [2]Anand Seetharam, [1]Chintha Tellambura
[1]Department of Electrical and Computer Engineering, University of Alberta, Canada
[2]Computer Science Department, SUNY Binghamton, USA
bitan@ualberta.ca, aseethar@binghamton.edu, chintha@ece.ualberta.ca

***Abstract*—Most caching strategies in information-centric networks (ICN) primarily focus on pushing popular content to the network edge. As such, these approaches make limited use of caches in the network core, reduce cache utilization due to content duplication and provide limited performance improvement. In this paper, we propose *Greedy Caching* to determine the set of content to be cached at each network node. *Greedy Caching* starts by caching the most popular content, calculated based on the total incoming request stream from users, at the network edge. The algorithm then recalculates the relative popularity of each piece of content based on the request miss stream from downstream nodes to determine the set of content to be cached in the network core. We perform exhaustive simulation in the Icarus simulator [1] using realistic Internet topologies (e.g., GARR, GEANT, and WIDE) and demonstrate that *Greedy Caching* provides significant improvement in content download delay (referred to as latency) over state-of-the-art routing and caching strategies for ICN for a wide range of simulation parameters. Simulation results suggest an improvement of 5-28% in latency and 15-50% improvement in hit rate over state-of-the-art policies.**

***Index Terms*—Greedy algorithm, caching, information-centric networking, latency, hit rate, hop count.**

I. Introduction

The explosive increase in content in recent years has lead to the proposal of a new Internet architecture called information-centric networking (ICN) which aims to evolve the current Internet from a host-centric model to a content-centric one. By caching content at storage-enabled network nodes, requests for content can be served from the content custodians (origin servers), as well as from intermediate caches. With the primary emphasis being content, if a cache enroute to the custodian has the requested content, the content will be returned to the user from the cache itself, thereby improving user performance. Serving a request from an intermediate cache has several benefits such as reduced content download delay, increased throughput and decreased network congestion.

Video delivery companies (e.g., YouTube , Netflix) already use simple forms of popularity based in-network caching in today's Internet to improve user performance. These video delivery applications primarily determine the popularity of multimedia content based on parameters such as release date, viewership of past series of a show and push popular content to the network edge [2], [3]. In recent years, caching policies proposed for ICN have also identified that caching popular content within the network is essential to improve the performance [4]. However, existing policies focus mainly on the network edge and fail to effectively leverage caches in the network core [5]. In ICN, deployment of network-wide caches is likely to be expensive. Therefore, it is important to design efficient caching and routing policies that maximize cache utilization, both at the network edge and in the network core and minimize unnecessary content duplication. While it is tempting to think that determining what content to cache at a node only requires local information, content cached at downstream nodes drastically impacts the request stream seen by upstream caches and may ultimately reduce network-wide cache utilization. We use the words cache and node interchangeably in this paper.

To this end, in this paper, we propose *Greedy Caching*, a simple caching policy that determines the optimized set of content to be cached at each network node based on the relative content popularity, with the goal of reducing content download delay (referred to as latency). *Greedy Caching* estimates the relative content popularity at each node based on the request stream from directly connected users as well as the request miss stream from downstream nodes and then uses a greedy algorithm to determine the content to be cached. The difficulty of the problem stems from the fact that different pairs of network nodes can forward requests to one another, resulting in interdependencies, and cycles in the underlying graph, thereby making it difficult to estimate the relative content popularity.

The main contributions of this paper are given below.

- We assume that the network has an underlying routing policy for forwarding requests for content towards the custodian. We propose *Greedy Caching*, a caching policy that greedily caches the most popular content at each cache based on their relative content popularity. To estimate relative content popularity, *Greedy Caching* first leverages routes provided by the routing algorithm to create a directed acyclic graph (DAG). For the single custodian case, DAG construction is relatively straightforward. However, for the multiple custodian case, simply combining the routes provided by the routing algorithm results in a cyclical graph, due to node pairs sending traffic to one another. *Greedy Caching* therefore uses the feedback arc set algorithm to prune this cyclical graph and construct a DAG. *Greedy Caching* then combines the request stream from users with the constructed DAG to determine the set of content to be cached at each network

ISBN 978-3-901882-94-4 © 2017 IFIP

node, starting from the network edge and ending at the custodians.

- We perform extensive simulations in Icarus [1], a simulator built exclusively for implementing and testing new ICN routing and caching policies to demonstrate the efficacy of *Greedy Caching*. We compare the performance of *Greedy Caching* against state-of-the-art caching and routing policies, Leave Copy Everywhere (LCE) [6], Leave Copy Down (LCD) [7], Cache Less for More (CL4M) [8], ProbCache [9], and Random Caching (Random) [10] on real world internet topologies (e.g., GARR, GEANT, and WIDE) [11]. We study the impact of various simulation parameters (e.g., cache size, content-universe, content popularity skewness) on the performance of *Greedy Caching* and demonstrate that it provides approximately 5-28% improvement in latency and 15-50% improvement in hit rate over state-of-the-art strategies.

The rest of the paper is organized as follows. We discuss related work in Section II. We describe the problem and the proposed *Greedy Caching* algorithm in Section III. Experimental results are presented in Section IV and we conclude the paper in Section V.

II. Related Work

Caching strategies have been proposed in literature, both in the context of Content Delivery Networks (CDN) as well as ICN. In this section, we mainly focus on existing literature in ICN and demonstrate how *Greedy Caching* differs from prior work. Some of the most widely accepted caching policies are LCD [7], CL4M [8], and ProbCache . All these caching policies aim to reduce cache redundancy by caching content based on parameters such as content popularity, connectivity of nodes. A modified version of LCD with chunk caching and searching (CLS) is proposed in [12], where a piece of content is cached one level downstream or upstream depending on whether a request is a cache hit or a cache miss. Similarly, a modified version of ProbCache, namely ProbCache+ [13] incorporates a new variable called cache weight to enforce fairness between content.

PopCache [14] primarily uses content popularity to determine whether to cache a particular content or not. Authors in [15] propose a caching strategy ProbPD, where the dynamic popularity of a content determines its caching probability. This dynamic popularity is calculated by incorporating the distance of a cache from a user, and incoming content request for a certain time interval. In MPC [16], authors dynamically calculate content popularity locally at each cache by maintaining a popularity table. Topology dependent caching strategies have also been proposed in literature. Authors in [17] develop a caching strategy called Progressive Caching Policy (PCP), where content is cached at the node one hop downstream of the serving node and another intermediate node that has number of incoming links greater than a threshold. Wang *et al.* propose CRCache [18], a caching strategy based on correlation between content popularity and network topology information. Hop-based Probabilistic Caching (HPC) [19] probabilistically caches content depending on the distance between the user and the cache.

Badov *et al.* propose a caching strategy to avoid congested links by caching content at the edge of congested link. A cooperative caching strategy, where off-path caching is explored by controlling the routing algorithm is proposed in [20]. Authors in [21] develop a hash function based joint routing and caching strategy, that helps i) caches decide whether or not to cache a particular content and, ii) routers route requests to relevant caches. Similarly, authors in [22] propose CPHR, a collaborative caching strategy which also uses hash functions. Each content is partitioned according to the hash function and these partitions are then assigned to network caches. Hash function based strategies generally require centralized control which results in high overhead. To overcome the shortfalls of centralized control, distributed cache management (DCM) [23] was proposed to improve cache utilization by sharing holistic information about request patterns and cache configuration. In our earlier work, we proposed a routing algorithm that leverages characteristic time information to forward content requests [24].

In contrast to existing literature, we propose a caching policy that adopts a locally optimal approach at each node to determine the set of content to be cached at each network node. *Greedy Caching* caches content at each node based on relative content popularity, which is calculated based on the request miss stream from downstream nodes. This approach not only maximizes hit rate at each network node, but it also increases cache utilization by reducing content duplication. We note that greedy algorithms for replica management in CDNs have been proposed in literature [25]–[27]. However, these papers assume that the incoming request rate at each cache is available, and thereafter, develop a greedy algorithm subject to several parameters such as cache size, distance from users, and access cost and do not consider the request miss streams from downstream nodes.

III. Greedy Caching Algorithm

A. Network Model

Let us consider an ICN which is represented by an undirected graph $\mathcal{G}(V, E)$, where V consists of all the nodes in the network including the users, caches and custodians and E consists of the set of interconnected links. We consider $\mathbb{U} = \{U_1, U_2, ...U_N\}$, $\mathbb{R} = \{R_1, R_2, ...R_M\}$ and $\mathbb{C} = \{C_1, C_2, ...C_L\}$ to denote the set of users, caches and custodians respectively. Therefore, the network comprises of N users, M caches and L custodians. We assume that each cache has the same amount of finite storage, $\mathcal{C}$.

We assume that content universe $F = \{f_1, f_2, ..., f_K\}$ is uniformly distributed among the custodians. Each piece of content is available at only one custodian and is permanently stored there. We assume content popularity follows a certain probability distribution (e.g., Zipf). We assume that user U_i generates request at rate $\Lambda_i = \{\lambda_{i1}, \lambda_{i2}, ...\lambda_{iK}\}$. We assume that these requests are forwarded towards the custodian depending on the underlying routing strategy (e.g., Dijkstra's shortest path routing). We abuse notation and let Λ_i also denote the outgoing request rate at any intermediate network node i (apart from the users). The incoming request rate at node i is denoted by Λ'_i. Note that Λ_i and Λ'_i can differ due to caching at node i.

Let $\mathcal{P}_{ij}(V_{ij}, E_{ij})$ denote the shortest path from U_i to the C_j with V_{ij} denoting the set of nodes on that path and E_{ij} denoting the set of directed edges tracing the path from U_i to the C_j. Additionally, for all edges $e_{ij} \in E_{ij}$ connecting nodes i and j, an indicator variable, I^k_{ij} is set to 1 if e_{ij} lies on the shortest path for content k, otherwise it is set to 0. We assume that the shortest path algorithm returns $\mathcal{P}_{ij}(V_{ij}, E_{ij})$ and also sets I^k_{ij}. Note that each request can traverses multiple caches enroute to the custodian. If a cache enroute to the custodian has the requested content it serves the content, otherwise the content is served by the custodian.

B. Motivating Example

Our goal in this paper is to propose a caching policy that decreases latency. We attempt to maximize the hit rate at each network node and eliminate unnecessary content duplication. We propose *Greedy Caching*, an optimized caching policy for ICN, that first estimates the relative content popularity at each node based on the request miss stream from downstream nodes. *Greedy Caching* then employs a simple greedy algorithm that caches the most popular content at each node based on the relative content popularity at that node.

To motivate the need for *Greedy Caching* and to illustrate the importance of relative popularity, let us consider a simple example. We consider a network of 3 users (U_1, U_2, U_3), 2 caches (R_1, R_2), and one custodian (C_1) as shown in Figure 1(a). Let us assume that the delay on each link is 1 second. We consider that there are only two unique pieces of content A and B, with probability of requesting content A and B being 0.6 and 0.4 respectively. We assume that R_1 and R_2 can cache only one piece of content and all users generate requests at same rate λ.

If content is cached based on absolute popularity, then this will result in content A being cached at both R_1 and R_2. At first glance, this appears to be a good idea, but if one considers the miss request stream from R_1 to R_2 (which is 0.8λ for content B), it is easy to understand that it is better to cache content A at R_1 and content B at R_2. In fact, for this simple network this is the optimal caching policy. Caching content A at R_1 and content B at R_2 decreases overall content download delay to 1.47 seconds in comparison to 1.67 seconds when content A is cached at both R_1 and R_2. Details of the *Greedy Caching* algorithm, which leverages this concept of relative content popularity are discussed next.

C. Greedy Caching

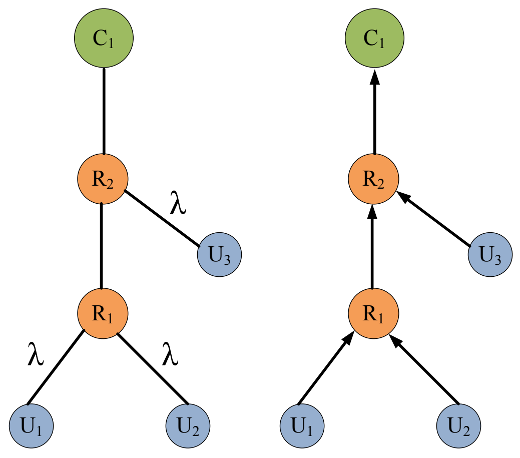
From our discussion in the previous subsection, it is evident that estimating the relative content popularity lies at the heart of *Greedy Caching*. At the highest level, the *Greedy Caching* algorithm starts by caching the most popular content at the network edge and then iteratively determines the content to be cached at the nodes in the network core by estimating the relative popularity. This iterative process stops when all network nodes have been visited. We first discuss *Greedy Caching* for the relatively simple scenario of an ICN with a single custodian and then move on to the more challenging multiple custodian case. Estimating the relative content popularity, especially for the multiple custodian case is non-trivial because of the interdependencies arising from pairs of network nodes forwarding requests to one another.

1) Single Custodian: To determine the relative content popularity with respect to a cache, *Greedy Caching* first combines the routes provided by the underlying shortest path routing algorithm for all users to generate a directed acyclic graph (DAG) $\Psi(V', E')$, where V' and E' are the number of vertices and edges in the DAG respectively. As there is only a single custodian in the network, it is easy to observe that combining these paths will result in a DAG. This is because if a node (say R_1) forwards requests through a node (say R_2) towards the custodian, R_2 lies on the shortest path from R_1 to the custodian. Therefore, R_2 cannot route the requests it receives through R_1. For each node i in Ψ, let $\mathbb{N}'_i$ denote the set of neighbors from which there is an incoming edge to i.

Greedy Caching then performs a topological sort on $\Psi(V', E')$ to determine Θ, an ordering of the vertices in Ψ. Let Θ_i denote the i^{th} vertex in Θ. For a DAG, topological sort provides a linear ordering of the vertices such that for every directed edge from vertex u to vertex v, u comes before v in the ordering. It is evident that the the users and the custodian will be first and last nodes in this topological ordering. *Greedy Caching* then visits the nodes in order.

Greedy Caching then caches the set of content with the highest incoming request rate (i.e., the content with the highest relative popularity). Note that nodes at the network edge will only have incoming edges from the users and thus will be the first group of nodes visited by the algorithm. Therefore, the *Greedy Caching* algorithm will cache the $\mathcal{C}$ most popular content at each edge node. Now, these nodes at the network edge will only forward requests for uncached content along their outgoing edges as determined by the routing algorithm. As a result, any node v, which appears in the topological ordering after the edge nodes will take into account the request stream from directly connected users and the request miss stream from nodes that appear earlier than it in the ordering to calculate the relative popularity. Node v will thus cache the C most popular content based on the calculated relative content popularity. Details of the *Greedy Caching* algorithm for a single custodian are provided in Algorithm 1.

Let us now revisit Figure 1(a) and see how *Greedy Caching* ends up caching content A at R_1 and content B at R_2. For this network, the DAG obtained by combining the shortest paths will be similar to the network itself and is given in Figure 1(b). The topological sort is given by $U_1, U_2, U_3, R_1, R_2, C_1$. Therefore, the algorithm visits R_1 first and caches A. Accounting for the miss stream from R_1 to R_2 and the request stream from U_3, it is easy to see that *Greedy Caching* will cache content B at R_2.



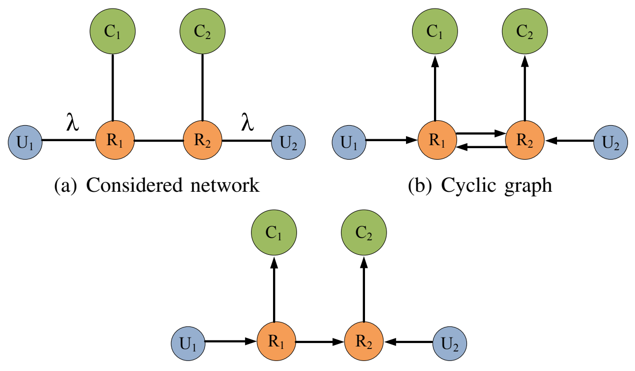
(a) Considered network (b) Resulting DAG

Fig. 1: Greedy Caching illustration for single custodian

Algorithm 1 Greedy caching for single custodian

```
1:  Input network $\mathcal{G}(V, E)$
2:  for $U_i \in \mathbb{U}$ do
3:      $\mathcal{P}_{i1}(V_{i1}, E_{i1})$ = ShortestPath$(U_i, \mathbb{C}_1)$
4:  end for
5:  $\Psi(V', E') = \bigcup_{i=1}^{N} \mathcal{P}_{i1}(V_{i1}, E_{i1})$
6:  $\Theta$ = TopologicalSort $\Psi(V', E')$
7:  procedure GREEDY CACHING($\Theta$, $\mathbb{R}$)
8:      for $i = 1$, $i \leq |V'|$, $i++$ do
9:          $r = \Theta_i$
10:         if $r \in \mathbb{R}$ then
11:             for each content $k$ do
12:                 $\lambda'_{rk} = \sum_{j \in \mathbb{N}'_r} I^k_{jr} \lambda_{jk}$
13:             end for
14:             $\Lambda'_{r_{sort}}$: Sort $\Lambda'_r$ in descending order
15:             Cache top $\mathcal{C}$ content in $\Lambda'_{r_{sort}}$
16:             for each content $k$ cached at $r$ do
17:                 $\lambda_{rk} = 0$
18:             end for
19:         end if
20:     end for
21: end procedure
```

2) Multiple Custodian: The multiple custodian scenario is more challenging, primarily due to the fact that simply combining the shortest paths from the users in the network may result in a cyclic graph ($\mathcal{G}'$), as shown in Algorithm 2. Let us consider Figure 2(a) to understand this. In this network, there are two users U_1 and U_2, two caches R_1 and R_2 and two custodians C_1 and C_2. Let us assume that one half of the content universe F_1 is available at C_1 and the other half F_2 is available at C_2. As is evident from the figure, R_1 with forward requests for content in F_2 from U_1 to R_2 and R_2 will forward requests for content in F_1 from U_2 to R_1. Combining the shortest paths will result in a cyclic graph as shown in Figure 2(b).



(a) Considered network (b) Cyclic graph

(c) Resulting DAG

Fig. 2: Greedy Caching illustration for multiple custodian

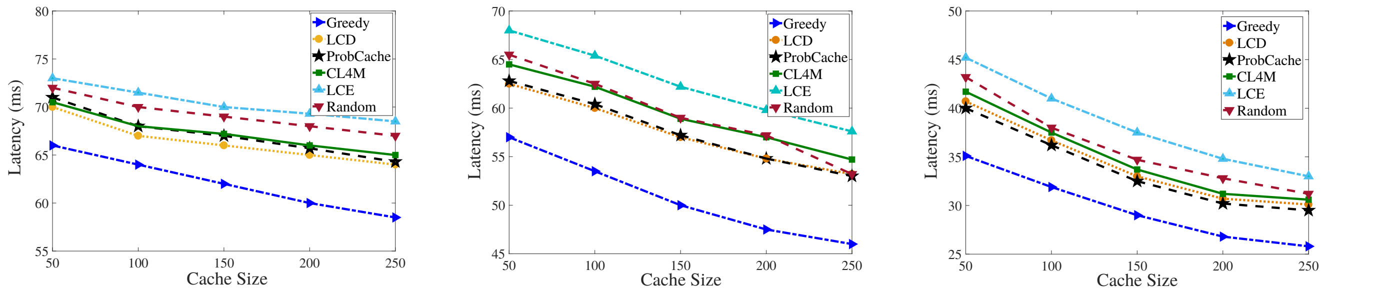
Greedy Caching attempts to eliminate this problem by leveraging the feedback arc set algorithm [28], [29] that provides the set of edges to be removed from the cyclic graph $\mathcal{G}'$ to create a DAG $\Psi(V', E')$. Therefore, applying the feedback arc set algorithm to Figure 2(b) will result in Figure 2(c). Note that in order to create a DAG from a cyclic graph, *Greedy Caching* leverages the feedback arc set algorithm that removes edges to eliminate cycles. This is clearly an approximation and thus *Greedy Caching* is a heuristic and can only provide an optimized solution. However, we will observe in Section IV, *Greedy Caching* outperforms state-of-the-art algorithms in real world settings. Figure 2(c) demonstrates DAG construction for the multiple custodian network given by Figure 2(a). Once the DAG is constructed, the relative content popularity at each node is determined in a manner similar to Algorithm 1.

Algorithm 2 Greedy caching for multiple custodian

```
1:  Input network $\mathcal{G}$
2:  for $U_i \in \mathbb{U}$ do
3:      for $C_j \in \mathbb{C}$ do
4:          $\mathcal{P}_{ij}(V_{ij}, E_{ij})$ = ShortestPath$(U_i, \mathbb{C}_j)$
5:      end for
6:  end for
7:  $\mathcal{G}' = \bigcup_{i=1}^{N} \bigcup_{j=1}^{L} \mathcal{P}_{ij}(V_{ij}, E_{ij})$
8:  Apply feedback arc set over $\mathcal{G}'$ to generate $\Psi(V', E')$
9:  $\Theta$ = TopologicalSort $\Psi(V', E')$
10: procedure GREEDY CACHING($\Theta$, $\mathbb{R}$)
11: end procedure
```

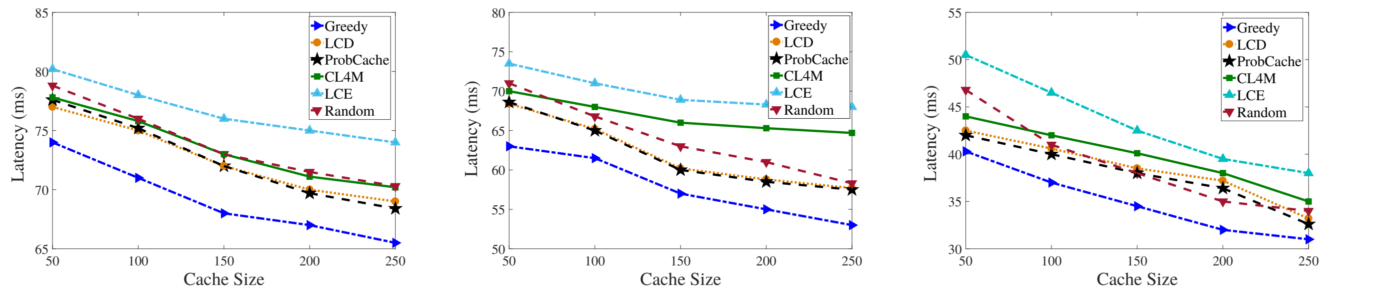
IV. Performance Evaluation

In this section, we first describe the experimental setup and then present simulation results. We compare the performance of *Greedy Caching* against state-of-the-art caching and routing policies, namely Leave Copy Everywhere (LCE), Leave Copy Down (LCD), Cache Less for More (CL4M), ProbCache, and Random Caching (Random). These strategies are explained below.



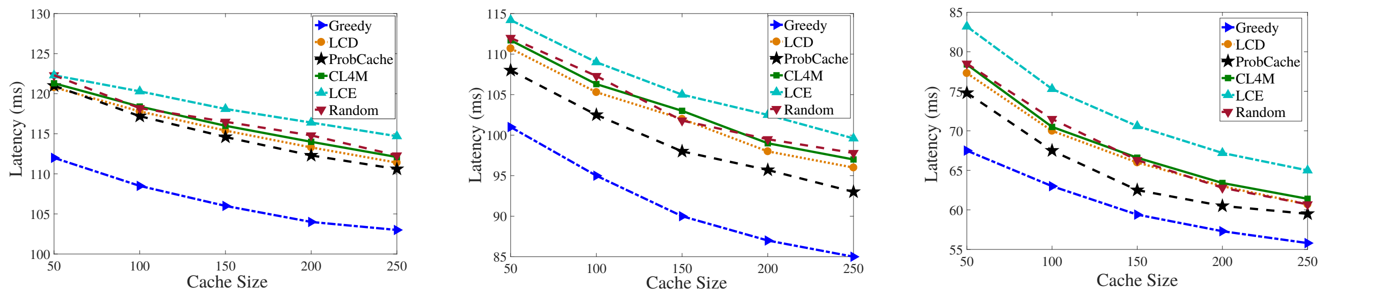
(a) $\alpha = 0.6$ (b) $\alpha = 0.8$ (c) $\alpha = 1.1$

Fig. 3: Latency performance of Greedy Caching for GARR with single custodian



(a) $\alpha = 0.6$ (b) $\alpha = 0.8$ (c) $\alpha = 1.1$

Fig. 4: Latency performance of Greedy Caching for GARR with 2 custodians

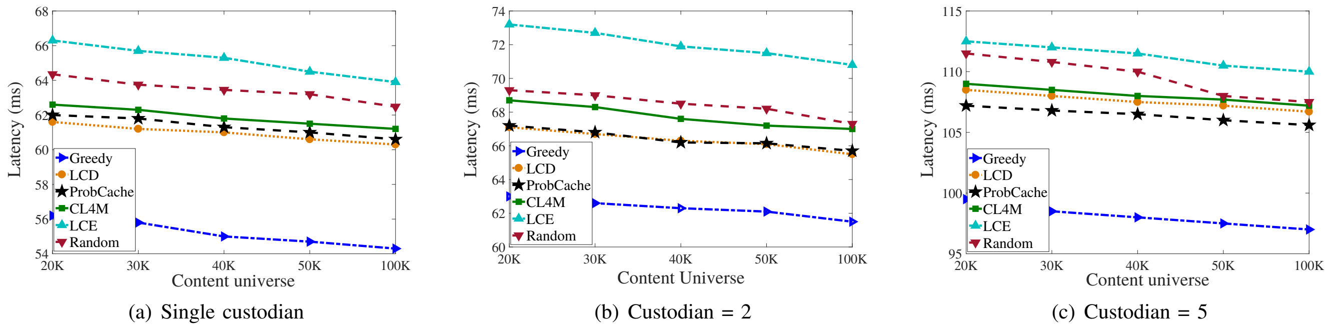


(a) $\alpha = 0.6$ (b) $\alpha = 0.8$ (c) $\alpha = 1.1$

Fig. 5: Latency performance of Greedy Caching for GARR with 5 custodians

- **LCE:** In this strawman approach, content is cached at every node along the path as it is being downloaded [6].
- **LCD:** In this policy whenever there is a cache hit, the content is replicated at the cache which is one hop downstream towards the requester [7].
- **CL4M:** This policy leverages the concept of betweenness centrality (i.e., the number of shortest paths traversing a cache) to make caching decisions [8]. This policy caches content at nodes with the greatest betweenness centrality, so as to maximize the probability of a cache hit.
- **ProbCache:** This policy reduces cache content redundancy [9] by probabilistically caching content at enroute caches.
- **Random:** In this caching strategy [10], content is cached at any one of the downstream node, and the node is selected randomly. Hence, the probability of caching content at any downstream node is inversely proportion to the path length. In the simulator, we use different seed values to select the node for caching.

We assume that the network uses Dijkstra's weighted shortest path routing to route content requests to custodians, where weights corresponds to the delay on links. If content is found at a cache enroute to a custodian, then it is served from that cache. We perform simulations on a discrete event based simulator Icarus [1], a simulator designed exclusively for ICN research. The simulator consists of four building blocks, scenario generation, experiment orchestration, experiment execution, and result collection. Each content request



(a) Single custodian (b) Custodian = 2 (c) Custodian = 5

Fig. 6: Latency performance for GARR topology for varying content universe

is considered as an event, and whenever an event occurs, a corresponding timestamp is stored. The result collection block gathers the results of the simulation. In Icarus, latency is calculated as the sum of delays on each link, traversed during content download.

We perform extensive experiments on various real world networks namely GARR (Italian computer network), GEANT (European academic network), and WIDE (Japanese academic network). GARR is the Italian national computer network for universities with 61 nodes and 89 edges. GEANT is an academic network spread around the world consisting of 40 nodes and 61 edges. The WIDE topology is the first network established in Japan and consists of 30 nodes and 33 edges. To avoid cluttering the paper with figures of similar nature, unless mentioned otherwise, all results shown in this paper are for the GARR topology.

In our simulations, prior to evaluation, all caches are warmed up. Except for large content universe, caches are always warmed up with 100000 requests and the subsequent 100000 requests are used for performance evaluation. We assume that the probability of requesting a content follows a Zipfian distribution with skewness parameter α. Nodes with the highest degree (i.e., number of connected links) are considered as custodians, and in case multiple nodes have the same degree, custodians are selected randomly among them. For the multiple custodian case, content is uniformly distributed between the custodians and each content is permanently stored at only one of the custodians. Nodes with degree of 1 are selected as the users. Unless otherwise mentioned, all results are generated for the following simulation configuration; content universe $F = 10000$, cache size of each node is varied between $50 - 250$, and content popularity skewness (α) is varied between $0.6 - 1.1$. Results in the figures are averaged over 5 runs of the experiment. We would like to mention that minimum feedback arc set algorithm is NP-complete. Hence, we consider an implementation of feedback arc set using the heuristic proposed in [29]. This implementation is suboptimal but effective from time complexity standpoint.

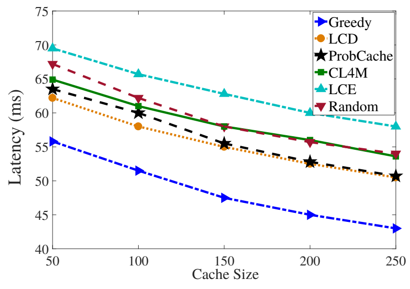
We consider various performance metrics such as latency, hit rate, and average hop count to demonstrate the superiority of *Greedy Caching*. In our results, performance improvement of using strategy A over strategy B is calculated by taking the percentage of the difference between the two strategies divided by the performance of strategy B. Our experiments demonstrate that *Greedy Caching* outperforms state-of-the-art strategies for a wide range of simulation parameters.

A. Discussion on latency performance

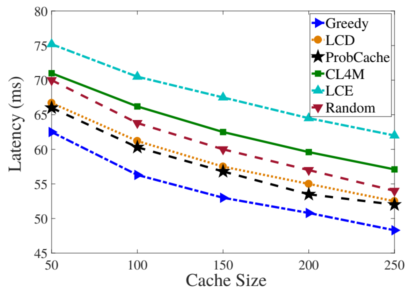
In this section, we discuss the latency performance of *Greedy Caching* for single custodian and multiple custodian cases for variety of different settings.

1) Performance for GARR: Figure 3 demonstrates the latency results for the different policies for GARR for the single custodian case while figures 4 and 5 show the latency results for the 2-custodian and 5-custodian scenarios for different α values $(0.6, 0.8, 1.1)$. We observe from the figures that *Greedy Caching* outperforms state-of-the-art strategies by 7-22 % for a wide range of cache sizes. The primary reason behind the superior performance of *Greedy Caching* is better cache utilization, especially in the network core. An interesting point to note is that the overall latency increases for all strategies for the multiple custodian scenario. We observe from our simulations that for the multiple custodian case, the custodians are further away from the users on average when compared to the single custodian case, with some shortest paths from users to a custodian traversing through another custodian. Internally in the simulator, links connected directly to a custodian have higher delay in comparison to the other links, thereby increasing the overall latency. This assumption is realistic, as higher incoming traffic at the custodian can result in additional congestion and queuing delay. We also demonstrate in Section IV-B, that the overall hit rate reduces for multiple custodian network.

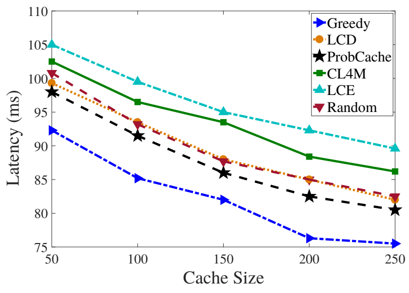
2) Performance for varying content universe: We study the impact of varying content universe on the performance of *Greedy Caching* for GARR (Figure 6). Results are generated for fixed content-to-cache ratio of 0.005, i.e., size of each cache is 0.5% of the content universe. Content universe is varied from 20000 to 100000. Our results suggest that *Greedy Caching* outperforms other strategies for large content universe as well. We observe from Figure 6 that latency decreases as content universe increases. As the absolute cache size increases with content universe (the content-to-cache ratio is



(a) Single custodian

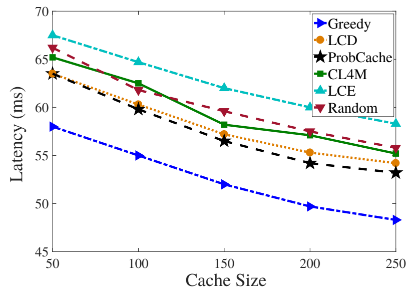


(b) Custodian = 2

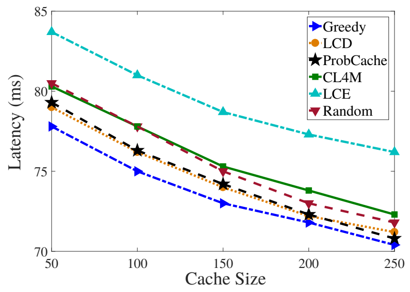


(c) Custodian = 5

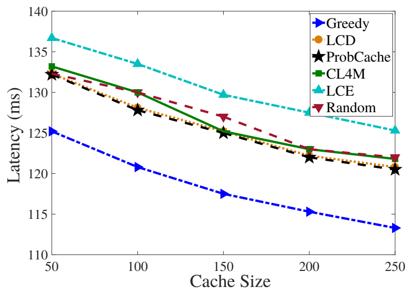
Fig. 7: Latency performance of Greedy Caching for GEANT



(a) Single custodian



(b) Custodian = 2



(c) Custodian = 5

Fig. 8: Latency performance of Greedy Caching for WIDE

fixed), popular content is readily available in caches leading to increased cache utilization. As the primary purpose of Figure 6 is to demonstrate the scalability of our algorithm, the horizontal axis in Figure 6 is not plotted in linear scale to preserve the aesthetics of the figure. These results demonstrate that *Greedy Caching* is efficient and can work with large catalogue sizes.

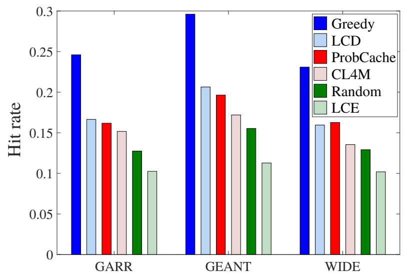
3) Performance for other topologies: Results for GEANT and WIDE topologies are shown in Figures 7 and 8 respectively for $\alpha = 0.8$. From the results we conclude that *Greedy Caching* performs best among the existing strategies. For both GEANT and WIDE, we make observations similar to the GARR topology. In general, we observe a) an overall increase in latency with increasing number of custodians and b) decrease in latency for greater cache size and higher values of content popularity skewness.

From our simulations, we observe that the performance improvement for *Greedy Caching* varies between 9-25%, 10-28% and 5-20% for GARR, GEANT and WIDE respectively. We notice that *Greedy Caching* gives better performance for GARR and GEANT compared to WIDE. Primary reason behind this improved performance is that for WIDE, the edge to node ratio (1.1) is significantly lower in comparison to GEANT (1.46) and GARR (1.52). It means that multiple paths are not available to reach a node from a randomly selected user in the multiple custodian scenario. We observe from the simulations that in case of WIDE with multiple custodians, most paths to a custodian are via another custodian, as alternate paths are not available. This scenario reduces cache hit significantly and decreases the performance gains.

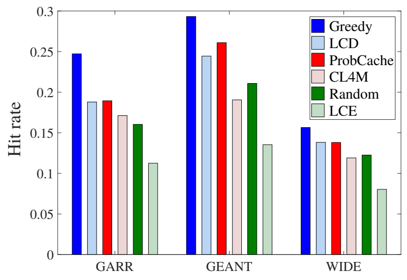
B. Discussion on cache hit rate

In addition to latency, hit rate is an important performance measure in ICN. If a request is served by a network cache, it is referred to as a hit and if it served by the custodian it is referred to as a miss. Therefore, higher hit rate suggests that more content requests are served from network caches, thereby reducing load on the custodians. Hit rate can therefore be considered as a measure for both traffic offloading and cache utilization. Figures 9 and 10 show the hit rate for two different cache sizes 50 and 150 respectively for fixed $\alpha = 0.8$, for single and multiple custodian scenarios. Each sub-figure in Figures 9 and 10 comprises of three topologies (GARR, GEANT, and WIDE). From the figures we observe that *Greedy Caching* improves hit rate by $15 - 50\%$. This can primarily be attributed to the intelligent content placement strategy adopted by *Greedy Caching* where every node caches content based on the total number of requests it receives from it downstream nodes.

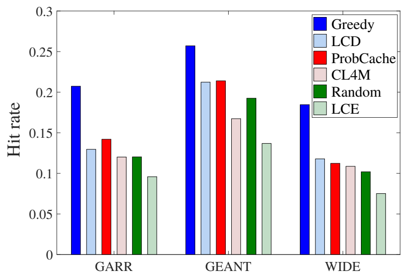
We also observe that hit rate performance of *Greedy Caching* degrades with increasing number of custodians. This performance degradation can be attributed to the removal of edges by the feedback arc set algorithm to break cycles for DAG construction. Moreover, we also observe that hit rate increases with cache size and content popularity skewness. Greater cache size provides an increased opportunity to cache



(a) Single custodian

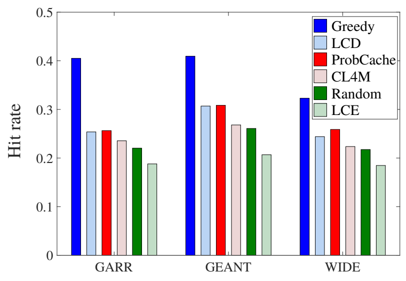


(b) Custodian = 2

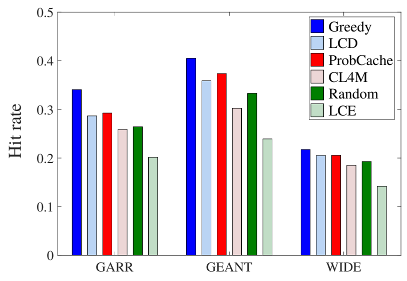


(c) Custodian = 5

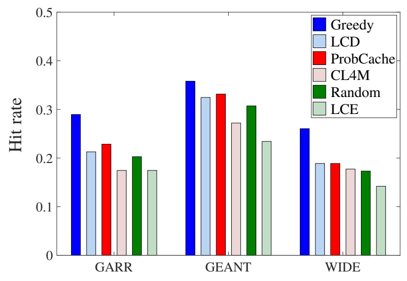
Fig. 9: Hit rate performance of Greedy Caching for $\alpha = 0.8$, $C = 50$



(a) Single custodian

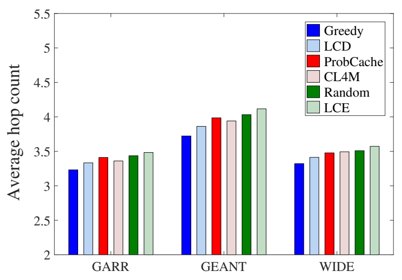


(b) Custodian = 2

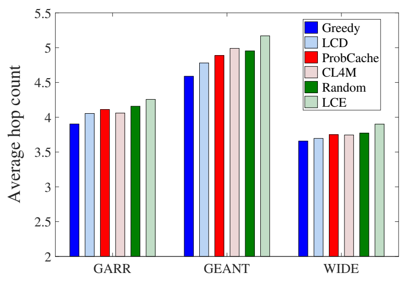


(c) Custodian = 5

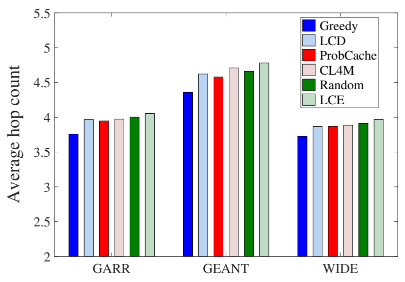
Fig. 10: Hit rate performance of Greedy Caching for $\alpha = 0.8$, $C = 150$



(a) Single custodian

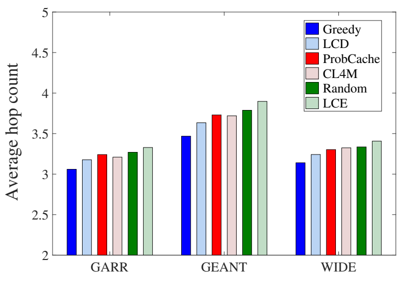


(b) Custodian = 2

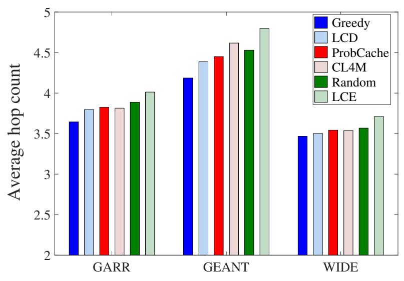


(c) Custodian = 5

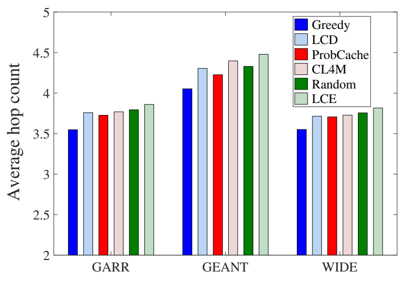
Fig. 11: Average hop count performance of Greedy Caching for $\alpha = 0.8$, $C = 50$



(a) Single custodian



(b) Custodian = 2



(c) Custodian = 5

Fig. 12: Average hop count performance of Greedy Caching for $\alpha = 0.8$, $C = 150$

popular content, while higher value of the skewness parameter results in popular content being requested most of the time.

C. Discussion on average hop count

Although hit rate provides an indication of the percentage of requests served from within the network, average hop count provides a measure of how far a request needs to travel to be satisfied. Figures 11 and 12 show the average hop count results for single and multiple custodian cases. As expected, we observe that *Greedy Caching* has the least average hop count. As *Greedy Caching* focuses on the miss stream from downstream nodes to calculate the relative popularity of content at each cache, it makes superior caching decisions at both the network edge and the network core, thereby leading to better overall performance.

We also perform simulations for different values of content request rate, and observe that request rate does not have a significant impact on latency. This result is expected and can be attributed to the design of the Icarus simulator. The simulator treats arrivals as an independent request stream and does not take into account increased network delays (due to congestion) because of increased arrival rates.

V. Conclusion

In this paper, we proposed *Greedy Caching*, a caching policy for ICN that works with any underlying routing algorithm and determines the content to be cached at each network node. *Greedy Caching* adopts a greedy approach that considers the request miss stream from downstream caches to make caching decisions at upstream caches. Therefore, the algorithm attempts to maximize the hit rate at each individual cache. Via extensive simulations, we showed that *Greedy Caching* significantly outperforms state-of-the-art caching and routing strategies. In future, we would like to extend this work to determine the optimal caching strategy for ICN.

References

[1] L. Saino, I. Psaras, and G. Pavlou, "Icarus: a caching simulator for information centric networking (ICN)," in *SimuTools*, 2014, pp. 66–75.

[2] W. Hoiles, O. N. Gharehshiran, V. Krishnamurthy, and N. Dào, "Adaptive Caching in the YouTube Content Distribution Network: A Revealed Preference Game-Theoretic Learning Approach," *IEEE Transactions on Cognitive Communications and Networking*, vol. 1, no. 1, pp. 71 – 85, Oct. 2015.

[3] "Netflix open connect." [Online]. Available: https://openconnect.netflix.com/en/

[4] M. Zhang, H. Luo, and H. Zhang, "A Survey of Caching Mechanisms in Information-Centric Networking," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 3, pp. 1473 – 1499, 2015.

[5] A. Dabirmoghaddam, B. Mirzazad-Barijough, and J. J. Garcia-Luna-Aceves, "Understanding Optimal Caching and Opportunistic Caching at The Edge of Information-Centric Networks," in *ACM ICN*, 2014, pp. 47–56.

[6] N. Laoutaris, S. Syntila, and I. Stavrakakis, "Meta algorithms for hierarchical web caches," in *IEEE International Conference on Performance, Computing, and Communications*, 2003, pp. 445 – 452.

[7] 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, July 2006.

[8] W. K. Chai, D. He, I. Psaras, and G. Pavlou, "Cache less for more in information-centric networks," in *IFIP Networking*, 2012, pp. 27–40.

[9] I. Psaras, W. K. Chai, and G. Pavlou, "Probabilistic in-network caching for information-centric networks," in *ACM ICN*, 2012, pp. 55–60.

[10] K. Cho *et al.*, "Wave: Popularity-based and collaborative in-network caching for content-oriented networks," in *IEEE INFOCOM (Workshop)*, Mar. 2012.

[11] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 83–86, Jan. 2006.

[12] Y. Li, T. Lin, H. Tang, and P. Sun, "A chunk caching location and searching scheme in content centric networking," in *IEEE ICC*, June 2012, pp. 1550 – 3607.

[13] I. Psaras, W. K. Chai, and G. Pavlou, "In-network cache management and resource allocation for information-centric networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 11, pp. 2920 – 2931, Nov. 2014.

[14] K. Suksomboon, S. Tarnoi, Y. Ji, M. Koibuchi, K. Fukuda, S. Abe, N. Motonori, M. Aoki, S. Urushidani, and S. Yamada, "PopCache: Cache more or less based on content popularity for information-centric networking," in *IEEE LCN*, Oct. 2014, pp. 236–243.

[15] A. Ioannou and S. Weber, "Towards on-path caching alternatives in information-centric networks," in *IEEE Conf. Local Comput. Netw. (LCN)*, 2014, pp. 362 – 365.

[16] C. Bernardini, T. Silverston, and O. Festor, "MPC: Popularity based caching strategy for content centric networks," in *IEEE ICC*, Jun. 2013, pp. 3619 – 3623.

[17] J. M. Wang and B. Bensaou, "Progressive caching in CCN," in *IEEE GLOBECOM*, Dec. 2012, pp. 2727 – 2732.

[18] W. Wang *et al.*, "CRCache: Exploiting the correlation between content popularity and network topology information for ICN caching," in *IEEE ICC*, Jun. 2014, pp. 3191 – 3196.

[19] Y. Wang, M. Xu, and Z. Feng, "Hop-based probabilistic caching for information-centric networks," in *IEEE GLOBECOM*, Dec. 2013, pp. 2102 – 2107.

[20] S. Saha, A. Lukyanenko, and A. Ylä-Jääski, "Cooperative caching through routing control in information-centric networks," in *IEEE INFOCOM*, Apr. 2013, pp. 100 – 104.

[21] L. Saino, I. Psaras, and G. Pavlou, "Hash-routing schemes for information centric networking," in *ACM ICN*, Aug. 2013, pp. 27–32.

[22] S. Wang, J. Bi, J. Wu, and A. V. Vasilakos, "CPHR: In-Network Caching for Information-Centric Networking With Partitioning and Hash-Routing," *IEEE/ACM Trans. Netw.*, vol. PP, no. 99, Oct. 2015.

[23] V. Sourlas, L. Gkatzikis, P.Flegkas, and L. Tassiulas, "Distributed cache management in information-centric networks," *IEEE Transactions on Network and Service Management*, vol. 10, no. 3, pp. 286–299, Sept. 2013.

[24] B. Banerjee, A. Seetharam, A. Mukherjee, and M. Naskar, "Characteristic time routing in information centric networks," *Elsevier Computer Networks*, vol. 113, pp. 148 – 158, Feb. 2017.

[25] S. Zaman and D. Grosu, "A distributed algorithm for the replica placement problem," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 9, pp. 1455 – 1468, Sep. 2011.

[26] L. Qiu, V. Padmanabhan, and G. Voelker, "On the placement of web server replicas," in *IEEE INFOCOM*, vol. 3, 2001, pp. 1587 – 1596.

[27] J. Kangasharju, J. Roberts, and K. Ross, "Object replication strategies in content distribution networks," *Computer Communications*, vol. 25, no. 4, p. Mar., 376 - 383 2002.

[28] V. Ramachandran, "Finding a minimum feedback arc set in reducible flow graphs," *Elsevier Journal of Algoriths*, vol. 9, no. 3, pp. 299 – 313, Sep. 1988.

[29] P. Eades, X. Lin, and W. F. Smyth, "A fast and effective heuristic for the feedback arc set problem," *Elsevier Information Processing Letters*, vol. 47, no. 16, pp. 319–323, Oct. 1993.