Asymptotically-Optimal Incentive-Based En-route Caching Scheme

Ammar Gharaibeh[†], Abdallah Khreishah[†], Issa Khalil[⋆], Jie Wu[⋄]
[†]New Jersey Institute of Technology, [⋆]Qatar Computing Research Institute, [⋄]Temple University
amg54@njit.edu, abdallah@njit.edu, ikhalil@qf.org.qa, jiewu@temple.edu

***Abstract*—Content caching at intermediate nodes is a very effective way to optimize the operations of Computer networks, so that future requests can be served without going back to the origin of the content. Several caching techniques have been proposed since the emergence of the concept, including techniques that require major changes to the Internet architecture such as Content Centric Networking. Few of these techniques consider providing caching incentives for the nodes or quality of service guarantees for content owners. In this work, we present a low complexity, distributed, and online algorithm for making caching decisions based on content popularity, while taking into account the aforementioned issues. Our algorithm performs en-route caching. Therefore, it can be integrated with the current TCP/IP model. In order to measure the performance of any online caching algorithm, we define the competitive ratio as the ratio of the performance of the online algorithm in terms of traffic savings to the performance of the optimal offline algorithm that has a complete knowledge of the future. We show that under our settings, no online algorithm can achieve a better competitive ratio than $\Omega(\log n)$, where n is the number of nodes in the network. Furthermore, we show that under realistic scenarios, our algorithm has an asymptotically optimal competitive ratio in terms of the number of nodes in the network.**

***Keywords*-En-route caching, caching incentive, competitive ratio, asymptotic optimality, quality of service.**

I. Introduction

Recently, content retrieval has dominated the Internet traffic. Services like Video on Demand accounts for 53% of the total Internet traffic, and it is expected to grow to 69% by the end of 2018 [1]. Content Delivery Network (CDN) uses content replication schemes at dedicated servers to bring the contents closer to the requesting customers. This has the effect of offloading the traffic from the origin servers, reducing content delivery time, and achieving better performance, scalability, and energy efficiency [2], [3]. Akamai, for example, is one of the largest CDNs deployed, delivering around 30% of web traffic through globally-distributed platforms [4]. The problem with CDN is the necessity of *dedicated servers* and that content replication is done offline.

Several techniques have emerged to overcome the limitation of caching at dedicated servers. For example, Content Centric Networking (CCN) [5] uses the content name instead of the IP address of the source to locate the content. This allows more flexible caching at intermediate nodes. In order to implement CCN, major changes in the TCP/IP protocol needs to be performed. When a client requests certain content, the client sends an *Interest Packet* to all its neighbors, which in turn send the packet to all of their neighbors except the one where the packet came from. The process continues until a node caching the desired content is found, which in turn replies with a *Data Packet* containing the desired content.

Clearly, caching a content will reduce the traffic on the upstream path, if the same content is being requested another time by a different client. Given the limited cache capacity, the questions to answer become 'What are the factors that affect achieving the maximum traffic savings?'and 'Which contents are to be cached in order to achieve the same objective?'

Several studies try to answer the above questions. The work in [6] investigates the dependence of the caching benefit on content popularity, nodes' caching capacities, and the distance between nodes and the origin server. The performance of CCN has been evaluated in [7] under different topologies, by varying routing strategies, caching decisions, and cache replacement policies. The results also show the dependence of CCN performance on content popularity.

Several techniques for content caching have been proposed in the literature. The work in [5] presents *Always Cache*, where a node caches every new piece of content under the constraint of cache capacity. The authors in [8] provide a push-pull model to optimize the joint latency-traffic problem by deciding which contents to push (cache) on intermediate nodes, and which contents to pull (retrieve) from the origin server. Most Popular Caching caches a content at neighboring nodes when the number of requests exceeds some threshold [9]. *ProbCache* aims to reduce the cache redundancy by caching contents at nodes that are close to the destination [10]. A cooperative approach in [11] leads to a node's caching decision that depends on its estimate of what neighboring nodes have in their cache. A collaborative caching mechanism in [12] maximizes cache cooperation through dynamic request routing. In [13], nodes try to grasp an idea of other nodes' caching policies through requests coming from those nodes.

Few works targeted the caching decision problem from the point of view of optimality, or providing incentives for nodes to cache. The work in [14] presents an offline solution through dynamic programming for content placement for en-route caching. Authors in [15] characterize the optimal content placement strategy under offline settings, in which all future requests are known to all nodes in the network. The work of [16] presents an online solution but with no efficiency or optimality proofs. Other works such as [17] and [18] consider incentives for nodes to cache. However, they provide high level solutions that do not scale well with large systems. The authors in [18] consider a special case with only 3 ISPs.

This paper provides a provably-optimal online solution for the first time under a setting that brings incentives for the nodes to cache. In order to provide incentives for the nodes to cache, nodes have to charge content providers for caching their contents. Adopting such charging policies forces the caching node to provide quality of service guarantees for content providers by not replacing their contents in the future, if the node decides to cache their contents. Since the number of contents far exceeds the nodes' cache capacities, and assuming that the charging price for every piece of content is the same, then the node has no preference in caching one content over the other, forcing the node to cooperate and apply our policy that achieves asymptotic optimality.

Specifically, we make the following contributions:

(1) We design an online, low complexity, and distributed caching decision algorithm that provides incentives for the nodes to cache, and quality of service guarantees for content providers. **(2)** Our algorithm performs en-route caching and thus can be implemented without radical changes to the TCP/IP protocol stack. **(3)** Under some realistic network settings, We show that our algorithm is asymptotically (in terms of the number of nodes in the network) optimal (in terms of traffic savings). **(4)** Through extensive simulations, we show that our algorithm outperforms existing caching schemes.

The rest of the paper is organized as follows: Section II states the definitions and settings of our algorithm. Section III describes the algorithm and practical issues. Optimality analysis of the algorithm is presented in Section IV. Section V provides simulation results. We conclude the paper in Section VI.

II. Settings and Definitions

In this Section, we provide the settings under which our algorithm takes place, followed by some definitions.

A. Settings

A network is represented by a graph $G(V, E)$, where each node $i \in V$ has a caching capacity of D_i. If the node does not have caching capability, its caching capacity is set to 0. Weights can be assigned to each link $e \in E$, but we consider all links to have the same weight. The input consists of a sequence of contents $\beta_1, \beta_2, ..., \beta_m$, the j-th of which is represented by $\beta_j = (S_j, r_j, T_j(\tau))$, where S_j is the source for content β_j, r_j is the size of β_j, and $T_j(\tau)$ is the effective caching duration in which more requests are expected for β_j when the first request appears at time slot τ. For simplicity, we assume a slotted time system and that $T_j(\tau)$ is an integer multiple of slots.

For each content, we define the following values:

(1) $b_i(j)$: Number of hops on the path from node i to S_j for β_j.

(2) $W_i(\tau, j)$: The expected number of requests for β_j to be served from the cache at node i at time slot τ, if all of the caching nodes cache β_j.

(3) $t_0(i, j)$: The time when a request for β_j appears at node i.

(4) $\mathcal{E}_i(\tau, j)$: The total expected number of requests for β_j to be served from the cache at node i per time slot τ. We assume that $\mathcal{E}_i(\tau, j)$ is fixed $\forall \tau \in \{t_0, \ldots, t_0 + T_j(t_0)\}$.

(5) $\tau_0(i, j)$: The time when β_j is cached at node i. For simplicity, we denote this value hereafter by τ_0 since the values of (i, j) can be inferred from the context.

(6) $d_i(\tau, j)$: Number of hops from node i to the first node caching β_j along the path to S_j at time τ. We assume that if node i caches β_j at time τ_0, then $d_i(\tau, j) = d_i(\tau_0, j), \forall \tau \in \{\tau_0, \ldots, \tau_0 + T_j(\tau_0)\}$.

Figure 1 shows a simple network to illustrate the aforementioned definitions. In this example, we have two contents β_1 and β_2, originally stored on v_1 and v_2, respectively. The triangles in the figure represent the subnetworks containing the set of non-caching nodes connected to the caching node. The values of $W_i(\tau, j)$ represent the expected number of requests for β_j coming from the subnetwork connected to node i.

Before any requests for β_j appears at any node, each node i will send its $W_i(\tau, j)$ to all nodes on the path from node i to the source of β_j, S_j. This process will lead to the calculation of the initial values of $\mathcal{E}_i(\tau, j)$.

For example, in Figure 1, before any request for β_1 appears at any node, $\mathcal{E}_3(\tau, 1) = W_3(\tau, 1) + W_4(\tau, 1)$, to a total value of 6. This is because, starting from the initial configuration while investigating the caching of content β_1 on node v_3, all the requests for β_1 coming from the subnetworks connected to v_3 and v_4 will be served from the cache of v_3, if we decide to cache β_1 on v_3. Similarly, $\mathcal{E}_2(\tau, 1) = 9$. Later on, if v_4 decides to cache β_1, then $W_4(\tau, 1)$ will be subtracted from all nodes along the path

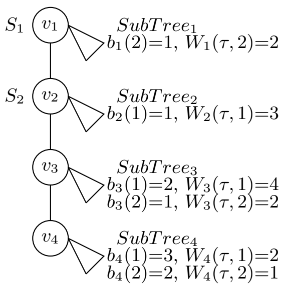

Fig. 1: Simple Caching Network.

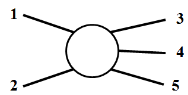

Fig. 2: A single node in CCN.

to S_1, until the first node caching β_1 is reached. This is because none of these nodes will serve the requests for β_1 coming from the subnetwork connected to v_4 after this point. In Sections III and III-B3, we provide details for the dynamic calculation and initialization of $\mathcal{E}_i(\tau, j)$, respectively.

We define the total traffic savings of caching in the time interval [0,t] as:

$$\sum_{\tau=0}^{t}\sum_{i=1}^{n}\sum_{j=1}^{m}\mathcal{E}_i(\tau_0, j)d_i(\tau_0, j)I(a_i(\tau, j)), \qquad (1)$$

where $I(.)$ is the indicator function and $a_i(\tau, j)$ is the event that β_j exists at node i at time τ. For example, referring to Figure 1, caching β_1 on v_3 alone for a single time slot will yield a saving of $\mathcal{E}_3(\tau, 1) \times d_3(\tau, 1) = (4 + 2) \times 2 = 12$.

We define the relative load on a caching node i at time τ when β_j arrives as

$$\lambda_i(\tau, j) = \sum_{\substack{k:k<j \\ k\in Cache_i(\tau)}} \frac{r_k}{D_i},$$

where $k < j$ refers to the indices of all β_k that are in the cache of node i at the time when considering β_j to be cached at node i. We use $k \in Cache_i(\tau)$ to represent the existence of β_k in the cache of node i at time τ.

As we mentioned in Section I, charging content providers for caching their contents will provide the nodes with the necessary incentives to cache. In return, the nodes have to guarantee quality of service for content providers by keeping their content cached for the required time period. We assume that content providers are charged the same to prevent the node from preferring contents with a higher prices. To this end, we consider *non-preemptive* caching to represent our system model, *i.e.*, once β_j is cached at node i, it will stay cached $\forall\tau \in \{\tau_0, \ldots, \tau_0 + T_j(\tau_0)\}$ time units. We elaborate more on $T_j(\tau)$ in Section III-B4.

B. Definitions

Offline vs. Online Algorithms: The main difference between the offline and the online algorithms is that the offline algorithm has a complete knowledge of the future. In our work, offline means that the algorithm knows *when*, *where*, and *how many times* a content will be requested. This knowledge leads to the optimal content distribution strategy that maximizes the performance in terms of traffic savings. On the other hand, online algorithms do not possess such knowledge, and have to make a caching decision for a content based on the available information at the time of the content arrival. Due to this difference, the offline algorithm's performance is better than that of the online algorithm.

Under our settings, we assume that the node does not know when a request for a content will come. However, once a request for a content arrives at a caching node, the node will know the content's size, the effective caching duration time, and the expected number of requests to be served from the cache of the caching node. Furthermore, all other caching nodes are informed about the arrival time of the request. We elaborate more on this issue in Section III-B4. For example, referring back to Figure 1, node v_3 does not know when a request for β_1 will come. Only when a request for β_1 arrives at v_3 at time t_0, does v_3 know $r_1, T_1(t_0), \mathcal{E}_3(\tau, 1)$, in addition to its own relative load, $\lambda_3(\tau, 1), \forall\tau \in \{t_0, \ldots, t_0 + T_1(t_0)\}$. However, node v_3 does not know when the next request for the same content will come.

To measure the performance in terms of *traffic savings*, as defined in (1), of the online algorithm against the offline algorithm, we use the concept of *Competitive Ratio*. Here, traffic savings refer to, but not limited, to the total number of hops saved using en-route caching, compared to the traditional no-caching case in which the request for a content is served by the content's source. The traffic savings can be based on other metrics like the actual distance or the energy consumption. Other works have used the concept of competitive ratio, but for different problems such as energy efficiency [19] or online routing [20]. Competitive ratio is defined as the performance achieved by the offline algorithm to the performance achieved by the online algorithm, *i.e.*, if we denote the offline performance as P_{off} and the online performance as P_{on}, the competitive ratio is:

$$\sup_{t} \sup_{\substack{all\ input \\ sequences\ in\ [0,t]}} \frac{P_{off}}{P_{on}}.$$

As the ratio gets closer to 1, the online performance gets closer to the offline performance. In other words, the smaller the competitive ratio, the better the online algorithm's performance.

We motivate the design of our online algorithm by the following reasoning; knowing the contents' popularities alone does not guarantee an optimal solution. The order in which the contents arrive makes a big difference.

In fact, we show that there is an upper bound on the savings achieved by the online algorithm when compared to the offline algorithm, and we develop an online algorithm that achieves that bound. We refer the reader to our technical report [21] for an example to show the difference between the offline and online algorithms.

III. Algorithm

In this Section, we present the Cost-Reward Caching (CRC) algorithm that achieves the optimal competitive ratio, along with some practical issues. We introduce the proof of optimality in the next Section.

A. CRC Algorithm

CRC takes advantage of en-route caching, *i.e.*, a request for a content is forwarded along the path to the content's source, up to the first node that has the content in its cache. The content then will follow the same path back to the requester.

In CCN, when an interest packet for a new content arrives at a node on a certain interface, the node will send the interest packet using all other interfaces. For example, Figure 2 shows a single node in CCN, where the numbers represent the interfaces of the node. When a request for β_j arrives at the node through interface number 2, and a match is not found in neither the cache nor the Pending Interest Table (PIT), the node will send the request on all interfaces except interface number 2. Our algorithm uses en-route caching, so the new interest packet is only forwarded on the single interface along the path to the content's source.

When a request for a content β_j appears at a node i at time t_0, node i sends a small control message up to the first node caching β_j along the path to the source of the content. Let w be that first node, then node w replies with a message containing r_j and the ID of node w. Every node u in the path from node w to node i stores a copy of the message, computes $d_u(t_0, j)$, and forwards the message to the next node along the path to node i. When Node i receives the message, it makes a caching decision according to Algorithm 2. If node i decides to cache β_j, it initializes a header field in the request packet to the value of $\mathcal{E}_i(\tau, j)$. If node i decides not to cache, it initializes the header field to 0.

The request packet is then forwarded to the parent node z. The parent first subtracts the value stored in the header field from its own value of $\mathcal{E}_z(\tau, j)$. Based on the new value of $\mathcal{E}_z(\tau, j)$, if node z decides to cache β_j, it adds its $\mathcal{E}_z(\tau, j)$ to the value in the header field. Otherwise, node z adds 0. The request packet is then forwarded to node z's parent, and the whole process is repeated until the request reaches the first node that has the content in its cache. The content then will follow the same path back to the requester, and every node in the path that decided to cache the content will store a copy in its cache. We describe the operation of our algorithm in Algorithm 1, and we refer the reader to Figure 3 in our technical report for an example describing the algorithm [21].

Algorithm 1 En-Route Caching

```
A request for β_j appears at node i at time t_0.
header = 0
if β_j ∈ Cache_i(t_0) then
    Reply back with β_j
else
    Send a control message to retrieve r_j, d_i(t_0, j)
    w ← first node on the path to S_j, where β_j ∈ Cache_w(t_0)
    Node w replies with r_j and ID
    ∀u ∈ Path(w, i), store r_j, d_u(t_0, j)
    for u_k ∈ Path(i, w), k = 1 : Length(Path(i, w)) do
        E_{u_k}(t_0, j) = E_{u_k}(t_0, j) − header
        Run Cost-Reward Caching algorithm
        if Caching Decision = TRUE then
            header = header + E_{u_k}(t_0, j)
```

The core idea of the Cost-Reward Caching algorithm is to assign an exponential cost function for each node in terms of the node's relative load. If the cost of caching a content is less than the traffic savings achieved by caching the content, the algorithm decides to cache. The choice of an exponential cost function guarantees that the node's capacity constraints are not violated. We show that in the next Section.

We define the cost of caching at a node i at time τ as:

$$C_i(\tau, j) = D_i(\mu^{\lambda_i(\tau, j)} - 1),$$

where μ is a constant defined in Section IV. The algorithm for Cost-Reward Caching is presented in Algorithm 2.

Algorithm 2 Cost-Reward Caching (CRC)

New request for β_j arriving at node i at time t_0

$\forall \tau \in \{t_0, \ldots, t_0 + T_j(t_0)\}$, Compute $\lambda_i(\tau, j), C_i(\tau, j)$

if $\sum_{\tau=t_0}^{t_0+T_j(t_0)} \mathcal{E}_i(\tau, j) d_i(t_0, j) \geq \sum_{\tau=t_0}^{t_0+T_j(t_0)} \frac{r_j}{D_i} C_i(\tau, j)$ **then**

  Cache β_j on node i

  $\tau_0(i, j) = t_0(i, j)$

  $\forall \tau \in \{t_0, \ldots, t_0 + T_j(t_0)\}, \lambda_i(\tau, j+1) = \lambda_i(\tau, j) + \frac{r_j}{D_i}$

else

  Do not cache

In the algorithm, when new content that is *not currently cached by node i* arrives at time t_0, node i computes the relative load ($\lambda_i(\tau, j)$) and the cost ($C_i(\tau, j)$) for every $\tau \in \{t_0, \ldots, t_0+T_j(\tau)\}$. This is because a currently cached content may be flushed before $t_0+T_j(t_0)$, thus the relative load and the cost should be adjusted for each time slot thereafter. We refer the reader to Figure 4 in our technical report [21] for an example on how to calculate the caching cost.

B. Practical Issues

So far, we developed a fully distributed algorithm that achieves asymptotic optimality in terms of traffic savings under some realistic assumptions. Before providing the optimality proof, we discuss in this section the practical issues that make the algorithm easy to implement. The major issues in our algorithm include providing incentives for the caching nodes and QoS guarantees for the content providers, the adoption of en-route caching, calculating the popularity expectation of each content, and updating the effective caching duration.

1) Providing Incentives and QoS Guarantees: In this work, the QoS measure is to guarantee the existence of the content in the cache for a certain period of time, so the content will be delivered quickly. In other words, once a caching node decides to cache a certain content, the content will not be replaced during the effective caching time of the content. Providing such a guarantee along with adopting an equal pay charging policy for all contents will provide the caching nodes with the necessary incentive to cache. Figure 3 shows the interaction between the ISP and the content provider.

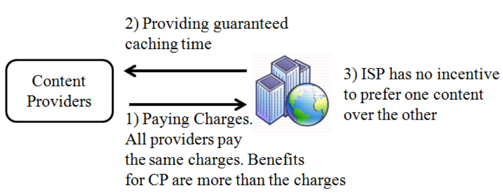

Fig. 3: Interaction between ISP and Content Provider.

We assume that the caching nodes should adopt charging policies, where every content provider is charged the same. This will prevent the caching node from preferring one content over the other. Moreover, such charging policies will enforce the caching nodes to cooperate and apply our CRC algorithm

2) En-Route Caching: In en-route caching, a request for β_j will be sent to the parent along the traditional path to the content's source, until the request reaches the first node caching the content or the content's source. The adoption of this en-route caching reduces the amount of broadcasted *Interest* packets as opposed to the currently deployed schemes in CCN, where the interest packets are broadcasted to all neighbors. Moreover, using en-route caching prevents the reception of multiple copies of the requested content as opposed to CCN. Furthermore, our algorithm can be easily implemented in the current Internet architecture.

3) Calculating the Initial Content Expectation Values: For each content, we start by building a caching tree rooted at the source of the content. The caching tree is the union of the traditional paths from the source of the content to all other nodes. We calculate the initial expectation value at a caching node for a certain content, when only node S_j holds the j-th content, based on the content's popularity and the number of end nodes in the subnetwork connected to that node. For example, in Figure 1, $W_3(\tau, j)$ at node v_3 for content β_j is proportional to the content's popularity and the number of end nodes in the subnetwork connected to node v_3. Our technical report [21] includes the algorithm for calculating $\mathcal{E}_i(\tau, j)$ for each content at each caching node before the appearance of any request at any node.

4) Effective Caching Duration: The effective caching duration of a content depends on its arrival time. For example, most people read the newspaper in a period of two hours, so the caching duration should be two hours beginning at the arrival of the first request. However, if a new request for the newspaper arrives at a node in the middle of the range and was cached by the algorithm, then the caching duration should be one hour. This requires the broadcast of the first arrival time to all other nodes in the network. The additional overhead incurred by such broadcasting is negligible compared to the reduction of the *Interest* packet broadcasting we achieve through the adoption of en-route caching.

IV. Performance Analysis

In this Section, we show that any online algorithm has a competitive ratio that is lower bounded by $\Omega(\log(n))$, then we show that our algorithm does not violate the capacity constraints, and achieves a competitive ratio that is upper bounded by $\mathcal{O}(\log(n))$ under realistic settings.

Proposition 1: Any online algorithm has a competitive ratio which is lower bounded by $\Omega(\log(n))$.

Due to space constraints, we include the proof in our technical report [21].

Before we start the proof of satisfying the capacity constraints and the upper bound, we need to state the following two assumptions:

$$1 \leq \frac{1}{n}.\frac{\mathcal{E}_i(\tau, j)b_i(j)}{r_jT_j(\tau)} \leq F \qquad \forall j, \forall i \neq S_j, \forall \tau, \tag{2}$$

and

$$r_j \leq \frac{\min D_i}{\log(\mu)} \qquad \forall j, \tag{3}$$

where F is any constant large enough to satisfy the assumption in (2), $\mu = 2(nTF+1)$, n is the number of caching nodes, and $T = \max(T_j), \forall j$. The assumption in (2) states that the amount of traffic savings for a content scales with the content's size and caching duration. The assumption in (3) requires that the caching capacity of any node should be greater than the size of any content, which is a practical condition to assume.

We start by proving that the CRC algorithm does not violate the capacity constraints. After that, we show that CRC achieves a $\mathcal{O}(\log(n))$ competitive ratio. In all of the subsequent proofs, $\tau \in \{t_0(i,j), \ldots, t_0(i,j) + T_j(t_0(i,j))\}$, where $t_0(i,j)$ is the arrival time of β_j at node i.

Proposition 2: The CRC algorithm does not violate the capacity constraints.

Proof: Let β_j be the first content that caused the relative load at node i to exceed 1. By the definition of the relative load, we have

$$\lambda_i(\tau,j) > 1 - \frac{r_j}{D_i}$$

using the assumption in (3) and the definition of the cost function, we get

$$\begin{aligned}\frac{C_i(\tau,j)}{D_i} &= \mu^{\lambda_i(\tau,j)} - 1\\ &\geq \mu^{1-\frac{r_j}{D_i}} - 1\\ &\geq \mu^{1-\frac{1}{\log\mu}} - 1\\ &\geq \frac{\mu}{2} - 1\\ &\geq nTF\end{aligned}$$

Multiplying both sides by r_j and using the assumption in (2), we get

$$\tfrac{r_j}{D_i}C_i(\tau,j) \geq nTFr_j \geq \mathcal{E}_i(\tau,j)b_i(j) \geq \mathcal{E}_i(\tau,j)d_i(t_0,j)$$

From the definition of our algorithm, β_j should not be cached at node i. Therefore, the CRC algorithm does not violate the capacity constraints. ■

The next lemma shows that the traffic saving gained by our algorithm is lower bounded by the sum of the caching costs.

Lemma 1: Let A be the set of indices of contents cached by the CRC algorithm, and k be the last index, then

$$2\log(\mu)\sum_{i,j\in A,\tau}[\mathcal{E}_i(\tau,j)d_i(t_0,j)] \geq \sum_{i,\tau}C_i(\tau,k+1) \quad (4)$$

Proof: By induction on k. When $k=0$, the cache is empty and the right hand side of the inequality is 0. When β_j is not cached by the online algorithm, neither side of the inequality is changed. Then it is enough to show, for a cached content β_j, that:

$$\begin{aligned}2\log(\mu)\sum_{i,\tau}&[\mathcal{E}_i(\tau,j)d_i(t_0,j)]\\ &\geq \sum_{i,\tau}[C_i(\tau,j+1) - C_i(\tau,j)]\end{aligned}$$

since summing both sides over all $j \in A$ will yield (4).

Consider a node i, the additional cost incurred by caching β_j is given by:

$$\begin{aligned}C_i(\tau,j+1) - C_i(\tau,j) &= D_i[\mu^{\lambda_i(\tau,j+1)} - \mu^{\lambda_i(\tau,j)}]\\ &= D_i\mu^{\lambda_i(\tau,j)}[\mu^{\frac{r_j}{D_i}} - 1]\\ &= D_i\mu^{\lambda_i(\tau,j)}[2^{\log\mu\frac{r_j}{D_i}} - 1]\end{aligned}$$

Since $2^x - 1 \leq x$ for $0 \leq x \leq 1$ and using the assumption in (3)

$$\begin{aligned}C_i(\tau,j+1) - C_i(\tau,j) &\leq D_i\mu^{\lambda_i(\tau,j)}[\frac{r_j}{D_i}\log\mu]\\ &\leq r_j\log\mu[\frac{C_i(\tau,j)}{D_i} + 1]\\ &\leq \log\mu[\frac{r_j}{D_i}C_i(\tau,j) + r_j]\end{aligned}$$

Summing over τ, i, and the fact that β_j is cached, we get

$$\begin{aligned}\sum_i\sum_\tau&[C_i(\tau,j+1) - C_i(\tau,j)]\\ &\leq \log\mu\sum_i\sum_\tau[\frac{r_j}{D_i}C_i(\tau,j) + r_j]\\ &\leq \log\mu[\sum_i\mathcal{E}_i(\tau,j)d_i(t_0,j) + \sum_i\sum_\tau r_j]\\ &\leq 2\log\mu\sum_i\mathcal{E}_i(\tau,j)d_i(t_0,j)\end{aligned}$$

■

In the next lemma, $d_i(\tau,j)$ is defined for the online algorithm.

Lemma 2: Let Q be the set of indices of contents cached by the offline algorithm, but not the CRC algorithm. Let $l = \arg\max_{j\in Q}(C_i(\tau,j))$. Then

$$\sum_i\sum_{j\in Q}\sum_\tau[\mathcal{E}_i(\tau,j)d_i(t_0,j)] \leq \sum_i\sum_\tau C_i(\tau,l)$$

Proof: Since β_j was not cached by the online algorithm, we have:

$$\begin{aligned}\sum_\tau\mathcal{E}_i(\tau,j)d_i(t_0,j) &\leq \sum_\tau\frac{r_j}{D_i}C_i(\tau,j)\\ &\leq \sum_\tau\frac{r_j}{D_i}C_i(\tau,l)\end{aligned}$$

$$\sum_i\sum_\tau\mathcal{E}_i(\tau,j)d_i(t_0,j) \leq \sum_i\sum_\tau\frac{r_j}{D_i}C_i(\tau,l)$$

Summing over all $j \in Q$

$$\sum_i \sum_{j \in Q} \sum_\tau \mathcal{E}_i(\tau, j) d_i(t_0, j) \leq \sum_i \sum_\tau C_i(\tau, l) \sum_{j \in Q} \frac{r_j}{D_i}$$
$$\leq \sum_i \sum_\tau C_i(\tau, l)$$

Since any offline algorithm cannot exceed a unit relative load, $\sum_{j \in Q} \frac{r_j}{D_i} \leq 1$. ■

Combining Lemma 1 and Lemma 2, we have the following lemma.

Lemma 3: Let A^* be the set of indices of the contents cached by the offline algorithm, and let k be the last index. Then:

$$\sum_{i, j \in A^*, \tau} \mathcal{E}_i(\tau, j) d_i(t_0, j)$$
$$\leq 2 \log(2\mu) \sum_{i, j \in A, \tau} \mathcal{E}_i(\tau, j) d_i(t_0, j)$$

Proof: The traffic savings of the offline algorithm is given by:

$$\sum_{i, j \in A^*, \tau} \mathcal{E}_i(\tau, j) d_i(t_0, j)$$
$$= \sum_{i, j \in Q, \tau} \mathcal{E}_i(\tau, j) d_i(t_0, j) + \sum_{i, j \in A^*/Q, \tau} \mathcal{E}_i(\tau, j) d_i(t_0, j)$$
$$\leq \sum_{i, j \in Q, \tau} \mathcal{E}_i(\tau, j) d_i(t_0, j) + \sum_{i, j \in A, \tau} \mathcal{E}_i(\tau, j) d_i(t_0, j)$$
$$\leq \sum_{i, \tau} C_i(\tau, l) + \sum_{i, j \in A, \tau} \mathcal{E}_i(\tau, j) d_i(t_0, j)$$
$$\leq \sum_{i, \tau} C_i(\tau, k+1) + \sum_{i, j \in A, \tau} \mathcal{E}_i(\tau, j) d_i(t_0, j)$$
$$\leq 2 \log(2\mu) \sum_{i, j \in A, \tau} \mathcal{E}_i(\tau, j) d_i(t_0, j)$$

■

Note that $d_i(\tau, j)$ in the previous lemmas is defined by the online algorithm. In order to achieve optimality using this proof technique, $d_i(\tau, j)$ of the online algorithm should be equal to $d_i(\tau, j)$ of the offline algorithm. In the next two corollaries, we show cases where $d_i(\tau, j)$ of the online algorithm is equal to $d_i(\tau, j)$ of the offline algorithm.

Corollary 1: When there is only one caching node in every path, then $d_i(\tau, j)$ of the online algorithm is equal to $d_i(\tau, j)$ of the offline algorithm, and our algorithm achieves asymptotic optimality.

Corollary 2: When every node in the path shares the same caching decision, then $d_i(\tau, j)$ of the online algorithm is equal to $d_i(\tau, j)$ of the offline algorithm, and our algorithm achieves asymptotic optimality.

V. Simulation Results

In this Section, we compare our CRC algorithm to some of the existing caching schemes.

A. Settings

We simulate the following caching schemes:

(1) CRC: This scheme represents our basic algorithm.

(2) CRC Version 2: This is similar to the CRC scheme, Version 1, except that we retrieve the content from the closest node that has the content in its cache, not necessarily along the path to the content's source.

(3) Cache All: This scheme caches every new content arriving at a caching node, as long as there is enough residual capacity to cache the new content.

(4) Random Caching Version 1: In this scheme, when a request for a content arrives at node i, the caching probability of the content depends on the content's popularity in the set of non-caching nodes connected to node i. If a randomly chosen number x between 0 and 1 is less than the content's popularity ($x \leq ContentPopularity$), and there is enough residual capacity in the cache of node i, then the content is cached.

(5) Random Caching Version 2: This is similar to Random Caching Version 1, except that the caching probability of the content depends on the content's popularity at node i, scaled by the fraction of the available residual capacity to the total capacity in the cache of node i denoted by f_i, *i.e.*, if we choose a random number x, and $x \leq f_i \times ContentPopularity$, then the content β_j is cached if there is enough room for it in the cache.

For every caching node i in the network, we assign a cache capacity D_i that is uniformly chosen in the range of [750,1000] GB. The number of the non-caching nodes connected to the caching node i is chosen uniformly at random in the range of 10 to 90 nodes.

For every content, we randomly chose one of the nodes to act as the source. Each content has a size chosen randomly in the range of [100,150] MB. The starting effective time of the content is chosen randomly. The end time is also chosen randomly within a fixed interval from the starting time. If the end time exceeds the end time of the simulation, it is adjusted to be equal to the end time of the simulation. The simulation interval is chosen to be 1000 time slots.

We simulate two different topologies, a random topology and a small world topology. For space constraints, we only provide the results for the random topology. Our technical report [21] provides all of our results.

B. Results on Random topologies

We start our evaluation on random backbone topologies, in which the caching nodes are generated as a random topology.

We simulate the effect of the number of caching nodes n in the network for three cases, $n = 30$, $n = 50$, and $n = 100$ nodes. For each case we use 10 random topologies, and report the average performance. We fix the effective caching duration and the number of contents to solely show the effect of the number of nodes on the performance of the CRC algorithm. The results are shown in Figure 4(a). As can be seen from the figure, CRC algorithm outperforms the other schemes by a range of 100% to about 300%. Another observation from the figure is that the performance of the CRC schemes increases as we increase the number of the nodes in the network. This shows that our scheme greatly benefits from adding more caching nodes to the network. It is also aligned with the property of asymptotic optimality of our scheme. On the other hand, not much improvement can be seen from the other schemes when the number of nodes is increased in the network.

We simulate the effect of changing the number of contents between 2000 and 10000 in steps of 2000. The results are averaged over 10 runs and are shown in Figure 4(b). The reason that the performance of the Cache All, Random 1, and Random 2 schemes increases and then decreases is that there is a saturation point after which the caches of the network cannot handle the requests. On the other hand, Our scheme reserves the cache capacity for contents with higher traffic savings, and achieves an improvement of 2 to 3-fold in terms of traffic savings.

Figure 4(c) shows the effect of the maximum effective caching duration for three cases, 50, 100, and 150 time slots. In this scenario, the difference between the start and end times for each content is drawn randomly from $\{1, \ldots, \max .caching\ duration\}$. The reason that the traffic savings decrease as the maximum effective caching duration increases after a certain point is that contents are cached for a longer period, so future contents are less likely to find enough residual capacity at the caching node.

In all of the results in Figure 4, the performance of CRC Version 2 is always less than the performance of CRC Version 1. This is because CRC Version 2 deviates from the settings under which we achieve optimality.

So far our performance measure was the traffic saving. In Figure 5, we measure the cost in terms of total number of hops to satisfy all of the requests. The results in Figure 5 are for a random topology with 100 caching nodes, the number of contents is 10000, and the maximum effective caching duration is 150 slots. The results in the figure shows that even when we measure the performance in terms of the total cost, our scheme reduces the cost by 30% to 50%.

In Figure 6 we measure the per topology improvement for the schemes with respect to the Random Caching Version 2 scheme. After that, the empirical CDF of the per topology improvement for 100 random topologies are plotted. The results in the figure show that there is at least one topology among the 100 different topologies where the improvement of our scheme over Random Caching Version 2 is more than 50 times. The results show that in all of the topologies the improvement of our scheme is at least 30%. In about 20% of the topologies, our scheme experiences about 4 times the improvements as that by Random Caching Version 2.

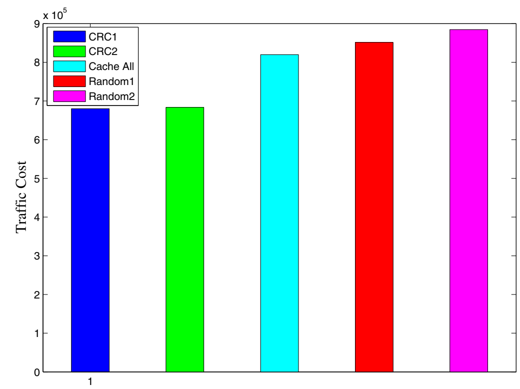

Fig. 5: Traffic cost.

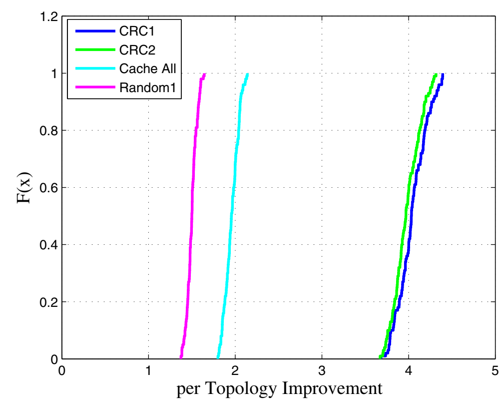

Fig. 6: The empirical CDF of the per topology improvement for random topologies with respect to Random Caching Version 2.

VI. Conclusion

Most of the previous works on caching policies assume that the caching nodes will cooperate and follow the caching policy. However, there is no incentive for the caching nodes to cooperate and cache. This work studies a new framework using en-route caching in which the caching nodes charge the content providers for storing their packets. In return, the caching nodes have to provide quality of service for the content providers by not replacing their contents within a given time, once the caching node agrees to cache the content. Moreover,

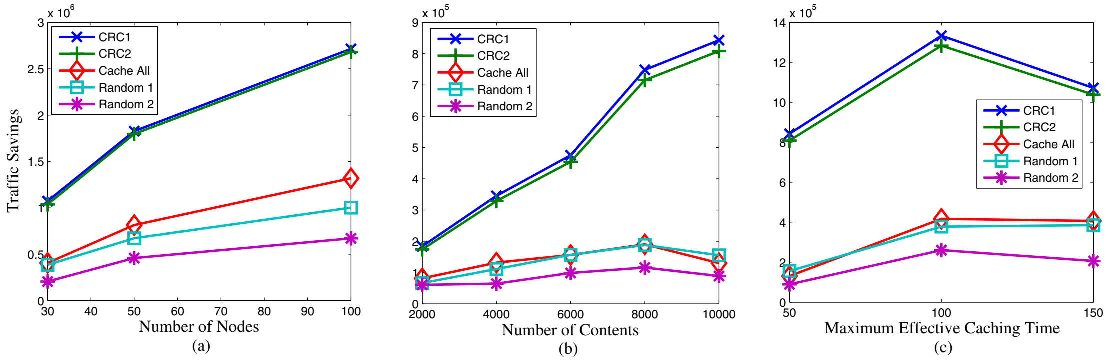

Fig. 4: The Effects of Different Factors on the Performance of the Random Topologies.

unlike CCN, the use of en-route caching does not require major changes to the TCP/IP model.

Under this new framework, we characterize the fundamental limit for the ratio of the performance of the optimal offline scheme to that of any online scheme. The offline scheme has a complete knowledge of all of the future requests, while the online scheme does not possess such knowledge. We also design an efficient online scheme and prove that the developed online scheme achieves optimality as the number of nodes in the network becomes large. Our simulation results affirm the efficiency of our scheme. Our future work includes the investigation of network coding [22], [23] under our settings.

Acknowledgement

This research is supported by NSF grant ECCS-1331018.

References

[1] V. Cisco, "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2012–2017," *Cisco Public Information*, 2012.

[2] A. Vakali and G. Pallis, "Content Delivery Networks: Status and Trends," *Internet Computing, IEEE*, vol. 7, no. 6, pp. 68–74, 2003.

[3] A.-M. K. Pathan and R. Buyya, "A Taxonomy and Survey of Content Delivery Networks," *CLOUDS Laboratory, University of Melbourne, Technical Report*, 2007.

[4] E. Nygren, R. K. Sitaraman, and J. Sun, "The Akamai Network: A Platform for High-Performance Internet Applications," *ACM SIGOPS*, vol. 44, no. 3, pp. 2–19, 2010.

[5] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking Named Content," in *Proceedings of the CoNEXT Conference*. ACM, 2009, pp. 1–12.

[6] I. Psaras, R. Clegg, R. Landa, W. Chai, and G. Pavlou, "Modelling and Evaluation of CCN-caching Trees," *NETWORKING*, pp. 78–91, 2011.

[7] D. Rossi and G. Rossini, "Caching performance of content centric networks under multi-path routing (and more)," *Relatório técnico, Telecom ParisTech*, 2011.

[8] X. Guan and B.-Y. Choi, "Push or pull? toward optimal content delivery using cloud storage," *JNCA*, 2013.

[9] C. Bernardini, T. Silverston, and O. Festor, "MPC: Popularity-Based Caching Strategy for Content Centric Networks," in *Communications (ICC)*. IEEE, 2013, pp. 3619–3623.

[10] I. Psaras, W. K. Chai, and G. Pavlou, "Probabilistic In-Network Caching for Information-Centric Networks," in *Proceedings of the second edition of the ICN workshop on Information-centric networking*. ACM, 2012, pp. 55–60.

[11] M. Fiore, F. Mininni, C. Casetti, and C. Chiasserini, "To Cache or Not To Cache?" in *INFOCOM*. IEEE, 2009, pp. 235–243.

[12] J. Dai, Z. Hu, B. Li, J. Liu, and B. Li, "Collaborative Hierarchical Caching with Dynamic Request Routing for Massive Content Distribution," in *INFOCOM*. IEEE, 2012, pp. 2444–2452.

[13] N. Laoutaris, G. Zervas, A. Bestavros, and G. Kollios, "The Cache Inference Problem and its Application to Content and Request Routing," in *INFOCOM*. IEEE, 2007, pp. 848–856.

[14] A. Jiang and J. Bruck, "Optimal Content Placement For En-Route Web Caching," in *NCA. Second IEEE International Symposium*. IEEE, 2003, pp. 9–16.

[15] J. Llorca, A. M. Tulino, K. Guan, J. Esteban, M. Varvello, N. Choi, and D. C. Kilper, "Dynamic In-Network Caching for Energy Efficient Content Delivery," in *INFOCOM*. IEEE, 2013, pp. 245–249.

[16] E. J. Rosensweig and J. Kurose, "Breadcrumbs: efficient, best-effort content location in cache networks," in *INFOCOM*. IEEE, 2009, pp. 2631–2635.

[17] J. Rajahalme, M. Särelä, P. Nikander, and S. Tarkoma, "Incentive-Compatible Caching and Peering in Data-Oriented Networks," in *Proceedings of the CoNEXT Conference*. ACM, 2008, p. 62.

[18] T.-M. Pham, S. Fdida, and P. Antoniadis, "Pricing in Information-Centric Network Interconnection," in *IFIP Networking Conference*. IEEE, 2013, pp. 1–9.

[19] S. Albers and H. Fujiwara, "Energy-Efficient Algorithms for Flow Time Minimization," *ACM Transactions on Algorithms (TALG)*, vol. 3, no. 4, p. 49, 2007.

[20] P. Jaillet and M. R. Wagner, "Generalized Online Routing: New Competitive Ratios, Resource Augmentation, and Asymptotic Analyses," *Operations research*, vol. 56, no. 3, pp. 745–757, 2008.

[21] A. Gharaibeh, A. Khreishah, I. Khalil, and J. Wu, "Asymptotically-optimal incentive-based en-route caching scheme," *arXiv preprint arXiv:1404.4639*, 2014.

[22] P. Ostovari, J. Wu, and A. Khreishah, "Network coding techniques for wireless and sensor networks," in *The Art of Wireless Sensor Networks*. Springer, 2014, pp. 129–162.

[23] A. Khreishah, I. Khalil, and J. Wu, "Distributed network coding-based opportunistic routing for multicast," in *Proceedings of the thirteenth ACM MobiHoc*. ACM, 2012, pp. 115–124.