Popularity-driven Coordinated Caching in Named Data Networking

Jun Li, Hao Wu, Bin Liu, Jianyuan Lu, Yi Wang
Dept. of Computer Science
Tsinghua University, Beijing

Xin Wang
Dept. of Electrical & Computer Engineering, State Univ. of New York
Stony Brook, New York

Yanyong Zhang, Lijun Dong
WINLAB, Rutgers University
North Brunswick, NJ

ABSTRACT
The built-in caching capability of future Named Data Networking (NDN) promises to enable effective content distribution at a global scale without requiring special infrastructure. The aim of this work is to design efficient caching schemes in NDN to achieve better performance at both the network layer and application layer. With the specific objective of minimizing the inter-ISP (Internet Service Provider) traffic and average access latency, we first formulate the optimization problems for different objectives and then solve them to obtain the optimal replica placement. Then we develop popularity-driven caching schemes which dynamically place the replicas in the caches on the en-route path in a coordination fashion. Simulation results show that the performances of our caching algorithms are much closer to the optimum and outperform the widely used schemes in terms of the inter-ISP traffic and the average number of access hops. Finally, we thoroughly evaluate the impact of several important design issues such as network topology, cache size, access pattern and content popularity on the caching performance and demonstrate that the proposed schemes are effective, stable, scalable and with reasonably light overhead.

Categories and Subject Descriptors
C.2.1 [**Computer Systems Organization**]: Network Architecture and Design

General Terms
Algorithms, Performance, Design.

Keywords
Named Data Networking; Modeling; Dynamic caching; Coordinated caching; Popularity-based.

1. INTRODUCTION
The modern usage of Internet has become largely content-oriented, i.e. users tend not to care where (from which host) and how (via which protocol) to obtain a piece of content, but are more interested in fast and reliable content retrieval. Meanwhile, driven by increasing content sizes and content types, Internet traffic has been rapidly growing at an unprecedented rate. This explosive growth in traffic poses a significant challenge to the underlying network, as network capacity cannot satisfy the exponentially growing demand. Content-centric overlay networks such as Content Delivery Network (CDN) and Peer-to-Peer (P2P) are then introduced to effectively improve the content distribution efficiency. However, these incremental designs have to deploy extra application-oriented overlay mechanisms and need dedicated components for the architecture, which leads to unscalable solutions. To meet the huge demand of content dissemination in the Internet, it is necessary to rethink the future Internet architecture which can bridge the gap between name-based content delivery and the underlying host-to-host communication infrastructure.

The clean slate Named Data Networking (NDN) [1], also called Content-Centric networking (CCN) [1], is recently proposed for this purpose and widely regarded as one of the most promising architectures for future networks. Quite different from the current IP-based network, this new paradigm features name-based routing and systematic in-network caching. To be specific, in-network caching can directly cache content at each node (say router) on the forwarding path. By typically caching the popular contents at the router, in-network caching can reduce both the overall network load and the access delay. Subsequent requests no longer need to be served directly by the content source which may be far away, but can be served by a closer NDN router along the routing path. Though Internet caching has already been extensively studied, caching in NDN faces a different set of challenges.

In today's Internet (like CDNs), caches are located in specific severs and replicas can be placed in any of these caches. In NDN, however, replicas of the objects are cached along the en-route paths so the requested objects can be obtained with much shorter latencies. This design significantly differs from traditional Internet caching, and can seamlessly integrate routing and content retrieval without introducing much overhead. In addition, NDN caching is universal as it not only applies to the content carried by any protocol, but also applies to all the content from users other than the content providers (e.g. CDNs). Since there is a tremendous amount of content in the Internet, line-speed packet processing is required by NDN

ANCS'12, October 29–30, 2012, Austin, Texas, USA.

[1] We use NDN and CCN interchangeably in the paper.

to support name-based data forwarding and caching. Therefore, the storage of each router in NDN is technologically limited by memory access speed. Due to limited cache capacity on each node, careful cache placement is critical to maximizing the benefit. Our goal in this paper is to find suitable caching locations according to specific objectives, given network topology, content access pattern and various caching constraints.

Among many metrics that can benefit from caching, the main objective of our work is to minimize the inter-ISP traffic at the administrative boundaries in NDN. Our motivations for this particular objective are as follows:

- Intra-ISP links are usually over-provisioned, while inter-ISP links tend to be the bottlenecks which often suffer from congestion [2]. Reducing inter-ISP traffic will significantly alleviate congestion and thus improve the network performance at a global scale.
- Since the inter-ISP links are much more costly than internal links, the reduction of Inter-ISP traffic will greatly reduce the deployment cost for ISPs and thus cut down the inter-ISP charging [2].
- By investigating popularity-based in-network caching strategies in NDN with the special objective, we intend to thoroughly remove the caching redundancy and accommodate as many diverse content items as possible in caching system, which yields highest cache hit rate as well as minimizing inter-ISP traffic. Meanwhile, since a fraction of requests are satisfied within the ISP, caching draws the most popular content closer to the end users and helps to reduce the number of access hops, which will in turn alleviate the traffic burden within an ISP.

The other objective of the work is to explore better caching algorithms to further reduce the access delay without increasing inter-ISP traffic. In addition, a fewer number of access hops can then result in light traffic load within the ISP. To summarize, our ultimate objective is thus to improve the overall network performance in terms of inter-ISP and intra-ISP bandwidth consumption as well as access latency, with effective in-network caching.

Intuitively, coordinated caching among the routers is a promising approach to achieving reduced inter-ISP traffic, but several important issues need to be addressed: 1) Caching principle. Although NDN suggests a multi-path usage to enhance the network performance, it is a non-deterministic variation depending upon the future protocol. It is difficult to model such kind of non-determinacy. For simplicity, at least at the beginning of NDN, we restrict content caching following the en-route principle as the first step towards a full-fledged one. 2) Caching consistency. Practically, an ISP has several gateways to interconnect with provider-ISPs or peer-ISPs. Obviously, maintaining caching consistency among multiple gateways is a very challenging problem. In this paper, we assume an ISP only has a single gateway and plan to extend our solution to multiple gateways in the future work.

The main problem addressed in the paper is to provide effective caching strategies that enable NDN routers within an ISP to coordinate their caching decisions. Unlike most of the earlier work in conventional CDN caching with a focus on minimizing the access latency without considering the resulting bandwidth consumption, we make the following contributions:

- We formulate the problem-solving models with the aim of concurrently minimizing inter-ISP traffic and minimizing the average number of access hops, in order to obtain an optimal solution to the replica placement.
- Guided by optimal replica placement, we present two popularity-based caching algorithms, named TopDown and AsympOpt, where caching is coordinated implicitly among the routers on the path and the routers can make online decision independently. The proposed algorithms can significantly reduce inter-ISP traffic as well as decrease access latencies. Particularly, AsympOpt can achieve the best overall performance, which is very close to the results of optimal solutions.
- We evaluate the performance of the proposed caching algorithms with optimum solutions and study the impact of a variety of factors such as network topology, request pattern, object popularity and cache capacity etc. Simulation results demonstrate that the proposed algorithms exhibit stability and scalability under a wide range of workloads without introducing much overhead.

The rest of the paper is organized as follows: Section 2 surveys the related work. The system model and problem statement are presented in Section 3. Section 4 describes the coordinated dynamic caching scheme. The simulation model, impact factors and the simulation results are discussed in Section 5. Finally, we conclude this paper in Section 6.

2. RELATED WORK

Internet Caching plays an important role in enhancing content delivery, as caching can reduce network traffic and alleviate server load, thereby decreasing access latencies and improving user-perceived Quality of Experience. A large body of research has been done in this field which has led to great successes, such as web proxies [3], object caches [4] and CDN [5], while caching in NDN is a new area and very little investigative work has been published, to our best knowledge. The similar idea of having Internet routers cache passing data as suggested in NDN has been studied in en-route caching, which equips each node in a network with a cache and enables the nodes along the routing path to cache formerly requested objects in the network for future reuse [6]. Dong et al. [7] presented independent content caching and replacement algorithms for intermediate nodes with limited storage, but the work can only reach local optimality with the mathematical model. Due to the complexity of solving the optimization problem, this scheme is limited to be used for small-scaled network. Walter et al. [8] presented an in-network caching architecture based on content routers, which discovers

resources in the network proximity. However, cooperation is limited to neighborhood and cannot reach the optimum in the network. In contrast, we strive to thoroughly remove the redundancy with implicit cooperation among caches in order to efficiently utilize the available cache space and minimize inter-ISP traffic.

The most recent works [9] [10] [11]have dived into the study on CCN caching. D.Rossi et al. [9] presented a quite thorough simulation study of CCN caching performance. However, they assumed that named content in CCN can in principle take any path in the network, while we argue that routing to the content sources is determined by the CCN routing protocol and content objects are cached along the en-route path. Kideok et al. [11] proposed a lightweight caching scheme named WAVE, in which the popular contents can be pushed closer to the end users. However, WAVE cannot eliminate the caching redundancy from perspective of the whole network and is limited to achieve good caching performance locally. Psaras et al. [10] focused on modeling caching trees of content-centric networking, which is complementary to our work.

In order to achieve better caching performance, we propose dynamic caching schemes based on content popularity. Traditional approaches towards network caching have placed large caches at specific points in the network, with little or no coordination between the caches. In contrast, routers with limited storage in our schemes independently make the caching decision based on the recent content requests of its subordinates; thus nodes implicitly share their caching information and coordinate in minimizing the redundancy. Besides, the proposed caching strategy can work online to adapt to various network changes.

3. SYSTEM MODEL AND PROBLEM STATEMENT

In this section, we first briefly introduce the NDN architecture and then present a system model of the routers/caches within a single ISP, which are followed by formulating our caching problem.

3.1 System Model

NDN architecture is featured with the availability of built-in network storage and receiver-driven chunk level transport. That is, each router on the Internet is equipped with a cache and can replicate passing contents to serve the subsequent requests without the need of forwarding them to their source servers. In addition, the unit used for transmission is the segment of content, named chunk.

NDN uses a globally unique identifier (e.g. a hash function of a URL) to recognize a content object. The content retrieval procedure is as follows: (1) The content names are published into network by different Internet applications. (2) An end user who is interested in a particular type of content sends out an *interest packet* with the name of the requested content. The *interest packet* propagates along the routing path towards the content source. (3) Each router receiving an *interest* checks whether the requested content is present in its local cache by looking up a *Content Store* (CS) table. If there is a hit, the router sends the matched data piece to the requester along the reverse path. Otherwise, it forwards the request to the interfaces determined by the *Forwarding Information Base* (FIB). Ongoing requests are recorded in a *Pending Interest Table* (PIT) for later sending back the requested data through the reverse path towards the sources of the interests. (4) When the target content travels from the source (or cache) downward to the requester, the router on the forwarding path will determine whether to replicate the content according to the caching strategy.

The requested content objects are distributed at the repositories outside the ISP in the model, due to the fact that the majority of the content requests can not be satisfied within an ISP without caching. Generally it takes fewer hops to retrieve a content object inside the ISP, so we do not consider caching those objects whose sources happen to reside in the investigated ISP, for fully exploiting cache capacity for better content retrieval performance. Hence, we simplify the model as illustrated in Figure 1. When an edge router (say R_{14}) receives an *interest* for content object (say O_1), it will forward the *interest* towards the content source guided by FIB. When the target content objects are sent in reply to *interest packets* and travel along the way back to the requester following the chain of PIT entries, the NDN router on the path determines whether to replicate the passing content according to the caching strategy. As in the example, node R_4 replicates the content object O_1 and serves the later requests for O_1 from the edge routers within its subtree, that is, R_{12}, R_{13}, R_{14}.

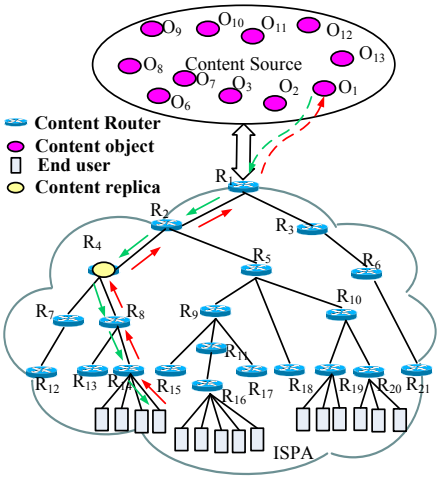


Figure 1. An illustrated caching model

From the above, all the content resources retrieved from the sources outside the given ISP pass through its gateway and are cached at some inner nodes on the path to the requesters. Thus, our caching infrastructure is hierarchical,

where the requests are only interfaced at the end nodes of the hierarchy and routed towards a single root cache (i.e., ISP gateway). Accordingly, the flat graph of ISP topology can be simplified as a tree model, with gateway being the root node. Figure 1 shows the example tree abstracted from the router-level topology of an ISP.

The purpose of our work is to design and evaluate the NDN caching strategies in order to achieve a system goal. To make full use of in-networking caching, we focus on minimizing the inter-ISP traffic and reducing the number of access hops by caching the requested contents within an ISP using the available caching space of routers. To be specific, we address the problem of online coordinated caching decision in the following environment: once a request for a piece of chunk fails to be satisfied by the caches inside an ISP, the requested object item will be fetched from the external source and traveled via the gateway of the investigated ISP. We aim to design appropriate caching strategies which can work online to achieve the best gain. Each router in the ISP independently determines whether to replicate the passing items in its associated cache. The objective of our work is to minimize the inter-ISP traffic and also minimize the average number of access hops by caching frequently requested objects at selected routers inside the ISP.

3.2 Problem Statement

It is impracticable to find the solution to an optimization problem including all the optimization objectives. In our case, the objective of minimizing inter-ISP traffic and the objective of minimizing average access hops are not always consistent. In this subsection, we first give out the definition and notations, and then we formulate individual problem-solving model for each objective and obtain the optimal solution to the replica placement respectively. The solutions are used to guide the caching design and taken as bounds to the performance.

3.2.1 Definition and Notations

According to the tree topology of our system model in Figure 1, we consider a caching tree (routing tree), whose node set N is partitioned into two parts: a set of end nodes U and a set of intermediate nodes ($N-U$), and the gateway node R_1 being the root of the tree. End nodes are responsible for managing the interest requests from their users. Each node j ($j \in N$) is equipped with a cache, whose storage capacity is C_j .

Let $[i \to j]$ be the unique routing path from node i up to node j , $i, j \in N$. We say j is the upstream node (or ancestor), while i is the downstream node (or descendant), if node j is closer to the root R_1 than node i . Let $Ancestors(j)$ denote the set of ancestors of node j , i.e., the nodes in the unique en-route path $[j \to R_1]$ (Node j is included).

Let O be the set of interested objects in the Internet. For the requested content object o_k ($o_k \in O$), we define:

s^k : The size of the object o_k .

q_i^k : Request rate for object o_k at node i ($i \in U$).

$Cache^k$: The node set which replicates the object o_k and serves the requests for object o_k within an ISP.

A_i^j : The hop counts between end node i ($i \in U$) and node j ($j \in Ancestors(i)$).

X_j^k is a Boolean variable which equals to 1 if node j is a cache of o_k , that is,

$$X_j^k = \begin{cases} 1 & j \in Cache^k \\ 0 & j \notin Cache^k \end{cases} \qquad j \in N \tag{1}$$

3.2.2 Optimization Problem for Minimizing inter-ISP Traffic

Given a set of caches and request rates, our optimization problem is to decide which objects should be stored in the caching system and where to cache them, in order to satisfy the objective of minimizing inter-ISP traffic. This optimization objective can be interpreted as maximizing the inter-ISP traffic savings by the caching system, with the limited cache storage at each router. That is,

$$\max\left(\sum_{k \in O} \sum_{i \in U} \sum_{j \in Ancestors(i)} q_i^k s^k X_j^k \right) \tag{2}$$

$$\text{s.t.} \quad \sum_{k \in O} s^k X_j^k \le C_j , \text{ for all } j \in N \tag{3}$$

$$\sum_{j \in Ancestors(i)} X_j^k \le 1 , \text{ for all } i \in U \tag{4}$$

The first constraint is a capacity constraint, which requires the amount of storage space occupied by objects cached at a router not exceed its capacity. The second constraint means that the number of the caches which serve the requests of object o_k from end node i is no more than 1, i.e., there is at most one replica for object o_k in the unique en-route path $[i \to R_1]$ from this end node i to the root R_1 .

3.2.3 Optimization Problem for Minimizing Average Access Hops

The objective of minimizing the average number of access hops for requesting contents can be formulated as follows,

$$\min(H_{avg})$$

$$\text{s.t.} \quad \sum_{k \in O} s^k X_j^k \le C_j , \text{ for all } j \in N$$

With the objective, our optimization problem is to decide which objects should be stored in the caching system and where to cache them, given a set of caches and requests rates. More specific, we try to find the optimal replica placement, i.e., the values of X_j^k ($o_k \in O$, $j \in N$), which leads to the smallest number of access hops on average, with the constraint of cache capacity. Thus, the problem

turns out to be a Mixed Integer Problem. In addition, for ease of mathematical expression, we change the objective of minimizing average response hops to the equivalent objective of maximizing the saved average hops by caching in network. The optimization problem is therefore given as,

$$\max\left(\sum_{o_k\in O}\sum_{i\in U}\sum_{j\in Ancestors(i)} q_i^k s^k (HOP - A_i^j) X_j^k\right) \quad (5)$$

$$\text{s.t.} \quad \sum_{o_k\in O} s^k X_j^k \le C_j, \ \text{for all} \ j\in N \quad (6)$$

$$\sum_{j\in Ancestors(i)} X_j^k \le 1, \ \text{for all} \ i\in U \quad (7)$$

Here, *HOP* is the average number of router traverse hops without caching in network. In today's Internet, packets traverse an average of around 12 to 14 hops. We remove 2 hops which account for the hops between the router and the user/content source and take 11 hops for *HOP* in our evaluation section [12] [13]. The first constraint is a capacity constraint, which requires the objects cached at a router could not exceed its storage capacity. The second constraint means that the number of the caches which serve the requests of object o_k from end node i is no more than 1, i.e., there is at most one replica of object o_k caching at the nodes falling in the set of the $Ancestors(i)$, in order to make the benefit of accommodating as many diverse objects as possible in caching system.

Each of the above linear programming problems is a Mixed Integer Problem (MIP). We obtain the optimal solution using GLPK [14], given the caching tree and request rate of each object at the end nodes. The complexity of solving the MIP problem mainly depends on the network size, content population and cache capacity of each router. For the given topology with 50 nodes (each node can accommodate 10 content items) and 1000 object items in the network, it takes several seconds to determine the cached objects and their optimal caching locations. However, as the tree topology enlarges to model the real ISP scale, say with the scale of 200 nodes (each accommodating 20 content items) and 5000 content items (which is still much fewer than the actual number of contents in Internet), it will take more than 3 weeks to solve the optimization problem on a high-end server. Obviously, the huge computational power prohibits the real-time online decision making. Again, considering the fact that the object request rates are not priori-known and typically difficult to be predicted, the optimal caching decision is impractical to be achieved. Thus instead, we will turn to developing online caching algorithms in Section 4, while taking the optimization solutions as the guide to caching algorithm design and the bounds for performance evaluation.

4. SYSTEM DESIGN AND CACHING ALGORITHMS

As mentioned earlier, the system can be modeled as a tree-like routing topology shown in Figure 1. We first clarify the two notations *Level* and *Tier* for the tree topology, as illustrated in Figure 2. *Tier* is denoted as the distance from the intermediate node to the closest end node (each end node being Tier 1), measured by the number of hops. Obviously, lower tier caches are closer to the end users. In contrast, *Level* is denoted as the distance from each node to the root R_1 (R_1 being Level 1), also measured by the number of hops. End nodes (nodes in Tier 1) correspond to the highest level caches and are responsible for monitoring the requests from the end users and the root node corresponds to the lowest level cache. The objects contained in the caches at the lower level can be shared and accessed by sub-tree nodes at the higher level. A user request travels from an end node (requester) towards the root, until the requested object is found. If the requested object cannot be found even at the root level, the request is redirected to the source content server which contains the interested object. Once the object is found, it is sent along the reverse path towards the requester. Each cache along the forwarding path independently decides the content replication according to a chosen caching strategy, which is determined by dynamic caching algorithms proposed later in this section.

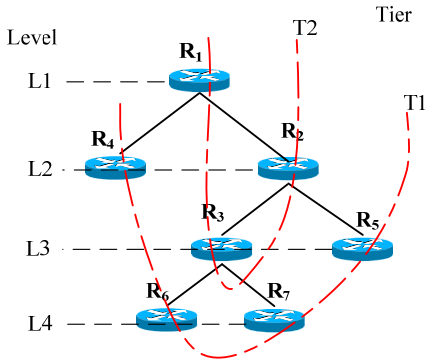


Figure 2. Toy model of caching topology

Our primary goal is to maximize the gain in cross-domain traffic from the ISP's perspective and in access latency from the user's perspective by dynamically creating replicas for popular contents. In this section, we first introduce the procedure of dynamic caching. We then propose two coordinated dynamic caching strategies to achieve our design goal. The strategies are also expected to run online.

4.1 Dynamic Caching

The access pattern at end nodes changes over time, so the caching strategy has to track the object request rate at end nodes, and adapts the caching decision to better achieve the design goal. The popularity of an object is measured by the request rate for the object and is generally stable during a short time period. The caching algorithm is invoked at regular time intervals to determine the replica placement positions based on the most recent histories of request statistics as well as the available cache capacity. As the object popularity varies over time, only the most recent

access histories are kept to reduce the memory occupation. The invocation interval for a caching decision is chosen according to the total arrival rates at the end nodes and the change frequency of the content popularity. A shorter interval is preferred for a higher request rate in order to adapt the caching decision quickly to the changing access patterns. However, a shorter interval incurs a higher overhead. On the contrary, a longer interval is suited better to the stable access patterns.

We go a step further into the details of the popularity-based dynamic caching process. Each access node maintains a set of request counters for selected objects and dynamically created object items and calculates the historical request statistics periodically to form the access profile. Herein, the selected objects refer to the most popular objects presented in the last caching interval, based on the observation that recently popular files will tend to be accessed more frequently than others in the near future. Those sustained in dynamic items region are the emerging popular contents in the caching round. Dynamic items are maintained according to the object arrival rate and those with longer arrival interval of requests for a particular object will make room for the subsequent requested objects. As a result, unpopular objects are screened out and the profile of content catalog is dramatically cut down.

Each node spreads the request information along the determined routing paths. In this way, each node gets the needed request profile and individually makes the placement decision according to the caching scheme. When an object is fetched after being requested, each router on the paths determines whether to replicate it or not based on the object tags which are set by the recent caching decision made in this round according to the caching algorithms presented in the next subsections. An object can be tagged by an *update* mark or a *replication* mark. The *replication* mark indicates that the fetched object can be cached at the router upon its arrival and its replica creation time is then set to the current time. Meanwhile, the *replication* mark turns to an *update* one. The *update* mark indicates that there is a replica of the object in the current cache, so a request for the object can be served by the cache until the lifetime of the replica exceeds a preset value. Once an object is obsolete, the request for this object has to be forwarded to the source instead of being served by the cache and the *update* mark of the object turns to a *replication* mark. Here the threshold of object lifetime is introduced for the purpose of keeping the cached object refreshed.

4.2 TopDown Caching Algorithm

TopDown caching algorithm consists of two procedures: information aggregation and decision making. In this algorithm, each node makes its caching decision for each object according to its popularity measured by the aggregated request statistics of its subtree.

The algorithm is invoked at the commencement of a new interval and starts the process of information aggregation from end nodes upwards to the root. To illustrate the algorithm, we use node R_j as an example, be it the end node or the intermediate node. An end node obtains the most recent request history covering the latest interval by calling *GetReqHistory*, while an intermediate node aggregates request records sent from all of its children by calling *Aggregate*. The obtained request records are then sorted in the descending order of the number of requests (#request) and those whose #request is less than the threshold are removed. The threshold can be used to screen out the unpopular objects for reducing the computing complexity. The result is stored in A_j. In this way, TopDown gets sorted request records at each node by aggregating request records from the bottom *level* (the highest *level*) up to the top *level* (the root). Here, the *level* is defined as the distance from the nodes in this layer to the root in terms of hops, as above mentioned.

Algorithm 1 TopDown Caching Algorithm

Information Aggregation (ReqHistory**)**

1 for layer ← bottom level to root do
2 for each node R_j in the layer do
3 if $R_j \in U$ then
4 ReqRec[]$_j$ ← Get ReqHistory( )
5 Else
6 ReqRec[]$_j$ ← Aggregate (Children(R_j), Records)
7 end if
8 A_j ← Sort-Dec (ReqRec[]$_j$, threshold)
9 end for
10 end for
11 Return (A_j)

TopDown decision making (A_j)

12 for layer ← root to bottom level do
13 for each node R_j in the layer do
14 for each record $r \in A_j$ do
15 if r.ObjectID Exist-In R_j 's CachedTable then
16 MarkUpdate (r.ObjectID)
17 Append ($DelTable_j$, r.ObjectID)
18 else if Available-Space ≥ Size (r.ObjectID) then
19 MarkReplicate (r.ObjectID)
20 Append ($DelTable_j$, r.ObjectID)
21 end if
22 end if
23 end for
24 Delete ($A_{Children(R_j)}$, $DelTable_j$)
25 end for
26 end for

With the request information, the decision making process is from the top to the bottom. Line 15 starts with the record from the top of A_j and fetches the record in turn from the top till reaching the number of objects (or chunks) which the node can cache. If the *ObjectID* of the record is found in the R_j 's *CachedTable* which is a list of cached object items determined by the previous interval, an *update* mark is set to the *ObjectID*. Otherwise, a *replication* mark is set

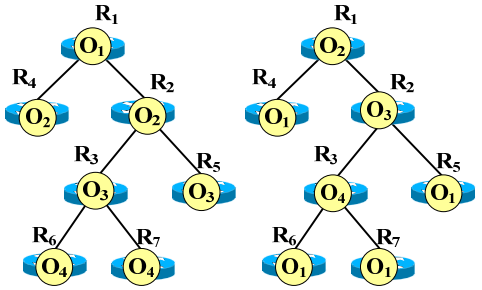
to the *ObjectID*. Thus, the corresponding router storage is assigned for the object item tagged by *update* and *replication* in this caching round and then actually implemented upon the arrival of the retrieved content.

For eliminating redundancy, Topdown creates an additional deletion tracking table called *Deltable*. The current node should append *ObjectID* of those objects marked to be cached locally (including *update* and *replication*) to a deletion tracking table and send the table to all its children. Each child will delete the corresponding records existing in the deletion tracking table from its request record table, and then makes the caching decision based on the table containing the local request records.

According to the TopDown algorithm, a caching example is illustrated in Figure 3 (a) with the request profile in Table 1. We assume each router can only accommodate one object in the example. In Figure 3, R_i stands for a content router, and O_j stands for an object. The circles with O_j indicate where the replicas of O_j are placed according to TopDown caching algorithm.

Table 1. Content request profile

Object	Request at R4	Request at R5	Request at R6	Request at R7
O_1	20	20	20	20
O_2	10	10	10	10
O_3	8	8	8	8
O_4	6	6	6	6
O_5	5	5	5	5
O_6	4	4	4	4



(a) TopDown (b) AsympOpt

Figure 3. Examples for Caching Algorithms

From the caching example, we can see that TopDown algorithm makes caching decision from root down towards end nodes and places the most popular content objects at the lower *levels*. We can also observe that TopDown can thoroughly eliminate the caching redundancy with global coordination, which can achieve the objective of minimizing inter-ISP traffic and maximizing cache hit rate.

4.3 AsympOpt Caching Algorithm

TopDown caching algorithm can eliminate caching redundancy and accommodate more diverse content objects, which increases the gain in reducing user access latency. From the observation on Figure 3(a) and the solution to the optimization for minimizing user access latency, we can draw the most popular content objects closer to the end nodes and further reduce the access delay, without sacrificing performance of hit rate or inter-ISP traffic. An example of caching placement determined by AsympOpt caching algorithm, which is an improvement of TopDown, is presented in Figure 3(b). AsympOpt caches the globally most popular content objects from the lowest *tier* to the highest *tier*. Each node in T1 (*tier* 1, end nodes layer) is tagged by first priority P_1 in T1. In other tiers, those having no parent in the same tier are tagged as P_1 in this tier. Otherwise, the node should be tagged after its parent's priority. Take the model in Figure 2 for instance. R_1, R_2, R_3 are in the same tier T2. R_1 has no parent in the this tier, while R_2 's parent is R_1 and R_3 's parent is R_2. Therefore, R_1 is tagged by priority P_1, R_2 is tagged by priority P_2, and R_3 is tagged by priority P_3. The values of Tier and priority are determined by the routing topology.

At starting time, all cache stores are empty and caching system will begin with *Information Aggregation* procedure of the TopDown algorithm in order to get the global popularity rank *GloRank* and distribute the result to each node. The subscript of Object O_j stands for its *GloRank*. Information Aggregation procedure is iterated periodically during the execution of AsympOpt caching. The iterative period is determined by the popularity change.

Given the routing topology and caching capacity C_j of router R_j, we have *StartValue*(j) and *RangeValue*(j) of R_j. Herein, *StartValue*(j) is determined by the R_j 's Tier and its priority, and *RangeValue*(j) is the total cache capacity on R_j 's forwarding route. With the value, router R_j picks out C_j global popular contents to be cached locally. If the local popularity of some object is not consistent with the global popularity and exceeds the range of popularity difference, those which have relatively low *GloRank* but are locally popular will be cached. In this way, we try to compensate for the difference between the global popularity rank and the local one (Line 10~Line 23).

Aggregate procedure and Mark procedure are similar to the corresponding procedures in Algorithm 1 except that the records of cached contents will be removed from the request profile which is sent to the parents. For brevity, we don't describe the procedures in detail in the paper.

We take an example to explain AsympOpt caching algorithm when the local and global popularity rank are not consistent. Let's consider the end node R_6. Its *StartValue*(6) =1 and *RangeValue*(6) = 4, provided each router can only cache one content object. If the request number for O_1 ranks lower than 5 in popularity and thus local popularity of O_1 exceeds the range value, O_5 which is not in the expected caching list for being globally popular but relatively locally popular is cached at R_6; if the request

number for O_1 ranks the first four in popularity, O_1 is still chosen to be cached.

Algorithm 2 AsympOpt Caching Algorithm

```
for layer ← lowest tier to highest tier do
 for each node R_j in the layer sorted by ascending priority do
    if R_j ∈ U then
       ReqRec[]_j ← Get ReqHistory( )
    Else
      ReqRec[]_j ← Aggregate (Children( R_j ), Records)
    end if
 end for
for each node R_j in the layer sorted by descending priority do
  A_j ← Sort-Dec (ReqRec[]_j, threshold)
 st ← StartValue(j ) obtained from child in the closest tier
 while k<= C_j do
 if each GloRank(st) Exist-in first C_j records of A_j then
  Mark(GloRank(st))
 else
    count ++
 end if
 st ++
end while
for k=1 to count do
 re ← top record of A_j
 if re.ObjectID is out of the RangeValue(j) then Mark( re )
end for
SendtoParent ( A_j )

Mark( r)
  if r.ObjectID Exist-In R_j 's CachedTable then
      MarkUpdate (r.ObjectID)
      Remove ( A_j , r)
  else
      MarkReplicate (r.ObjectID)
      Remove ( A_j , r)
  end if

Aggregate (Children( R_j ), Records)
Initialize (ReqRec[]_j)
for each child R_k in Children ( R_j ) do
   for each record r ∈ A_k do
      if r.objectID Exist-In ReqRec[]_j then
         r_j. ReqCount= r_j. ReqCount + r. ReqCount
      else
         Insert the record r
      end if
   end for
end for
return (ReqRec[]_j)
```

5. PERFORMANCE EVALUATION

In this section, the experimental results of the caching algorithms are presented and analyzed.

5.1 Simulation Settings

Since NS2 has some limitation to simulate the new NDN paradigm [10], we establish our own simulation to evaluate the presented caching schemes.

5.1.1 Simulation environment

Our algorithms are tested and evaluated using both synthetic and real network topologies that have different structural properties.

We employ the Georgia Tech Internetwork Topology Model (GT-ITM) toolkit [15] to generate the router-level network topology using the Transit-Stub model. A shortest path tree for rooting level topology is abstracted from the generated graph. Each node is equipped with a cache. User requests are sent to the end nodes of the tree. Each node, including the end nodes, checks the requested object in its local cache before forwarding the request. If the object is not found, the request will be forwarded to the next-hop node along the routing path towards the root until it reaches a node that caches the requested object or out of the ISP via a gateway node towards the source of the requested object. In either case, an object copy is sent along the reverse path to the requesting end node. Each node on the path can replicate the passing object based on the caching criteria.

We employ different network topologies in the evaluation, including the topology of University of Wisconsin AS59, UUNet Alternet AS701and UNINETT AS224 as listed in Table 2. These topologies vary in size and ISP type. AS701 is a tier-1 ISP, while AS59 and AS224 are Stub ISPs. We abstract the routing topologies from the listed network by ospf (open shortest path first) routing algorithm.

Table 2. Real Network Topology

Network	AS number	Nodes	Number of Tiers	Number of Layers
Uni. of Wisconsin	AS59	41	3	5
UUNet Alternet	AS701	75	3	8
UNINETT	AS224	208	5	9

5.1.2 Input data

We mainly use the synthetic input data in the caching performance evaluation, and also use the actual trace collected from Tsinghua University for validation purpose.

5.1.2.1 Synthetic input data

Let $O = \{o_1, o_2 ... o_N\}$ denote a set of cacheable objects. We assume that requests are identical and independently distributed (i.i.d.) within the set O in a considered time frame such that each request refers to an object o_k with a probability p_k without memorizing previous requests. The objects are ordered according to the decreasing access probabilities $p_1 \geq p_2 \geq ... \geq p_N$.

The requests at each end node follow the Poisson arrival $P_n(t) = (\lambda t)^n e^{-\lambda t} / n!$. We assume the average request rate λ

at edge routers follows a uniform distribution and the objects' popularity is governed by the Zipf distribution $f(i) = i^{-\alpha} / \sum i^{-\alpha}$ ($0.5 \le \alpha \le 2$) [16], where α is the skewness factor indicating the concentration degree of object access. Zipf distribution defines the probability of accessing an object at rank i out of N available objects.

5.1.2.2 Real trace

The real trace is collected from Tsinghua University campus network. The duration of the trace is one hour and the trace accounts for around 100G bytes. We apply DPI (Deep Packet Inspection) to parse the trace and use the urls in HTTP as the input data.

5.1.3 Performance metrics for evaluation

Our goal is to find the optimum caching locations to maximize the benefits. The most typical metrics are inter-ISP traffic or hit rate (The difference of the two metrics mainly lie in whether content size is involved), and the access delay measured by the number of hops that a given request travels in the network. So, the two metrics are tested.

- Saving Rate of inter-ISP Traffic (SR-CDT): the ratio of the Inter-ISP traffic saved by caching with respect to the total Inter-ISP traffic incurred without caching.
- Saving Rate of Hops (SR-Hops): we define the response hops as the number of the routers traveled by the response packets from the source (or cache) to the requester. SR-Hops is the ratio of the average number of response hops reduced by caching over the number of response hops without caching (11 hops as above mentioned).

5.1.4 Caching schemes for comparisons

We compare our caching schemes with other caching schemes: Leaving Copies Everywhere (LCE) [17], Leaving Copies with Probability (LCProb), and Leaving Copies with Uniform Probability (LCUniP). LCE is currently used in most hierarchical caches and the same caching algorithm was applied in the most influential article for NDN [1]. In LCE, each cache on the delivery path replicates the copy of the object with the Least Recently Used (LRU) replacement algorithm. LCE is widely used due to its high performance and ease of implementation. LCUniP and LCProb are similar to LCE except that the retrieved content is not blindly cache at each passing node, but selectively cached by probability to eliminate redundancy. LCUniP caches the passing content with uniform probability at each router, while LCProb is with caching probability 1/(hop count along the path).

In addition, we compare our caching algorithms with the optimization solutions.

5.2 Comparison with Optimization Solutions

As mentioned above, it takes an extremely long time to solve the optimization problem with a 200-node network and it is thus impractical to compare the solution at such a scale. Therefore, we compare our algorithms with the optimal solutions in the 50-node topology with each router caching 10 objects (or chunks), serving for 1000 object (or chunk) interests in the network. The object request arrivals follow the Poisson distribution and the popularity of requested objects follows the Zipf distribution with skewness parameter $\alpha = 0.9$.

In Table 3, Optimal Solution 1 stands for the solution to the objective of minimizing inter-ISP traffic, while Optimal Solution 2 stands for the solution to the objective of minimizing average number of access hops. We can observe that both AsympOpt and TopDown closely approach the bound of SR-CDT performance, so they both work well on achieving the objective of minimizing inter-ISP traffic and maximizing cache hit rate. As for SR-Hops which measures the access latency, AsympOpt is much closer to the bound and achieves the optimum performance from the user perspective. In contrast, though TopDown slightly outperforms AsympOpt in terms of SR-CDT, it is much inferior to AsympOpt in terms of SR-Hops. In summary, the proposed algorithms can achieve nearly optimal performance in small-scale networks. Particularly, the performance of AsympOpt is very close to the bounds obtained from the optimization solutions.

Table 3. Comparison with Optimal Solution

Metric	*Optimal Solution 1*	*Optimal Solution 2*	*AsympOpt*	*TopDown*
SR-CDT	0.4834	/	0.4775	0.4834
SR-Hops	/	0.4500	0.4462	0.3713

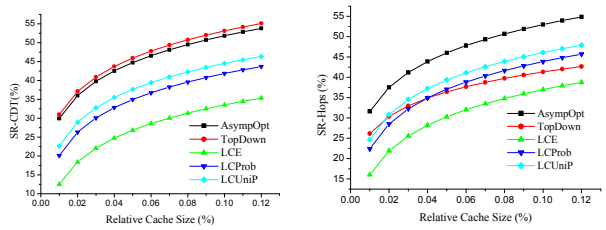
5.3 Performance Impact Factors

The efficiency of caching depends on factors such as network topology, request pattern, cache capacity and object popularity. In this section, we compare our algorithms with baseline algorithms LCE, LCProb and LCUniP (10% probability) considering these factors

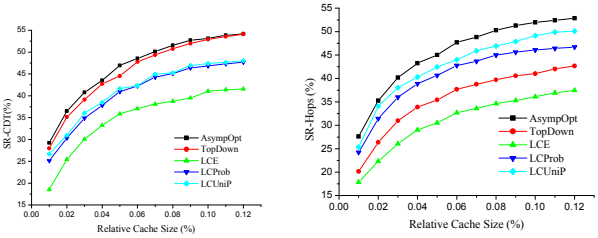
The default settings are listed as follows. The ISP routing topology is a tree with 200 nodes including 103 end nodes, where each node is equipped with a cache that can serve for 100,000 object items. The number of average requests at each end node (which follow the Poisson arrival with parameter λt) follows a uniform distribution U(20000,40000), where λ is the request arrival rate at the end node and t is the observation interval for caching decision. Object popularity follows a Zipf distribution with $\alpha = 0.9$.

5.3.1 Impact of the cache size on performance

The cache size at each node is described as the value relative to the total size of all objects available in the network and called relative cache size. We compare the effectiveness of different caching algorithms across a range of cache sizes, from 0.01% percent to 0.12% percent, with the total object size of 100,000 chunks. (The default relative cache size 0.04% will be used in the following simulations.)



(a) Synthetic input



(b) Real trace

Figure 4. Caching performance vs. cache size

Figure 4 compares the saving rate of inter-ISP traffic (SR-CDT) and saving rate of average response Hops (SR-Hops) with the synthetic load and real trace, respectively. The simulation results show that all the algorithms provide steady performance improvement as the cache size increases. In general, AsympOpt significantly outperform LCE, LCProb and LCUniP both in SR-CDT and SR-Hops. TopDown performs best, but achieves a marginal improvement in terms of SR-CDT compared with AsympOpt. However, AsympOpt performs much better in terms of SR-Hops than TopDown. Therefore, AsympOpt is preferable for achieving better whole performance. Among the three baseline algorithms, LCUnip (10%) performs the best, while LCE performs the worst as expected in terms of both metrics. We also test the naïve random en-route replica placement, that is LCUnip with 50% probability and find that LCUnip (10%) outperforms LCUnip (50%) in both of the performance metrics.

We also observe that as the relative cache size increases, the slope of the curves turns to be flatter, that is, the performance gain decreases with the increased capacity. Considering the tradeoff between the cost and performance gain, we can deploy suitable cache capacity with the curves.

Although the result curves are largely consistent with the curves created by the synthetic input, the advantage of our algorithms over baseline algorithms seem to decline in case of real trace input. It may be because the trace we've got is not general enough and we only analyze the web requests. The comprehensive evaluation on real traces will be our future work.

5.3.2 Impact of the request pattern on performance

5.3.2.1 Popularity Skewness

We assume that the request pattern follows the Zipf distribution, that is, the frequency of a request is the inverse of its rank in the request popularity. The Zipf skewness parameter α indicates the degree of concentration of object requests. When the values of α are close to 1, it indicates that a few objects (also known as "hot spots") attract the majority of the requests; while when values are close to 0, it means the object popularity is almost uniform. We examine the impact of request frequency distribution on the effectiveness of our caching schemes.

Figure 5 shows the performance curves as a function of Zipf parameter α over a range from 0.5 to 2, with the relative cache size of 0.04%. The curves remain approximately constant with the skewness factor. Because of the large base of content population, the most popular contents are almost cached in different cases.

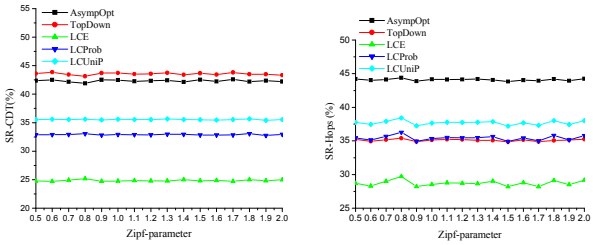


Figure 5. Caching performance vs. Zipf skewness

5.3.2.2 Object popularity fluctuation

As the object popularity changes with time owing to the variation in user interests, we further study the impact of varying object popularity on the caching performance. The X axis shows the popularity variation range, with maximum change of 60 popularity rank for each round. Though there is some fluctuation because of the random generation for dataset, in general slight variation on object popularity has advantage over sharp variation. It is more difficult for the caching algorithms to adapt quickly in response to severe and quick popularity alteration. Both SR-CDT and SR-Hops are the worst in case of sharp variation in popularity. As shown in Figure 6, all the algorithms can keep good efficiency and stability regardless of the popularity change. The insensitiveness of the proposed algorithms provides better adaptation to different network environment and makes the assumption of unchanged popularity reasonable.

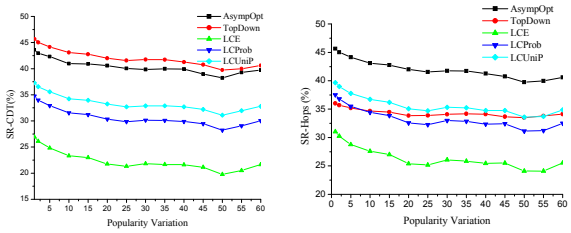


Figure 6. Caching performance vs. Popularity variation

5.3.3 Impact of content population on performance

Internet contents are anticipated to dramatically increase due to the explosive growth in user-generated contents and

many emerging applications. To shed light on this issue, we have conducted experiments to gain a deeper insight into the impact of object population increase on the performance of caching algorithms, with a special focus on its scalability and robustness.

Figure 7 shows that the proposed algorithms still gain much better performance than the baseline algorithms, when increasing the number of object items while keeping the caching capacity fixed. Our simulation also shows that when enlarging the cache capacity with the increase in #objects, the caching performance in terms of SR-CDT and SR-Hops remains good. Besides, as the content population exceeds 100,000, caching performance tends to be convergent. The stable property is very favorable, as the content items increase rapidly nowadays.

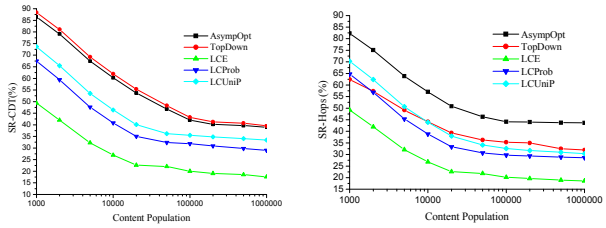


Figure 7. Caching performance vs. content population

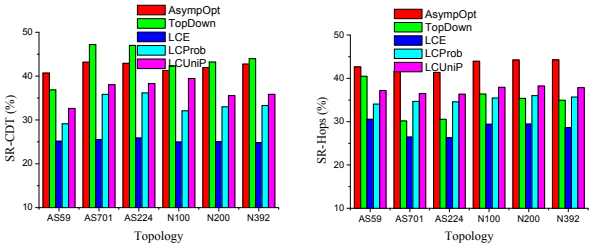


Figure 8. Caching performance vs. topology

5.3.4 Impact of the ISP topology on performance

We generate three different router-level topologies by GT-ITM and extract the routing trees rooting from the gateway: 100 nodes with 53 end nodes, 200 nodes with 103 end nodes, and 392 nodes with 262 end nodes, respectively. And we test the algorithms over real networks as well. Figure 8 illustrates that the performances of the presented caching schemes are insensitive to the topology change, which ensures the scalability of our proposed caching algorithms and the ease of deployment.

5.3.5 Discussion of simulation results

From the simulation results presented above, we reach some conclusions about our caching schemes.

1) Effectiveness

From Figure 4-8, we can see that AsympOpt outperforms the baseline algorithms both in SR-CDT and SR-Hops, and TopDown performs well in saving inter-ISP traffic, which is our first order objective. In all the cases, the algorithms are effective and seem to achieve better performance in presence of more randomness in access pattern.

2) Scalability

The proposed caching algorithms show great scalability as demonstrated in Figure 7, 8. The caching performance of the schemes increases with the increasing cache size. Moreover, enlarging ISP's network scale will not impose a negative impact on algorithms' performance, possibly because caching favors popular contents, which are not pertinent to object population. Further, when object population is large enough, caching performance is convergent to a reasonable value, which is good news for the content explosive era.

3) Stability

The algorithms are insensitive to the variation in user behaviors on object requests. The relative performance remains stable regardless of the distribution of requested objects and popularity alteration, even in the worst case where there is extremely dynamic change of object popularity. Besides, AsympOpt is superior to the baseline algorithms in almost all cases with various parameters.

5.4 Preliminary measurement of overheads

We analyzed the caching scheme and conducted preliminary experiments mainly on the default 200-node topology for measuring the cost on three folded: storage use, communication overhead and execution time.

5.4.1 Communication overhead

In order to characterize communication overhead, we first list the related notations as follows,

D_i : the number of nodes who is i hops apart from the root.

L: the longest distance between end node and root

C: homogeneous cache capacity

M: the number of content profile sent for information aggregation

N: the number of all nodes. $N = \sum_{i=0}^{L} D_i$

F: content population

H_{ij} : hop count to satisfy the request for O_j at end node i

Then, the average communication overhead is as follows,

AsympOpt $4\sum_{i=0}^{L} D_i \cdot [M - (L - i + 1) \cdot C] / N$

TopDown: $4 \cdot [\sum_{i=0}^{L-1} \sum_{j=i+1}^{L} D_j \cdot C + M \cdot (N - 1)] / N$

LCProb: $4 \cdot \sum_{i=1}^{N} \sum_{j=1}^{F} E[H_{ij}] / N$

LCProb: $\sum_{i=1}^{N} \sum_{j=1}^{F} E[H_{ij}] / N$

LCE: 0

Take 200-node topology for instance, the communication overhead for all the algorithms is : AsympOpt 11.3 Kbytes ,

TopDown 12.464Kbytes, LCProb 424M bytes, LCUniP 105 Mbytes.

5.4.2 Storage overhead

Storage cost for LCE, LCProb and LCUniP is negligible, while AsympOpt needs (4F-2/N+8) bytes and TopDown needs (4F-2/N+8) bytes.

5.4.3 Execution time

The job execution time during each caching round is listed as follows: 1304ms for AsympOpt, 2577ms for TopDown, 402540ms for LEC, 400784ms for LCProb, and 398672 for LCUniP. Our algorithms save hundreds of seconds of running time and account for limited communication overhead mostly because the algorithms only deal with access statistics of selected objects and are well scaled with the increasing object population, while the baseline algorithms have to cope with each arriving object. Therefore, we can expect our algorithms to gain favorable result under the large-scale network serving numerous objects.

6. CONCLUSION AND FUTURE WORK

We have developed coordinated caching schemes to reduce the redundant traffic going through the NDN networks, trying to minimize both inter-ISP traffic and average number of access hops. The main goal of this paper is to propose efficient caching algorithms that can make dynamic caching decisions on the fly. The proposed algorithms achieve the performance that is close to the optimum (especially for AsympOpt) with the favorable saving rate in inter-ISP traffic and considerably improve the performance of access delay and intra-ISP link consumption measured by the number of hops traveled. A variety of factors that can impact the caching performance are considered in the simulations and our algorithms are demonstrated to be effective, stable and scalable with varying network topology, cache capacity, objects request pattern, popularity variation and population covered by end routers. As part of our future work, we plan to extend our dynamic caching solution by considering multiple gateways, as well as the use of multiple paths in NDN.

7. ACKNOWLEDGMENTS

This paper is partially supported by NSFC (61073171), China Postdoctoral Science Foundation (Grant No. 023230012), Tsinghua University Initiative Scientific Research Program (20121080068), Specialized Research Fund for the Doctoral Program of Higher Education of China (20100002110051), and Ningbo Natural Science Foundation (Grant No. 2010A610121). We would like to thank Greg Byrd and the anonymous reviewers for their helpful comments and suggestions.

8. REFERENCES

[1]. Jacobson, V., Smetters, D. K., and Thornton, J. D. et al., "Networking named content", Proceedings of the 5th international conference on Emerging networking experiments and technologies, ACM, 2009, pp.1-12.

[2]. Hefeeda, M. and Noorizadeh, B., "On the Benefits of Cooperative Proxy Caching for Peer-to-Peer Traffic," Parallel and Distributed Systems, IEEE Transactions on, vol. 21, no. 7, pp.998-1010, 2010.

[3]. Rodriguez, P., Spanner, C., and Biersack, E. W., "Analysis of web caching architectures: hierarchical and distributed caching," Networking, IEEE/ACM Transactions on, vol. 9, no. 4, pp.404-418, 2001.

[4]. Hefeeda, M., Hsu, C. H., and Mokhtarian, K., "Design and Evaluation of a Proxy Cache for Peer to Peer Traffic," IEEE Transactions on Computers, vol. 60, no.7, pp.964-977, 2011.

[5]. Pallis, G. and Vakali, A., "Insight and perspectives for content delivery networks," Communications of the ACM, vol. 49, no. 1, pp.101-106, 2006.

[6]. Bhattacharjee, S., Calvert, K. L., and Zegura, E. W., "Self-organizing wide-area network caches", Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom'98), pp.600-608, 1998.

[7]. Lijun Dong, Dan Zhang, Yanyong Zhang, and Dipankar Raychaudhuri, "Optimal Caching with Content Broadcast in Cache-and-Forward Networks", ICC' 2011, 2011.

[8]. Walter Wong, Marcus Giraldi, Mauricio F.Magalhaes, and Jussi Kangasharj, "Content Routers: Fetching Data on Network Path", ICC'2011, pp.1-6, 2011.

[9]. D.Rossi and G.Rossini, "Caching performance of content centric networks under multi-path routing (and more)," Telecom ParisTech, Technical Report, 2011.

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

[11]. Cho, K., Lee, M., Park, K., Kwon, T. T., Choi, Y., and Pack, S., "WAVE: Popularity-based and Collaborative In-network Caching for Content-Oriented Networks", Infocom'2012 workshops, pp.316-321, 2012.

[12]. Schwartz, Y., Shavitt, Y., and Weinsberg, U., "A measurement study of the origins of end-to-end delay variations", Passive and Active Measurement, Lecture notes in Computer Science, vol. 6032/2010, pp.21-30, 2010.

[13]. P.V.Mieghem, "Performance analysis of communications networks and systems",Cambridge Univ. Press, 2006, pp.358.

[14]. "GLPK: GNU Linear Programming Kit," http://www.gnu.org/s/glpk/, 2012.

[15]. Calvert, K. I., Doar, M. B., and Zegura, E. W., "Modeling internet topology," Communications Magazine, IEEE, vol. 35, no. 6, pp.160-163, 1997.

[16]. Hefeeda, M. and Saleh, O., "Traffic modeling and proportional partial caching for peer-to-peer systems," IEEE/ACM Transactions on Networking (TON), vol. 16, no. 6, pp.1447-1460, 2008.

[17].Laoutaris, N., Syntila, S., and Stavrakakis, I., "Meta algorithms for hierarchical web caches", IEEE International Conference on Performance, Computing and Communications, pp. 445-452, 2004.