Characteristic Time based Routing in Information Centric Networks

Bitan Bandyopadhyay\(^1\), Anand Seetharam\(^2\), Amitava Mukherjee\(^3\), Mrinal Kanti Naskar\(^1\)
\(^1\)Department of Electronics and Telecommunications Engineering, Jadavpur University, India
\(^2\)Computer Science Program, California State University Monterey Bay, \(^3\)IBM India Pvt. Limited

Abstract—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 \([1]–[6]\) to improve the user-level performance (e.g., content-access delay) in ICN. In this paper, we propose a simple routing strategy that leverages the concept of characteristic time to improve content-access delay. 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 (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 policies and thus can be implemented in ICN prototypes with minimal effort. We perform exhaustive simulations using realistic Internet topologies (e.g., GEANT, WIDE, TISCALI, ROCKETFUEL \([7]\)) and demonstrate that the CTR algorithm provides approximately 10-50\% improvement in delay over state-of-the-art routing and caching management strategies for ICN for a wide range of simulation parameters.

I. 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 on efficient caching, if a cache en-route to the custodian has the requested content, the content will be returned to the user from the cache itself, thereby improving user performance.

One of the main challenges in ICN is to develop efficient routing and cache management policies which improve user performance (e.g., decreasing content-access delay and increasing throughput.) In this paper, we develop a simple routing strategy which leverages the concept of characteristic time to decrease delay. Characteristic time for a content in a cache indicates the duration of time the content is likely to be present in the cache. Our routing algorithm uses this concept to direct requests (interest packets) for content to caches (apart from the custodian) where the content is likely to be found.

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 \([4], [6], [8]\).

The main contributions of this paper are summarized below.

- 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., LCE \([9]\), LCD \([8]\)) and improves delay performance. Our proposed strategy maintains state at each requester node in the form of a time-to-live (TTL) based lookup table. The lookup table is populated based on the characteristic time information sent to the requester from the cache serving the request. The lookup table noted the cache from where a content is downloaded and the duration of time for which the content is likely to be available in the cache in future. To enable the requester to explore additional paths apart from the shortest path to the custodian, the lookup table is periodically updated using information obtained via a local neighbor search. Note that the lookup table is an additional data structure that is maintained at each requester along with data structures native to ICN implementations such as Forwarding Information Base (FIB), Pending Interest Table (PIT), Content Store (CS) \([10]\).

- We perform extensive simulations on Icarus \([11]\), a simulator built exclusively for implementing and testing new ICN routing and caching policies to demonstrate the efficacy of our proposed algorithm. We compare the performance of our approach against state-of-the-art policies (LCE, LCD, CL4M, ProbCache, Hash Routing) on real Internet topologies (e.g., GEANT, WIDE, TISCALI, ROCKETFUEL) and show that our routing strategy gives approximately 10-50\% improvement in delay over these policies for a wide range of simulation parameters.

The rest of the paper is organized as follows. We discuss the problem and our proposed characteristic time routing in Section II and III respectively. We describe the simulation setup and results in Section IV. We conclude the paper in Section V.

II. PROBLEM STATEMENT

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

We consider that the network has a native routing strategy (for our simulations, we assume Dijkstra’s shortest path routing) for forwarding requests and implement our CTR algorithm on top of it. All nodes also adopt some cache management (e.g., LCE [9], LCD [8]) and cache replacement policy (e.g., LRU).

Our goal in this paper is to propose a simple routing strategy that augments an existing routing policy 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; our CTR algorithm can easily be incorporated into ICN prototypes.

III. CHARACTERISTIC TIME ROUTING

In this section, we outline our CTR algorithm. Whenever a requester desires a particular content, it generates an interest for that content. While the content is downloaded from a particular cache (also known as source) by the requester, 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 delay.

To achieve this, each requester maintains state in the form of a lookup table which keeps track of the sources from where it has downloaded content recently. Note that the lookup table is in addition to the Forwarding Information Base (FIB) that is used by nodes for forwarding interests toward a source. 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 sources 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. In Section IV, we test the efficacy of the CTR algorithm via simulations in Icarus [11], a simulator designed exclusively for ICN.

To ensure freshness, each entry in the lookup table has a time-to-live (TTL) associated with it. The intuitive idea 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 toward the custodian, it can route it toward cache A.

Algorithm 1 describes the operation of a requester. Whenever a requester generates an interest, it searches its lookup table to determine the source to which the interest should be forwarded. If there is any existing entry in the lookup table for the requested content that satisfies the TTL criteria, the interest is forwarded towards that source. If there are multiple sources that satisfy the TTL criteria, the interest is sent to the source closest to the requester.

If there is no valid entry 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 valid 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 sources from where the requester could potentially obtain the desired content. In case of a successful search, the requester forwards the interest toward the nearest source. In case of an unsuccessful search, the interest is forwarded toward the custodian. For design simplicity and for minimizing the control overhead, we restrict the search only to neighbors of the requester. 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 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. It is evident that there are several optimizations possible to Algorithm 1. As our primary goal is to propose a simple algorithm that improves overall performance, we have not implemented them.

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 toward a source where the requested content is unlikely to be found. The source will

Algorithm 1 Functions of receiver

1: Generation of interest packet:
2: if Valid entry found in the lookup table then
3: Find out possible sources
4: Forward interest to the nearest source
5: else
6: Search neighbors
7: if Useful source information obtained then
8: Forward interest to the nearest source
9: else
10: Forward interest to the content custodian
11: end if
12: end if
13: Downloading content:
14: Fetch characteristic time information from header
15: Update lookup table
then reroute the requests toward the custodian, resulting in increased delay in obtaining the content. A naive approach is to adopt a heuristic method to determine the TTL, but this method will produce TTL values dependent on the simulation parameters.

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. We assume that every cache in the network can calculate the characteristic time for a residing content, based on the content popularity and the traffic flowing through it. 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 [12]. Martina et al. demonstrate that similar expressions can be determined for other cache replacement policies such as FIFO and Random [13]. In this paper, we assume that the cache replacement policy is LRU and use the approximation in [11] to determine the characteristic time. This approach is based on the method outlined in in [12]; the major difference being the authors in [11] incorporate a factor of 'miss-rate ratio of the content' in the approximation given in [12] to obtain the characteristic time for a content in a cache.

**Algorithm 2 Operation of source**

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

Algorithm 2 describes the operation of the source. There are two possible scenarios when a source receives an interest from a requester - hit or miss. If the interest results in a cache hit, the source attaches the characteristic time information in the header of the content being downloaded by the requester. Content is downloaded by the requester using the symmetric path, i.e., the content is downloaded along the reverse path through which the interest arrived at the cache. Once the content is downloaded, the requester extracts the characteristic time information from the header and updates its lookup table (Algorithm 1). If the interest results in a miss, then the source forwards the interest toward the custodian.

IV. Performance Evaluation

In this section, we briefly describe the state-of-the-art routing and cache management policies (namely LCE, LCD, CL4M, ProbCache, Hash-routing) developed for ICN with which we compare the CTR algorithm. All the above policies except Hash-routing route interests for content toward 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 interest.

- **LCE:** In this policy, a copy of the requested content is stored at every cache along the path the content is downloaded [9].
- **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 [8].
- **CL4M:** This policy uses betweenness centrality (i.e., 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, they map the content identifier to a specific cache and forward the request to that particular cache. Although this policy decreases delay, it increases the internal link load and has a high overhead associated with it [5].

A. Simulation setup

The CTR algorithm is simulated using the Icarus simulator [11]. The simulator consists of four building blocks, scenario generation, experiment orchestration, experiment execution, and result collection [11]. The scenario generation block configures the network topology and request generation for the simulation. Experiment orchestration is primarily concerned with implementing the different strategies (e.g., CTR, LCE). 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 delay) over the links. Additional details are available in [11].

The network parameters for the simulation are, $K = 500$, $\lambda = 12$, $\alpha = 0.6 - 1.1$, and $C = 50 - 250$, where $\alpha$ is content popularity skewness associated with the Zipfian distribution. Characteristic time is calculated using the method outlined in the Icarus simulator [11]. Each requester’s lookup table can be implemented as a multidimensional array or as a hash table. In our simulation, we assume that the lookup table can hold 2000 entries. If the table becomes full, outdated entries are replaced by new ones. In our experiments, the network is initially warmed up with 100000 requests and performance results are calculated over additional 100000 requests. Confidence
The performance metric evaluated in this paper is delay. We evaluate the performance of CTR and other strategies for simple topologies like binary tree, grid and 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 39 nodes. The WIDE topology is the first network established in Japan and consists of 29 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 [14], [15].

### B. Simulation Results

In this subsection, we present performance results for the four real 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 reader with visual clarity, performance of CTR is compared with LCE, LCD, CL4M and ProbCache. Note that we also compared the delay performance of CTR with the high overhead Hash-routing and observed that we obtained 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 [16].

We assume that the cache replacement policy is LRU for all algorithms evaluated. The CTR algorithm can work with any cache management policy; for the results presented in this paper, the CTR algorithm works with the LCD cache management policy. We discuss the results obtained by running CTR on top of other cache management policies later. Due to lack of space we present extensive simulation results only for the GEANT topology. Figs. 1(a)-1(c) and Figs. 1(d)-1(f) demonstrate the delay performance for the different strategies for the GEANT topology for different values of \( \alpha \) and cache size respectively. We observe from Figs. 1(a)-1(c) that the CTR algorithm significantly outperforms the other approaches with the percentage reduction in delay (10-42%) increasing as the cache size increases. Additionally, from Fig. 1 we observe that delay reduction of CTR is more significant (15-50%) for realistic \( \alpha \) values between 0.6 and 0.8, and this holds true for wide range of cache sizes.

Fig. 2 shows the delay 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 delay being for the WIDE topology. Delay reduction for WIDE, TISCALI, and ROCKETFUEL are 14-30%, 10-23% and 10-24% respectively. Note that the confidence interval on the plots in figures 1 and 2 is small which increase our confidence on 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 paths apart from the shortest path and obtain content from sources in the neighborhood. This is in contrast to 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 and purge expired entries from the lookup table. In our simulation, we observe that a requester can obtain useful information from its lookup table for approximately 25% to 50% of requests for different values of \( \alpha \) and \( C \) (considering all topologies). Additionally, each time a requester searches it neighbors, it obtains fruitful information approximately 40-60% of the time.

In our simulation, we also compare the internal link load in the network [11] which provides a measure of the amount of congestion in the network. We observe that the CTR algorithm has lower link load when compared to the other approaches for ROCKETFUEL and TISCALI (8-15%), and higher link load for WIDE and GEANT (35-40%). In the results presented so far, we assume that the native cache management strategy is similar to LCD. As the CTR algorithm can work with any cache management strategy we also experimented with the LCE, CL4M and ProbCache policies. We observe from our experiments that the average reduction in delay for the GEANT topology when CTR is used along with LCE, CL4M or ProbCache as the cache management policy is 1-15%, 10-50%, and 10-50% respectively.

Overall our simulation results indicate the superior performance of the CTR algorithm over the state-of-the-art approaches for a wide range of simulation parameters and considering different topologies.

V. CONCLUSION

In this paper, we designed a characteristic time based routing algorithm (CTR) which works on top of a native routing algorithm (e.g., Dijkstra’s algorithm) and works with existing cache management and cache 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 outperforms state-of-the-art routing and cache management strategies. In future, we plan to improve the performance of the CTR algorithm by optimizing the size of the lookup table and the granularity and scope of the local search.

REFERENCES

[16] [Online]. Available: http://tinyurl.com/n9e5h23