Espresso: A Data Naming Service for Self-summarizing Transport

Jongdeog Lee, Md Tanvir Al Amin, Tarek Abdelzaher
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois
{jlee700, maamin2, zaher}@illinois.edu

***Abstract*—Recent work suggested that, in the age of data overload produced by sensors, social media, and IoT devices, a key new type of network transport protocols will be one that offers *representative summaries* of requested data, retrieved at a consumer-controlled degree of granularity. Given the over-abundance of data, consumers will seldom need *all* data on a topic, but rather will increasingly favor an appropriate *sampling* for summarization purposes. The paper explores such sampling as a novel service enabled by information-centric networking paradigms that name data objects, not hosts. By naming data objects, it becomes possible to selectively retrieve them, but the properties of the resulting sampling depend on the naming scheme. This paper describes an automated object naming service, called Espresso, that facilitates content sampling over information-centric networks. We show how Espresso, combined with a trivial retrieval policy, translates the sampling problem into a naming problem, and customizes the naming to different applications' sampling needs. Experimental results show that the computational overhead of automated naming is affordable. The service is first evaluated in simulation, demonstrating a higher sampled-data utility to the consumer, while balancing retrieved data importance and diversity. Social network applications are then introduced, where naming is produced by Espresso. Results demonstrate the advantages of Espresso, compared to baselines, in terms of retrieving meaningful media data summaries.**

I. Introduction

This paper describes *Espresso*, a data naming service for the age of sensing, social media, and the Internet of Things. The goal of this service is to automatically name content in a manner that facilitates efficient retrieval of extractive *summaries*[1] over information-centric networks (ICNs). Recent work on ICNs argues for use of machine-generated names [1], because users are less interested in remembering cumbersome identifiers (even when they are human-readable). In this paper, names are automatically generated to support *content summarization* at different degrees of granularity.

The work is motivated by the view that modern sensing and social media applications will increasingly need content sampling services to combat data overload [2]. Data users will require well-selected samples of available data, as opposed the entire set. This is already true of social networks, for example, where users who search for content on a topic (e.g., via Twitter or Instagram) do not really need to see all matching tweets and pictures, but rather a representative selection thereof – a well chosen subset that summarizes the key issues. Thus, services are needed to support sampling or summarizing content.[2]

Espresso adopts a producer-consumer model proposed in recent literature [2]. The producer exports a data set of interest. Different consumers want to retrieve samples (*i.e.*, summaries) of the data set or parts thereof, at a consumer-specified degree of granularity.

We consider summarizing data sets of *homogeneous objects*, such as sets of microblog entries, pictures, news headlines, or product reviews. An ideal sampling method should select objects in a manner that maximizes some notion of information utility of the retrieved summary. Intuitively, an object should be included in the summary if it is both (i) more important in some application-specific sense and (ii) less redundant with previous objects already included in the summary.

ICNs can map the problem of *content sampling* to one of name-space design if the naming can allow retrieving objects matching a query in an order that meets the two aforementioned criteria. *Espresso* is such a naming service, developed for ICNs with hierarchical name spaces.

Espresso names data hierarchically, based on an application-specific content similarity metric, such that more similar data are closer together in the name space. It allows consumers to retrieve content summaries by traversing the name space in a particular generic order. Moreover, applications can zoom in and out to retrieve content at different degrees of detail. We describe the design and implementation of *Espresso* as well as its evaluation both in simulations and in the context of a Twitter-based application.

Simulation results demonstrate that *Espresso* achieves a higher utility for retrieved data than compared baselines. An empirical evaluation of a Twitter-based news application supports the same observation. The evaluation shows that consumers can control the level of summarization (of news extracted from Twitter) by zooming in or out on the topics.

The rest of this paper is organized as follows. A high-level description of naming and summarization is provided in Section II. Section III and Section IV explain overall design and implementation of *Espresso*, respectively. Performance

[1]Extractive summarization refers to summarization achieved by extracting samples of the data set being summarized, as opposed to generating a brief high-level description.

[2]In the rest of this paper, we use the term *sampling* and *summarization* interchangeably.

978-1-5090-6599-8/17/$31.00 ©2017 IEEE

experiment and simulation are provided in Section V. Section VI demonstrates two use-case applications that benefit from *Espresso*. We discuss related work in Section VII and conclude paper in Section VIII.

II. Naming, Retrieval Order, and Summarization

To explain the relation between naming, retrieval policy, object retrieval order, and data set summarization, consider the hierarchical name tree shown in Figure 1. We shall use it below to illustrate how the problem of representative sampling can be translated into one of name space design in information-centric networks that support hierarchical names.

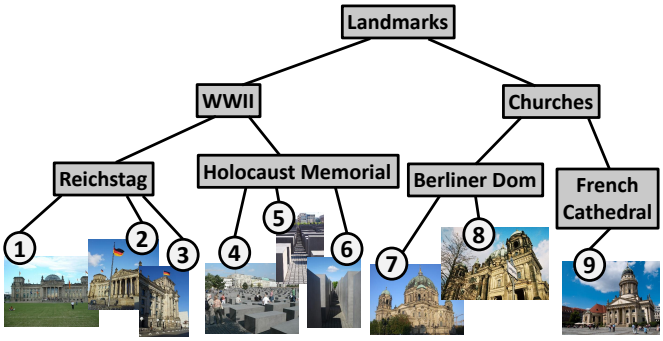


Fig. 1: Berlin Landmarks Set

A. Representative Sampling

Let us first informally discuss what we mean by representative sampling. In general, representative sampling is a sampling that preserves the desired statistic of the sampled phenomenon. Hence, what constitutes a representative sampling of data differs from one application to the next depending on what statistics are sought. If the goal is to determine average traffic speed on different streets, then a roughly equal number of samples from cars on each street may suffice, independently on the total number of cars on each street. However, if the goal is to tally (an approximate histogram of) good and bad reviews of a product, it is more representative if the samples of each opinion category were chosen proportionally to their prevalence, as opposed to independently of the number of reviews in the category.

In a later section, we show how *Espresso* allows applications to customize their own notion of representative sampling. For the purposes of the illustrative example below, let representative sampling refer to a sampling that groups data into categories (of partially redundant objects), and returns a roughly equal number of representatives of each category, chosen in the order of quality, as defined by the application. Below, we show a naming scheme, then explain how it leads to representative sampling.

B. Naming

Consider the tree shown in Figure 1. The data objects are images of various landmarks in the city of Berlin. All data objects are *leaves* of the tree. The rest of the tree comprises a hierarchical name space, where each leaf has a UNIX-like name. For example, the name `Landmarks/WWII/Reichstag/1` refers to the leftmost picture; a front view of the famous Reichstag building.

We call the set of objects at the leaves, the *data set*. Note that, the data set is partially redundant. Pictures are grouped into categories, each representing the same landmark. There are multiple pictures of each landmark. Furthermore, in the figure, pictures of the same landmark are organized left-to-right in decreasing degree of some application-specific quality metric. The metric chosen here is completeness (namely, percentage of the landmark visible in the camera's field of view). Partial views of the same landmark are to the right of their siblings, whereas more complete views are to the left. This is just one illustrative application-specific organization for sibling nodes by quality. Other ways to define quality are possible. In general, let siblings be organized left-to-right by decreasing value of some application-specific quality metric we call *weight*, w_i, defined for each object, O_i. (For now, let us apply this order only to the leaves.)

Finally, note that the tree is organized such that *more related landmarks share a longer name suffix*. For example, all landmarks related to World War II start with `Landmarks/WWII`, and all pictures of the Reichstag start with `Landmarks/WWII/Reichstag`. In this work, each data name is logically divided into two: a prefix and a suffix. The prefix is a common name of all the data within the data set, whereas the suffix denotes an individual data item followed by the prefix. Hence, the name tree has the property that objects that share a longer name suffix are potentially more redundant and similar.

C. Retrieval Policy and Retrieval Order

Intuitively, a representative sampling of the above tree would be to retrieve at least one picture of each landmark or higher-level landmark category (depending on desired granularity). We call the set of rules that algorithmically define how to choose the next object to retrieve from a name space, the *retrieval policy*. The actual sequence of objects selected when the policy is applied is called, *retrieval order*. Let us consider the following simple retrieval policy:

1) First, associate a counter with each branch in the tree (to count how many times it was traversed) and initialize it to zero.
2) To retrieve an object, start at a specified vertex, Q, and recursively descend down the branch with the lowest counter value (or the leftmost branch in case of a tie). As you traverse a branch, the counter of the traversed branch is incremented.
3) When you reach a leaf, select the object for retrieval and go to step 2 again to find the next object.

We shall call it the InfoMax policy, referring to the work where it was first presented [2]. Starting at the root of the tree shown in Figure 1 (i.e., $Q =$`Landmarks`), the above policy retrieves the pictures in the order 1, 7, 4, 9, 2, 8, 5, 3, 6.

D. Properties of Summary

For our example tree, the policy, mentioned above has two favorable properties. Namely, (i) it samples representatives of each cluster in decreasing granularity, and (ii) selects pictures of a given landmark in decreasing order of quality.

Specifically, starting with node Q =`Landmarks`, it first selects a picture from each landmark category (picture 1 from `WWII` then picture 7 from `Churches`), then it continues the selection to include a picture of each individual landmark (retrieving 4 and 9 that cover the Holocaust memorial and the French Cathedral after having retrieved 1 and 7, covering the Riechstag and Berliner Dom). This is a consequence of the hierarchical organization of the tree by content similarity. Finally, it retrieves subsequent pictures of each landmark in decreasing order of quality. This is a consequence of the tie-breaking rule (that favors the leftmost branch), combined with our tree organization that orders sibling objects left-to-right by decreasing weight.

The above favorable properties are independent of the starting node, Q. For example, starting with node Q =`Landmarks/WWII`, the algorithm retrieves pictures in the order 1, 4, 2, 5, 3, 6. Note how all WWII landmarks are covered first, then additional lower weight images of each landmark follow.

It is important to note that the above favorable properties are not inherent in the InfoMax retrieval *policy*. Rather, they arise from interaction of the policy with the *specific way the name tree of is organized*. The example illustrates that an appropriate hierarchical naming scheme, combined with a simple and generic retrieval policy, allow retrieving objects matching a query (that specifies a starting node, Q) in a manner that achieves representative sampling.

In the presented example, the name tree was manually generated. The question addressed in this paper is: how to do so automatically, such that the above favorable summarization properties hold for the resulting retrieval order?

III. The Design of Espresso

In this section, we describe the design of *Espresso*, illustrating automatic generation of name spaces that lead to representative sampling when retrieved using the simple rules comprising the InfoMax policy [2].

A. Overview

The overall architecture of *Espresso* can seen in Figure 2. *Espresso* runs on the producer side. To name objects, it takes three inputs from the producer application: a (pointer to a) data set, a distance metric (to customize the application-specific notion of object similarity), and configuration parameters that customize the notion of representative sampling. The first two inputs customize the generation of names by the *naming engine*. The last input customizes the retrieval policy implemented by the *prioritizer*.

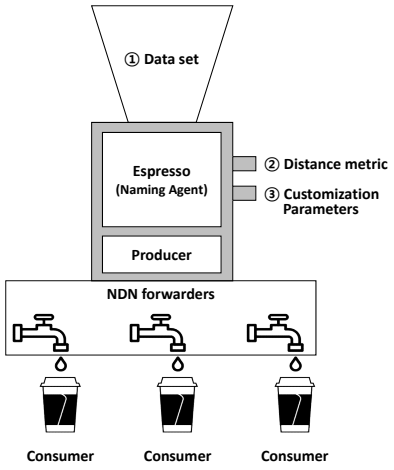


Fig. 2: Overall architecture of *Espresso*

B. Automated Naming

The purpose of the naming engine is to build a hierarchical name space for a given data set. Note that, an automated name is a *suffix* and *Espresso* generates a full name by concatenating it with a given *prefix*. In ICNs, routing is only associated with the *prefix*. This property nicely decouples naming concerns related to routing efficiency from naming concerns related to efficiency of summarization. This paper is concerned with the latter only.

Espresso requires an application-specific distance metric that defines the notion of redundancy from the application's perspective. For example, if data objects comprise sensory data, the distance function might simply be the difference in value. If objects refer to pieces of text, a distance function might be the cosine similarity, a commonly used metric for deciding how similar two pieces of text are. If objects comprise images taken, a distance function might be the color correlogram, or might depend on metadata such as time and location of images, and associated image tags. In our Twitter application example, the Jaccard index, a text similarity metric [3], was used as a distance metric for tweets.

Inspired by the approach described in SocialTrove [4], the overall procedure is illustrated in Figure 3. The example shows how unstructured image data is converted into a hierarchical name space. A detailed explanation for each step follows.

1) Graph Constructions: The first step is to build a graph from a data set. The graph consists of nodes and edges, where a node represents a data object and an edge denotes distance between two data objects calculated by the given distance function. In this particular example, the distance metric can be defined as location distance. That is, images taken at nearby places will have a shorter edge. For example, two Berliner Dom images in Figure 3 may have the shortest distance among all since both images are taken from the approximately same place.

2) Hierarchical Naming: The second step is to build a tree from the graph in the previous stage. To this end, a hierarchical

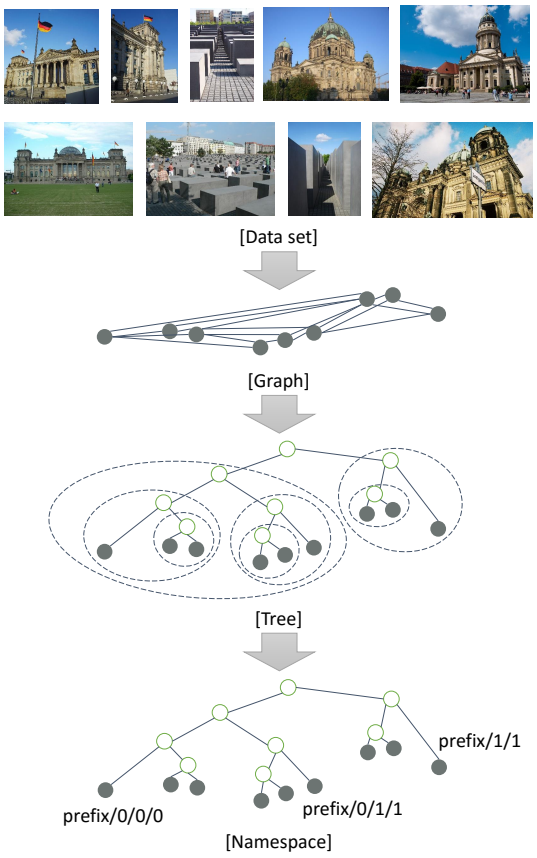


Fig. 3: Procedure to convert a data set to a hierarchical name space

clustering algorithm is applied to group similar objects. One naive and expensive algorithm is to calculate a pair-wise distance among all data, which takes $O(n^2)$. Considering the massive volume of data being generated in this age, however, it is simply not affordable. Instead, we exploit an efficient approximation of the K-means clustering algorithm. Nodes are recursively grouped into K disjoint sets based on distance. Each set is further subdivided until single objects are reached. Figure 3 illustrates the process, where K is 2. Siblings are ordered according to the application-supplied weight metric, and leaf objects are named by concatenating the branch IDs that lead to the leaf from the root. In the case of a binary tree, a branch can have an ID 1 or 0. Our implementation currently supports only binary trees following the example of SocialTrove [4]. Figure 3 shows the names assigned to each leaf node.

Although the K-means clustering algorithm is known to be NP-hard, there are several heuristic algorithms that can accelerate the running time. For example, time complexity of Lloyd's algorithm is $O(wnkd)$ for k centers, n vector points in d dimensions, and w iterations until convergence [5]. Note that, w may vary depending on data points, but the Lloyd algorithm often has polynomial smoothed complexity as shown in [6] except few extreme cases. If the complexity of K-means clustering is linear, the complexity of hierarchical clustering described above would be somewhere between $O(n \log n)$ and $O(n^2)$ depending on balance of the binary tree. That is, if the tree is well-balanced, then the algorithm would be closer to $O(n \log n)$. Otherwise, it would be closer to $O(n^2)$. In practice, the time complexity of *Espresso* is even close to $O(n)$ as shown in the evaluation section.

Indeed, a clustering algorithm that is not dependent on a pre-defined number of clusters, K, would be able to construct more adaptive name trees according to data distributions. However, it is known to be hard and itself could be an interesting research topic [7]. Instead, *Espresso* allows an application to control prioritization parameters to support various sampling requirements. More details can be found in the following.

C. Customizable Prioritization

Once a tree is constructed, applications may configure how the tree is traversed. The following parameters are defined to offer configurability:

- *Weight*, w_i: defined for each object and denotes its priority among its siblings.
- *Aggregation*: a function that specifies how the aggregate weight of a node is computed given the weights of each of its children. Examples include sum or max.
- *Merge order*: a function that defines how sibling branches are multiplexed given aggregate branch weights and individual object weights within each branch.

Logically, leaves are organized in decreasing order of weight, w_i. Intermediate nodes are organized in decreasing order of the output of the aggregation function, applied to each sibling. To generate a retrieval order, branches are merged bottom up. When merging sibling branches, nodes of different branches are multiplexed. For example, their nodes may be strictly interleaved in a round-robin fashion, or they may be merged proportionally according to their branch weights.

In all cases, the output is a sequence that gives rise to an ordered list of objects associated with each node in the tree, leaves in the subtree hanging from that node. Below are example orders produced by the aforementioned customization:

- *Diversity only*: It samples branches equally in a round robin fashion, regardless of weights.
- *Proportional to aggregates*: This order selects more samples from the bigger clusters, proportionally to their size.
- *High aggregates*: It is the default *Espresso* order that samples branches equally, most important first.
- *Above threshold*: This order samples only those branches that have higher weights than a threshold.

Each order above can meet different application sampling needs.

The retrieval policy used is modified version of merge sorting. Intermediate nodes merge leaf nodes from bottom to top and sort them in the desired retrieval order. The process terminates after every data object gets sorted.

D. Handling dynamic updates

Espresso is primarily designed for organizing static data. If data is dynamically updated, *Espresso* needs to re-construct

Retrieval patterns	Weight	Aggregation	Branching order
Diversity only	*	*	Least visited
Proportional to aggregates	1	Sum	Proportional to sum
High aggregates	From application	Max	Least visited, High aggregates
Above threshold	From application	Max	Least visited, Above threshold

TABLE I: Variations of *Espresso*

Algorithm 1 Bottom-up prioritization algorithm

```
1:  procedure BUPRIORITIZER(Node N)
2:      L ← empty list
3:      if N is leaf then
4:          L.append(N)
5:          return L
6:      for each children C of N do
7:          L.append(BUprioritizer(C))
8:      L ← merge(L, N)
9:      return L
10:
11: procedure MERGE(List L, Node N)
12:     List ← empty list
13:     agg ← weight of N.children
14:     cnt ← 0
15:     while L is non-empty list do
16:         selected ← mergingOrder(agg, cnt)
17:         cnt[selected] ← cnt[selected] + 1
18:         node ← L[selected].pop(0)
19:         List.append(node)
20:         updateAggregate(agg[selected], node.weight)
21:     return List
```

the hierarchy. This will introduce extra overhead especially when data changes frequently.

In order to prevent these problems, *Espresso* uses time windows. A name tree is constructed for data residing in the same window, once the window is closed (i.e., the time corresponding to the window range elapses). Hence, the generated tree never changes. New objects are put in a new window. *Espresso* allows applications to control the size of the time window in accordance with their data freshness needs, essentially balancing freshness and tree construction overhead. For example, in our Twitter news application, we have set a 1-hour time window to refresh news-feeds. Hence, generated summaries are at most one hour old.

The main reason to use the above approach is the data immutability principle of ICNs [8]. That is, data packets are uniquely named and they are immutable after being published. Deleting the existing content or updating objects would violate this principle. Naming with a version may help solve this problem, but it may cause cache inefficiency as one data item will have multiple names. The time-window-based approach guarantees that data is uniquely named, and thus complies well with the principle. Moreover, increasing the number of trees does not cause a serious scalability problem. This is because the constructed trees are considered to be small as they only contain pointers to content, not actual content.

IV. Implementation

We implemented *Espresso* on top of the named data networks (NDN) code-base; specifically, the producer and consumer context. The overall procedure of exchanging sampled data objects between a producer and a consumer is sequentially outlined as follows.

Espresso names individual data objects, added by a producer, and creates a hierarchical name space for a given data set and uses it to generate a retrieval order associated with each node in the name space (for objects in its subtree) using the *Espresso* prioritizer. The retrieval order for each subtree is cut into lists of object names. These lists themselves are treated as objects having the same prefix as the objects named on the list.

A consumer application requests samples of a subtree by issuing a request (interest) with a specific prefix and a list number (starting with list 1). Upon receiving this request, the producer sends the corresponding list. Next, the consumer application requests the actual data objects in the order they appear on the list. If all the named objects in the received list are consumed, the consumer may request the next name list to retrieve more samples. This process is iterative until the consumer decided that it has seen enough data.

Espresso is piggybacked on *InfoMax* to implement the above protocol. *InfoMax* is implemented as one of the data retrieval protocols in the consumer APIs of NDN [9], [10]. It is designed to provide more representative samples first by transmitting data in the least-shared-suffix order. The underlying assumption is that more similar content shares a longer name suffix. In this sense, *Espresso* is easy to plug into *InfoMax* since the automated namespace naturally satisfies this requirement. Also, the *Espresso* prioritizer exploits *InfoMax* name lists to support sampling customizability. That is, the sampling order decided by the prioritizer can be not only the least-shared-suffix, but also one of several variations that meets the application needs.

V. Evaluation

This section first evaluates latency of automatic naming mainly caused by the hierarchical clustering algorithm. It then presents functional performance analysis of *Espresso*. The following subsections describe each experiment in detail.

A. Latency of automatic naming

The main purpose of this experiment is to understand the overhead of *Espresso*. To this end, we have measured the running time of automated naming for a set of tweets with

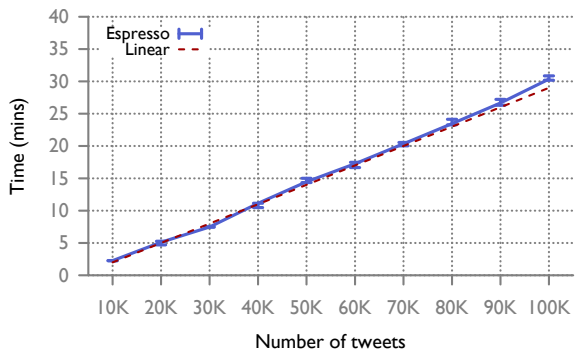


Fig. 4: The running time to construct a hierarchical name space with increasing the number of tweets

increasing the number in order to evaluate performance of *Espresso*. Note that we have prepared 10 different data sets with the same size to avoid the data-dependent result. The result is based on average and standard error for all measurements. The measured latency includes the cost of graph construction and hierarchical naming for the given number of tweets. As we discussed in the design section, the main cost of *Espresso* is latency induced by hierarchical clustering. We have shown that the iterative 2-means clustering method would be approximately $O(n \log n)$ in the average case and we will support this argument with the following experiment result.

Figure 4 shows that latency of *Espresso* increases slightly faster than the linear trend-line. We believe that this successful result is due to several optimization efforts such as exploiting the triangle inequality to avoid unnecessary distance calculation as described in [11]. In practice, we could not find overhead problems in the Twitter news application when setting 1-hour time window because the number of tweets crawled for a certain topic (keywords) during the specified period mostly does not exceed 10,000.

The latency can be further optimized by the distributed manner. Since divided groups are independent to each other, they can be executed simultaneously. Therefore, with multiple workers, it would be able to further reduce the currently measured latency. The assumption of multiple workers is reasonable because *Espresso* is run at the producer-side which is often considered to have rich computing resources. Further optimization technique can be studied in future.

B. Simulation for functional performance

We have designed a simulator that can generate various data sets, where objects are separated in some customizable manner. Specifically, the simulator can randomly distribute object metadata in a two-dimensional Euclidean plane, then compute Euclidean distance as a distance metric. Each point gets assigned a random weight based on the normal (Gaussian) distribution with specified mean (μ) and standard deviation (σ).

We assume that the utility of receiving an object O_i is:

$$U_i = W_i \times D_i \ (0 \leq D_i \leq 1)$$

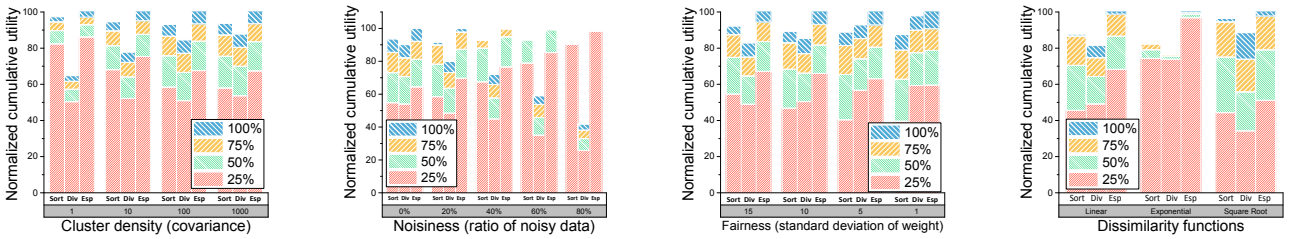
where U_i, W_i, and D_i denote ith object's marginal utility, weight, and dissimilarity. D_i is a function of distance, where longer distance means less redundancy. The dissimilarity function is non-decreasing function, and there can be many ways to define it because notion of redundancy can vary depending on applications. The simulator can choose three types of dissimilarity functions, *linear*, *exponential*, and *square root*. The *linear* dissimilarity function is defined as follows: $D_i = \min(\frac{d_i}{t},\ 1)$ where d_i denotes the minimum distance of the ith object from any of the previously transmitted items. For example, if (0, 0) and (0, 5) are previously sent, then the minimum distance of (0, 1) from either of them is 1. This is because redundancy would be decided by the most similar data (*i.e.*, the closest data). The parameter, t, denotes the threshold that distinguishes two objects as non-overlapped. That is, if the distance between two objects is longer than t, they are considered to be different.

The simulation results are summarized in Figure 5. The graphs show cumulative information utility, ΣU_i, at 25%, 50%, 75%, and 100% transmission of a data set. As can be seen in the figures, marginal utility is diminished as getting more and more data objects. For obvious comparison among different algorithms, ΣU_i is normalized to 100, the maximum score of each experiment. Note that the utility value is computed from the utility function defined above. Three algorithms are compared, *sort*, *diversity*, and *Espresso*. *Sort* is the sorted order by weight, W_i, from the largest to the smallest. *Diversity* is the order of maximizing dissimilarity, D_i. *Espresso* considers harmony between the two and is configured to maximize utility for received objects. As described in III, Espresso can adjust branching (or merging) function according to application-specific utility function. In this case, it first divides aggregates of nodes by visit counts and selects the highest value. In this manner, *Espresso* achieves the highest marginal utility than two baselines by pursuing high U_i, not simply W_i or D_i.

The main observation that can be found from Figure 5 is customizability of *Espresso*. Other monotonic algorithms have certain pros and cons in accordance with data characteristics. For example, *diversity* is favorable for unfairly weighted clusters whereas *sort* is not. *sort*, on the other hand, would be preferable for highly noisy data over *diversity*. *Espresso* always achieves the best utility among all even when different dissimilarity functions are applied. This is because *Espresso* allows applications to customize the branching (or merging) order to their utility functions. This fine-tuning customization only requires a few lines of code that define selection orders for given aggregates and counts. The properly defined function shall allow *Espresso* to inherently achieve the best marginal utility than other baselines.

VI. Application

This section describes applications supported by *Espresso*. Two applications that have different sampling needs are demonstrated: Twitter news and product review sampler. The following subsections explain details on each application with outputs on real data.



(a) cluster density (b) noisiness of data (c) weight distribution fairness (d) dissimilarity functions

Fig. 5: Normalized information utility in accordance with different cluster density, proportion of noisy data, weight distribution fairness, and dissimilarity functions

A. *Twitter news*

An example application enabled thanks to *Espresso* is automatic extraction of news from Twitter. The Twitter news producer uses the Twitter search API [12] to crawl actual tweets by keywords that represent multiple topics such as disasters, politics, civil unrest, epidemics, and sports. Consumers request the news-feed.

Espresso automatically names individual tweets. Tanimoto distance, a vector expansion of the Jaccard index, is selected as a string distance metric for this particular application. Each tweet is assigned a weight which specifies the number of times it was retweeted.

After the hierarchical name space is constructed, tweets are sampled by *Espresso*'s default policy, *high aggregates*. Prioritized lists and actual data are published through NDN. News readers are then able to request the feed.

To evaluate this application in a controlled manner, we manually collected data sets on four topics of different popularity. We labeled them Canada fire, Baltimore protest, Superbug, and UEFA champions league (UCL). Canada fire includes tweets about a large and intense wildfire in 2016 that burned more than 595,000 acres in the northern Alberta town of Fort McMurray. Baltimore protest describes campaigns against violence triggered during the trial of police officers investigated in the death of Freddie Gray. The protest began with the hash tag, #BlackLivesMatter, on a social network service. Superbug includes tweets on the spread of multidrug-resistant bacteria, causing two patients' death in a California hospital. Lastly, the soccer data set describes UEFA champions league, one of the most popular soccer tournaments in the world. This particular data set was collected right after the final match was finished in 2016. The data set has about 25,000 tweets in total. Almost half of them are on the Canada fire (12,000). The second biggest data set was the Baltimore protest (8,000), and two other sets split the remaining 5000 tweets.

A summary of the top five tweets (according to the retrieval order) was then retrieved by the consumer, using three different retrieval policies: *sort*, *diversity*, and our recommended order. The *sort* and *diversity* orders preferably select high-scoring and less-redundant tweets, respectively. Our recommend setting, labeled *Espresso*, finds a balance between the two.

Table II shows the topics retrieved by *sort*. Note that three out of five tweets are on the topic of Canada fire. The remaining two are on the Baltimore protest. The topics are skewed towards the most popular news, missing other topics altogether like Superbug and UCL.

On the other hand, as seen in Table III, *diversity* does not return high quality tweets. This is because it does not consider the weight parameter.

Table IV shows headline news created by our recommended retrieval order, labeled *Espresso* (dafault). Compared to other two baselines, it balances importance and diversity. Selected tweets are meaningful and cover all topics.

Table V further exhibits the benefit of *Espresso*. Because the name space is constructed from semantic hierarchy, readers can select the level of summaries by pointing out a subtree in which they are interested. By going deeper down the tree, *Espresso* shall provide more detailed news. Table V shows the result of zooming in from the original Soccer tweet in Table IV.

In the real application, forward and backward buttons are mapped to zoom-in and zoom-out functions. Suppose that a consumer viewing a summary of `/news` zooms in on a tweet, named `/news/0/0/0/0`. When the user clicks the forward button (for this tweet), the summary for the subtree `/news/0` would be retrieved. In this manner, readers are able to retrieve more detailed news by clicking the forward button on specific topics (i.e., tweets). Conversely, the backward button retrieves a summary of the parent node.

B. *Product review sampler*

The last application is a consumer product review sampler that summarizes customer reviews for a requested product. Many customers are interested in reading reviews in order to learn experiences of previous customers that used or are using the product. However, there are often too many reviews to read at once. Just checking the average rating may not provide much details on user experience with respect to the product.

Our illustrative application provides a subset of reviews that can represent samples of different degrees of satisfaction. To that end, reviews are clustered by rating. In this case, Espresso is configured such that they are sampled proportionally to the cluster size. Table VI shows the top 10 sample reviews for a

TABLE II: Headline news based on *sort*

Topic	Retweets	Tweet
Canada Fire	408	Wildfire Update - May 4, 4 a.m. Residential damage assessment (approximate) #ymmfire #ymm https://t.co/X8t2Fr3NF8
BLM	214	UC Davis spent $175,000 to bury this story of police brutality. We're writing about it so they fail. https://t.co/DoEk83kjK2
Canada Fire	212	Now a #ymmfire wildfire status update https://t.co/mClUYvcMu4
Canada Fire	201	My thoughts are with people affected by the fire in Fort McMurray tonight. Stay safe and remember to follow evacuation orders. #ymmfire
BLM	184	Black Lives Matter and America's long history of resisting civil rights protesters https://t.co/cKpq1mP18L

TABLE III: Headline news based on *diversity*

Topic	Retweets	Tweet
Soccer	1	Ｊ２の"四国ダービー"愛媛ＦＣ徳島ヴォルティスの今季初戦は６月１２日、愛媛県松山市のニンジニアスタジアムで行われる（１９時キックオフ）。０６年にスタート ... https://t.co/NfiWdiT4lu
Canada Fire	34	IF YOU STILL HAVE ANIMALS IN YOUR HOME AND YOU ARE NOT HOME PLEASE contact Pulse 780 743 7000 or RMWB by-law at 780 788 4200 #ymmfires
Soccer	1	@BustAFern Sure ... Enjoy the tournament. https://t.co/iaahMwiw5L https://t.co/AZZeb8Qqb2
Superbug	2	What makes the LA superbug story significant is that it is one thing to have it hit a hospital, another to have it loose in the community.
Canada Fire	1	The #FortMcMurray wildfire, seen from the air. #ymmfire #Alberta https://t.co/ViPcfIRiLN

TABLE IV: Headline news based on *Espresso*

Topic	Retweets	Tweet
Canada Fire	408	Wildfire Update - May 4, 4 a.m. Residential damage assessment (approximate) #ymmfire #ymm https://t.co/X8t2Fr3NF8
BLM	184	Black Lives Matter and America's long history of resisting civil rights protesters https://t.co/cKpq1mP18L
Soccer	76	Congratulations to realmadriden! ChampionsLeague #europeanchampion
Superbug	143	2 deaths have been linked to this superbug. This is what you should know about CRE: http://t.co/kOmkHkavs1 http://t.co/XLxDhPzxOq
BLM	214	UC Davis spent $175,000 to bury this story of police brutality. We're writing about it so they fail. https://t.co/DoEk83kjK2

humidifier in Best Buy. The total number of reviews in the original data set was 145, of which a rating of 5, 4, 3, 2, and 1 was given by 51, 48, 15, 13, and 8 customers, respectively.

Note, from the table, that the 10 reviews comprising the summary contain 3, 4, 1, 1, and 1 reviews of the respective ratings. Indeed, the data are sampled (approximately) proportionally.

TABLE V: Zoom-in news to soccer

Retweets	Tweet
76	Congratulations to realmadriden! ChampionsLeague #europeanchampion
59	Here comes the scorer of the realmadriden winning penalty in the #uclfinal Cristiano! https://t.co/h6xamTXjPH
40	22 de 27, 72 goles... el 'efecto Zizou' convierte la temporada en histórica — https://t.co/v1OA5UHOV4 https://t.co/367fR0yzmO
17	CHAMPIONS OF EUROPE! Congratulations realmadrid! #UCLfinal https://t.co/XkjKSjpPer
21	Mexican police mount big hunt for kidnapped soccer player: https://t.co/m2cKVEyfiq https://t.co/YpTJn6LZhc

TABLE VI: Top ten reviews for a humidifier with *proportional to cluster size*

rating	comments
5	The humidifier works great, it keeps the dust down, your air is more breathable, the plants love it, I have an indoor jungle growing and it runs silent. You can control the amount of humidity and fan speed. It holds a decent amount of water, so you don't have to keep filling it up.
5	Love this humidifier. It works quiet enough with just the right amount of background noise. It has helped my children through the dry winters with their runny noses And coughs.
5	we have used it overnight and it works wonderful,it is not noisy
4	After being stuffed up and congested for 2 months this product really helped make it easy to breath. A plus it that it is quite and I was finally able to get some sleep.
4	This product works well. I purchased it for new born daughter room thats soon to arrive and have been happy with it. Ive tried it out in my own room and worked as expected. Only issue I really had and didnt really like was having to dump out the water if there was any left in it.
4	Works well for a large bedroom. Noticed a difference the first night.
4	Overall happy with purchase. Keeps room nice and humid. I like that it has a small footprint but I wish makers of humidifiers would consider the ease of filling these containers. They are always awkward and slippery. This is a replacement for one that cracked when i dropped it while filling and they no longer make the reservoir.
3	Very noisy. Filter needs replacement frequently. Get a better one than this.
2	This humidifier does its job without drenching the surrounding floor etc. It can be set at a quieter level for sleeping hours. Would recommend.
1	This is a classic example of where buying something expensive doesn't mean it's good. First of all this thing is loud, you can't use it at night because even on the lowest setting it is loud. Also, you need to replace filters every so often. If not, you start circulate moldy bacteria in the air. I am not looking for an air filter, just a air humidifier. This thing is the worst. If you want mine you can take it, it'll be in the garbage.

VII. RELATED WORK

Many ideas were proposed for information-centric networks (ICNs), as versions of a new network paradigm that switches from host- to content-oriented networking. Examples include DONA [13], PURSUIT [14], CCN [15] and [16]. The rationale for ICNs is that most applications access networks for information, as opposed to connecting to a specific host, which calls for a new addressing abstraction.

In this work, we aim to design an automatic naming engine, and implement it on top of named data networking (NDN) [16]. Naming is an active research topic in NDN. Yu

et al. describe current naming conventions in NDN [8]. Fan *et al.* introduced a way to search data names through a name server called Catalog (similar to DNS in IP) [17]. To the best of our knowledge, this is the first work to propose automatic naming service designed specifically for summarization.

There are several transport frameworks in ICNs [18], [9], [19], [20]. Recently, services that support summarization were introduced such as the Information Funnel [19] and InfoMax [20]. The main difference between the two is in that the former describes how one consumer may collect data samples from multiple producers, whereas the latter discusses how one producer may disseminate samples to multiple consumers, given appropriate naming. Espresso automates the naming.

Our name space construction is inspired by SocialTrove [4]. This unique work provides a general summary model for any given data set, represented as a binary tree, if a correct distance metric is defined. While SocialTrove did not aim at data naming, it clustered data by recursively dividing objects into two disjoint sets using 2-means clustering algorithm [21]. We use the same approach for naming.

Espresso is also differentiated in that it flexibly balances importance of content versus diversity. While both predecessors only focus diversity, *Espresso* allows applications to control the trade between diversity and importance in a customizable manner.

VIII. Conclusions

In this work, we proposed *Espresso*, a generic data naming service for ICNs. The primary benefit of this service is to create an automated namespace for an unstructured data set in a manner that facilitates representative sampling. It allows customizing retrieval policies, balancing notions of diversity and importance, to offer representative summaries to meet application needs. It further supports data zoom-in and zoom-out functionality, based on the name hierarchy. The authors are currently working on developing applications that exploit advantages of *Espresso*.

Acknowledgements

Research reported in this paper was sponsored in part by NSF under grants CNS 13-45266, CNS 16-18627 and CNS 13-20209, and in part by the Army Research Laboratory under Cooperative Agreement W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory, NSF, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

[1] M. F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. L. Bertrand Mathieu, "A survey of naming and routing in information-centric networks," *IEEE Comunications Magazine*, pp. 44–53, December 2012.

[2] J. Lee, A. Kapoor, M. T. A. Amin, Z. Wang, Z. Zhang, R. Goyal, and T. Abdelzaher, "InfoMax: An information maximizing transport layer protocol for named data networks," in *International Conference on Computer Communication and Networks (ICCCN'15)*. IEEE, August 2015, pp. 1–10.

[3] W. W. Cohen, P. Ravikumar, and S. E. Fienberg, "A comparison of string distance metrics for name-matching tasks," in *In Proceedings of IJCAI-03 Workshop on Information Integration*, August 2003, pp. 73–78.

[4] M. T. A. Amin, S. Li, M. R. Rahman, P. T. Seetharamu, S. Wang, T. Abdelzaher, I. Gupta, M. Srivatsa, R. Ganti, R. Ahmed, and H. Le, "SocialTrove: A self-summarizing storage service for social sensing," in *International Conference on Autonomic Computing (ICAC'15)*. IEEE, July 2015, pp. 41–50.

[5] G. Hamerly and J. Drake, "Accelerating lloyd's algorithm for k-means clustering," in *Partitional Clustering Algorithms*, M. E. Celebi, Ed. Springer International Publishing, 2014, pp. 41–78.

[6] D. Arthur, B. Manthey, and H. Röglin, "Smoothed analysis of the k-means method," *J. ACM*, vol. 58, no. 5, pp. 19:1–19:31, Oct. 2011. [Online]. Available: http://doi.acm.org/10.1145/2027216.2027217

[7] S. K. Bhatia, "Adaptive k-means clustering." in *FLAIRS Conference*, 2004, pp. 695–699.

[8] Y. Yu, A. Afanasyev, Z. Zhu, and L. Zhang, "Ndn naming conventions," http://named-data.net/publications/techreports/ndn-tr-22-ndn-memo-naming-conventions/, UCLA, Tech. Rep., 2014.

[9] I. Moiseenko, L. Wang, and L. Zhang, "Consumer / producer communication with application level framing in named data networking," in *Proc. 2nd International Conference on Information-Centric Networking (ACM ICN'15)*, September 2015.

[10] "Consumer-Producer-API," https://github.com/named-data/Consumer-Producer-API.

[11] C. Elkan, "Using the triangle inequality to accelerate k-means," in *ICML*, vol. 3, 2003, pp. 147–153.

[12] "Twitter search apis." [Online]. Available: https://dev.twitter.com/rest/reference/get/search/tweets

[13] 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*. ACM, August 2007.

[14] N. Fotiou, P. Nikander, D. Trossen, and G. C. Polyzos, "Developing information networking further: From psirp to pursuit," in *Broadband Communications, Networks, and Systems*, I. Tomkos, C. J. Bouras, G. Ellinas, P. Demestichas, and P. Sinha, Eds. Berlin: Springer Berlin Heidelberg, 2012.

[15] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies (CoNEXT'09)*. ACM, December 2009, pp. 1–12.

[16] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, kc claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," in *SIGCOMM Computer Communication Review (CCR)*. ACM, July 2014.

[17] C. Fan, C. Olschanowsky, S. Shannigrahi, C. Papadopoulos, S. DiBenedetto, and H. Newman, "Managing scientific data with named data networking," in *Network-aware Data Management Workshop*, November 2015.

[18] J. Chen, M. Arumaithurai, X. Fu, and K. K. Ramakrishnan, "SAID: A control protocol for scalable and adaptive information dissemination in ICN," *CoRR*, vol. abs/1510.08530, 2015. [Online]. Available: http://arxiv.org/abs/1510.08530

[19] S. Wang, T. Abdelzaher, S. Gajendran, A. Herga, S. Kulkarni, S. Li, H. Liu, C. Suresh, A. Sreenath, H. Wang, W. Dron, A. Leung, R. Govindan, and J. Hancock, "The information funnel: Exploiting named data for information-maximizing data collection," in *Proc. International Conference on Distributed Computing in Sensor Systems (DCOSS'14)*. IEEE, May 2014, pp. 92–100.

[20] J. Lee, A. Kapoor, M. T. A. Amin, Z. Wang, Z. Zhang, R. Goyal, and T. Abdelzaher, "InfoMax: A transport layer paradigm for the age of data overload," in *Advances in Computer Communications and Networks - from green, mobile, pervasive networking to big data computing*, J. Fagerberg, D. C. Mowery, and R. R. Nelson, Eds. River Publisher, 2016.

[21] C. C. Aggarwal and C. K. Reddy, Eds., *Data Clustering: Algorithms and Applications 1st ed*. Oxford: Chapman & Hall/CRC, 2013.