Popularity-driven Coordinated Caching in Named Data Networking

Jun Li, Hao Wu, Jianyuan Lu, Yi Wang, Bin Liu
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) will enable effective content distribution at a global scale without special infrastructure. The aim of this work is to design efficient caching schemes in NDN to achieve better network layer performance and application layer performance. With the specific objective of minimizing both inter-ISP (Internet Service Provider) traffic and average access latency, we first formulate the optimization problems for different objectives and solve the problems 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 our caching algorithms are very closer to the optimum and outperform the widely used algorithms in terms of minimizing inter-ISP traffic and the average 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 a low 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. the users tend not to care where (from which host) and how (via which protocol) to obtain a piece of content, but essentially be interested in fast and reliable content retrieval. Meanwhile, driven by increasing content size and content types, such as multimedia content and user-generated content, Internet traffic has been rapidly growing both at an unprecedented rate and scale. This explosive growth in traffic poses a significant challenge to the network, as network capacity cannot satisfy exponentially growing demand for content distribution. 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 mechanism and need dedicated components for the architecture, which makes the solutions not scalable for future Internet. To meet the huge demand of content dissemination in the Internet, it is necessary to rethink of the future Internet architecture which can bridge the mismatch between name-based content delivery and underline 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 a most promising architecture for future networks. Quite differing from the current IP-based network, this new paradigm is characterized as name-based routing and systematic in-network caching. To be specific, in-network caching can store content at each node (say router) when it traverses along the delivery path. By typically caching the popular contents at the router, NDN can reduce both the overall network load and the access delay, as subsequent requests no longer need to travel until the content source, but are served by a closer NDN router along the routing path. As such, less traffic will travel across ISP domain boundary, which saves the expensive inter-ISP traffic and possibly relieves backbone network from congestion.

Since the storage of each router in NDN is technologically limited by memory access speed due to the requirement of the line-speed packet processing, careful cache placement is needed to achieve the maximal benefit with the given constraint of caching capacity. Among many variables that can be impacted by caching, we strive to minimize the inter-ISP traffic at the administrative boundaries in NDN as a first-order objective. 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 of the Internet and where congestion occurs. Reducing inter-ISP traffic will significantly improve the whole network performance.
- In addition, because the inter-ISP links are much more costly than internal links, the gain in Inter-ISP traffic

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

saving will greatly reduce the cost incurred by 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 can expect to thoroughly remove the caching redundancy and accommodate as many diverse content items as possible in caching system, which yields highest cache hit rate[2] 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 access hops and potentially alleviates the traffic burden within an ISP.

On the basis, the other objective is to explore the better caching algorithm to further reduce the access delay without sacrifice of the hit rate or the gain of inter-ISP traffic. Less access hops result in light traffic load within the ISP. As above mentioned, the ultimate objective is to improve the whole network performance in terms of inter-ISP and intra-ISP bandwidth consumption, and access latency by 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) Practically, an ISP has several gateways to interconnect with others, either provider-ISPs or peer-ISPs. Obviously, solving caching coherently among multiple gateways is a very challenging problem. In this paper, we focus on single-gateway situation, and plan to extend our solution to multiple gateways in our future work.

The main problem addressed in the paper is provide effective caching strategies that enable the routers within an ISP in NDN to coordinate in making online caching decision dynamically. Our goal is to find the suitable caching locations according to the objectives, and based on given network topology, content access pattern, and various caching constraints. Though caching has already been extensively studied, a number of architecture details make caching in NDN a relatively new and unexplored research topic. Unlike most of the earlier work in conventional CDN caching which has focused on minimizing the access latency without considering the resulting bandwidth consumption [2], we made the following contributions to this new research field:

- We formulate the problem-solving models for the objective of minimizing inter-ISP traffic and minimizing average access hops, and obtain optimal solutions to the replica placement, respectively.
- Guided by optimal replica placement, we present two popularity-driven coordinated caching algorithms, named TopDown and AsympOpt, where caching is coordinated implicitly among routers on the path and routers can make online decision independently. The proposed algorithms significantly improve the caching performance in terms of reducing inter-ISP traffic and diminishing response hops. Especially, AsympOpt can achieve the best whole performance, very closer to the results of optimal solutions.
- We evaluate the performance of the caching algorithms compared with optimum solutions and study the impact of a variety of factors such as the network topology, the request pattern, object popularity and cache capacity etc. Simulation results demonstrate that the proposed algorithms retain distinct stability and scalability under a wide range of changes pertaining to the most influenced factors, with reasonable 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 schemes and proposes two caching algorithms. 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 top 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, however the work can only reach local optimality with the mathematical model. Due to the complexity of solving the optimization problem, the presented scheme is limited to be used in small-scale networks. 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.

[2] A cache hit occurs when a request finds a match at a router along the path. Otherwise, there is a cache miss. Cache hit rate is the ratio of the number of cache hits to the total number of requests

The most recent work [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 content 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 achieving 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 among the caches. In contrast, routers with limited storage in our schemes independently make the caching decision based on the recent history of the content requests of its subordinates, thus nodes implicitly share their caching information and coordinate in minimizing the redundancy. Besides, our 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 should firstly check whether the requested content is present in its local cache by looking up a Content Store table. If there is a hit, the router sends out 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) As the target content travels from the source (or cache) downward to the requester, the router on the 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 estimation that 80% of the content requests can not be satisfied within an ISP without caching. And, generally it takes fewer hops to retrieve a content object inside the ISP, so we are not ready to cache those objects reside in the investigated ISP and 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 content according to the deterministic caching strategy. As in the example, node R_4 replicates the content object O_1 and can serve the later requests for O_1 from the edge routers within its subtree, that is, R_{12}, R_{13}, R_{14}.

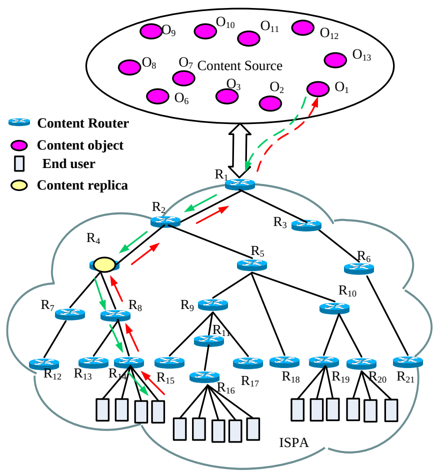


Figure 1. An illustrated caching model

From the above, all the content resources retrieved from 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 an inversed 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. Different from caching in general networks where caches are located in some specific severs and replicas can be positioned in any of the caches, in NDN the replicas of objects are cached along the en-route paths so the requested objects can be obtained accordingly when the routers forward the content requests towards the content source. This design would seamlessly integrate the routing and content retrieval for the optimal system performance in a light-weight manner. As a result, conventional caching algorithms can not be directly applied in NDN networks.

To make full use of in-networking caching, we focus on minimizing the inter-ISP traffic and taking into account access hops as well 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 further minimize the average access hops by caching frequently requested objects at selected routers inside the ISP, other than the commonly used performance gain in simply reducing the access latency from the user perspective or gain in link bandwidth consumption from the ISP perspective.

3.2 Problem Statement

It is impracticable to find the solution to an optimization problem including all the optimization objectives. Especially, 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 optimal solution to the replica placement respectively. The solutions are used to guide the caching design and taken as the bounds to caching performance.

3.2.1 Definition and Notations

According to the tree topology of our system model in Figure1, 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 ($N-U$), 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 $Ancestors(i)$ denote the set of ancestors of node j , i.e., the nodes in the unique en-route path $\left[j \to R_1 \right]$ (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. The optimization objective can be interpreted as maximizing the inter-ISP traffic saved by the caching system, with the fixed and 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 objects cached at a router could 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 $\left[i \to R_1 \right]$ 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 access hops for the requesting contents can be formulated as follows,

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

s.t. $\sum_{k \in O} s^k X_j^k \le C_j$, 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. In 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 lead to the least average access hops, with the constraint of cache capacity. Thus, the problem turns to be a Mixed Integer Problem. In addition, for ease of mathematical expression, minimizing average response hops equal to maximizing the saved average hops by caching in network. The optimization problem is therefore specified 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 having 20 content items) and 5000 objects (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 on-line 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 take the optimization solutions as the guide to caching algorithm design and evaluation bounds for caching performance.

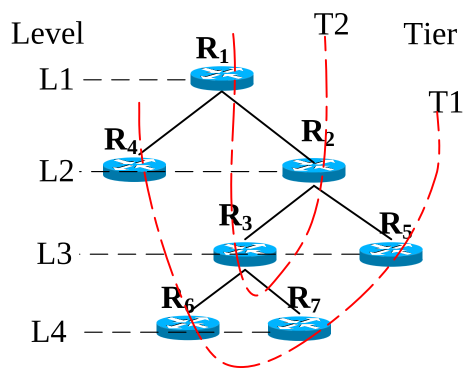


Figure 2. Toy model of caching topology

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), 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 caches at the lower level can be shared and accessed by sub-tree nodes at the higher level. A user request travels from a given end node towards the root node, 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 object of interest. When the object is found, it is passed down along the reverse path to the end node which sends the request. Each cache on the reverse path decides whether to cache the object or not according to a chosen strategy, which is determined by dynamic caching algorithms proposed later in this section.

Our primary goal is to maximize the gain in cross-domain traffic from the ISP's perspective and 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, selected objects refer to the most popular objects in presence in the last caching interval based on the fact 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 this caching round. Dynamic items are maintained according to the object arrival rate and those with longer arrival interval will make room for the subsequent requested objects. In this way, 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 recent caching decision in this round. An object can be tagged by an *update* mark or a *replication* mark. The *replication* mark of an object is set based on a caching strategy presented in the next subsection. When a fetched object arrives at each node, an object with a *replication* mark is cached at the router and the replica creation time of the object is set to the current time. Meanwhile, the *replication* mark turns to an *update* one. An *update* mark of an object 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 goes beyond a preset value. Once an object is obsolete, the corresponding request for the object is forwarded to the source and the *update* mark of the object turns to a *replication* mark, for the purpose of keeping the cached object refreshed. In our scheme, the nodes along the transmission path between the gateways and the requester make a caching decision coordinately. The caching decision can be made based on one of the caching algorithms described in the next subsections: Topdown caching algorithm and AsympOpt caching algorithm.

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 (local request statistics for the end nodes).

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.

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 the 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.

Algorithm 1 TopDown Caching Algorithm

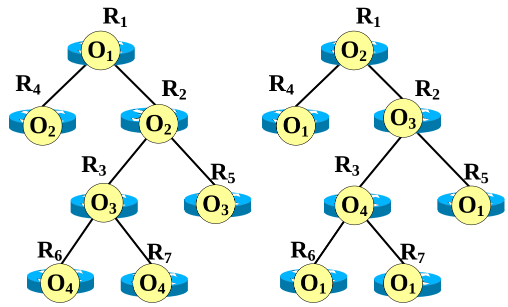
```
Information Aggregation (ReqHistory)
for layer← bottom level to root do
 for each node R_j in the layer do
   if R_j ∈ U then
     ReqRec[]_j ← Get ReqHistory( )
   Else
     ReqRec[]_j ← Aggregate (Children( R_j ), Records)
   end if
   A_j ← Sort-Dec (ReqRec[]_j, threshold)
  end for
end for
Return ( A_j )
TopDown decision making ( A_j )
for layer←root to bottom level do
 for each node R_j in the layer do
    for each record r ∈ A_j do
      if r.ObjectID Exist-In R_j 's CachedTable then
        MarkUpdate (r.ObjectID)
        Append ( DelTable_j , r.ObjectID)
      else if Available-Space ≥ Size (r.ObjectID) then
        MarkReplicate (r.ObjectID)
        Append ( DelTable_j , r.ObjectID)
        end if
     end if
   end for
   Delete ( A_Children(R_j) , DelTable_j )
  end for
end for
```

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

Table 1. Content request profile

Object	Request at R4	Request at R5	Request at R6	Request at R7
O1	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

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 ensure the objective of minimizing inter-ISP traffic and maximizing cache hit rate.



(a) TopDown (b) AsympOpt

Figure 3. Examples for Caching Algorithms

4.3 AsympOpt Caching Algorithm

TopDown caching algorithm can eliminate caching redundancy and accommodate more diverse content objects, which benefits the gain in 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 reducing 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*, i.e., end nodes 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_2 . The values of Tier and priority are determined by the routing topology and obtained before hand.

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 the each node. The subscript of Object O_j stands for its *GloRank*. Information Aggregation procedure will be iterated periodically during the AsympOpt caching as well. The period is dependant on 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 . With the value, router R_j picks out C_j global popular contents to cache locally. If the local popularity of some object is not consist with the global popularity and exceed the range of the popularity difference, those not expected to be cached because of the low *GloRank*, while popular locally will be cached. In this way, we try to compensate the difference between global popularity rank and local one (Line 10~Line 23).

Algorithm 2 AsympOpt Caching Algorithm

```
1   for layer ← lowest tier to highest tier do
2    for each node R_j in the layer sorted by ascending priority do
3       if R_j ∈ U then
4         ReqRec[]_j ← Get ReqHistory( )
5       Else
6       ReqRec[]_j ← Aggregate (Children( R_j ), Records)
7       end if
8    end for
9   for each node R_j in the layer sorted by descending priority do
10    A_j ← Sort-Dec (ReqRec[]_j, threshold)
11   st ← StartValue(j ) obtained from child in the closest tier
12   while k<= C_j do
13   if each GloRank(st) Exist-in first C_j records of A_j then
14    Mark(GloRank(st))
15   else
16      count ++
17   end if
18   st ++
19  end while
20  for k=1 to count do
21   re ← top record of A_j
22   if re.ObjectID out of the RangeValue(j) then Mark( re )
23  end for
24  SendtoParent ( A_j)
25
26  Mark( r)
27    if r.ObjectID Exist-In R_j 's CachedTable then
28        MarkUpdate (r.ObjectID)
29        Remove (A_j, r)
30    else
31        MarkReplicate (r.ObjectID)
32        Remove (A_j, r)
33    end if
34
35  Aggregate (Children( R_j ), Records)
36  Initialize (ReqRec[]_j)
37  for each child R_k in Children ( R_j ) do
38     for each record r ∈ A_k do
39        if r.objectID Exist-In ReqRec[]_j then
40           r_j. ReqCount= r_j. ReqCount + r. ReqCount
41        else
42           Insert the record r
43        end if
44     end for
45  end for
46  return (ReqRec[]_j)
```

Aggregate and Mark procedure are similar to corresponding procedures in Algorithm 1 except that the records of cached contents will remove from the request profile sent to the parents. For clarity, we don't describe 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 for O_1 is less than 5 and local popularity of O_1 exceed the range value, O_5 which is not in the expected caching list but is a relatively popular content is cached at R_6 ; if the request for O_1 stays in the first 4 popular record, O_1 is still cached.

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 in 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 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 node, checks the requested object in its local cache. If the object is not found, the request will be forwarded to the next-hop node along the routing path to the root until it reaches a cache node with the desired object or out of the ISP via gateway to the source of the requested object. Then, an object copy is sent back along the reverse path to the requesting end node. Each node on the path can replicate the passing object based on the collected request records as well as the recent caching decision.

We employ different network topologies in the evaluation, including the topology of Wisconsin University 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
Wisconsin University	AS59	41
UUNet Alternet	AS701	75
UNINETT	AS224	208

5.1.2 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 Poisson arrival $P_n(t)=(\lambda t)^n e^{-\lambda t}/n!$. Considering edge routers may cover different amount of user population, we assume the average request rate λ at edge routers follows the uniform distribution. The average number of arriving requests at each node is also determined by the chosen interval. The objects' popularity is governed by Zipf distribution $f(i)=i^{-\alpha}/\sum i^{-\alpha}$ ($0.5 \leq \alpha \leq 2$) [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, and will be used to generate the input requests in this paper.

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 a given request passes in the network. So, the two metrics are tested.

- Saving Rate of inter-ISP Traffic (SR-CDT): the ratio of the Inter-ISP traffic saving gained by caching 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 to satisfy an object request from the end router. SR-Hops is the ratio of the average number of response hops reduced by caching over the number of hops without caching (11 hops as suggested in [12], here we assume the evaluated ISP connected directly to the end hosts for comparison).

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 scheme, a request is routed upwards on the path from the responsible end node to the root until it reaches the cache of the object or the source. Once the requested object is found, it is sent back along the reverse path to the end node and each cache on the path replicates the copy of the object with the Least Recently Used (LRU) replacement policy independently. LCE is widely used due to its high performance and ease of implementation. LCUniP and LCProb are similar to LRU 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 (10% in the paper) at each router, while LCProb is with caching probability 1/(hop count along the path).

Besides, we compare our caching algorithms with the optimization solutions to measure the performance difference with the bounds.

5.2 Comparison with Optimization Solutions

As mentioned above, it takes extremely long time to solve the optimization problem with a 200-node topology of ISP network and this is impractical to find the solution for performance comparison purpose. Therefore, we just simply 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 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 access hops. We can observe that both AsympOpt and TopDown closely approach the bound of SR-CDT performance, so they are both effective to achieve the objective of minimizing inter-ISP traffic and maximizing cache hit rate. As for SR-Hops which a measure for access latency, AsympOpt is very 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 the context of small scale, the proposed algorithms can achieve good performance. Particularly, AsympOpt can achieve an overall best performance, even 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 the 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 in terms of SR-CDT and SR-Hops.

The default setting parameters 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 serving for 100,000 object items. The number of average requests λt of Poisson arrival at each end node follows the 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 Zipf distribution with skewness parameter $\alpha = 0.9$.

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 taken in most of the following simulations).

Figure 4 compares the saving rate of inter-ISP traffic (SR-CDT) and saving rate of average response Hops (SR-Hops), 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 SR-Hops than TopDown. Therefore, AsympOpt is preferable for achieving better whole performance. Among the three baseline algorithms, LCUnip performs the best, while LCE performs the worst in terms of both metrics.

We can also observe that as the relative cache size increases, the slope of the curves turns to be flatter, that is, the performance gain brought by capacity increase will decrease. Considering the tradeoff between the cost and performance gain, we can plan the most suitable cache capacity with the curves.

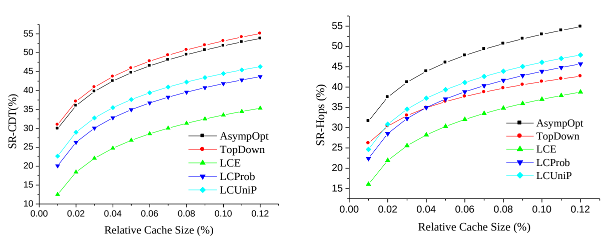


Figure 4. Caching performance vs. cache size

5.3.2 Impact of the request pattern on performance

5.3.2.1 Popularity concentration of object

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 homogeneous. 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 0.04%. The curves almost remain approximately constant with the varying zipf parameter. It might be because of the large base of content population, the most popular contents are almost cached in different cases.

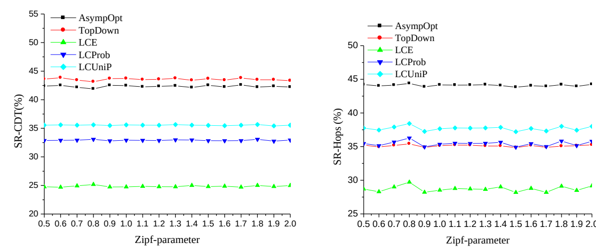


Figure 5. Caching performance vs. Zipf skewness

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.

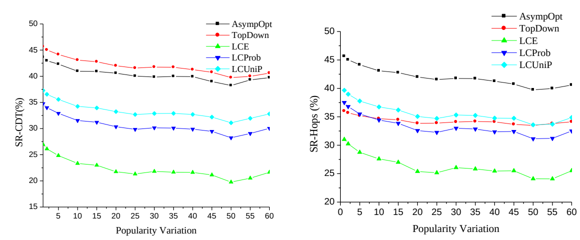


Figure 6. Caching performance vs. Popularity variation

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.

5.3.3 Impact of content population on performance

The Internet contents are anticipated to dramatically increase due to the explosive growth in user-generated contents and new emerged applications. In view of the issue, we have conducted experiments to gain a deeper insight into the impact of the object items increase on the performance of our algorithms, in order to examine its scalability and robustness.

Figure 7 shows that the proposed algorithms still gain much better performance than the baseline comparison 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.

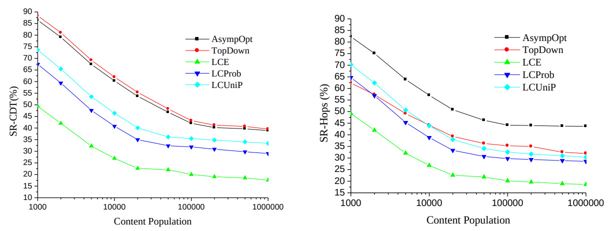


Figure 7. Caching performance vs. content population

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 network AS59, AS701 and AS224 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.

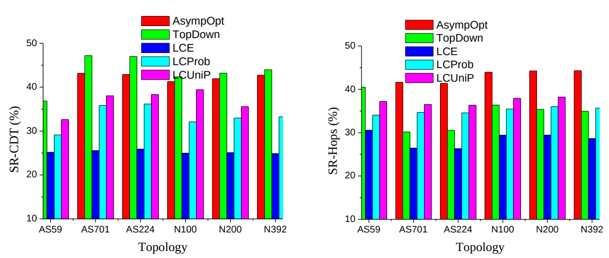


Figure 8. Caching performance vs. topology

5.3.5 *Discussion of simulation results*

From the simulation results presented above, it comes to some conclusions about our caching schemes.

1) Effectiveness

From Figure 4-8, we can see that all the proposed algorithms yield significantly better performance for both SR-CDT and SR-Hops than LCE, LCProb and LCUniP. In all the studied 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 2, 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, as object population is large enough, caching performance is convergent to a reasonable value, which is a very 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 algorithm is superior to baseline algorithms in almost all cases with a wide range of 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 explain 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: $4\cdot\sum_{i=1}^{N}\sum_{j=1}^{F} E[H_{ij}]/N$

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

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 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 networks, minimize both inter-ISP traffic and average access hops, exploiting the caching capability of routers in NDN. The main goal of this paper is to propose efficient caching algorithms that can make the dynamic caching decision on-the-fly. The proposed algorithms achieve close to globally optimum performance (especially AsympOpt) with favorable saving rate in inter-ISP traffic and considerably improve the performance of access delay and intra-ISP link consumption measured by hops traveled. A variety of impact factors are considered in the simulations and our algorithms are demonstrated to be effective, stable and scalable with the varying network topology, cache capacity, objects request pattern, popularity variation and population covered by end routers. We plan to extend our dynamic caching solution to multiple gateways, and consider the requirements of NDN’s multi-path usage as well.

7. REFERENCES

[1]. Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M. F., Briggs, N. H., and Braynard, R. L., 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, 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.Proceedings (Infocom'08), IEEE, 1998, pp.600-608.

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

[8]. Walter Wong, Marcus Giraldi, et.al, "Content Routers: Fetching Data on Network Path", ICC'2011, 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 Workshop, pp. 316-321, 2012.3

[12]. Schwartz, Y., Shavitt, Y., and Weinsberg, U., "A measurement study of the origins of end-to-end delay variations", Passive and Active Measurement, Springer, 2010, pp.21-30.

[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/, 11.

[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", Performance, Computing, and Communications, 2004 IEEE International Conference on, IEEE, 2004, pp.445-452.