

# Scalable Keyword-Based Data Retrievals in Future Content-Centric Networks

Ying Mao

Department of Computer Science  
University of Massachusetts Boston  
ying.mao001@umb.edu

Bo Sheng

Department of Computer Science  
University of Massachusetts Boston  
shengbo@cs.umb.edu

Mooi Choo Chuah

Department of Computer Science and Engineering  
Lehigh University  
chuah@cse.lehigh.edu

**Abstract**—The emergence of powerful mobile devices has allowed users to publish more contents in the Internet in recent years. The existing Internet architecture cannot cope with such exponential growth in users published contents. Content-centric networks have been proposed recently to allow future Internet to be data-centric rather than network centric. Several content centric networking approaches have been proposed, but most of them assume that users know the unique identifiers of the contents that are of interests to them. SECON [1] proposed a content centric mobile network solution that provides keyword-based retrievals. However, the authors do not provide detailed description on how their solution can be made scalable. In this paper, we propose two scalable solutions for keyword-based retrievals in content centric networks. Our preliminary simulation results indicate that our solutions are scalable.

## I. INTRODUCTION

Smartphones are the most popular mobile devices today. The emergence of powerful smartphones and tablets that combine multiple functionalities, e.g., cellular phone, personal computer, multimedia player and etc., has made such smart devices indispensable in our daily lives. A recent report showed that there are more than 700 million smartphones worldwide [2]. With more smartphone usages, more feature rich smartphone applications have emerged in various market places such as Google Play and Apple stores. For example, a report shows that there are over 470K applications in Google's official Android Market by July 2012 [3]. Such rapid advancements of mobile technology provide new opportunities for mobile users to have easy access to real time data, and stay connected with friends, colleagues or business partners. In addition, more and more information have been pushed to the Internet. Search engines such as Yahoo or Google allow us to locate useful information amidst the large volume of data published in the Internet. With such exponential explosion of published information in the Internet, users often desire to access information which is most relevant to their interests rather than consuming all published data.

Publish/subscribe systems built based on IP-based network have been proposed and designed in the past to cater to such needs. However, such systems can be inefficient and are not flexible enough to meet emerging requirements. For example, one group of sport fans may be interested in reading all types of sport news from ESPN while another group of sport fans may be interested in reading only soccer related news. Thus,

ESPN data servers may have to let the first group subscribe to `//espn.com/news/sport` while the second one will subscribe to `//espn.com/news/sports/soccer`. The data delivery mechanism in existing publish/subscribe may have to replicate the same news article to let it reach these two different groups of fans. Hence, the existing systems can be inefficient and not flexible enough to meet emerging requirements.

Recently, content-centric networks have been proposed to allow users more flexibility in accessing published information. For example, in [4], the authors propose a content centric network approach called NDN where publishers only publish data that users have expressed interests in. However, in real life, there are multiple scenarios where publishers publish information first before users express any interests in receiving such information, e.g., breaking news and new movies. In another approach called DONA [5], the authors propose using flat, self-certifying names for information objects. However, the scalability of flat names is questionable. In a recent paper [6], the authors present a design called DMAP that maps content identifiers to IP addresses, and use that mapping information to route relevant content locator information to the appropriate servers that support their approach.

However, none of the existing solutions provide keyword-based content searches. A user who is interested in retrieving all country music published between years 2010 and 2012 may not know where to start unless there is a server that houses all the relevant content identifiers related to country music and provides a mechanism for users to retrieve such information easily. Hence, any future information sharing systems should provide keyword-based content searches such that users only retrieve information of interests to them. In SECON [1], the authors propose a solution that provides push/pull-based data disseminations, keyword-based data retrievals. However, the authors did not elaborate on how their solution can be made scalable.

In this paper, we propose two solutions which provide keyword based searches for users to access information of interest to them in future content-centric networks. Our first solution called independent search and merge (ISM) scheme allows publishers to insert content identifiers together with independent keywords that are used to describe the content. Users can submit their searches based on keywords. Intermediate content routers retrieve content identifiers that match

different keywords. The client programs in users devices then do an intersection of all retrieved content identifiers to obtain a final list of content identifiers that match their interests. After that, users can retrieve these matched data contents. Our second solution called Integrated Keyword Search (IKS) scheme allows users to submit their interests based on keywords. Intermediate routers retrieve content identifiers that match all these keywords. Hence, the client programs within users devices will receive a list of content identifiers that match all these keywords. We evaluate our proposed solutions using the realistic Internet topology described in [6].

## II. RELATED WORK

Several content-centric networking approaches have been proposed for future Internet, e.g., the Data Oriented Network Architecture (DONA) [5], the Networking Named Content [4], and the Network of Information (NetInf) [7]. The authors in DONA suggest replacing DNS names with flat, self-certifying names and a name-based anycast primitive above the IP layer. The names in DONA are a cryptographic digest of the publisher's key and potentially user-friendly label. DONA is a pull-based approach where contents need to be published with a tree of trusted resolution handlers to enable retrieval. NetInf [7] proposed a solution to retrieve data objects based on their unique identifiers. NetInf process typically involves two major steps, (a) name resolution that locates an object in the network and routing which forwards the object retrieval query to its storage locations, and (b) forwarding the retrieved data object from its storage location to the requesting client. To facilitate the object location process, an information object (IO) is proposed for NetInf in [8] where the IO is a unique identifier with cryptographic properties and meta-data which can be used to verify data integrity. Some content-centric applications have been proposed [9], [10]. In [9], the authors proposed using content centric networking features to disseminate traffic information while in [10], it is used to disseminate college related information for prospective students.

NDN [4] proposed a content centric network architecture where content sources register their availability using URI-like names and such names (or aggregated versions of the names) are announced for global reachability. Two types of packets are supported, namely interest and data. An interest packet is sent by a consumer to query for data. Any data provider who receives the internet packet and has matching data responds with a data packet. NDN routers forward interests based on the content name within the interest packet. Data packets follow the reverse path established by the corresponding Interest. In NDN, a publisher is allowed to publish data items only after prior interest packets have been received. However, in real life, there are situations where users express interests only after they are made aware of certain published data items. Furthermore, NDN assumes that queriers know the unique content identifiers and no keyword-based queries are supported in NDN. In the COPS approach described in [11], the authors provided two additional types of packets, namely publish/subscribe packets. Publish packets are used

by publishers to express their intention to publish while subscribe packets are used by querying users to express their interests in certain topics e.g. `"/query/sports/football"` rather than expressing interests in a specific content using a unique content identifiers. COPS also use multicast feature to improve on delivery performance.

Some of these NDN limitations have been addressed in the SECON design [1] which supports both push and pull features. Publishers can push data contents to the routers before any user express interests. Furthermore, SECON supports keyword-based interests and intentional named delivery where interests packets can be addressed to intentional named based content resolution routers, e.g., content resolution routers within a certain geographical area. In addition, SECON supports content-centric security features where contents are encrypted and can only be decrypted when the querying users possess the right attributes that are used to encrypt the data contents.

The content naming approaches for various content-centric solutions that have been proposed are different. For example, NDN uses hierarchical names typically corresponding to organizational structures. This implies name persistence with respect to owner or organizational changes is not satisfied. NDN security concept requires the content identifier be signed by an entity trusted by the users and hence is not as flexible when trust changes due to owner or organizational structure change. NetInf related researchers propose their secure naming solution in [8] to deal with such issues.

Recently, the authors of DMAP [6] presented the design and evaluation of a distributed shared hosting approach for managing dynamic identifier to locator mappings in Future Internet. They suggested hashing object identifiers to IP-address-like identifiers, and using such mapped IP addresses to distribute the object identifier to locator mappings among Autonomous Systems. However, they do not describe how the object identifiers are generated by data owners and how querying users find out about such identifiers. Furthermore, their approach currently does not support keyword based queries.

## III. SOLUTIONS

Assume each content is hosted at a network address  $CA$  with a globally unique identifier  $CID$ .  $CA$  can refer to typical servers or any personal devices such as laptops and smartphones. The host of a content will advertise the mapping between  $CID$  and  $CA$  with a set of keywords that describe the content. Users can access a content through two methods

- 1) If users know the  $CID$ , they can query the corresponding  $CA$ , and then fetch the content from the host.
- 2) If users do not know the  $CID$ , they may use keywords to search all contents. The result will be a list of  $CIDs$  that contains the specified keywords. Then users can apply the first method to obtain the contents.

The host will 'insert' a content in the following form,

$$(CID, CA, \{t_1, t_2, \dots, t_m\}),$$

where  $\{t_1, t_2, \dots, t_m\}$  represents  $m$  keyword terms.

In this paper, we propose two approaches to support keyword search, Independent Search and Merge (ISM), and Integrated Keywords Search (IKS).

### A. General Setting

First, we assign an identifier to each keyword by applying a pre-defined hash function,  $KID = h_{kid}(keyword)$ .  $KID$  and  $CID$  belong to different domains, i.e.,  $\{CID\} \cap \{KID\} = \phi$ , so that they can be distinguished by their values. In practice, we define a set of general IDs  $GID$  which includes both  $CID$  and  $KID$ , i.e.,  $\{GID\} = \{CID\} \cup \{KID\}$ . For any  $g \in \{GID\}$ , the first bit is an indicator,

$$g \in \begin{cases} CID & \text{if the first bit is 0;} \\ KID & \text{if the first bit is 1.} \end{cases}$$

Second, we use another hash function to map any  $GID$   $g$  to an IP address,  $IP_g = h_{ip}(g)$ . Note the result  $IP$  must exist in current Internet. If  $IP_g$  does not exist, the hash function will be repeatedly applied until we generate a valid IP address. Due to the page limit, we omit the details here and we refer the readers to [6] for further discussions.

For both ISM and IKS, the insert process includes two steps, inserting the content and inserting the keywords. The first step of inserting the content is identical in both solutions and similar to DMap [6]. First, the content host applies the hash function  $h_{ip}$  on  $CID$  and generate an IP address  $IP$ . Then, a content insert request  $(CID, CA)$  is sent to the AS that owns  $IP$ . Each AS keeps a content search table which stores the mapping information between  $CID$  and  $CA$ . Once receiving a content insert request, the AS will add the new mapping entry in the table.

For content query, we focus on the keyword search methods, i.e., how to obtain the  $CIDs$  based on the specified keywords. With  $CIDs$  available, users can convert each  $CID$  to an IP address by applying  $h_{ip}$ . Then a content query will be transferred to the AS which is the owner of  $IP$ . After looking up in the content search table, if there is a matching  $CID$ , the AS will send the corresponding  $CA$  back to the user.

In the rest of this section, we present two solutions and focus on keywords-related operation, i.e., how the content host inserts the affiliated keywords, and how users search with keywords and find the  $CIDs$  of their interested contents.

### B. Independent Search and Merge (ISM)

1) *Insert Keywords*: When inserting a set of keywords  $\{t_1, t_2, \dots, t_m\}$ , the content host applies the basic steps in the following Algorithm 1. For each term  $t_i$ , the host converts it to a  $KID$  and then maps the  $KID$  to an IP address  $IP_i$ . Finally, the insert request  $(KID_i, CID)$  is sent to the AS that owns  $IP_i$  following BGP protocol. Therefore, inserting keywords requires  $m$  insert request messages.

On the AS's side, a keyword search table is established to store the mapping information between keywords and  $CIDs$ , i.e., each entry contains a keyword and a list of  $CIDs$  whose contents are described by the keyword,

$KID$	$CID_1, CID_2, \dots$
-------	-----------------------

---

### Algorithm 1: ISM: Insert Keywords

---

```

1 for each keyword  $t_i$  do
2    $KID_i = h_{kid}(t_i)$ ;
3    $IP_i = h_{ip}(KID_i)$ ;
4   Send  $(KID_i, CID)$  to the AS that owns  $IP_i$ ;
5 end

```

---

Upon receiving the insert request  $(KID, CID)$ , the AS will add it in the keyword search table. If  $KID$  is a new keyword, the AS will create a new entry for it and map it to  $CID$ . Otherwise, the AS will look up the  $KID$  in the table and append the  $CID$  at the end of the  $CID$  list of the corresponding entry.

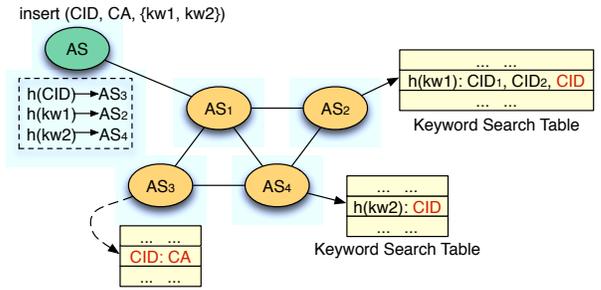


Fig. 1: Example: Insert in ISM

2) *Keyword Search Query*: When querying contents with a set of keywords, the user follows Algorithm 2. Each keyword  $t_i$  is first converted to its  $KID_i$  (Line 3) and then an IP address  $IP_i$  (Line 4). A keyword search query containing  $KID_i$  is routed to the AS that owns  $IP_i$ . The AS will search for  $KID_i$  in its keyword search table. If found, the corresponding list of  $CIDs$  ( $CL$  in Line 6) will be sent back to the user. In Algorithm 2, the variable  $ret$  is the return list of  $CIDs$  and set to empty initially. Upon receiving the  $CL$  from the ASes, the user updates  $ret$  by intersecting the current  $ret$  with  $CL$  (Line 10). Overall, there are  $q$  query messages and  $q$  lists of  $CIDs$  transmitted in this process.

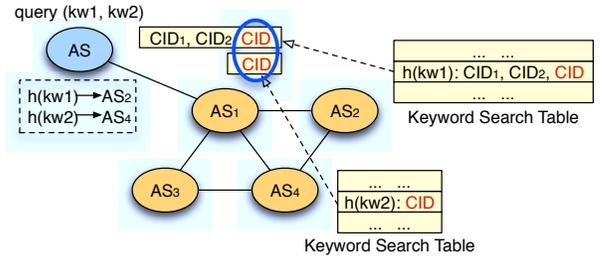


Fig. 2: Example: Query in ISM

### C. Integrated Keywords Search (IKS)

1) *Insert Keywords*: In IKS, the content host considers the combinations of keywords when inserting them. For a set of keywords  $\{t_1, t_2, \dots, t_m\}$ , there are  $2^m - 1$  possible combinations containing at least one keyword. IKS inserts an

---

**Algorithm 2: ISM: Keyword Search Query**

---

**Input:** a set of query terms  $QT = \{t_1, t_2, \dots, t_q\}$ **Output:**  $CIDs$  whose affiliated keywords include  $QT$ 

```
1  $ret \leftarrow \phi$ 
2 for each keyword  $t_i$  do
3    $KID_i = h_{kid}(t_i)$ ;
4    $IP_i = h_{ip}(KID_i)$ ;
5   Send a query with  $KID_i$  to the AS that owns  $IP_i$ ;
6   Let  $CL$  be the returned  $CID$  list from the AS;
7   if  $ret = \phi$  then
8      $ret \leftarrow CL$ ;
9   else
10     $ret \leftarrow ret \cap CL$ ;
11  end
12 end
13 end
14 return  $ret$ ;
```

---

entry for each of these combinations. The details are presented in Algorithm 3. For each subset of keywords, the host first sorts all the terms based on their alphabetical order. Then, one  $KID$  is generated for the entire subset by applying the hash function on the concatenation of all the terms (Line 3). The rest of the algorithm is similar to Algorithm 1. The  $KID$  is mapped to an IP address and then the insert request ( $KID, CID$ ) is sent to the AS that owns  $IP$ . IKS needs  $2^m - 1$  request messages for inserting the keywords. Note that this approach is only applicable to a short keyword list, i.e., when  $m$  is small. For a large  $m$ , there are too many combinations to enumerate which is not feasible in practice.

---

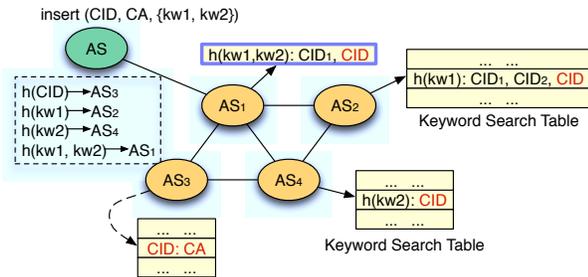
**Algorithm 3: IKS: Insert Keywords**

---

```
1 for each subset of keywords,  $KS = \{t_{c_1}, t_{c_2}, \dots, t_{c_{m'}}\}$ ,
    $m' \leq m$  and  $\forall i \in [1, m'], c_i \in [1, m]$  do
2   Sort the terms in  $KS$  in alphabetic order;
3    $KID = h_{kid}(t_{c_1}, t_{c_2}, \dots, t_{c_{m'}})$ ;
4    $IP = h_{ip}(KID)$ ;
5   Send ( $KID, CID$ ) to the AS that owns  $IP$ ;
6 end
```

---

The insert protocol for ASes is the same as in ISM. Each AS maintains a keyword search table. Once a insert request arrives, the AS will add the new mapping data to the table.

**Fig. 3: Example: Insert in IKS**

2) *Keyword Search Query*: Compared to ISM, keyword search query in IKS is simpler as shown in Algorithm 4. The user first sorts all the terms in the query and then generates a  $KID$  for the sorted keyword set (Line 2). After mapping the  $KID$  to the IP address  $IP$ , the keyword search query is sent to the AS that owns  $IP$ . Similarly, the AS will search for  $KID$  in its keyword search table and return the corresponding list of  $CIDs$  ( $CL$  in Line 5) to the user. In Algorithm 4,  $CL$  will be returned as the final result. Therefore, there is only one query message and response in IKS.

---

**Algorithm 4: IKS: Keyword Search Query**

---

**Input:** a set of query terms  $QT = \{t_1, t_2, \dots, t_q\}$ **Output:**  $CIDs$  whose affiliated keywords include  $QT$ 

```
1 Sort the terms in  $QT$  in alphabetic order;
2  $KID = h_{kid}(t_1, t_2, \dots, t_q)$ ;
3  $IP = h_{ip}(KID)$ ;
4 Send a query with  $KID$  to the AS that owns  $IP$ ;
5 Let  $CL$  be the returned  $CID$  list from the AS;
6 return  $CL$ ;
```

---

#### D. Storage-enhanced Solutions

In practice, the basic solutions we propose above cause unbalanced storage usage among all ASes due to the following two reasons. First, ASes form a hierarchical structure with big diversity as some ‘big’ ASes host very large IP blocks while other ‘small’ ASes may be in charge of fewer IP addresses. When randomly mapping keywords to IP addresses, as a consequence, big ASes may eventually host a large portion of the keywords which consume more storage resources than small ASes. Second, the content list for each keyword is not balanced. Some hot keywords may have a large volume of matching contents while other rarely used keywords may appear in only one content. Therefore, the ASes hosting popular keywords are likely to spend more storage in storing the corresponding matching  $CIDs$ . In summary, in basic solutions ISM and IKS, the storage requirement is unbalanced and some ASes may exhaust the storage resource much faster than others. In this subsection, we propose two enhanced versions that work with both ISM and IKS to improve the storage balance.

1) *-fair solution*: In this enhanced solution, we allocate  $k$  candidate ASes for each insert and try to balance the storage usage among these  $k$  ASes. The basic protocol is as follows.

- Step 1: When inserting a  $(KID, CID)$  pair, the user applies hash functions to find  $k$  distinct destination ASes.
- Step 2: The user sends an initial request to each of  $k$  candidate ASes asking for their current storage usage, i.e., how many  $KIDs$  and  $CIDs$  have been stored there.
- Step 3: Upon receiving the replies from the candidate ASes, the user selects the AS with the minimum storage usage to send  $(KID, CID)$  insert request to.

Fig. 4 illustrates an example of insert in -fair solution. The user first contacts  $k = 3$  candidate ASes ( $AS_2, AS_3, AS_4$ ) and

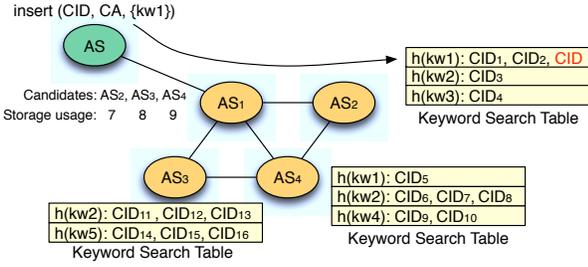


Fig. 4: Example: Insert in -fair solution ( $k = 3$ )

obtains their current storage usages. After comparison, the user picks  $AS_2$  as the destination AS and sends the request to it.

When querying for a keyword, the user has to contact all  $k$  candidate ASes to fetch the  $CID$  lists. Thus a  $q$ -keyword query needs to be sent to  $k \cdot q$  ASes.

2) *-split solution*: The motivation of this solution is to avoid a long list of  $CID$ s. We set a threshold  $\tau$  as the limit of the longest  $CID$  list. When AS  $A$  receives an insert request ( $KID, CID$ ) and finds that the  $CID$  list for this  $KID$  exceeds  $\tau$  items, AS  $A$  splits the  $CID$  by half and migrates the second half to another AS  $A'$ . AS  $A$  records  $A'$  in the remaining content list for  $KID$ . AS  $A$  may split the list again when more contents are added and the  $CID$  list goes over the limit.

When a user issues a query for  $KID$ , the AS  $A$  will return all the  $CID$ s in the list as well as the AS numbers in the list such as  $A'$ . Then the user will further send the query to the received ASes such as  $A'$ . The query process is terminated when there is no AS number in the returned list.

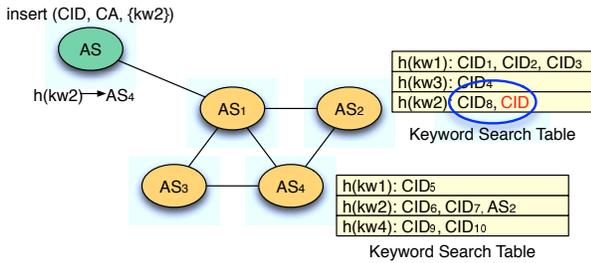


Fig. 5: Example: Insert in -split solution ( $\tau = 3$ )

Fig. 5 shows an example of insert. Assume a user inserts  $kw2$  and the destination AS is  $AS_4$  where 3  $CID$  ( $CID_6, CID_7, CID_8$ ) have already been stored for  $h(kw2)$ . After receiving the new insert request,  $AS_4$  finds that the resulting  $CID$  list will exceed the threshold  $\tau = 3$ . Thus, it splits the list and migrates the second half ( $CID_8, CID$ ) to another AS ( $AS_2$ ). In addition,  $AS_4$  adds  $AS_2$  in the content list of  $h(kw2)$ . We will evaluate both enhancements in Section IV. By default, we set  $k = 5$  for -fair solution and  $\tau = 100$  for -split solution.

#### IV. PERFORMANCE EVALUATION

In this section, we evaluate our proposed solutions based on simulation. We implement ISM and IKS based on the DMap [6] simulator and use the real AS topology and IP prefix allocation which consists of more than 26,000 ASes.

#### A. Workload Characteristics

1) *Contents and keywords*: The content/keyword workload is generated from flickr.com's picture titles. We have crawled around 240,000 random pictures hosted at flickr.com. After we eliminate non-English characters, digits, and special characters, 21,754 pictures have remained each with an effective title. We regard each word in the title as a keyword for the content (picture). In our simulation, we assign each picture a  $CID$  and insert it with the corresponding keywords (i.e., the words in the title). Therefore, our simulation includes 21,754 contents and the host of each content is a randomly selected AS. In total, there are 23,500 unique keywords in our setting.

	Average	Maximum	Std Deviation
# of keywords	2.61	38	2.05
Content freq	2.42	1284	13.35

TABLE I: Content Characteristics: number of keywords per content and content frequency per keyword

Table I lists some basic characteristics of content/keyword workload. The first row is about the number of keywords for each content. The average number is 2.61, the maximum is 38, and the standard deviation is 2.05. The second row represents the statistical information about *content frequency*, which is the number of contents containing a given keyword. More detailed distribution is shown in Fig. 6 as a log-log scale plot. The axis  $x$  is the popularity rank of a keyword, and the axis  $y$  indicates the value of content frequency, i.e., the number of occurrences of the keyword in all contents. Apparently, the keywords follows a power-law distribution as the few top-ranked keywords frequently appear in contents while there is a long tail of low-ranked keywords with much smaller content frequency. Note that for evaluating IKS, we have to set a limit for the number of keyword ( $m$  in Section III-C). In our simulation, we set  $m = 5$  and remove all the contents with more than 5 keywords from the workload.

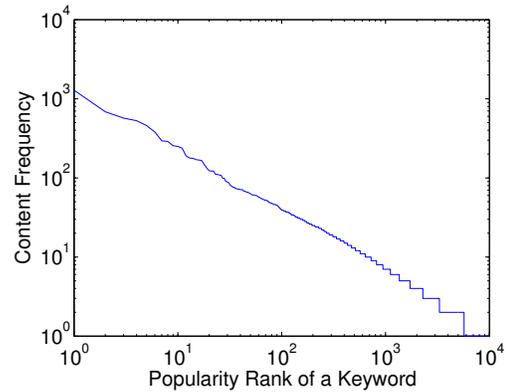


Fig. 6: Keyword Distribution

2) *Keyword Query*: In this paper, we simulate  $q$ -keyword query for  $q \in \{1, 2, 3, 4\}$ . For each value of  $q$ , we generate 5,000 queries each randomly chooses  $q$  keywords from the keyword list in our trace. In order to guarantee that at least one matching contents can be found, we first randomly choose one content  $C$  from a pool of contents which contain at least  $q$  keywords. And then, we randomly pick  $q$  keywords from the

content  $C$ 's keyword list. The table below shows the size of the pool for selecting  $q$  keywords, i.e., the number of contents with at least  $q$  keywords.

$q$ 's value	1	2	3	4
size of content pool	21754	13815	8623	5062

In addition, every query is submitted from a randomly selected AS. The following Table II presents the statistics of the number of matching contents for each type of 5,000 queries. Basically, queries for more keywords tend to have fewer matching contents and smaller deviation on the number of returned contents.

	Average	Maximum	Std Deviation
1 keyword	47.98	1189	171.41
2 keywords	2.23	269	10.40
3 keywords	1.12	111	2.11
4 keywords	1.02	29	0.45

TABLE II: Query Characteristics: # of matching contents

### B. Query Response Time

In this subsection, we evaluate the query response time (RT), one of the major performance metrics we consider. In our keyword search query, the user often needs to contact multiple ASes. For example, if the query contains multiple keywords, those keywords may map to different ASes. Also, the returned results are likely to contain more than one matching contents which may be hosted at different ASes. Thus, the time needed to fetch  $CIDs$  is varying. In our setting, we measure the response time of a query at the user's side from sending out the query to receiving the LAST matching  $CID$ , i.e., the longest response time among those for obtaining all matching  $CIDs$ .

Intuitively, if the requesting AS has to contact some ASes far away with large latency, the response time will be large. Since in our solutions, the mapping to the destination ASes is based on hash functions, each destination AS has equal probability to have a large latency. Statistically, contacting more ASes in a query will lead to a longer response time.

In our simulation, we find that the response time in both solutions roughly follows the trends in the following Fig. 7 which is the case of searching one keyword in ISM (5000 queries). Fig. 7a illustrates the longest 3000 response time in a descend order and Fig. 7b plots the histogram of response time with a bin size of 5ms. First, the sorted response time is a power-law distribution with a long tail (Fig. 7a). Second, within a certain range of time values, the response time is close to a normal distribution, e.g., from 0 to 60ms in Fig. 7b.

Table III compares the average response time in ISM and IKS. We observe that the response time of  $q$ -keyword query in ISM is increased sublinearly when  $q$  increases. The reason is that in ISM, the user fetches a list of  $CIDs$  for each keyword and applies intersection to obtain the matching contents. Therefore, query with an additional keyword requires contacting one more AS or the same number of ASes if the new keyword is mapped to one of the existing destination ASes. In contrast, the response time in IKS is consistent for different number of keywords, because the user always contact one AS for keyword search.

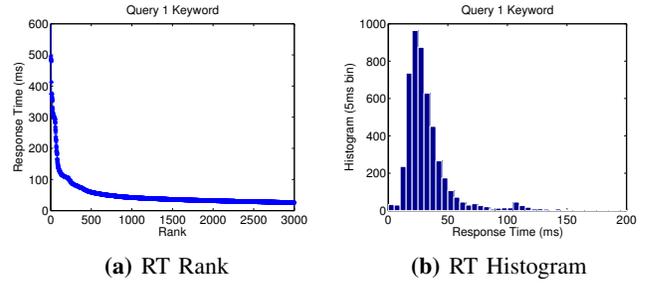


Fig. 7: Query Response Time Distribution in ISM (1 keyword)

	1 keyword	2 keywords	3 keywords	4 keywords
ISM	37.3	45.1	52.3	57.8
IKS	37.3	36.8	38.3	37.4

TABLE III: Average Query Response Time (ms)

The CDF distributions of response time in ISM and IKS are presented in Fig. 8 and Fig. 9 respectively. In Fig. 8, all curves have a flat start, and then increase sharply around their mean values. Most of the queries are answered within 150ms. We also observe that querying more keywords incurs longer latency. For example, when the user searches for one keyword, 87.8% queries are responded in 50ms. With 4 keywords, however, only 66.4% queries can be completed in 50ms. In Fig. 9, we can see that there is almost no difference among the four curves for IKS and they are very close to searching one keyword in ISM.

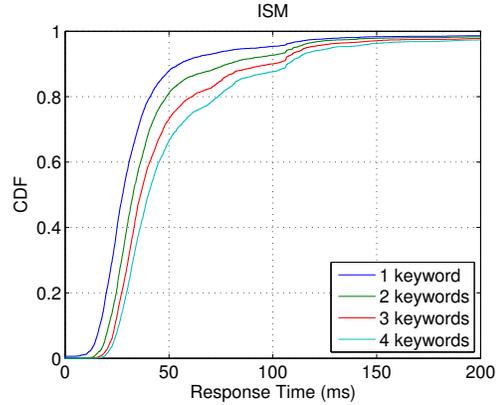


Fig. 8: Query Response Time in ISM

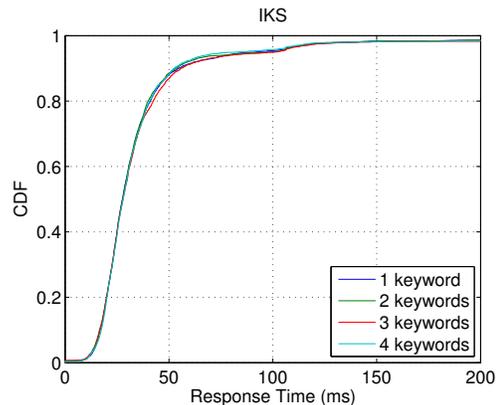


Fig. 9: Query Response Time in IKS

### C. Storage Requirement

In this subsection, we evaluate how much storage space is required for both ISM and IKS. In our solutions, a keyword search table is maintained at each AS which consists of a column of *KIDs* and a list of *CIDs* for each *KID*. For simplicity, we assume *KIDs* and *CIDs* have the same bit length, e.g., 128 bits in our setting. Thus, the storage requirement can be quantified and compared by the number of *KIDs* and *CIDs* in the keyword search table.

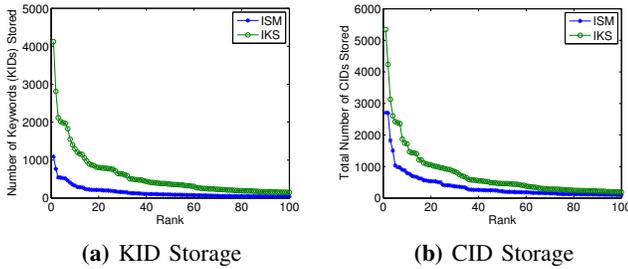


Fig. 10: Comparison of Storage Requirements

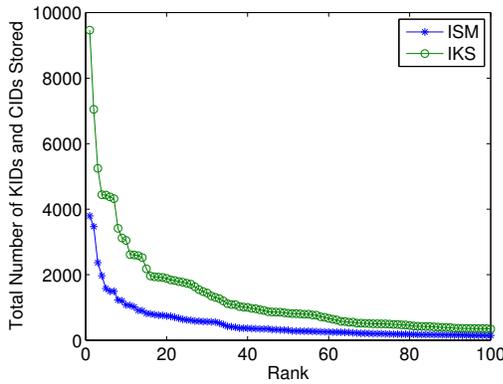


Fig. 11: Total Storage in ISM

Fig. 10 compares the storage needed for *KIDs* and *CIDs* in ISM and IKS based on the rank of all ASes (in terms of the storage requirement, i.e., higher ranked ASes require more storage). The total storage requirements are shown in Fig. 11. First, we can see IKS needs more space than ISM as expected because we enumerate and insert combinations of keywords in IKS. On average, each AS in ISM stores 3.64 *KIDs* and 8.7 *CIDs*. In IKS, however, the average values become 14.16 and 18.25 respectively. Second, as we mentioned earlier, the storage on each AS is not balanced. The following Table IV gives a quantitative view by measuring the standard deviations. Both solutions yield large deviations and IKS is more dynamic than ISM in terms of storage allocation among ASes. The largest storage requirement for an AS in ISM is to store 3797 *KIDs* and *CIDs* in total which need 60.75K bytes. In IKS, the largest storage is 9465 *KIDs* and *CIDs* (151.44K bytes).

	<i>KID</i> storage	<i>CID</i> storage	total storage
ISM	27.6	72.8	99.6
IKS	106.3	139.9	245.8

TABLE IV: Standard Deviation of Storage

### D. Storage-enhanced Solutions

To address the issue of unbalanced storage, we have proposed two storage-enhanced solutions. In this subsection, we present their performance and due to the page limit, we focus on the enhanced versions for ISM. The total storage requirements are compared in Fig. 13 and the breakdowns are shown in Fig 12.

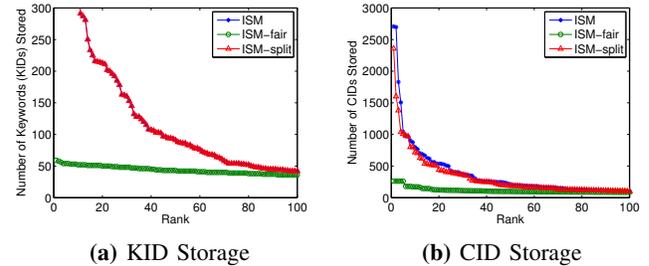


Fig. 12: Comparison of Storage Requirements in ISM

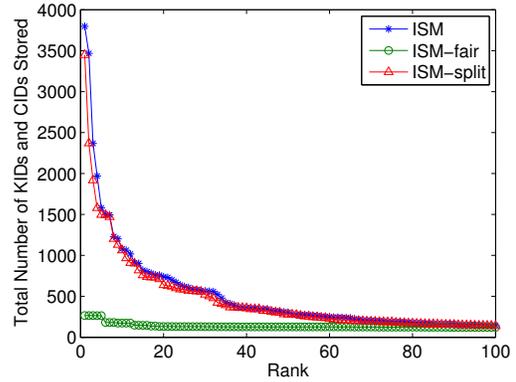


Fig. 13: Total Storage Requirement in ISM

Basically, ISM-split tries to avoid a long list of *CIDs* for a particular keyword while ISM-fair targets on balancing the total storage usage among all (5) candidates when inserting a keyword. As shown in Fig. 12a, ISM-split does not help in balancing the *KID* storage because the keywords are distributed in the same way as in ISM. ISM-fair, however, significantly improve the keyword allocation with a much more smoother curve. The maximum number of keywords hosted at an AS has dropped to 59 in ISM-fair. In Fig. 12b, both enhancements improve the storage balance. Due to the impact from keyword allocation, ISM-fair is still much superior to ISM-split and ISM. The values of their standard deviations are listed in the table below.

	<i>KID</i> storage	<i>CID</i> storage	total storage
ISM	27.6	72.8	99.6
ISM-fair	8.0	20.3	26.7
ISM-split	27.6	59.8	87.1

TABLE V: Standard Deviation of Storage

On the other hand, both enhanced versions sacrifice response time to achieve the more balanced storage. In ISM-fair, the user needs to contact 5 candidate ASes for each keyword. Thus, a *q*-keyword query in ISM-fair is roughly equivalent to

5 ·  $q$ -keyword query in ISM. For the same threshold of 50ms for the response time, ISM-fair can finish 64.7% one keyword queries and only 11.5% 4-keyword queries. In ISM-split, the number of ASes the user has to contact is not certain. The user will keep sending request until there is no AS number in the returned list. Overall, the response time performance in ISM-split is much better than ISM-fair especially for multiple keywords search. Compared to ISM, however, there is still a big degradation, e.g., for 4-keyword queries, ISM can respond to 87.6% of them while ISM-split can finish only 61.8%.

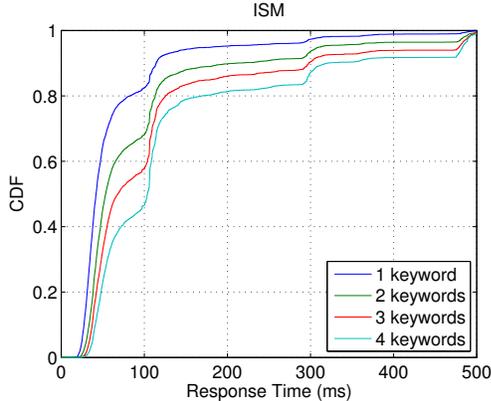


Fig. 14: Response Time in ISM-fair ( $k = 3$ )

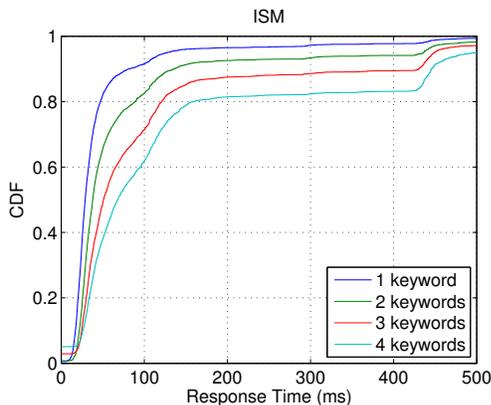


Fig. 15: Response Time in ISM-split ( $\tau = 100$ )

## V. DISCUSSIONS AND FUTURE WORK

### A. Performance of Inserts and Updates

For simplicity, this paper omits the study on the performance of insert requests. In the future work, we plan to measure the overhead of inserting keywords and contents, especially study how a bursty insert workload may impact the AS system. In addition, we consider a stable workload setting in this paper including AS topology, content/keyword, and the network addresses of contents (CA). Our future work will investigate more dynamic environments considering BGP updates, keyword updates for contents, and CA updates for content host changes (e.g., mobile devices that attach to different entry networks).

### B. Content/Keyword Replicas

To improve the response time, one common approach is to deploy replicas for both contents and keywords (it is

also discussed in [6]). We have implemented the support for multiple replicas in both ISM and IKS. By exploiting the advantage of locality, storing multiple replicas reduces the response time in both solutions. As a tradeoff, the storage requirements are increased too. Due to the page limit, we omit the results in this paper.

### C. Keyword Filtering

The keyword workload in our simulation can be further filtered or pruned. We plan to obtain a bigger trace file and filter the non-existing words and ‘stop words’ such as ‘the’ and ‘a’ as usual web mining does. We hope the modified keyword workload would provide us a better understanding on the performance in a realistic environment. It is also interesting to explore how we can modify our solutions to deal with typos in keywords.

## VI. CONCLUSION

This paper proposes two solutions, ISM and IKS, that enable users to search for keywords in a content-centric network. Our framework is based on and compatible with the current BGP system formed by ASes. A keyword search table is maintained at each AS to assist the user to locate the matching contents. In addition, we propose two enhanced versions that improve the storage balance. Finally, we evaluate our solution based on simulation with realistic workload and the results show that our solutions yield a small keyword query response time with a reasonable storage requirement.

## REFERENCES

- [1] M. C. Chuah and X. Xiong, “Secure content centric mobile network,” in *GLOBECOM*, 2011.
- [2] Pingdom, “The mobile web in numbers,” <http://royal.pingdom.com/2011/12/07/the-mobile-web-in-numbers/>, 2011.
- [3] Appbrain, “Number of available android applications,” <http://www.appbrain.com/stats/number-of-android-apps>, Jul. 2012.
- [4] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, “Networking named content,” in *CoNEXT '09*, 2009.
- [5] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, “A data-oriented (and beyond) network architecture,” in *SIGCOMM '07*, 2007.
- [6] T. Vu, A. Baid, Y. Zhang, T. D. Nguyen, J. Fukuyama, R. P. Martin, and D. Raychaudhuri, “Dmap: A shared hosting scheme for dynamic identifier to locator mappings in the global internet,” in *ICDCS*, 2012.
- [7] B. Ahlgren, M. D’Ambrosio, M. Marchisio, I. Marsh, C. Dannewitz, B. Ohlman, K. Pentikousis, O. Strandberg, R. Rembarz, and V. Vercellone, “Design considerations for a network of information,” in *CoNEXT '08*, 2008.
- [8] C. Dannewitz, J. Golic, B. Ohlman, and B. Ahlgren, “Secure Naming for a Network of Information,” in *INFOCOM IEEE Conference on Computer Communications Workshops*, 2010, Mar. 2010, pp. 1–6.
- [9] L. Wang, A. Afanasyev, R. Kuntz, R. Vuyyuru, R. Wakikawa, and L. Zhang, “Rapid traffic information dissemination using named data,” in *Proceedings of the 1st ACM workshop on Emerging Name-Oriented Mobile Networking Design - Architecture, Algorithms, and Applications*, ser. NoM '12, 2012.
- [10] Z. Qin, X. Xiong, and M. Chuah, “Lehigh explorer: Android application utilizing content centric features,” in *1st GENI Research and Educational Experiment Workshop*, 2012.
- [11] J. Chen, M. Arumathurai, L. Jiao, X. Fu, and K. K. Ramakrishnan, “Copss: An efficient content oriented publish/subscribe system,” in *ANCS '11*, 2011.