Distributed Online En-route Caching

Ammar Gharaibeh, *Student Member, IEEE*, Abdallah Khreishah, *Member, IEEE*, Issa Khalil, *Member, IEEE*, and Jie Wu, *Fellow, IEEE*

Abstract—Content caching at intermediate nodes is an 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 in literature, including techniques that require major changes to the Internet architecture. In this work, we present a low complexity, distributed, and online caching algorithm based on content popularity. Our algorithm performs en-route caching using a simple cost-reward comparison. Therefore, it can be integrated with the current TCP/IP model. We use the concept of competitive ratio to measure the performance of any online caching algorithm, in terms of traffic savings, with respect 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. We also study several extensions to the basic algorithm and show their effectiveness through extensive simulations.

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

1 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 even further to 69% by the end of 2018 [2]. Content Delivery Network (CDN) uses content replication schemes at dedicated servers to bring the content 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 [3], [4]. Akamai, for example, is one of the largest CDNs deployed, delivering around 30% of web traffic through globally-distributed platforms [5]. 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) [6] 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 [7] 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 [8] 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.

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

• *Ammar Gharaibeh and Abdallah Khreishah are with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102 USA, Issa Khalil is with Qatar Computing Research Institute, Hamad bin Khalifa University, 13th Floor, Tornado Tower, Doha, Qatar, and Jie Wu is with the Department of Computer & Information Sciences, Temple University, Philadelphia, PA.*
E-mail: {amg54,abdallah}@njit.edu, ikhalil@qf.org.qa, and jiewu@temple.edu.

This research was supported in part by NSF grants ECCS 1331018. Part of this work was presented in IEEE MASS 2014 [1].

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. We also show the efficiency of several extensions of our algorithm with respect to the existing caching schemes.

The rest of the paper is organized as follows: We perform a literature review in Section 2. Section 3 states the definitions and settings of our algorithm. Section 4 describes the algorithm and practical issues. Optimality analysis of the algorithm is presented in Section 5. Section 6 describes the extensions of our algorithm. Section 7 provides simulation results. We conclude the paper in Section 8.

2 Related Work

Several techniques for content caching have been proposed in the literature. The work in [6] presents *Always Cache*, where a node caches every new piece of content under the constraint of cache capacity. The authors in [9] 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 [10]. *ProbCache* aims to reduce the cache redundancy by caching contents at nodes that are close to the destination [11]. A cooperative approach in [12] 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 [13] maximizes cache cooperation through dynamic request routing. In [14], 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 [15] presents an offline solution through dynamic programming for content placement for en-route caching. Authors in [16] characterize the optimal content placement strategy under offline settings, in which all future requests are known to all nodes in the network. The works in [17], [18] study the content allocation problem in the traditional IP-based network and CCN from an offline point of view by presenting a Mixed Integer Linear Program. The work of [19] presents an online solution but with no efficiency or optimality proofs. Other works such as [20] and [21] consider incentives for nodes to cache. However, they provide high level solutions that do not scale well with large systems. The authors in [21] consider a special case with only 3 ISPs. The work in [22] considers the joint caching and routing problem in Data Center networks from an offline point of view to minimize the total cost. The work in [23] presents an optimization problem for joint caching and routing to minimize the total energy consumption. The work in [24] considers caching in cellular network from an offline point of view, while the work in [25] presents an online algorithm for caching in cellular networks. Our work is different in that we provide an online algorithm with provable performance for en-route caching.

3 Settings and Definitions

In this Section, we provide the settings under which our algorithm takes place, followed by some necessary definitions. Lastly, we prove that En-route Caching is NP-hard.

3.1 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 a 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 set of non-caching nodes in the subnetwork connected to node i.

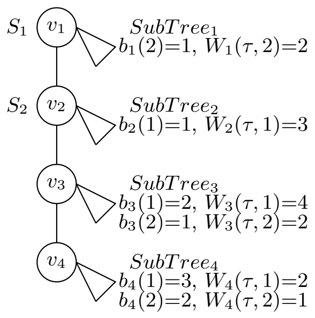
S_1
v_1
$SubTree_1$
$b_1(2)$=1, $W_1(\tau,2)$=2
S_2
v_2
$SubTree_2$
$b_2(1)$=1, $W_2(\tau,1)$=3
v_3
$SubTree_3$
$b_3(1)$=2, $W_3(\tau,1)$=4
$b_3(2)$=1, $W_3(\tau,2)$=2
v_4
$SubTree_4$
$b_4(1)$=3, $W_4(\tau,1)$=2
$b_4(2)$=2, $W_4(\tau,2)$=1

Fig. 1: Simple Caching Network.

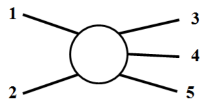

Fig. 2: A single node in CCN.

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 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 4 and 4.2.3, 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:

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

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 1, 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 slots. We elaborate more on $T_j(\tau)$ in Section 4.2.4.

3.2 Definitions

Offline vs. Online Algorithms: The main difference between the offline and the online algorithms is that the offline algorithm has complete knowledge of the future. In our work, offline means that the algorithm knows *when*, *where*, and *how many times* 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. Online algorithms 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 4.2.4. 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 optimal 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 [26] or online routing [27]. Competitive ratio is defined as the performance achieved by the optimal offline algorithm to the performance achieved by the online algorithm, *i.e.*, if we denote the optimal 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 optimal 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. Due to space limitations, the motivating example is provided in the supplemental material.

3.3 Proof of NP-Hardness

Due to space limitations, this section is provided in the supplemental material.

4 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.

4.1 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.

Algorithm 1 En-Route Caching

```
 1: A request for $\beta_j$ appears at node $i$ at time $t_0$.
 2: $header = 0$
 3: if $\beta_j \in Cache_i(t_0)$ then
 4:     Reply back with $\beta_j$
 5: else
 6:     Send a control message to retrieve $r_j$, $d_i(t_0, j)$
 7:     $w \leftarrow$ first node on the path to $S_j$, where $\beta_j \in Cache_w(t_0)$
 8:     Node $w$ replies with $r_j$ and $ID$
 9:     $\forall u \in Path(w, i)$, store $r_j, d_u(t_0, j)$
10:     for $u_k \in Path(i, w), k = 1 : Length(Path(i, w))$ do
11:         $\mathcal{E}_{u_k}(t_0, j) = \mathcal{E}_{u_k}(t_0, j) - header$
12:         Run Cost-Reward Caching algorithm
13:         if Caching Decision = TRUE then
14:             $header = header + \mathcal{E}_{u_k}(t_0, j)$
15:         end if
16:     end for
17: end if
```

For example, Figure 3 shows a simple network where a content β_1 is originally stored at S_1. We removed the triangles representing the set of non-caching nodes for the sake of clarity. If a request for β_1 appears at v_0, node v_0 will send a control message up to the first node caching β_1, which is S_1, and retrieves the values of r_1 and $d_0(t_0, 1) = 1$. Based on these values, if v_0 decides to cache β_1, it will send the request for β_1 to its parent, which is S_1, with the header field initialized to $\mathcal{E}_0(t_0, 1) = 14$. Node S_1 will simply reply with a data packet containing β_1, and v_0 will cache β_1. Later on, if another request for β_1 appears at v_5 while β_1 is still cached at v_0, node v_5 will send a control message up to the first node caching β_1, which is v_0. Node v_0 sends a message containing the values of r_1 and its ID to node v_2. Node v_2 will store the value of r_1, sets $d_2(t_0, j) = 1$, and forwards the message to v_5. Node v_5 in turn will store the value of r_1 and set $d_5(t_0, j) = 2$. Based on these values, if v_5 decides to cache β_1 it will send the request for β_1 to its parent, which is v_2, with a header field initialized to $\mathcal{E}_5(\tau, 1) = 2$. When the request reaches v_2, it will first subtract the value in the header field from its own $\mathcal{E}_2(\tau, 1)$, so the new value of $\mathcal{E}_2(\tau, 1)$ is $\mathcal{E}_2(\tau, 1) = \mathcal{E}_2(\tau, 1) - header = 4 - 2 = 2$. The reason that node v_2 has to subtract the header field from its own $\mathcal{E}_2(\tau, 1)$ is because the requests for β_1 coming from the subnetwork connected to node v_5 will not be served from the cache of node v_2 since v_5 decided to cache β_1. Based on these values, if v_2 decides *not* to cache β_1, it will add 0 to the header field and forward the request to its parent v_0. Node v_0 will simply reply with a data packet containing β_1, and only v_5 will cache β_1.

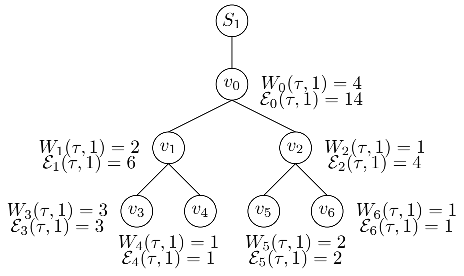
S_1
v_0 $W_0(\tau,1)=4$ $\mathcal{E}_0(\tau,1)=14$
$W_1(\tau,1)=2$ $\mathcal{E}_1(\tau,1)=6$ v_1
v_2 $W_2(\tau,1)=1$ $\mathcal{E}_2(\tau,1)=4$
$W_3(\tau,1)=3$ $\mathcal{E}_3(\tau,1)=3$ v_3
v_4 $W_4(\tau,1)=1$ $\mathcal{E}_4(\tau,1)=1$
v_5 $W_5(\tau,1)=2$ $\mathcal{E}_5(\tau,1)=2$
v_6 $W_6(\tau,1)=1$ $\mathcal{E}_6(\tau,1)=1$

Fig. 3: Simple Caching Network 2.

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 5. The algorithm for Cost-Reward Caching is presented in Algorithm 2.

Algorithm 2 Cost-Reward Caching (CRC)

```
1: New request for $\beta_j$ arriving at node $i$ at time $t_0$
2: $\forall\tau \in \{t_0, \dots, t_0 + T_j(t_0)\}$, Compute $\lambda_i(\tau, j), C_i(\tau, j)$
3:
4: 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
5:
6:     Cache $\beta_j$ on node $i$
7:     $\tau_0(i, j) = t_0(i, j)$
8:     $\forall\tau \in \{t_0, \dots, t_0+T_j(t_0)\}, \lambda_i(\tau, j+1) = \lambda_i(\tau, j) + \frac{r_j}{D_i}$
9: else
10:    Do not cache
11: end if
```

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 $\tau \in \{t_0, \dots, t_0 + T_j(\tau)\}$. 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.

For example, Figure 4 shows the relative load at a node for the next 10 time slots starting from t_0, which is the arrival time of a new content β_4. The node has three cached contents, β_1, β_2, and β_3 that are going to be flushed at times $\tau_1 = t_0+3$, $\tau_2 = t_0+9$, and $\tau_3 = t_0+7$, respectively. When a β_4 arrives at this node at $\tau = t_0$ with $T_4(t_0) = 10$, the cost calculation should include three cached contents for 3 time slots, two cached contents for 4 time slots, one cached content for 2 time slots, and 0 cached content for 1 time slot. If the total savings for caching β_4 is greater than the aggregated cost, then β_4 will be cached on node i, and the relative load is updated to include the effect of β_4.

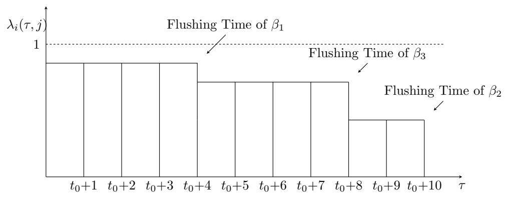

Fig. 4: Relative Load Calculation Example. The figure shows the state of the cache in one node when it considers a new content β_4 for caching at time t_0 and $T_4(t_0) = 10$. We have three contents, β_1, β_2, and β_3, that are to be flushed at times $\tau_1 = t_0 + 3$, $\tau_2 = t_0 + 9$, and $\tau_3 = t_0 + 7$, respectively.

4.2 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.

4.2.1 *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 5 shows the interaction between the ISP and the content provider.

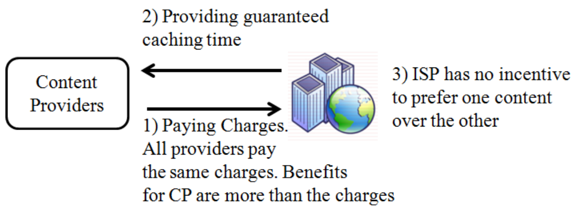
2) Providing guaranteed caching time
Content Providers
3) ISP has no incentive to prefer one content over the other
1) Paying Charges. All providers pay the same charges. Benefits for CP are more than the charges

Fig. 5: 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

4.2.2 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.

4.2.3 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.

Algorithm 3 shows how to calculate $\mathcal{E}_i(\tau, j)$ for each content at each caching node before the appearance of any request at any node. The expectations are calculated in a distributed way, where each node only needs to know the expectation values of its children in the caching tree. In the simulation, we investigate the effect of having error margins in the expectation calculation.

Algorithm 3 Initial Content Popularity Expectation Calculation

```
for each content $\beta_j = \{S_j, r_j, T_j(\tau)\}$ do
    $CachingTree(j) \leftarrow$ build the traditional path tree rooted at $S_j$
    for each caching node $i \in CachingTree(j)$ do
        Calculate $W_i(\tau, j)$
        Initialize $\mathcal{E}_i(\tau, j) \leftarrow W_i(\tau, j)$
    end for
    for each node $z \in Ancestor(i)$ in $CachingTree(j)$ do
        $\mathcal{E}_z(\tau, j) = \mathcal{E}_z(\tau, j) + W_i(\tau, j)$
    end for
end for
```

For example, referring back to Figure 3, and before a request for β_1 appears at any node, the values of $\mathcal{E}_i(\tau, j)$ are calculated as described in Algorithm 3. Take node v_2 for example, then $\mathcal{E}_2(\tau, 1) = W_2(\tau, 1) + W_5(\tau, 1) + W_6(\tau, 1) = 4$. The final expectation values for the rest of the nodes are shown in the figure.

4.2.4 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.

4.2.5 Imperfect Knowledge of the Input Parameters for CRC

Our CRC algorithm achieves asymptotic optimality under the assumption of having exact knowledge of the values of the content popularity expectation, $\mathcal{E}_i(\tau, j)$, and the effective caching duration time, $T_j(\tau)$. Nevertheless, we show the resiliency of our algorithm with respect to errors in the values of $\mathcal{E}_i(\tau, j)$ and $T_j(\tau)$ through simulations.

4.2.6 Scalability

Now we analyze the complexity of executing the algorithm. Since the execution of the CRC algorithm (Algorithm 2) constitutes of simple cost calculation and cost-reward comparison, the complexity of executing the algorithm is $\mathcal{O}(1)$ in terms of the number of contents, the content size, the number of nodes, and the number of users. As for the complexity of executing the En-route Caching algorithm (Algorithm 1), the complexity arises from executing the CRC algorithm at all nodes along the path up to the content's source (or up to the first node that is already caching the content), which is independent from the number of contents, the content size, and the number of users. We recognize that the complexity of Algorithm 1 might be $\mathcal{O}(n)$ only if the network is a line. However, in general networks, the number of nodes along the path up to the content's source is much less than the total number of nodes in the network (the average hop count is estimated to be around 15 hops according to [28], and as the contents are cached at different nodes, the average hop count will decrease).

5 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 $\Omega(\log(n))$.*

Proof: We show this proposition by giving an example network, such that the best online algorithm competitive ratio is lower bounded by $\Omega(\log(n))$. Consider a network which consists of $n+2$ nodes, as shown in Figure 6. All contents are originally placed at node S, and node C is the only node with caching capability with a unit cache capacity. All other nodes can request the contents. We consider a 2-time slots system where all contents are to be requested at the beginning of each time slot, though sequentially. Sequentially means that the algorithm has to make a caching decision for a content before considering the next one.

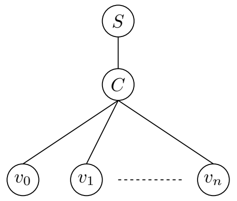

Fig. 6: Network for Lower Bound Proof.

Consider a $\log(n)+1$ phases of contents. For each phase $0 \le i \le \log(n)$, we have $1/\alpha$ identical contents, each with size $\alpha \ll 1$ and a caching time equal to 2 time slots. Contents in the same phase are destined for the same 2^i nodes. The reason behind considering a 2-time slots system is that when a node caches a content, the traffic saving is considered for future requests.

Let x_i be the fraction of contents stored from phase i and G_i be the traffic saving of the online algorithm gained from phase i, then

$$G_i = x_i 2^i$$

Consider the first k phases, then the online traffic saving of these k phases, denoted by $G(k)$, is

$$G(k) = \sum G_i = \sum_{i=0}^{k} x_i 2^i$$

The offline algorithm will cache the contents from phase k only, gaining a traffic saving of 2^k.

Now consider the ratio of the online traffic saving to the offline traffic saving:

$$\begin{aligned}\sum_{k=0}^{\log(n)} \frac{G(k)}{2^k} &= \sum_{k=0}^{\log(n)} \sum_{i=0}^{k} \frac{x_i 2^i}{2^k} = \sum_{i=0}^{\log(n)} \sum_{k=i}^{\log(n)} x_i 2^{i-k} \\ &= \sum_{i=0}^{\log(n)} x_i \sum_{k=i}^{\log(n)} 2^{i-k} \le 1 * 2 \le 2\end{aligned}$$

Hence, there exist some k such that $\frac{G(k)}{2^k} \le \frac{2}{\log(n)}$. This means that the saving of the offline algorithm is at least within a $\log n$ factor of the savings achieved by any online algorithm. □

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

$$1 \le \frac{1}{n} . \frac{\mathcal{E}_i(\tau, j) b_i(j)}{r_j T_j(\tau)} \le F \qquad \forall j, \forall i \ne S_j, \forall \tau, \tag{2}$$

and

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

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

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

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

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

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*

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

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

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

□

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 $l = \text{argmax}_{j\in Q}(C_i(\tau,j))$. 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:

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

Summing over all $j \in Q$

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

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:*

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

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

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

□

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 (e.g. at the Point of Presence (POP) of an ISP or at the base stations of a cellular network), 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 (i.e., the content is cached either by all nodes in the path or none of the nodes[1]), then $d_i(\tau,j)$ of the online*

1. This can be realized by adding a single bit to the request's header to indicate if any node decided not to cache the content.

algorithm is equal to $d_i(\tau, j)$ of the offline algorithm, and our algorithm achieves asymptotic optimality.

Note that the term *asymptotic optimality* means that our algorithm achieves the optimal competitive ratio as the number of caching nodes (n) in the network increases. Specifically speaking, the optimal competitive ratio is $\Omega(\log(n))$ as shown in Proposition 1, while the competitive ratio of our algorithm is $\mathcal{O}(\log(\mu)) = \mathcal{O}(\log(nTF))$ as shown in Lemma 3. Therefore, as n increases and becomes the dominant factor, the competitive ratio of the proposed algorithm becomes closer to the optimal competitive ratio.

6 Extensions to CRC Algorithm

In this section, we provide some extensions to the CRC algorithm. We show the efficiency of these extensions with respect to currently deployed caching schemes through extensive simulations.

6.1 Energy-CRC

The basic CRC algorithm measures the traffic savings for caching a content based on the number of hops. In this section, we provide an extension for the basic CRC algorithm where the savings are measured based on the energy saved on the upstream path from the caching node up to the first node that already has the content in its cache.

The settings for the Energy-CRC algorithm are the same for the basic CRC algorithm except for the definitions of $b_i(j)$ and $d_i(\tau, j)$, where we define $b_i(j)$ as the energy consumption from S_j to node i, and $d_i(\tau, j)$ as the energy consumption on the path from the first node that is currently caching β_j to node i along the path to S_j at time τ.

Specifically, let $\alpha_{(u,v)}$ denote the energy consumption to transfer a unit-size content from node u to node v via a direct link, then the energy consumption to transfer β_j at time τ from node w to node i, where w is the first node along the path from node i to S_j that is currently caching β_j, is given by:

$$d_i(\tau, j) = \sum_{\substack{(u,v):\\ (u,v)\in Path(w,i)}} (r_j \alpha_{(u,v)}).$$

Based on the values of $d_i(\tau, j)$, we apply the basic CRC algorithm.

In the case where renewable energy is used to power the caching nodes, the objective will be to reduce the amount of the consumed non-renewable energy. Therefore, the definition of $d_i(\tau, j)$ changes to reflect the new objective. We measure the traffic savings based on how much non-renewable (brown) energy is saved.

Specifically, let $gr_u(\tau)$ denotes the amount of available renewable energy at node u at time τ, then:

$$d_i(\tau, j) = \sum_{\substack{(u,v)\\ (u,v)\in Path(w,i)}} (\max\{r_j \alpha_{(u,v)} - gr_u(\tau), 0\}).$$

We assume that every caching node has a prior estimation of how much renewable energy will be available in the near future.

6.2 Replacement-CRC

The basic CRC algorithm provides quality of service guarantees for content providers by not replacing their contents once they are cached. Content providers, in return, are charged to provide incentives for the caching nodes based on the caching policy discussed in section 4.2.1. In this section, we present an extension for the basic CRC algorithm that allows content replacement.

The settings for Replacement-CRC are the same as for the basic CRC algorithm. However, there is no restriction on keeping a content β_j in the cache of node i for the whole effective caching duration time $T_j(\tau)$, as β_j may be replaced by another content.

We present the details of the Replacement-CRC algorithm in algorithm 4.

Algorithm 4 Replacement-CRC

```
A new request for $\beta_j$ appears at node $i$ at time $t_0$
$\forall\tau \in \{t_0, \dots, t_0 + T_j(t_0)\}$, Compute $\lambda_i(\tau, j)$, $C_i(\tau, j)$
**if** $\sum_\tau \mathcal{E}_i(t_0, j) d_i(t_0, j) \geq \sum_\tau \frac{r_j}{D_i} C_i(\tau, j)$ **then**
    Cache $\beta_j$ at node $i$
    $\tau_0(i, j) = t_0(i, j)$
    $\forall\tau \in \{t_0, \dots, t_0 + T_j(t_0)\}, \lambda_i(\tau, j+1) = \lambda_i(\tau, j) + \frac{r_j}{D_i}$
**else**
    $\forall\beta_k \in Cache_i(t_0) \cup \beta_j, \forall\tau \in \{t_0, \dots, t_0 + T_k(t_0)\}$, Compute
    $\lambda_i^k(\tau, j) = \lambda_i(\tau, j) + \frac{r_j}{D_i} - \frac{r_k}{D_i}$
    $C_i^k(\tau, j) = D_i[\mu^{\lambda_i^k(\tau,j)-1}]$
    **if** $\lambda_i^k(\tau, j) \leq 1$ **then**
        $Diff(k) = \sum_\tau \mathcal{E}_i(\tau_0, k) d_i(\tau_0, k) - \sum_\tau \frac{r_j}{D_i} C_i^k(\tau, j)$
    **end if**
    $l = \text{argmin}_k(Diff)$
    **if** $l \neq j$ **then**
        Replace $\beta_l$ with $\beta_j$
        $\forall\tau \in \{t_0, \dots, t_0 + T_j(t_0)\}, \lambda_i(\tau, j+1) = \lambda_i^l(\tau, j)$
    **end if**
**end if**
```

Algorithm 4 states that if the traffic savings gained by caching a new content β_j is greater than the caching cost at node i, then the algorithm decides to cache. Otherwise, we compare the difference between the traffic savings and the caching costs for every $\beta_k \in Cache_i(\tau)$, if it is replaced by β_j without violating the capacity constraints. We then choose the content with the minimum difference to replace with β_j.

6.3 Energy-CRC with Replacement

This extension combines Energy-CRC with Replacement-CRC, where the traffic savings are measured based on the brown energy savings, and where content replacement is allowed. We show the efficiency of this extension against currently deployed caching schemes such as Least Recently Used (LRU) through extensive simulations.

7 Simulation Results

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

7.1 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) All Cache: This is a modified version of [6]. This scheme caches every new content arriving at a caching node, as long as there is enough residual capacity to cache the new content. The content is kept in the cache for the whole effective cache duration.

(4) Random Caching Version 1: This is a modified version of [11]. 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 at node i. The popularity of a content β_j at node i denoted by Pop_j, is defined as the ratio of the number of requests for β_j coming from the subnetwork connected to node i denoted by N_i^j, to the total number of non-caching nodes in the subnetwork connected to node i denoted by N_i. Mathematically speaking, $Pop_j = N_i^j/N_i$. If we choose a uniform random number x between [0,1], and $x \leq Pop_j$, then the content β_j is cached if there is enough room for it in the cache. Otherwise, the content is not 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 uniform random number x between [0,1], and $x \leq f_i \times Pop_j$, then the content β_j is cached if there is enough room for it in the cache. Otherwise, the content is not cached.

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. The values of $W_i(\tau, j)$ are calculated by multiplying the popularity of the j-th content by the number of non-caching nodes connected to node i. The popularity of each content at node i is chosen independently from other nodes according to a Zipf distribution [29] with parameter $\zeta = 0.8$.

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.

7.2 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 to 150 slots and the number of contents to 10000 contents to solely show the effect of increasing the number of nodes on the performance of the CRC algorithm. The results are shown in Figure 7(a).

As can be seen from the figure, increasing the number of the caching nodes will result in better performance in all schemes since more contents can be cached. Another observation from the figure is that the performance of CRC schemes increases at a higher rate than other schemes 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 from 2000 to 10000. The results are averaged over 10 runs and are shown in Figure 7(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 7(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 7, 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 8, we measure the cost in terms of total number of hops to satisfy all of the requests. The results in Figure 8 are for a random topology with 100

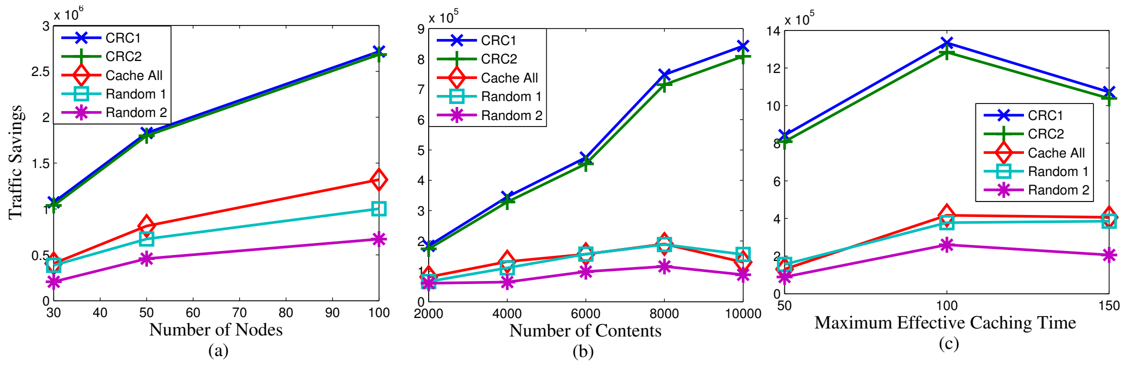

Fig. 7: The Effects of Different Factors on the Performance of the Random Topologies.

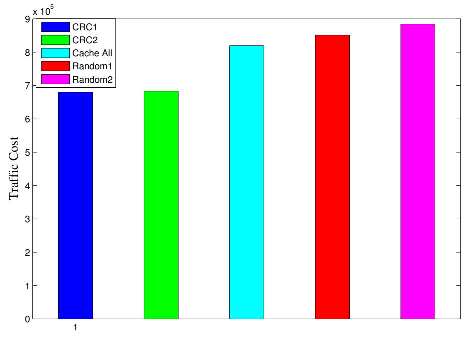

Fig. 8: Traffic cost.

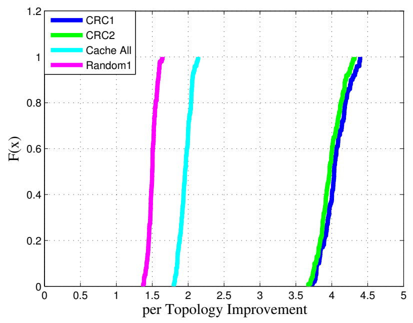

Fig. 9: The empirical CDF of the per topology improvement for random topologies with respect to Random Caching Version 2.

caching nodes, the number of contents is 10000, and the maximum effective caching duration is 150 slots. The results in the figure show that even when we measure the performance in terms of the total cost, our scheme reduces the cost by the range of 30% to 50%.

In Figure 9, we measure the per topology improvement for all schemes with respect to Random Caching Version 2 scheme. Here, we measure the performance of all schemes for 100 different random topologies. For each topology, we normalize the performance of all schemes with respect to the performance of Random Caching Version 2. Denote the performance of the CRC scheme and Random Caching Version 2 scheme for topology s as $P_{CRC}(top_s)$ and $P_{Random2}(top_s)$, respectively. We compute the normalized performance of CRC scheme with respect to Random Caching Version 2 scheme for topology s as $R_{CRC}(top_s) = P_{CRC}(top_s)/P_{Random2}(top_s)$. After that, the empirical CDF of the vector $R_{CRC} = [R_{CRC}(top_1), R_{CRC}(top_2), \ldots, R_{CRC}(top_{100})]$ for the 100 random topologies is plotted. We do the same process for the other two schemes. The results in the figure show that our scheme experiences about 4 times the improvements as that by Random Caching Version 2.

7.3 Results on a Small-word generated topology

In [30] it is shown that the Internet topology exhibits a small-world structure defined in [31]. In this Section we perform simulations based on the small world-structure.

Figure 10 is similar to Figure 7, but for the small-world topologies. The results follow the same trend as the results for the random topologies except for two differences. The first difference is that is that CRC Version 1 achieves better performance than CRC Version 2 as we increase the number of nodes. The second difference is that all of the schemes' performances increase along with increasing the effective caching time. One of the reasons is due to the sparsity of the small-world topologies, which results in the fact that the requests are distributed over multiple domains inside the topology.

7.4 Results for Replacement-CRC

We compare the performance of Replacement-CRC against the following schemes:

(1) Least Recently Used (LRU): In this scheme, when a request for a content β_j appears at node i, Least Recently Used replacement is performed at all nodes along the path from node i to the source of the content β_j.

(2) Random Replacement: In this scheme, when a request for a content β_j appears at node i, every node along the path from node i to the source of the content β_j will randomly choose a cached content to be replaced with β_j, as long as the capacity constraints are satisfied.

(3) CCN: This scheme represents the Content Centric Network as described in [6], where a request for a content is broadcasted until the closest node with a copy of the content in its cache is found. The content then follows the path from the closest node to the requester, and all nodes along that path caches the content as

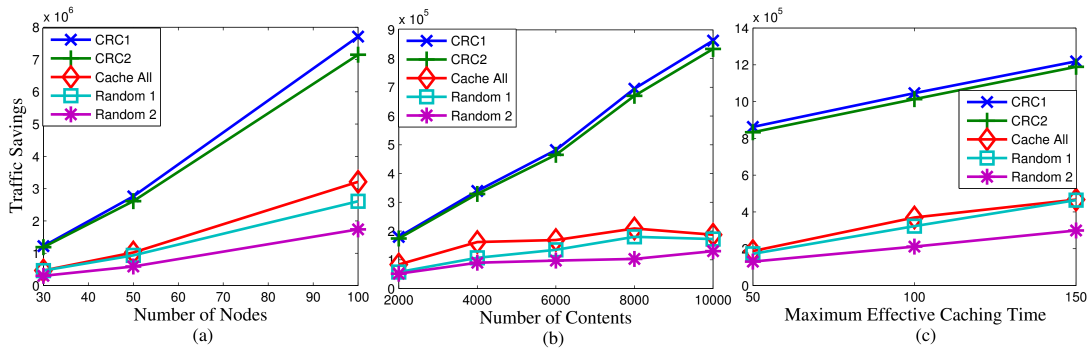

Fig. 10: The Effects of Different Factors on the Performance of the Small-world Topologies.

long as the capacity constraints are satisfied, or performs replacement using LRU if content replacement is needed.

We use the same settings as described in Section 7.1, and we simulate the effect of increasing the number of caching nodes in the network, the effect of increasing the number of contents, and the effect of increasing the cache capacity of the caching nodes. The results are shown in Figure 11.

Figure 11(a) shows the performance of all schemes as we increase the number of the caching nodes in the network. From the figure, the performance of all schemes increases with increasing the number of caching nodes. This is because adding more caching nodes will increase the overall caching capacity of the network, which results in more cached contents. Moreover, as the topology grows with adding more nodes, the average distance between the nodes in the network increases. The figure shows that Replacement-CRC outperforms the existing replacement schemes by 30% to 60%.

Figure 11(b) shows the performance of all schemes as we increase the number of contents. As we increase the number of contents, the performance of all schemes increases since more contents are available for caching. Replacement-CRC achieves better performance than the other schemes, since it is able to identify the contents with higher traffic savings and the replacement is done less frequently than the other schemes.

In Figure 11(c), we investigate the effect of increasing the caching size of the caching nodes on the performance of all schemes. We increased the caching size of each node until we reach a saturation point, where all of the nodes are able to cache all of the contents without the need for replacement. At this saturation point, all schemes achieves the same traffic savings. Another observation from the figure is that the performance of Replacement-CRC at 500GB is similar to the performance of the other schemes at 1500GB. This means that Replacement-CRC can achieve the same performance of the other schemes with only 30% of the cache capacity.

7.5 Energy-CRC with Replacement vs. Cache Size

We investigate the effect of varying the cache size on the performance of Energy-CRC with replacement algorithm as well as currently deployed caching schemes. The simulation was run on a world-wide topology consisting of 41 nodes spanning 41 different cities around the world as reported in [32]. The distances between the nodes were taken from [33] and the energy consumption for transferring data was taken from [34]. The amount of renewable energy available at each node was taken from [35]. Figure 12 shows the brown energy consumption for the Energy-CRC with replacement, Least Recently Used (LRU), and Random replacement vs. the average cache size. As we increase the average cache size of the caching nodes, the caching nodes can cache more contents and achieve lower brown energy consumption, until we reach a point where every caching nodes can cache all the contents and reach the lowest brown energy consumption. From the figure, we see that our algorithm achieves a maximum gain of 60% over other schemes when the average cache size is 500 GB. This is because our algorithm reserves the cache space for contents with high brown energy consumption. This means that our algorithm can lower the brown energy consumption without the necessity of excessive increment in the cache size.

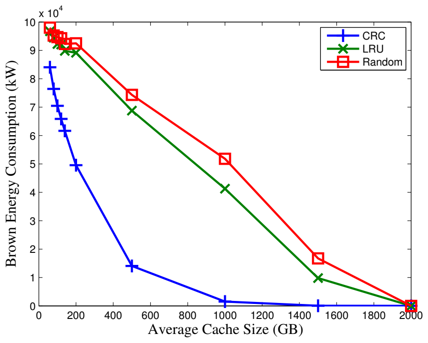

Fig. 12: Performance of CRC algorithm vs. Cache Size.

7.6 CRC vs. Error Margins in Input Parameters

We investigate the effect of having error margins in estimating the expectation values and in estimating the maximum effective caching time on the performance of the basic CRC algorithm. Although data mining techniques or stochastic process modeling based on the history [36]

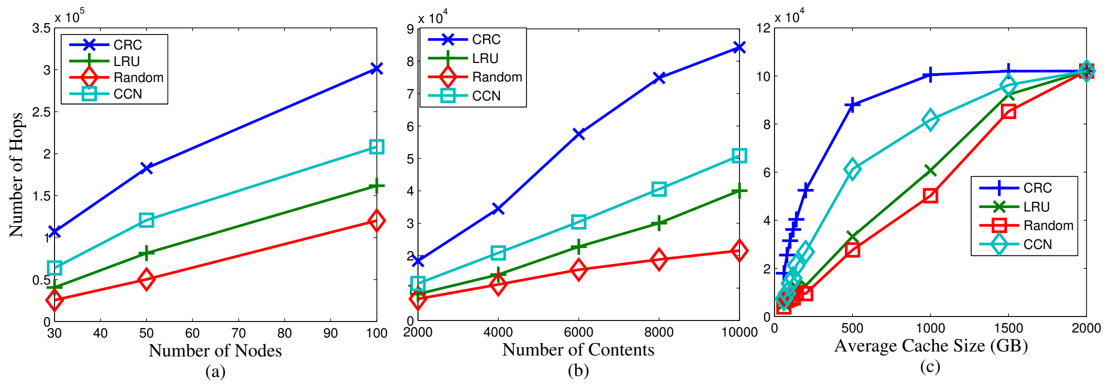

Fig. 11: The Effects of Different Factors on the Performance of Different Replacement Schemes.

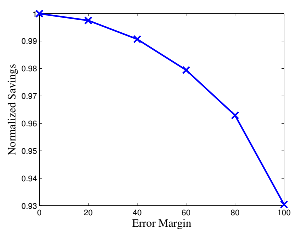

Fig. 13: Performance of CRC algorithm vs. Errors in Expectation Values.

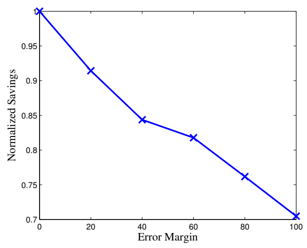

Fig. 14: Performance of CRC algorithm vs. Errors in Effective Caching Time.

can be used to provide good expectation values, we run the simulation as the error margin is changed from 0% to 100%. Denote the error margin by ϵ, then the new values of the expectation $\hat{\mathcal{E}}_i(\tau, j)$ is chosen randomly from a uniform distribution in the range $[(1-\epsilon)\times\mathcal{E}_i(\tau, j), (1+\epsilon)\times\mathcal{E}_i(\tau, j)]$, and the new values of the effective caching time $\hat{T}_j(\tau)$ is chosen randomly from a uniform distribution in the range $[(1-\epsilon)\times T_j(\tau), (1+\epsilon)\times T_j(\tau)]$. Since the basic CRC algorithm achieves asymptotic optimality when having perfect knowledge of the expectation and effective caching time values, introducing errors in these values will cause the performance of the CRC algorithm to decrease.

Figure 13 shows the performance of the CRC algorithm versus error margins in estimating the expectation values, while Figure 14 shows the performance of the CRC algorithm versus error margins in estimating the effective caching time. From the figures, we see that our algorithm is more sensitive to error margins in estimating the effective caching time. We believe this is due to the adaptability of our algorithm to error margins in estimating the expectation values, that even with such errors, the actual number of requests will still be served. However, error margins in estimating the effective caching time have greater effect on the performance of the basic CRC algorithm, since once a content is cached, the content has to stay in the cache for the whole effective caching duration time.

8 Conclusion

Caching at intermediate nodes has the advantage of bringing the contents closer to the users, which results in traffic offloading from the origin servers and lower delays. To achieve this, caching schemes such as en-route caching and CCN have been investigated. Unlike CCN, the use of en-route caching does not require major changes to the TCP/IP model. Previous works have studied en-route caching under offline settings to achieve the optimal content placement strategy. In this work, we study the framework of en-route caching under online settings.

Under this 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 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. Moreover, we introduce some extensions to the algorithm. Our simulation results affirm the efficiency of our scheme and its extensions. Our future work includes the investigation of network coding [37], [38] under our settings.

References

[1] A. Gharaibeh, A. Khreishah, I. Khalil, and J. Wu, "Asymptotically-Optimal Incentive-Based En-Route Caching Scheme," *IEEE MASS 2014*, 2014.

[2] V. Cisco, "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2013–2018," *Cisco Public Information*, 2014.

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

[4] A.-M. K. Pathan and R. Buyya, "A Taxonomy and Survey of Content Delivery Networks," *Grid Computing and Distributed Systems Laboratory, University of Melbourne, Technical Report*, 2007.

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

[6] 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 5th international conference on Emerging networking experiments and technologies*. ACM, 2009, pp. 1–12.

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

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

[9] X. Guan and B.-Y. Choi, "Push or pull? toward optimal content delivery using cloud storage," *Journal of Network and Computer Applications*, 2013.

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

[11] 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.

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

[13] 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.

[14] 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.

[15] A. Jiang and J. Bruck, "Optimal Content Placement For En-Route Web Caching," in *Network Computing and Applications (NCA)*. IEEE, 2003, pp. 9–16.

[16] 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.

[17] Y. Kim and I. Yeom, "Performance analysis of in-network caching for content-centric networking," *Computer Networks*, vol. 57, no. 13, pp. 2465–2482, 2013.

[18] M. Mangili, F. Martignon, and A. Capone, "A comparative study of content-centric and content-distribution networks: Performance and bounds," in *GLOBECOM*. IEEE, 2013, pp. 1403–1409.

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

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

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

[22] A. Khreishah, J. Chakareski, A. Gharaibeh, I. Khalil, and Y. Jararweh, "Joint data placement and flow control for cost-efficient data center networks," in *ICICS*. IEEE, 2015, pp. 274–279.

[23] A. Khreishah, I. Khalil, A. Gharaibeh, H. B. Salameh, and R. Alasem, "Joint caching and routing for greening computer networks with renewable energy sources," in *FiCloud*. IEEE, 2014, pp. 101–106.

[24] P. Ostovari, A. Khreishah, and J. Wu, "Cache content placement using triangular network coding," in *WCNC*. IEEE, 2013, pp. 1375–1380.

[25] A. Gharaibeh, A. Khreishah, B. Ji, and M. Ayyash, "A provably efficient online collaborative caching algorithm for multicell-coordinated systems," 2015.

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

[27] 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.

[28] A. Fei, G. Pei, R. Liu, and L. Zhang, "Measurements on delay and hop-count of the internet," in *IEEE GLOBECOM*, vol. 98, 1998.

[29] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *INFOCOM*, vol. 1. IEEE, 1999, pp. 126–134.

[30] T. Bu and D. Towsley, "On Distinguishing between Internet Power Law Topology Generators," in *INFOCOM*, vol. 2. IEEE, 2002, pp. 638–647.

[31] D. J. Watts and S. H. Strogatz, "Collective dynamics of small-world networks," *nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[32] "Air miles calculator." [Online]. Available: http://www.airmilescalculator.com/

[33] "Mapping the unmappable: Visual representations of the internet as social constructions." [Online]. Available: https://scholarworks.iu.edu/dspace/bitstream/handle/2022/171/wp00-05B.html

[34] V. Sivaraman, A. Vishwanath, Z. Zhao, and C. Russell, "Profiling per-packet and per-byte energy consumption in the netfpga gigabit router," in *Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2011, pp. 331–336.

[35] "Soda: Solar radiation data." [Online]. Available: http://www.soda-pro.com/web-services/radiation/helioclim-4

[36] P. Blasco and D. Gunduz, "Learning-based optimization of cache content in a small cell base station," *arXiv preprint arXiv:1402.3247*, 2014.

[37] 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.

[38] A. Khreishah, I. Khalil, and J. Wu, "Distributed network coding-based opportunistic routing for multicast," in *Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing*. ACM, 2012, pp. 115–124.

Ammar Gharaibeh is a PhD student at the ECE department of New Jersey Institute of Technology. He received his M.S. degrees in Computer Engineering from Texas A& M University in 2009. Prior to that, he received his B.S. degree with honors from Jordan University of Science & Technology in 2006. His research interests spans the areas of wireless networks and network analysis and design.

Abdallah Khreishah received his Ph.D and M.S. degrees in Electrical and Computer Engineering from Purdue University in 2010 and 2006, respectively. Prior to that, he received his B.S. degree with honors from Jordan University of Science & Technology in 2004. In Fall 2012, he joined the ECE department of New Jersey Institute of Technology as an Assistant Professor. His research spans the areas of network coding, wireless networks, congestion control, cloud computing, and network security.

Issa Khalil received his B.Sc. and M.S. degrees from Jordan University of Science and Technology in 1994 and 1996, and received his PhD degree from Purdue University, USA, in 2006, all in Computer Engineering. He joined the Faculty of Information Technology (FIT) of the United Arab Emirates University (UAEU) in August 2007, where he was promoted to an associate professor in September 2011. In August 2012, he was appointed as the chair of the Information Security department at UAEU. In June 2013, he joined the Qatar Computing Research Institute (QCRI) as a senior scientist in the cyber security group. Khalil's research interests span the areas of wireless and wireline communication networks. He is especially interested in security, routing, and performance of wireless Sensor, Ad Hoc, and Mesh networks.

Jie Wu is the chair and Laura H. Carnell professor in the Department of Computer and Information Sciences, Temple University. Prior to joining Temple University, he was a program director at the National Science Foundation and was a distinguished professor at Florida Atlantic University.

His research interests include wireless networks and mobile computing, routing protocols, fault-tolerant computing, and interconnection networks. He has published more than 600 papers in various journals, conference proceedings, and books. He serves on the editorial board of the IEEE Transactions on Computers and the Journal of Parallel and Distributed Computing. He has served as an IEEE computer society distinguished visitor. Currently, he is the chair of the IEEE Technical Committee on Distributed Processing (TCDP) and ACM distinguished speaker. Dr. Wu is a Fellow of the IEEE.