Characteristic time routing in information centric networks

Bitan Banerjee\textsuperscript{a,}\textsuperscript{*}, Anand Seetharam\textsuperscript{b}, Amitava Mukherjee\textsuperscript{c}, Mrinal Kanti Naskar\textsuperscript{d}

\textsuperscript{a}Department of Electrical and Computer Engineering, University of Alberta, Canada
\textsuperscript{b}Computer Science Department, SUNY Binghamton, USA
\textsuperscript{c}IBM India Pvt. Ltd., India
\textsuperscript{d}Department of Electronics and Telecommunication Engineering, Jadavpur University, India

\textbf{A R T I C L E  I N F O}

Article history:
Received 17 February 2016
Revised 5 November 2016
Accepted 16 December 2016
Available online 18 December 2016

Keywords:
Characteristic time
Routing
Information-centric networks
Latency

\textbf{A B S T R A C T}

Information centric networking (ICN) aims to transform today’s Internet from a host-centric model to a content-centric one by caching content internally within the network at storage-enabled nodes. Recently, multiple routing and cache management strategies have been proposed \cite{1,2,3,4,5,6} to improve the user-level performance, primarily latency in ICN. In this paper, we define latency as the download time for a piece of content. In this paper, we propose a simple routing strategy that leverages the concept of characteristic time to improve latency. Characteristic time for a content in a cache indicates the amount of time in future a recently accessed content is likely to remain in that cache. Our proposed algorithm namely, Characteristic Time Routing (CTR) uses characteristic time information to forward requests to caches where the content is likely to be found. CTR augments native routing strategies (e.g., Dijkstra’s algorithm), works with existing cache management and cache replacement policies and thus can be implemented in ICN prototypes with minimal effort. We perform exhaustive simulation in the Icarus simulator \cite{7} using realistic Internet topologies (e.g., GEANT, WIDE, TISCALL, ROCKETFUEL \cite{8}) and demonstrate that the CTR algorithm provides approximately 10–50% improvement in latency over state-of-the-art routing and caching management strategies for ICN for a wide range of simulation parameters.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

An exponential increase in content in recent years has resulted in the development of a flexible network architecture called Information Centric Networking (ICN) which proposes to evolve the current Internet from a host-centric model to a content-centric one. By caching content at storage-enabled nodes (also referred to as caches), requests for content can be served not only from the content custodian (origin servers), but also from intermediate caches. With the primary emphasis being content, if a cache en-route to the custodian has the requested content, the content will be returned to the requester from the cache itself, thereby improving user performance.

One of the main challenges in ICN is to develop efficient routing and cache management policies that improve user performance (e.g., decreasing latency and increasing throughput). In this paper, we develop a simple routing strategy that leverages the concept of characteristic time to decrease latency. In this paper, we define latency as the download time for a piece of content. Characteristic time for a content in a cache indicates the expected amount of time a content remains in that cache, given that it has been requested recently. Each requester maintains a time-to-live (TTL) based lookup table that is populated using characteristic time information. The requesters use their lookup tables to direct requests for content to caches, where the content is likely to be found, apart from the custodian. The proposed algorithm can be implemented on top of a native routing algorithm and works with any cache management policy. In contrast to our work, most prior research has focused primarily on shortest path routing, with different variants of en-route caching to improve performance \cite{4,6,9}.

We propose a characteristic time based routing algorithm (CTR) which executes on top of existing shortest path routing algorithms (e.g., Dijkstra’s algorithm) and alongside existing cache management policies (e.g., Leave Copy Everywhere (LCE) \cite{10}, Leave Copy Down (LCD) \cite{9}) and decreases latency. Fig. 1 illustrates the primitives of the CTR algorithm. Let us consider a network of cache-enabled routers with two requesters $U_1$ and $U_2$ and a custodian $C$. Let us assume that $U_1$ initiates a request for content $ID_1$. If an entry corresponding to $ID_1$ is found in its lookup table, $U_1$ forwards the request towards that node. In case of an unsuccessful search, $U_1$ searches the lookup table of neighboring requesters (in

\textsuperscript{*} Corresponding author.

E-mail addresses: bitan@ualberta.ca (B. Banerjee), anand@cs.binghamton.edu (A. Seetharam), amitava.mukherjee@in.ibm.com (A. Mukherjee), mrinaletce@gmail.com (M. Kanti Naskar).

http://dx.doi.org/10.1016/j.comnet.2016.12.009
1389-1286/© 2016 Elsevier B.V. All rights reserved.
this case $U_2$) for entries corresponding to $ID_1$. In case of an unsuccessful neighbor search, $U_1$ forwards the request towards the custodian $C$. A more detailed explanation of the CTR algorithm is given in Section 3.3.

The main contributions of this paper are summarized below. This paper is an extension of our previous work [11].

- We propose a novel routing strategy based on characteristic time. Our proposed strategy maintains state at each requester in the form of a time-to-live (TTL) based lookup table. Each entry of the lookup table consists of tuples denoting the content, the cache serving the content and the corresponding TTL information. TTL for an entry in the table is calculated as the expected time in future till which the content will remain in the cache that recently served the content. The lookup table is populated based on the characteristic time information sent to the requester from the cache serving the request. To enable the requester to explore additional paths apart from the shortest path to the custodian, the lookup table is also periodically updated using information obtained via a local neighbor search.

- We perform extensive simulations in Icarus [7], a simulator built exclusively for implementing and testing new ICN routing and caching policies to demonstrate the efficacy of the CTR algorithm. We compare the performance of CTR against state-of-the-art policies (LCE, LCD, CL4M, ProbCache, Hash Routing) on realistic Internet topologies (e.g., GEANT, WIDE, TISCALLI, ROCKEETFUEL). We study the impact of various simulation parameters (e.g., caching policy, content-universe, lookup table size, request arrival rate) on the performance of CTR and demonstrate that CTR provides approximately 10–50% improvement in latency over state-of-the-art policies for a wide range of simulation parameters.

The rest of the paper is organized as follows. We provide an overview of related work in Section 2. We discuss the problem and the proposed characteristic time routing (CTR) algorithm in Section 3. We describe the simulation setup and discuss the experimental results in Section 4. We conclude the paper in Section 5.

2. Related work

In this section, we provide an overview of related work and discuss the main differences between our work and existing work. We compare existing cache management and routing policies on the basis of the following attributes, namely, on-path caching, alternate-path caching and neighbor search. On-path caching denotes if content (while it is being downloaded) is cached at nodes along the shortest path, as determined by the routing algorithm. Alternate-path caching denotes that requests are forwarded along paths other than the shortest one based on some predefined rule and correspondingly the content is cached (while it is being downloaded) at nodes on these paths. Neighbor search indicates if the algorithm leverages a requester’s neighbors to discover new paths for obtaining the desired content i.e., instead of always routing to the custodian, a requester periodically searches its neighbors to find out if the desired content is available nearby. We note that both neighbor search and alternative path caching is cache-aware routing and neighbor search is a subgroup of alternative path caching. A brief comparison between the proposed algorithm (CTR) and existing routing and caching strategies is provided in Table 1.

We next describe the different algorithms listed in Table 1. Among the caching strategies, LCE [10] is the simplest one. In LCE, nodes cache all content that passes through them. To improve content diversity, in LCD [9] content is only replicated at the node that is one hop downstream from the cache serving the content. A modified version of LCD with chunk caching and searching (CLS) is proposed in [13] where a content chunk is cached one level downstream or upstream depending on whether a request is a cache hit or a cache miss. ProbCache [4], CL4M [6] and PopCache [14] all aim to improve content diversity by selectively caching content as it passes through a node. In ProbCache [4], content is probabilistically cached at nodes along the downloading path while in CL4M content is cached at a node depending on its centrality. PopCache leverages content popularity to determine whether to cache a particular content or not.

Cooperative caching [3,12,19] and hash table based routing [5] have also been proposed in the literature to improve ICN performance. Hash-routing [5] requires routers and caches to implement a hash function, that helps (i) caches decide whether or not to cache a particular content and, (ii) routers route requests to relevant caches. Cache-aware routing (CAR) [19] determines the optimal request forwarding path based on content request patterns. Distributed Cache Management (DCM) [17] improves cache utilization by using holistic information about request patterns and cache configuration. A congestion-aware routing and forwarding strategy is proposed in [3], while the benefits of forwarding requests to multiple custodians in parallel are explored in [15]. In contrast to the above-mentioned works, we adopt a more systematic approach to routing by leveraging the concept of characteristic time. Characteristic time [20] for a cache determines the expected amount of time a content is likely to remain in the cache, once it has been requested. In our CTR algorithm, we use the characteristic time information to determine where to forward content requests. We also augment the CTR algorithm with local search so that our approach can explore additional paths.

We would like to mention that there is also a significant amount of research studying the statistical attributes of caching. Rafii et al. [21] in one of the seminal papers study the efficiency of LRU caches for various reference models, e.g., locality model, Denning and Schwartz model, Markovian model, A$\phi$ model, and independent reference model (IRM). An approximate analysis of hit rate for LRU and FIFO caches has been studied in [22,23]. The performance of time-to-live (TTL) based caches has also been extensively studied in the literature [24,25].

3. Characteristic time routing

3.1. Problem statement

Let us consider a network of $N$ storage-enabled nodes. We assume that there are $M$ content custodians (origin servers, where the content is always available). The remaining nodes in the network are provided with a cache of size $C$. We assume that the content universe is of size $K$ and interests are generated by users according to a Poisson process at rate $\lambda$. The content popularity varies according to some known distribution such as a Zipfian or a Pareto or a Zeta distribution (in our simulation we assume a Zipfian distribution). We also assume that requests for content follow an Independent Reference Model (IRM) [26,27].

We consider that the network has a native routing strategy (for simulation, we assume Dijkstra’s shortest path routing) for for-
warding requests and implement the CTR algorithm on top of it. All nodes adopt some cache management (e.g., LCE [10], LCD [9]) and cache replacement policies (e.g., LRU). We assume that content is downloaded along the same path taken by the request to reach the cache from the requester, but in the opposite direction.

Our goal in this paper is to propose a simple routing strategy that augments an existing routing strategy to improve network performance and works with existing cache management and cache replacement policies. To enable rapid network-wide deployment and testing, in our CTR algorithm, we propose minimal changes to existing approaches; hence the CTR algorithm can be easily incorporated into ICN prototypes.

### 3.2. CTR algorithm

In this section, we outline the CTR algorithm. Whenever a requester desires a particular content, it generates an interest for that content. While the content is being downloaded from a particular cache, the content may be cached on en-route caches depending on the caching policy. Our goal is to obtain content along the fastest possible route so as to minimize latency.

To achieve this, each requester maintains state in the form of a lookup table which keeps track of the nodes from where it has downloaded content recently. Note that the lookup table is in addition to the Forwarding Information Base (FIB) that is used for forwarding interests towards a cache. We assume that the FIB is populated by an existing routing algorithm. Our CTR algorithm runs on top of the native routing algorithm and leverages the lookup table to determine the caches where the desired content is likely to be found. The CTR algorithm thus augments existing ICN designs and requires each requester to maintain another data structure, the lookup table.

Each entry in the lookup table consists of tuples denoting a content and the cache serving the content. To ensure freshness, each entry in the lookup table has a time-to-live (TTL) associated with it. The TTL denotes the time in future till which this entry remains in the table. When current time exceeds the TTL value of an entry, this entry is removed from the lookup table. We describe how we set the TTL value later in this section. The intuitive idea behind using a TTL is the following. If a requester has recently downloaded content from a particular cache (say A), it is highly likely that the content will be available in cache A in the near future. Therefore, instead of routing the interest towards the custodian, the requester can route it towards cache A.

Algorithm 1 describes the operation of any requester U. Whenever U generates an interest for content i, it searches its lookup table \( T_U \) to determine the cache to which the interest should be forwarded. If there are multiple entries in \( T_U \), the interest is forwarded to the nearest cache. If there is no entry for the requested content in the lookup table, the requester searches its neighbors for useful information (i.e., if there are other requesters among its neighbors, it searches their lookup tables to see if there are entries for the requested content). The search process begins with the requester forwarding the interest to all neighbors. In case of a match, a neighbor replies with an interest reply message that indicates the set of caches from where the requester could potentially obtain the desired content. In case of a successful search, the requester forwards the interest towards the nearest cache. In case of an unsuccessful neighbor search, the interest is forwarded towards the custodian. Note that if the interest encounters another cache en-route to the cache/custodian with the desired content, the content will be downloaded from the intermediate cache.

There is a subtle point to note here. Though the requester forwards the request towards a cache, it is not guaranteed that the cache will have the content as the content could have been evicted from the cache. Algorithm 2 describes the operation of the cache. There are two possible scenarios when a cache receives an interest from a requester - hit or miss. If the interest results in a cache

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Routing and cache management policies for ICN.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>On-path caching support</td>
</tr>
<tr>
<td>LCE [10]</td>
<td>✓</td>
</tr>
<tr>
<td>LCD [9]</td>
<td>✓</td>
</tr>
<tr>
<td>CLAM [6]</td>
<td>✓</td>
</tr>
<tr>
<td>ProbCache [4]</td>
<td>✓</td>
</tr>
<tr>
<td>CPHR [12]</td>
<td>✓</td>
</tr>
<tr>
<td>CLS [13]</td>
<td>✓</td>
</tr>
<tr>
<td>PopCache [14]</td>
<td>✓</td>
</tr>
<tr>
<td>Strategic Caching [15]</td>
<td>✓</td>
</tr>
<tr>
<td>Coupling Caching [16]</td>
<td>✓</td>
</tr>
<tr>
<td>DCM [17]</td>
<td>✓</td>
</tr>
<tr>
<td>Optimal Caching [2]</td>
<td>✓</td>
</tr>
<tr>
<td>OCRP [18]</td>
<td>✓</td>
</tr>
<tr>
<td>Hash-routing [5]</td>
<td>✓</td>
</tr>
<tr>
<td>CAR [19]</td>
<td>✓</td>
</tr>
<tr>
<td>CACS [3]</td>
<td>✓</td>
</tr>
<tr>
<td>CTR</td>
<td>✓</td>
</tr>
</tbody>
</table>
Algorithm 2 Operation of cache.

1: Cache hit;
2: Attach TTL to header of content being downloaded
3: Content downloaded along symmetric path
4: Cache miss;
5: Forward interest to the custodian

hit, the cache serves the content to the requester, else the cache forwards the request to the custodian.

Note that in our algorithm, the local search is performed only when there is no information available in the requester’s lookup table for the generated interest. The local neighbor search plays an important role in the CTR algorithm; it enables the requester to explore paths other than the shortest path. Without the local search, the CTR algorithm will converge to simple shortest path routing with the adopted en-route caching policy. For design simplicity and for minimizing the control overhead, we restrict the search only to neighbors of the requester. The neighbor search can be improved by using an approach similar to scoped-flooding outlined in [28]. The authors in [28] develop a method for obtaining the optimal neighbor search radius based on a network growth model. A similar approach can be used with CTR as well, assuming that the topological details of the network is readily available to the requesters. Determining the optimal search radius when the requested content is unavailable in a requester’s lookup table by taking into account the tradeoff between performance and control overhead is part of our future work. Note that when a requester searches its neighboring requesters, it waits until it obtains a response from them before forwarding an interest for the content. We assume that the content download time is much larger in comparison to the time needed to search a requester’s neighbors and for interest routing. Therefore, we only consider the content download time and not the time for neighbor search and for interest routing.

The idea behind the CTR algorithm is intuitive, but the biggest challenge lies in determining the value of the TTL. Too small a TTL will result in entries getting purged frequently and will trigger a local search. In contrast, large TTL will result in requests being routed towards a cache where the requested content is unlikely to be found. The cache will then reroute the requests towards the custodian, resulting in increased latency in obtaining the content. This happens because the content is downloaded along the same path as traversed by the request, but in the reverse direction.

In this paper, we leverage the concept of characteristic time to model the TTL value. Characteristic time for a content in a cache is the expected amount of time in future a content remains in that cache, given that a request for it has just been served from the cache. There are multiple approaches for calculating the characteristic time. A simple approach is to empirically determine the amount of time a content remains in a cache. Che et al. proposed an analytical method to determine the characteristic time for an LRU cache [20]. Martina et al. demonstrated that similar expressions can be determined for other cache replacement policies such as FIFO and Random [29]. In this paper, we assume that the cache replacement policy is LRU and mainly use Che’s approximation [7] to determine the characteristic time. This approach is based on the method outlined in [20]; the major difference being the authors in [7] incorporated a factor of ‘miss-rate ratio of the content’ in the approximation given in [20] to obtain the characteristic time for a content in a cache. Later, in Section 4.6, we study the performance of the CTR algorithm using Laoutaris’ approximation [30] for calculating the characteristic time.

We next describe how the characteristic time information is calculated and used to update a requester’s lookup table. We assume that every cache in the network can calculate the characteristic time ($T_c$) for a residing content, based on the content popularity and the traffic flowing through it. Let us assume that a cache receives a request for a content at time $t$. When a cache serves a content, it attaches the TTL = $t + T_c$ to the header of the content being sent to the requester (Algorithm 2). Once the content is downloaded, the requester extracts the TTL information from the header and updates its lookup table (Algorithm 1).

In Section 4, we test the efficacy of the CTR algorithm via simulations in Icarus [7], a simulator designed exclusively for ICN. We note that the CTR algorithm can be extended to Content-Centric Networks (CCN) as well. Vasilis et al. [31] enhance the CCN router architecture with satisfied-interest table (SIT), where information about each download is stored. CTR can be implemented in CCN by incorporating the TTL information in the SIT table.

3.3. An illustrative example

We illustrate the operation of CTR using a simple ring topology, as shown in Fig. 2. We consider a network with one custodian $C$, two routers $R_1$, $R_2$, and three interconnected requesters $U_1$, $U_2$, and $U_3$. The content universe is of size 2 with content ids $ID_1$ and $ID_2$. We assume that $U_1$, $U_2$, and $U_3$ do not have caches while $R_1$ and $R_2$ can cache content; no items are cached at $R_1$ while $ID_1$ is cached at $R_2$. We also assume that the lookup tables of $U_1$ and $U_2$ are empty. The lookup table for $U_3$ is shown in Fig. 2. Note that the lookup table only includes information about obtaining content from sources other than the custodian. Further, we also assume that the shortest path to the custodian for $U_1$ and $U_2$ is through $R_1$ and for $U_3$ is through $R_2$. We explain the functioning of the network by describing two simple scenarios.

Let us consider the state of the network at time $t = t_1$. At time $t = t_1$, $U_1$ generates a request for $ID_2$. As $U_1$’s lookup table is empty, it will search its neighbor $U_2$. As $U_2$’s lookup table is also empty, the neighbor search will be unsuccessful. $U_1$ will then forward the request to the custodian. While the content is being downloaded it is cached at $R_1$. Note that as the request is satisfied by the custodian, $U_1$’s lookup table is still empty as it is currently unaware of the fact that the content has been cached at $R_1$.

Now, let us assume that $U_2$ generates a request for $ID_1$ at time $t_2 > t_1$. As $U_2$’s lookup table is empty, it will search the lookup tables of $U_1$ and $U_2$. $U_1$’s lookup table being empty, no useful information is obtained from it. But, $U_2$ will provide $U_1$ the information that $ID_1$ is cached at $R_2$. $U_2$ then forwards the request to $R_2$. Let us
assume that the request reaches $R_2$ at time $t_3$, $R_2$ then serves the content to $U_2$. Assuming that the characteristic time for $ID_1$ at $R_2$ is $T_{d1}$, the TTL information ($TTL_1$) included along with content $ID_1$ is $t_3 + T_{d1}$. When the content is downloaded from $R_2$, an entry for $ID_1$ will be created in $U_2$’s lookup table with the above-mentioned TTL. As the shortest path to the custodian for obtaining content for $U_2$ is through $R_3$, this example illustrates that neighbor search enables a requester to explore additional paths for obtaining the desired content.

### 4. Performance evaluation

In this section, we first describe the experimental setup and then show the simulation results obtained by comparing the performance of the CTR algorithm against the state-of-the-art routing and cache management policies (namely LCE, LCD, CL4M, ProbCache, Hash-routing). All the above policies except Hash-routing route interests for content towards the custodian according to Dijkstra’s algorithm and adopt some variant of en-route caching. In all these policies, if a cache en-route to the custodian has a copy of the content, it serves the request. Overall, our simulation results demonstrate that the CTR algorithm decreases latency by nearly 10–50% over state-of-the-art approaches for varied topologies and a wide range of simulation parameters.

- **LCE**: In this policy, a copy of the requested content is stored in every cache along the path the content is downloaded [10].
- **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 [9].
- **CL4M**: This policy uses betweenness centrality (i.e., the number of shortest paths traversing a cache) to make caching decisions [6]. This policy aims to place content in caches with the greatest betweenness centrality, so as to maximize the probability of a cache hit.
- **ProbCache**: This policy attempts to reduce redundancy of content between caches [4] by probabilistically caching content at en-route caches.
- **Hash routing**: In Hash-routing [5], edge nodes in a network compute a hash function upon receiving a content request. Using the hash function, edge nodes map the content identifier to a specific cache and forward the request to that particular cache.

#### 4.1. Simulation setup

The CTR algorithm is simulated using the Icarus simulator [7]. The simulator consists of four building blocks, scenario generation, experiment orchestration, experiment execution, and result collection [7]. The scenario generation block configures the network topology and request generation for the simulation. Experiment orchestration is primarily concerned with implementing the different routing and caching strategies. The experiment execution block is the heart of the simulator and implements the actual forwarding of requests and the caching of content. The simulator is modeled as a discrete event based one. All network topologies in our simulation are considered as undirected graphs with interest arrival rate following a Poisson process. Each interest is considered as an event, and whenever an event occurs, a corresponding timestamp is stored. The result collection block’s functionality is to gather the results of the simulation. The native routing algorithm is Dijkstra’s algorithm; the weights of the edges in the graph indicate inverse of the capacity (i.e., the latency) over the links. Additional details are available in [7].

The network parameters for the simulation are the content universe $K$, the arrival rate $\lambda$, the cache size $C$, the size of the lookup table $L$ and the content popularity skewness $\alpha$ for the Zipfian distribution. Characteristic time is calculated using the Che’s approximation as outlined in the Icarus simulator [7]. Each requester’s lookup table can be implemented as a multidimensional array or as a hash table. In our experiments, the network is initially warmed up with 100,000 requests and performance results are calculated over additional 100,000 requests. Confidence intervals are obtained over 5 runs of the simulation. The performance metric evaluated in this paper is latency.

We evaluate the performance of CTR and other strategies for real Internet topologies, GEANT (European academic network), WIDE (Japanese Academic Network), TISCALI (pan-European commercial ISP), ROCKETFUEL. GEANT is an academic network spread around the world consisting of 40 nodes. The WIDE topology is the first network established in Japan and consists of 30 nodes. TISCALI is a commercial ISP network consisting of 240 nodes while ROCKETFUEL is an ISP topology mapping engine, used to generate ISP router level networks. ROCKETFUEL topology used in this paper has 161 nodes. Additional details regarding these networks can be found in [32,33].

#### 4.2. Simulation results

In this subsection, we present performance results for four Internet topologies, GEANT, WIDE, TISCALI and ROCKETFUEL for different cache sizes and content popularity skewness ($\alpha$). To avoid cluttering the graph with multiple lines and to help the reader with visual clarity, performance of CTR is compared with LCE, LCD, CL4M, ProbCache, and basic Hash-routing. Note that we have also compared the latency performance of CTR with the high overhead versions of Hash-routing and observed that CTR gives similar performance. Additionally, the internal link-load is significantly higher for Hash-routing when compared to the CTR and the other algorithms. Our implementation can be found in [34]. To avoid repetition, we present all results for the GEANT topology and few results for the other topologies.

We assume that the cache replacement policy is LRU for all algorithms evaluated. The CTR algorithm can work with any cache management policy. We first present results obtained by running the CTR algorithm along with the LCD cache management policy. We discuss results obtained by running CTR on top of other cache management policies later. Simulation results are generated for $K = 500$, $\lambda = 20$, $L = 300$, and $\alpha$ and $C$ are varied between 0.6–1.1 and 50–250, respectively. Unless mentioned otherwise, simulation parameters are kept at the aforementioned values throughout the paper. Fig. 3(a)–(c) and Fig. 3(d)–(f) demonstrate the latency performance for the different strategies for the GEANT topology. We observe from Fig. 3(a)–(c) that the CTR algorithm significantly outperforms other caching approaches with the percentage reduction in latency (7–35%) increasing as the cache size increases. We also observe that Hash-routing closely follows CTR for low values of $\alpha$ (0.6, 0.8), latency reduction is 1–7%, but for higher $\alpha$ (11) latency reduction is 20–38%. Similarly, from Fig. 3(d)–(f) we observe that CTR outperforms the state-of-the-art approaches, and this holds true for a wide range of cache sizes.

Fig. 4 shows the latency performance of TISCALI, ROCKETFUEL and WIDE for $\alpha = 0.8$ for varying cache sizes. We observe from the figure that the CTR algorithm significantly outperforms the other strategies for these topologies as well, with the greatest reduction in latency being for the WIDE topology. Latency reduction for WIDE, TISCALI, and ROCKETFUEL when compared to the state-of-the-art caching policies (LCE, LCD, CL4M, ProbCache) is 2–31%, 6–25% and 3–24% respectively. However, when compared to Hash-routing, the latency reduction for WIDE, TISCALI, and ROCKETFUEL is 21–45%, 9–37% and 10–35%. Note that the confidence interval on
the plots in Figs. 3 and 4 is small which increase our confidence in these results.

The main reasons for the superior performance of the CTR algorithm are the use of local search and the concept of characteristic time. By searching locally, the CTR algorithm is able to explore alternate paths apart from the shortest one and obtain content from caches in the neighborhood. This is in contrast to majority of state-of-the-art approaches which restrict themselves to shortest path routing. Additionally, the use of a characteristic time based lookup table helps to keep track of caches where the content might be present. All results presented in this paper so far are for certain fixed values of the simulation parameters. We next explore the impact of the various simulation parameters (e.g., \( K, L, \lambda \)) on performance.

### 4.3. Effect of varying content universe

In this section, we study the impact of \( K \) on the performance of CTR. Performance of CTR for different values of \( K \), for the GEANT topology for \( \alpha = 0.8 \) is shown in Fig. 5. From Fig. 5, we observe that CTR outperforms the state-of-the-art approaches for all values of \( K \) with the performance gains increasing with increasing cache size. We observe that the average latency for all strategies increases with increasing \( K \). For a fixed cache size, as the value of \( K \) increases, a greater percentage of requests to any cache will result in a cache miss, thereby resulting in increased latency. We note that we observe similar performance for other topologies as well.

We also study the performance of CTR with varying content universe for fixed content-to-cache ratio of 0.05, 0.1, and 0.2, i.e., cache size is 5\%, 10\%, and 20\% of the size of the content universe in Fig. 6 for the GEANT topology for \( \alpha = 0.8 \). Content universe is varied from 5000 to 1 million. As the primary purpose of Fig. 6 is to demonstrate the scalability of our algorithm, the horizontal axis in Fig. 6 is not plotted in linear scale to preserve the aesthetics of the figure. The CTR algorithm is efficient and can work with large catalogue sizes. Hence, it can be easily implemented in a real ICN.

Primary reason behind superior performance of CTR in Fig. 6 is the utilization of neighbor caching. We note that as the content universe is large, even with an increased number of warmup requests the caches may not reach steady state, i.e., caches do not have all the popular content. The improved diversity of cached content complements neighbor search as it helps explore new paths and increases the cache hit probability. This situation is true in a real world setting as well, where it almost impossible for the caches to reach the steady state due to generation of new content.
4.4. Effect of varying lookup table size

We also consider the effect of lookup table’s size on the performance of CTR. Results for varying sizes of lookup table (L = 300, 100, 50 and 30) are shown for fixed values of K = 1000, α = 0.6 and for varying C for the GEANT topology in Fig. 7. We observe that for higher values of L (300, 100), CTR outperforms the other approaches. We also observe from Fig. 7 that the performance of CTR starts deteriorating for small lookup table sizes when L is less than 10% of the content universe (L = 50), the performance of CTR is comparable to the shortest path routing with LCD caching mechanism. For a lookup table size of about 5% of the content universe (L = 30), the performance of CTR is worse than LCD. The explanation for this behavior is the following. When the size of the lookup table is small, information related to additional caches from where content can be downloaded is unavailable from the table. Moreover, a smaller lookup table can lead to increased neighbor searches that increase latency and adversely impact performance. As the size of the lookup table increases, the probability of finding alternative content sources other than the custodian increases leading to improved performance. However beyond a certain size, increasing the lookup table size does not help, as the extra entries become obsolete due to TTL constraint. We note that the CTR algorithm will require additional memory for storing the lookup table. But this additional memory incurred by the lookup table will only be a small fraction in comparison to the memory required in ICN routers [35].

4.5. Effect of other caching mechanisms

In the results presented so far, we have assumed that the caching mechanism used with the CTR algorithm is LCD. As CTR is a routing strategy, it can be combined with any caching policy and is not bound to any underlying caching policy. A superior caching policy can thus augment the performance of CTR. In this section, we study the performance of CTR for different caching mechanisms (Fig. 8).

Fig. 8 shows the performance of the different caching policies in combination with CTR for the GEANT topology. From Fig. 8, we can conclude that CTR in combination with ProbCache provides the best performance in terms of latency. Additionally, we observe that the average performance improvement of the different caching policies (LCE, LCD, CL4M, and ProbCache) in combination with CTR in comparison to using the caching policies (LCE, LCD, CL4M, and ProbCache) with shortest path routing is 1–20%, 7–24%, 10–30%, and 10–30% respectively.
We next explore the reasons behind the improved performance of CTR with CL4M and ProbCache. Both CL4M and ProbCache are better caching strategies in comparison to LCD and LCE as they cache content based on betweeness centrality and hit probability respectively and increase the diversity of content among the different caches. CTR being a routing algorithm thus performs better with CL4M and ProbCache. We hypothesize that a superior caching strategy will improve the overall performance of CTR.

4.6. Effect of other characteristic time approximation

Characteristic time estimation lies at the heart of the CTR algorithm. In this paper, so far we have used Che’s approximation [20] for determining the characteristic time. We next discuss the impact of the accuracy of the characteristic time approximation on latency. Another widely adopted policy for calculating the characteristic time is Laoutaris’ approximation [30]. The main difference between the two approaches is that Laoutaris’ approximation is obtained by performing a Taylor series expansion on the characteristic time obtained by Che’s approximation and considering the first three terms. Therefore, Laoutaris’ approximation is likely to be less accurate when compared to Che’s approximation. However, as Laoutaris’ approximation only uses the first three terms of the Taylor expansion, the overhead required for its computation is lower in comparison to Che’s approach.

In this section, we compare the performance of CTR using Che’s and Laoutaris’ approximations for GEANT, TISCALI, ROCKETFUEL, and WIDE topologies are given in Fig. 9. We observe that the performance of CTR using Laoutaris’ approximation is poorer when compared to Che’s approximation. This is expected because Laoutaris’ approximation provides a looser bound on the characteristic time in comparison to Che’s approximation. Interestingly, our simulations demonstrate that the performance degradation is only about 1–7%. Therefore, Laoutaris’ approximation can be used to determine the characteristic time for the CTR algorithm in scenarios where computational overhead is the bottleneck (e.g., ICN consisting of battery-powered sensor nodes.)

4.7. Discussion on hit rate

In this subsection, we study the performance of CTR in terms of cache hit rate. In this experiment, if a request is served from in-network caches it is considered to be a hit and if it is served from the custodian, it is considered to be a miss. Fig. 10 illustrates the hit rate performance of CTR and other considered strategies. We report average hit rate after considering different values of C (50, 100, 150, 200, and 250). We observe that CTR outperforms the other algorithms in terms of hit rate which can be attributed as the root cause for the decreased latency in CTR.

We next examine the performance of CTR under a microscope and shed light on how and where (within the network) requests are satisfied for the CTR algorithm. The performance of CTR, in terms of average percentage of requests that result in a hit or a miss for GEANT, TISCALI, and WIDE are shown in Fig. 11. Bars 1 to 3 represent the performance of CTR for the GEANT topology for $\alpha = 0.6$, 0.8, and 1.1 respectively, while bars 4, 5 and 6 illustrate the performance of CTR for TISCALI, ROCKETFUEL, and WIDE topology for $\alpha = 0.8$ respectively. In Fig. 11 too average percentage is reported after considering different values of C (50, 100, 150, 200, and 250).

Each bar comprises of multiple parts, namely, CTR early, CTR selected, Neighbor, CTR neighbor, Custodian early, and Custodian. Recall that a requester first searches it lookup table to determine alternate sources (apart from the custodian) from where it can obtain the desired content. If a cache is found from the lookup table, the requester forwards the request towards the cache. CTR selected denotes the fraction of requests served by the cache selected from the lookup table. CTR early denotes the fraction of requests served by nodes en-route to the cache selected from the lookup table. When a source is not obtained from the lookup table, the requester searches the lookup table of its neighbors. Neighbor corresponds to the fraction of requests satisfied by the cache obtained from the neighbor search or by nodes en-route to that cache. CTR neighbor stands for the fraction of requests served using lookup table entries, obtained from earlier neighbor search information. The requests that result in a miss are routed to the custodian. Custodian and Custodian early denote the fraction of requests that are satisfied by the custodian and by nodes en-route to the custodian respectively.
From Fig. 11 we observe that nearly 20–35% of all requests are satisfied by the requester's own lookup table or via neighbor search. Although it might appear from the figure that neighbor search does not satisfy many requests, it is important to note that when a content is obtained from a source as provided by the neighbor search, this information is included in the lookup table of the requester. Therefore, future requests for that content can be satisfied by the requester's own lookup table.

There is another subtle point that needs to be highlighted here. The lookup table for a requester includes information about sources that lie en-route to the custodian. From the bar plot in Fig. 11, we observe that approximately 10–20% of all requests satisfied using a requester's lookup table are based on the earlier information obtained using neighbor search. This validates our claim that the CTR algorithm explores alternate paths other than the shortest one. Additionally, from Fig. 11 we observe that the percentage of requests satisfied by a requester’s lookup table and via neighbor search increases as the value of α increases. As α increases, some pieces of content become far more popular than other pieces of content and are available at multiple caches within the network. As requests for popular content are generated frequently, a larger fraction of requests can be satisfied by a requester’s lookup table and via neighbor search.

### 4.8. Discussion on overhead

In our discussion so far, we have not considered the overhead of the CTR algorithm. In the CTR algorithm, a requester searches the lookup table of its neighbors for possible sources of the requested content when it does not obtain a valid entry for that content in its own lookup table. This local search introduces a small amount of overhead in the network, and in this section we discuss the trade-off between latency and overhead.

In our simulation, we observe that a requester can obtain useful information from its lookup table for 18–30% of requests for different values of α and C (considering all topologies). Additionally, when a requester searches its neighbors, approximately 40–60% of the time neighbor search returns fruitful information. A natural question arises – is it necessary to initiate a neighbor search each time a valid entry for a content is not found in the requester's lookup table? We introduce a probability p with which a requester searches its neighbors when there is no valid entry in the lookup table to explore this trade-off between latency and network overhead. Note that in all simulation results presented so far we consider p = 1.

In Table 2, we present the tradeoff between latency and overhead for the GEANT topology for C = 150. In this table overhead represents the percentage of all requests that trigger a neighbor search. Note that the overhead is different from p, as p represents the probability of a local search if a valid entry for the requested content in not found in the lookup table. As expected, as the value of p increases the overhead increases and the latency decreases. Depending on requirements, it is possible for a network operator to select an appropriate value of p. We leave the task of determining the optimal value of p based on this latency-overhead tradeoff as future work.

### Table 2

<table>
<thead>
<tr>
<th>Search Probability</th>
<th>Latency (ms)</th>
<th>Overhead (percentage)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>62.4</td>
<td>0</td>
</tr>
<tr>
<td>0.1</td>
<td>61.2</td>
<td>5.3172</td>
</tr>
<tr>
<td>0.4</td>
<td>58.75</td>
<td>21.4836</td>
</tr>
<tr>
<td>0.5</td>
<td>57.5</td>
<td>27.0158</td>
</tr>
<tr>
<td>0.6</td>
<td>57</td>
<td>31.5066</td>
</tr>
<tr>
<td>0.7</td>
<td>55.5</td>
<td>37.833</td>
</tr>
<tr>
<td>0.9</td>
<td>51.3</td>
<td>47.968</td>
</tr>
<tr>
<td>1.0</td>
<td>47</td>
<td>50.190</td>
</tr>
</tbody>
</table>

### 4.9. Effect of varying request arrival rate

We also perform simulations for different values of λ (10, 20, 50, and 100), and observe that arrival 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 as independent request stream and does not take into account increased network delays (due to congestion) because of increased arrival rates.
5. Conclusion

In this paper, we designed a characteristic time based routing algorithm (CTR) for ICN that runs on top of a native routing algorithm (e.g., Dijkstra’s algorithm) and alongside existing cache management and replacement strategies. The CTR algorithm leveraged the concept of characteristic time to route requests for content to caches where it is likely to be found. Via extensive simulations, we showed that the CTR algorithm significantly outperformed state-of-the-art routing and cache management strategies. In future, we plan to improve the performance of the CTR algorithm by optimizing the granularity and scope of the local search. We also plan to evaluate the effectiveness of the CTR algorithm for correlated request streams.

References


[34] URL http://tinyurl.com/p9g5h23.

Bitan Banerjee is a graduate student at the University of Alberta, Department of Electrical and Computer Engineering. He received his Bachelor of Engineering degree from the Department of Electronics and Telecommunication Engineering, Jadavpur University, India. His research interests include computer communication, wireless sensor networks, body area networks. He published several peer-reviewed articles, including IEEE Transactions on Computers, IEEE ICNC, ACM BodyNets.

Anand Seetharam is an assistant professor at SUNY Binghamton, USA. Prior to joining SUNY Binghamton, he was an assistant professor at California State University Monterey Bay. He obtained his Ph.D. from the University of Massachusetts Amherst. He is broadly interested in the field of computer networking. His research encompasses information-centric networks, wireless networks, multimedia streaming, internet-of-things, cybersecurity, and cyberphysical systems. He has published numerous papers in peer-reviewed journals and conferences. He has co-organized the IEEE INFOCOM 2016 MuSc workshop and the IEEE MASS 2015 CCN workshop. He has also served on the TPC of multiple conferences including IEEE ICC, IEEE ICCCN and IEEE WoWMoM. He has won multiple awards including the ACM ICN 2014 runners up to best paper award, University of Massachusetts Amherst Outstanding Synthesis Award, University of Massachusetts Amherst Portfolio with Distinction Award and has a U.S. patent on video streaming systems.

Amitava Mukherjee is Senior Manager of IBM India from Oct 2002. And has over 33 years of experience in leading and managing competencies in IBM GBS India, and Application/Implementation and R&D projects for Domestic/International Clients. Amitava had been on sabbatical from IBM India (Jan 2003–Mar 2005), and visited University of New South Wales, Sydney as visiting Professor (2003–2004) and Royal Institute of Technology, Stockholm as Senior Researcher (2004–2005). He was Senior and Principal Consultant at PwC India from May 1995 to Jun 1999 and Jul 1999 to Sep 2002 respectively. From 1983 to 1995, he was with the Department of Electronics and Telecommunication Engineering, Jadavpur University, Kolkata, India in research and teaching positions. Amitava’s contributions are in the design of network architecture, routing protocol and pagiing strategy in the field of large communication networks specifically in Wireless Sensor, Cellular communication, Mobile Ad hoc and Pervasive, and Optical Networks. Currently, his main focus of research is on 5G wireless networks, controllability of complex network, cognitive radio network, nano communication network, software defined networking (SDN), information centric networking (ICN), compressive sampling (CS). He has around 150 published papers in journals and conference proceedings of international repute, four patents, five books and two book chapters in pervasive computing, wireless communication, societal engineering, nano communication network and controllability of complex network respectively. He is a Senior Member of IEEE Communications Society and member of ACM. He has been serving as member of Technical Program Committee for a large number of conferences like ICC, GLOBECOM, WPMC, etc. He is the reviewer of major IEEE Transactions on (TON, TMC, TWC, PDS, etc.). He is currently serving 1913 WC on IEEE Standard on Software-Defined Quantum Communication (SDQC). He served as an active member of 1906.1 IEEE Standard WC on Nano networking, the emerging field of research from 2012–2015. He was the Vice-Chairman of IEEE Communication Society, Calcutta and a member of ACM. Amitava was one of experts of Program Analysis Task Force (PNTF) of IEEE Communication Society, Head Quarter, New York. Amitava had received Ph.D. degree in Computer Science from Jadavpur University, Kolkata, India.

Mrinal Kanti Naskar received his B.Tech. (Hons) and M.Tech degrees from E&ECE Department, IIT Kharagpur, India in 1987 and 1989 respectively and Ph.D. from Jadavpur University, India in 2006. He served as a faculty member in NIT, Jamshedpur and NIT, Durgapur during 1991–1996 and 1996–1999 respectively. Currently, he is a professor in the Department of Electronics and Telecommunication Engineering, Jadavpur University, Kolkata, India where he is in charge of the Advanced Digital and Embedded Systems Lab. His research interests include ad hoc networks, optical networks, wireless sensor networks, wireless and mobile networks and embedded systems. He is an author/co-author of numerous peer-reviewed articles.