The Information Funnel: Exploiting Named Data for Information-maximizing Data Collection

Shiguang Wang *, Tarek Abdelzaher *, Santhosh Gajendran *, Ajith Herga *, Sachin Kulkarni *, Shen Li *,
Hengchang Liu ¶, Chethan Suresh *, Abhishek Sreenath *, Hongwei Wang *, William Dron †, Alice Leung †,
Ramesh Govindan ‡, John Hancock §
*University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
†BBN Raytheon Technologies, Cambridge, MA 02138, USA
‡University of Southern California, Los Angeles, CS 90089, USA
§ArtisTech, Inc. Fairfax, VA 22030, USA
¶University of Science and Technology of China, Suzhou, Jiangsu 215123, China

***Abstract*—This paper describes the exploitation of hierarchical data names to achieve information-utility maximizing data collection in social sensing applications. We describe a novel transport abstraction, called the *information funnel*. It encapsulates a data collection protocol for social sensing that maximizes a measure of delivered information utility, that is the minimized data redundancy, by diversifying the data objects to be collected. The abstraction leverages named-data networking, a communication paradigm where data objects are named instead of hosts. We argue that this paradigm is especially suited for utility-maximizing transport in resource constrained environments, because hierarchical data names give rise to a notion of distance between named objects that is a function of only the topology of the name tree. This distance, in turn, can expose similarities between named objects that can be leveraged for minimizing redundancy among objects transmitted over bottlenecks, thereby maximizing their aggregate utility. With a proper hierarchical name space design, our protocol prioritizes transmission of data objects over bottlenecks to maximize information utility, with very weak assumptions on the utility function. This prioritization is achieved merely by comparing data name prefixes, without knowing application-level name semantics, which makes it generalizable across a wide range of applications. Evaluation results show that the information funnel improves the utility of the collected data objects compared to other lossy protocols.**

I. Introduction

The expanding proliferation of sensors available in social spaces (such as smartphone sensors, cameras, and GPS devices) and the exponential growth in digital data generated in recent years, far outstrip the human capacity to consume the resulting information. This trend suggests that an important category of future networked applications and services will focus around information sampling to bridge the widening gap between data generation rate and human consumption capacity.

Current transport abstractions, such as reliable transmission in TCP, offer pipes where each bit of input must be delivered at the output. In the future, driven by information overload, a new higher-level abstraction will become increasingly important: namely, one that offers at the output a *representative sampling* of information at the input, thereby reducing the large body of input to a readily consumable size. For wider applicability, this sub-sampling must be done in an application-independent manner. Nevertheless, it must do better than random selection. The *information funnel* implements such an abstraction.

The information funnel is targeted for scenarios, where resource constraints (e.g. limited transmission bandwidth or constrained power) or efficiency considerations prevent transmission of *all* collected data. In these scenarios, with limited number of data objects can be transmitted, the information funnel tries to sample a representative subset of the data objects that maximizes the information utility by minimizing data redundancy. Much prior work on data collection in sensor networks addressed the challenge of optimal data selection (based on different application-level metrics) (e.g.[15]), that judiciously chooses the best data objects to transmit when transmission of all data is impossible or undesirable. Unlike ours, such protocols are application specific, as they use application-specific information and optimize application-specific performance metrics. Therefore, their work is not general; it will have a poor performance or even not work in a different application.

We also distinguish ourselves from work on sampling theory that determines how to sub-sample time series data in ways that generically minimize a measure of loss(e.g. [7] [14]). In contrast to these approaches, and in seeking a general service, we do not impose any specific requirements on the underlying data type. For example, the collected data may constitute images, text, or sound clips, as opposed to numeric data types.

The paper complements the aforementioned literature by exploring the potential and limitations of *application-independent* maximization of delivered information utility (that is to minimize delivered redundant data). By application-independent, we mean that the solution does not use any application-specific knowledge or semantics. It only uses the hierarchical data names, treated as bit strings with no semantic interpretation. This is a major difference from sampling theory that requires understanding the application-specific semantics of data objects.

We exploit the *named-data networking* (NDN) paradigm [9] as an enabler for information utility maximization. NDN is originated because of the fact that people care more about *what* data they received but not *where* they get the data.

978-1-4799-4618-1/14 $31.00 © 2014 IEEE
DOI 10.1109/DCOSS.2014.32

Therefore, NDN names data objects, not hosts, which distinguishes it from the mainstream communication paradigm based on TCP/IP. In NDN, the information consumer (e.g., the data collection point) sends interests in information objects described by a given name prefix. Objects that match the specified prefix are returned in response to the respective interests. The information funnel is implemented on top of NDN as a thin layer that decides on the order of data transmission.

The paper investigates (and confirms) the hypothesis that *by giving data objects hierarchical names, where the length of the common prefix between two names is a rough measure of similarity between the corresponding objects, information-maximizing ordering can be achieved using policies that diversify the transmitted names.* Here the similarity between two data objects refers to the possibility that the value of one can be approximated by the other's. Our service is geared for social sensing applications in which a receiver (such as a remote back-end server) acts as a collection point for a group of (typically mobile) nodes that report data from the physical environment. Often the nodes are disconnected and come only into sporadic contact with the collection point. The data collected usually carries much overlap. For example if data names encode location and time of data collection, the more similar the names are, the more likely the overlap between the named measurements and the less is their aggregate utility. A data collection protocol that *diversifies the collected names* will tend to maximize information utility as well. It remains to show how exactly names should be diversified in the presence of resource constraints, which is the topic of this paper.

The rest of the paper is organized as follows. Section II reviews related work. Section III presents our notion of optimality and suggests heuristics with near-optimal behavior. Section IV presents evaluation results. The paper concludes with Section V.

II. Related Work

Cyber-physical and sensing applications, where data objects are collected from the physical world, typically exhibit significant redundancy in collected data. This calls for prioritizing data collection in a way that reduces redundancy received. The problem was recently described by the authors in PhotoNet [20], where a picture-collection service was developed for disaster-response applications that maximize situation-awareness. Dron *et. al.* [5] proposed a novel caching design aiming to maximize the information utility. Another illustration of such redundancy was covered in CarSpeak [11], where autonomous vehicles share cloud-point data for obstacle avoidance. In these papers, application-specific data prioritization policies were described that aim at improving an information utility metric.

Our paper points out one aspect of application utility maximization (for data originating in the physical environment) that can be broadly generalized independently of other application details. Namely, within the scope of a query, received information is maximized when redundancy is minimized, since redundancy reduces information content. Hence, it makes sense to dedicate such information-maximization to a layer that is independent of and supports the specific application.

NDN enables the development of such a layer. One big benefit of NDN is that data has name, which makes the designing and implementing some kinds of applications easier, like the dataset synchronization [31]. Specifically, if data names can approximately denote similarity between objects, information maximizing data ordering (e.g., in data transmission across bottlenecks) can be done in a completely generic fashion by applying a breadth-first traversal algorithm to the subtree of an application's name space that falls within the scope of the query. This paper focuses on persistent queries, where the relevant name-space subtree is expressed as a name prefix associated with the information funnel.

Our work is related to general efforts that attempt to handle network resource constraints in an efficient manner. Prior literature explores how to efficiently transfer (spatially, temporally or spatio-temporally) correlated data samples over multi-hop networks to a sink. Usually, one of two approaches is adopted: either compress samples to reduce redundancy or select a small subset of nodes to aggregate samples before sending them to the sink. A few notable examples of the first approach include Cristescu *et al.* [4], Pattem *et al.* [18] and Vuran *et al.* [21]. The second approach includes schemes such as selecting an energy-efficient correlation-dominating set [7], clustering based on correlations [14], clustering based aggregation [16], routing through a set of mobile sinks [25], sensor data dissemination for mobile users [17] and distributed data collection by localized coding [29]. In [15], authors propose distance entropy as a metric to formalize communication cost for collecting correlated data. While the above work focused on time-series data, some more recent work [20], [10], [24] focuses on more complex data types (such as pictures). In contrast to the above literature, this paper focuses on generic prioritization policies based on named-data networking.

Several approaches were considered for efficient data collection in intermittently connected networks and DTNs. These include interest profiles [6], cooperative sensing [30], vehicular data collection [22], publish/subscribe methods [27], [13], and subscription of channels [12], [3]. In [26], the authors studied caching, where nodes cache data based on popularity, such that future queries can be answered with less delay. Some research efforts [19], [8] improve data accessibility from infrastructure networks such as WiFi Access Points [8] or the Internet [19]. Our work is complementary in that we focus on maximizing information utility by minimizing redundancy in collected data. Specifically, we do so by designing an appropriate name space in the context of named-data networking.

III. The Information Funnel

Social sensing applications share in common the fact that they (i) collect data objects from one or more (typically mobile) sources, (ii) do not need all sender data to operate correctly, and (iii) perceive a quality of information wherein receiving more data on the same "topic" has diminishing

return. For example, to estimate the current speed of traffic on city streets, it is sufficient to obtain representative speed measurements from a subset of vehicles. More data will have diminishing return. Similarly, in a disaster-response application where first-responders pictorially document damage and report it to a rescue site, only a few pictures of each problem spot are needed to understand the situation. More pictures have diminishing return. This motivates the *information funnel* abstraction, described below. We discuss challenges in implementing it and define a notion of optimality. Finally, we present its design and implementation on top of a named-data-networking stack.

A. The Basic Abstraction

The information funnel targets applications that implement persistent data collection tasks. We require that data objects have a hierarchical name space. A funnel is associated with a name prefix in that space (analogous to a path prefix in a UNIX directory tree), defining the subtree in which data of interest to the application resides. Content that belongs to the subtree starting with that prefix is the target of collection. Senders publish content under appropriate names. If those names fall within the target subtree, the corresponding objects become targets for collection.

For example, in an urban traffic speed monitoring application, the name space might look something like this: `/ndn/app/city/street/block/speed`. The collection point creates a funnel (defined by the name prefix of the above tree, say `/ndn/app`) to populate this space with data from senders. The design of the name space is up to the application developer. In applications where mobile entities share a physical environment in which they measure some quantity, such as cars measuring traffic speed, the name space might associate names with parts of the physical environment (e.g., street blocks). In this case, mobile sensors will assign data names depending on what part of the environment they are sensing. In applications where sensors are fixed, such as security cameras in rooms, data names may be associated with sensor IDs.

An important goal of our design is to accommodate mobility and disconnected operation. Hence, we assume that the normal state of senders is "offline". For example, mobile sensors may not have connectivity until they meet an access point. Smart phone users may disallow an application from using their 3G/4G data plan quota. First responders in a post-disaster scenario may communicate only using short-range radios, and thus be disconnected unless in close proximity, because other communication infrastructure is out of power or destroyed.

In general, at a given time, the receiver has a partially populated content tree. When a sender has a transmission opportunity (e.g., encounters an access point), the receiver needs to be updated on any new data the sender has, under that tree, that the receiver has not yet received. Two interesting questions arise: how to inform the sender efficiently of data gaps at the receiver, and in what order should such missing data objects be sent? Below, we first describe the underlying challenges, then define a notion of optimality and present our algorithm.

B. Data Ordering Challenges

To implement an update between a sender and the receiver of the funnel, the trivial solution is to have each sender collect data under `/ndn/app` and forward it when possible to the receiver. Once data is delivered, it can be discarded at the sender locally or marked as delivered. This solution works well when senders populate *non-overlapping* parts of the content tree, and when they do not exchange their collected data among themselves for uploading to the server. In this case, each sender can easily tell which of its data does not yet reside on the receiver. In social sensing applications, many senders may report data on the same event. Hence, sending all of one's newly collected data to the receiver may be suboptimal because the receiver may have already received that (or similar) content from another sender. The receiver needs to tell the sender (either exactly or approximately) what information it already has.

Summarizing the receiver's information state to the sender is easy when data has a linear order (e.g., "I have all data up to time-stamp X"). This, unfortunately, is not true in our case. Two factors compound our problem. First, the receiver may have only partial data that populates the name space sparsely. Hence, many gaps exist in data coverage, making their exact enumeration hard. Second, the receiver may not know the totality of data generated under a given name subtree. For example, in our vehicular sensing application, a receiver can never tell that it has "all data from Main Street" because it does not know how many vehicles drove on Main Street that may not have uploaded their data yet. Hence, there is no easy way to prune subtrees from consideration on account of completeness.

Another question is regarding the order in which a sender should send the data that its receiver is missing. If the sender sends such data in the order it was collected, the receiver may receive a lot of data from one branch of the name space and no data from other branches, resulting in a very unbalanced coverage of content. Instead, it is better to diversify by sending a sampling of data from each branch. The need for diversification calls for a definition of optimal information utility to guide the data ordering algorithm, as formulated below.

C. Optimal Transmission Order

Consider the problem where a sender must order a set of data objects for transmission to a receiver (that fall within the content space of the funnel). An optimal transmission order is sought, where the utility of the transmitted data to the receiver is maximized. In this section, we formulate this problem more carefully and describe our solution. We initially assume that all objects have the same size and the same importance (weight). In subsequent sections, these assumptions will be removed.

A primary design objective is to keep the formulation *as simple as possible*, since quantities such as data utility are

notoriously hard to compute exactly. Clearly, if utilities are set subjectively or arbitrarily, then optimizing them does not make much sense. To render the problem of finding an optimal transmission order meaningful, we must seek an approach *that makes minimal assumptions about utility curves.* For example, we explicitly stay away from schemes that require computing absolute utility values for data objects, since those are subjective.

Instead, we assume that the following two properties hold regarding the marginal utility of data objects at different parts of the content tree:

*a) **The hierarchical similarity property**:* The hierarchical name space is designed such that items that share a longer name prefix (measured in the number of tree levels) are more similar. By similarity, we mean that one can approximately be substituted by the other. For example, speeds at `/app/urbana/main/1200` and `/app/urbana/main/1000` are more similar than speeds at `/app/urbana/main/1200` and `/app/urbana/green/1100` since the former pair shares a longer common prefix. A corollary is that the marginal utility of a data object increases with the decreasing length of the longest common prefix between itself and any of the previously collected items. This is because the smaller that prefix, the less substitutable the item is by any of the ones already collected, thus the higher marginal utility it has.

*b) **The diminishing return property**:* The marginal utility of adding a data object to a name subspace is diminishing; adding the first item to `/ndn/app/urbana/main` has a larger marginal utility than adding the second item, which in turn has a larger marginal utility than adding the third one, and so on.

The *hierarchical similarity assumption* implies that, at each step, the optimal transmission order must pick the data object whose maximum common prefix (with all previously collected ones) is shortest. If there is a tie between two (or more) such items, the *diminishing return assumption* implies that we pick the one that has the least populated prefix. In other words, we count how many data objects were collected under each prefix, then pick the one whose prefix has the smaller count. We call it the *occupancy count* of the prefix. If there is a tie again, we break the tie by picking the item on the left-most branch (which is the one with the most recent timestamp assuming branches are chronologically sorted).

More formally, let $\ell_i(\mathcal{J})$ denote the longest common name prefix of a data object i with respect to a data set $\mathcal{J}$, $|\ell_i(\mathcal{J})|$ denote its length, and $\overline{\ell_i(\mathcal{J})}$ denote its occupancy count. We denote the marginal utility of a data object i with respect to a data set $\mathcal{J}$ as $\mathbb{U}(i|\mathcal{J})$. Given any two data objects, i and j, and a data set $\mathcal{J}$, we can compare the marginal utilities of the two data objects as follows.

Marginal utility comparison rules:

- If $|\ell_i(\mathcal{J})| < |\ell_j(\mathcal{J})|$, then $\mathbb{U}(i|\mathcal{J}) > \mathbb{U}(j|\mathcal{J})$,
- Otherwise, if $|\ell_i(\mathcal{J})| = |\ell_j(\mathcal{J})|$ and $\overline{\ell_i(\mathcal{J})} < \overline{\ell_j(\mathcal{J})}$, then $\mathbb{U}(i|\mathcal{J}) > \mathbb{U}(j|\mathcal{J})$,
- Else (*i.e.*, if $|\ell_i(\mathcal{J})| = |\ell_j(\mathcal{J})|$ and $\overline{\ell_i(\mathcal{J})} = \overline{\ell_j(\mathcal{J})}$), $\mathbb{U}(i|\mathcal{J}) = \mathbb{U}(j|\mathcal{J})$.

An optimal transmission order is one *that maximizes the marginal utility, for every count k of transmitted objects, over all ways of picking k objects for transmission.* A greedy solution is to transmit the object with the minimum $|\ell_i(\mathcal{J})|$. In case of a tie, transmit the object with the minimum $\overline{\ell_i(\mathcal{J})}$. In case of a a second tie, break the tie arbitrarily (e.g., transmit the left-most child). In the following section, we illustrate our idea through an example and discuss optimality.

D. An Example

An example of the proposed algorithm is shown in Figure 1. The square boxes indicate data objects at the leaves of the content tree. The circles are intermediate nodes (directories) in the name space.

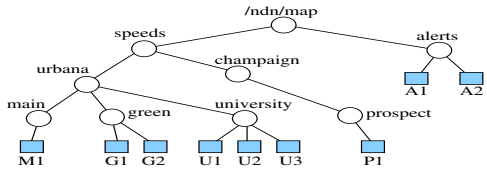


Fig. 1. An Example

Let the receiver have no data objects initially. There is a tie between all the items in that none share a common prefix with what the receiver has, and the name space has zero occupancy count. Picking the left-most branch, we send item `M1` first. Next, the items that minimize the longest the common prefix with `M1` are `A1` and `A2`. Their longest common prefix with previous items (i.e., `M1`) is the same; namely, `/ndn/map`, which has zero occupancy count. The tie is broken by following the leftmost branch (i.e., send `A1`). The next items that minimize the longest common prefix with those transmitted earlier are `P1` and `A2`. Their respective longest common prefixes with earlier items are `/ndn/map/speeds` and `/ndn/map/alerts`. Tying on occupancy count (namely, one item was transmitted under each prefix), the leftmost item (i.e., `P1`) is transmitted next. Following that, `A2` minimizes the longest common prefix and is transmitted next. The next items that minimize the longest common prefix with those transmitted earlier are `G1`, `G2`, `U1`, `U2`, and `U3`. They tie on occupancy of their prefixes and so the leftmost one is transmitted (i.e., `G1`). Next, items `U1`, `U2`, and `U3` tie and `U1` is transmitted. It can be seen that, following the above logic, we then transmit `G2`, `U2`, and `U3` in that order.

E. Receiver Feedback

In the previous section, we have not addressed the case where the receiver already has a partially populated name space. To accommodate this scenario, when a sender comes in contact with the receiver, the receiver first sends the sender a packet that contains occupancy counts of all prefixes up to a configurable tree level n. The sender will initiate the occupancy number of the name tree based on the receiver's feedback. The initialization will cause transmission to favor data that resides in prefixes that the receiver has less (or no) data from. Consider again the example shown in Figure 1. Assume that the receiver already has items `G0` and `P0` that

reside at the same prefixes as G1 and P1, respectively. The transmission order will be A1, then A2 (minimizing longest common prefix with previously collected items and favoring the prefix with lowest occupancy count), followed by M1 and P1 (since they are the next to minimize the longest common prefix with previously collected items), then U1, G1, U2, G2 and U3 (tie on longest common prefix, so round robin on occupancy count).

F. The Algorithm

In this section, we present the prioritization algorithm and prove its optimality. Its time complexity analysis is in the appendix. At the first look of the above tree traversal, it seems that our algorithm is just a simple breadth-first traversal, with round robin. However, after a careful examination, the breadth-first traversal does not always give the optimal prioritization by the marginal utility comparison rules; it only guarantees optimality when all the data nodes (leaves) reside at the same tree level. We implement the algorithm in a recursive fashion, as shown in Algorithm 1.

Algorithm 1 The prioritization algorithm

Input: The application root name prefix R, the named data set $\mathcal{I}$ of the sender, the occupancy tree T of the receiver
Output: A prioritized order of object names

```
1: Return PRI(R, I, T)
2:
3: procedure PRI(name prefix P, data set I, occupancy tree T)
4:     if P is leaf node then
5:         Return the corresponding data object I.get(P)
6:     end if
7:     order = []                          ▷ Initiate an empty list of lists
8:     for Each branch b of P do
9:         order.append(PRI(P/b, I, T)
10:    end for
11:    result = []                                ▷ Initiate an empty list
12:    while order is not empty do:
13:        S = MINCHILD(order, T)
14:        e = LEASTOCCUPANCY(S, T)
15:        result.append(e)
16:        UPDATEOCCUPANCY(e, T)
17:        if result[e.setIndex] is empty then
18:            result.pop(e.setIndex)
19:        end if
20:    end while
21:    Return result
22: end procedure
```

In this algorithm, the input parameters include the application root name prefix R under which all the application data resides in the name space, the data set at the sender side $\mathcal{I}$, and the occupancy tree T summarizing the name space occupancy at the receiver side. The algorithm calculates the prioritization order for each node at each level in the name tree in a bottom-up fashion. To compute the prioritization order of an inner node in the name tree, it "merges" the prioritization results of all its children nodes in a prioritized order. The merge process has three steps: (1) finding the data objects having the least common prefix with respect to the data set on the receiver side and assign highest priority to them (in the procedure MINCHILD), (2) balancing the occupancy tree at the receiver side by finding the data object residing at the name tree branch with least occupancy number (in the procedure LEASTOCCUPANCY) , and (3) update the occupancy number of the occupancy tree T (by the procedure UPDATEOCCUPANCY). The return of Algorithm 1 is the prioritized order of the data objects at the sender side.

Theorem 1: By marginal utility comparison rules in Section III-C, Algorithm 1 returns the optimal prioritization order of the data objects at the sender side when data objects have the same size and weight.

The proof of Theorem 1 is in the appendix. Please note that the optimality of Algorithm 1 holds *without assumption on the number of data objects transmitted in one transmission session.* In other words, for *any* k data objects transmitted in one transmission session Algorithm 1 is optimal, where k is no greater than the total number of data objects at the sender side.

G. Variable Object Length and Differentiated Service

In the above discussion, we assumed that all objects have the same length. In general, objects in some parts of the tree might be longer than others. For example, one branch might contain images with high quality (*i.e.* high resolution), whereas another contains images with low quality. To balance data collection from different branches, rather than maintaining a collected occupancy count for different prefixes, we maintain the number of collected bytes. Hence, when an object is selected, the occupancy count of its ancestor nodes in the name space is incremented by its length, as opposed to by one in UPDATEOCCUPANCY (see Algorithm 1). The approach will balance the bytes collected instead of objects. Besides the difference in balancing occupancy compared with the uniform data size, in this variable data object size case, we also want to transmit the data objects with the highest marginal utility "density", which means that in the MINCHILD procedure, if two data objects under the same name prefix have the same marginal utility (*i.e.* the same length of the common name prefixes), the one with smaller size will be selected to transmit first. By modifying Algorithm 1, we can guarantee that the output prioritization order of the procedure PRI is in decreasing order of the data marginal utility density. However, due to the occupancy balancing of Algorithm 1, the property of decreasing marginal utility density is not guaranteed of the final returned prioritization order as shown in Theorem 3 in the appendix.

Finally, we can offer some prefixes preference over others by specifying a transmission weight, w_p for each prefix p. Accordingly, the function UPDATEOCCUPANCY in Algorithm 1 updates the occupancy count by the total bytes transmitted *divided by the weight of the prefix*. Hence, prefixes with higher weights will grow their (weighted) occupancy count at a slower rate resulting in an amount of received content that is proportional to the weight of the prefix.

H. System Design and Implementation

The system contains three layers as shown in Fig. 2; (1) the application layer, (2) the information funnel layer, and (3) the NDN layer. The application layer contains all the application specific tasks, such as sensing and naming the data objects. The information funnel layer is designed for the generalized application-independent information-maximizing transmission. This novel design of separating application specific tasks from application independent tasks greatly simplifies the application development for both the sensing application on mobile devices (the clients) and the data collection application running on the backend server. In the rest of this section, we first introduce the APIs provided by the information funnel layer to the applications on both the mobile devices and the backend server respectively. Then we present how the information funnel layer interacts with the NDN layer.

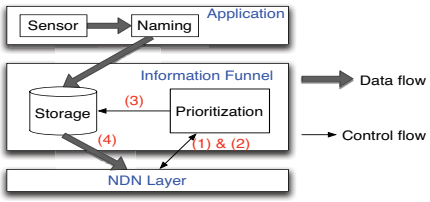


Fig. 2. Information Funnel Structure

For mobile sensing applications, the funnel layer provides three APIs. The first one is CREATEFUNNELSOURCE() to start the *client funnel thread* for the information-maximizing data transmission. This API takes two parameters, the name prefix of the funnel and the device ID. The client funnel thread first allocates the *funnel repo* to cache the data objects under the name prefix of the funnel, and it actively probes the WiFi connection status. Once the connection is built, the thread initiates the data transmission in the prioritized order, which will be discussed later. Another API is PUTTOFUNNEL() to put data objects to the funnel repo. Its parameters are the funnel prefix, the data name, and the data object pointer. This function checks the name of the data object and only put the data with the funnel name prefix to the repo. The third API is RELEASEFUNNELSOURCE() for removing the funnel and recycling the resources to the OS. It takes only one parameter which is the name prefix of the funnel.

For the backend server application, the funnel layer also provides three APIs. All the three APIs only take one parameter, the name prefix of the funnel. The first API is CREATEFUNNELSINK() to allocate a local repo and start a thread called *server funnel thread* with a name prefix, the second one is EXTRACTFROMFUNNEL() to extract data objects from repo to the above server application, and the third one is RELEASEFUNNELSINK() to remove the funnel and recycle the OS resource.

After introducing the funnel APIs to the application layer, we present how the funnel interacts with the NDN layer for information-maximizing data transmission. The client funnel thread running on the mobile device actively probes the WiFi connection status. Once the WiFi connection is setup, it broadcasts an interest packet with the funnel name prefix. For example, `/ndn/uiuc/maps/[ID]/[timestamp]/summary`, where `ID` is the device ID and `timestamp` is the current local time. If the server funnel thread is created under the same name prefix, say `/ndn/uiuc/maps`, then the server responds with a data packet that contains the local occupancy tree of the name space (summarized to some level) as defined in Algorithm 1. Meanwhile, the sever funnel thread sends an interest packet with name `/ndn/uiuc/maps/[ID]/[new_timestamp]/list` to ask for the prioritized name list, where `ID` is the mobile device ID and `new_timestamp` is the server local time. After the client receives the occupancy summary of the server, it runs Algorithm 1 to prioritize the cached data objects in the repo and generate a name list. Upon receiving the interest packet `/ndn/uiuc/maps/[ID]/[new_timestamp]/list` from the server, the client funnel thread responds with a data packet of the name list. Then, the server fetches the data objects one-by-one according to the list.

Note that we add `timestamp` to the interests that are either sent by client asking for the server occupancy tree or sent by the server requesting the name list. The `timestamp` guarantees that those interests finally reach the end node rather than some intermediate cache. Although the in-net caching design of NDN accelerates the data transmission (for example, data dissemination from a content provider to content consumers), in our application we need those requests reach the end nodes because the state of either the server or the mobile device probably has already changed since last communication, thus the cached data probably be meaningless. However, the interests requesting data objects do not contain `timestamp`. With the assumption that the application will name different data objects differently, the funnel can use the NDN in-net caching to accelerate the data transmission.

IV. EVALUATION

In this section, we study the performance of our algorithm to maximize the marginal information utility. We first introduce our methodology for the evaluation, and then evaluate the performance of the information funnel.

A. Methodology

We evaluate two aspects of the system: (1) the overhead of the prioritization, and (2) its performance by comparing with other state-of-the-art solutions. To measure the overhead in practical scenarios, we implement the information funnel on Google Galaxy Nexus phones [2]. Each phone is equipped with a 1.2 GHz dual-core CPU, 1GB RAM, and running Android OS 4.1. The information funnel is implemented using the Java programming language on top of the PARC's CCNx prototype software [1]. The data set used in the evaluation is the T-Drive data set [28] collected by MSRA. We use the taxi traces in the urban area of Beijing, China, with GPS coordinates from latitude 39.5^oN to 40.5^oN and from longitude 116^oE to 117^oE, where most data points reside.

In the evaluation, we assume the social sensing application provides a hypothetical service called "city view everyday", which is an improved version of the Google street view, where the user can see up-to-date street changes day-by-day as recorded by cameras in cars on street. This social sensing application needs to collect data objects (*i.e.* pictures) continuously from participants (*i.e.* cars).

To study the prioritization performance of the information funnel, we run a simulation on the T-Drive data set with assumptions that:

1) There are two WiFi sinks (gateways to one central server) to collect data that are located on two busy streets as shown in Fig 3(c),
2) The coverage range of each WiFi gateway is 100 meters,
3) The pictures are 100KB each,
4) The WiFi bandwidth is from 700Kbps to 1Mbps, which is estimated using the campus WiFi network, and
5) The speed of each cab is from 40km/h to 80km/h, which is estimated from the street speed limits of Beijing.

We simulate for 10 hours during which 50,000 data objects are collected by cabs (of which only 15% are uploaded to the server) and we assume that at the very beginning of the simulation the server does not have any data.

The area in the simulation is partitioned into 400 tiles, and each tile is further partitioned into 16 cells. The name of each data object (picture) is following the structure defined as `/citysense/`**`tile_idx`**`/`**`cell_idx`**`/filename`. So there are two possible levels of summary for the occupancy tree at the receiver (the central server) side.

We compare the performance of the Information Funnel with three baseline algorithms: (1) **FIFO**, which transmits the data objects in the fifo order of their time stamps, (2) **Distance-based** prioritization algorithm in PhotoNet [20] which always transmits the data object with the longest minimum distance from the data objects at the receiver side first, and (3) **Coverage-based** prioritization in Minerva [23], which always transmits the data object with the largest marginal coverage, where the side length of the coverage area of each data object defined to be 100 meters and we consider the information space is 2D. (Please refer to Minerva [23] for the detailed explanation of the configuration.) In the following section, we present the evaluation results of the information funnel, and we henceforth call the algorithm used in the information funnel as the "name-based" algorithm.

B. Evaluation Results

The computational overhead of data ordering results are shown in Table I. The prioritization computation of the Information Funnel is on a Google Galaxy Nexus phone, because it is a client-side algorithm, while the computations of the distance-based and coverage-based algorithms are on a desktop with a 3.2GHz Intel i5 quad-core CPU, because they are designed to run on the data collection server. The average, maximum, and 80th percentile overheads are shown in the table, where the 80th percentile means that in 80% of the transmission sessions the computation time is no more than this value. In the table, we also compare different levels of receiver feedback, denoted by X (as in Named-based(X)), where $X = 0$ means no feedback, and $X = 1$ (*resp.* $X = 2$) means the receiver summarizes the occupancy of the top one level (*resp.* two levels) of its name tree of all the local data objects under the funnel's prefix. Note how the computational overhead introduced by data ordering in the named-based algorithm is much less than that introduced by distance-based and coverage-based algorithms. This is because the previous algorithms were quadratic in the number of items to prioritize, where ours is in the order of $O(n \log n)$ (Theorem 2 in appendix). Considering that the WiFi connection time is around 2 seconds, the computational overhead introduced by the Information Funnel is negligible.

TABLE I
OVERHEAD STUDY RESULTS

Algorithm	avg(ms)	max(ms)	80%(ms)
Name-based(0)	0.000	0.001	0.000
Name-based(1)	0.310	7.491	0.180
Name-based(2)	0.315	7.658	0.189
Distance-based	14.212	609.992	6.441
Coverage-based	14.418	369.931	7.286

Fig. 3 shows coverage performance of the algorithms, where we consider a cell of the map covered if at least one picture was uploaded from there. Please note that more uniform distribution of points in the figure implies a larger coverage. From Fig. 3, we clearly observe that FIFO is the worst algorithm, since the data collected by it covers the smallest area, and the distance-based and coverage-based algorithms performs better than FIFO, whereas our name-based algorithm is the best (*covers the largest area*).

TABLE II
COVERAGE STUDY RESULTS

Algorithm	tile cover.	cell cover.
Name-based(2)	100%	95.54%
Name-based(1)	100%	94.23%
FIFO	85.96%	68.90%
Distance-based	94.74%	78.74%
Coverage-based	98.25%	89.76%

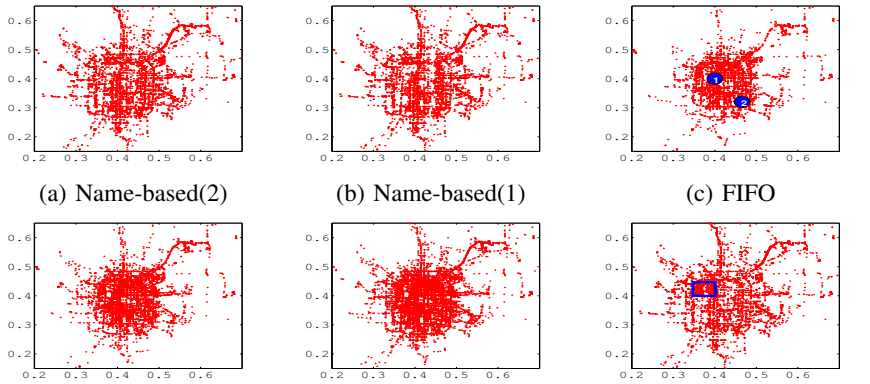
The actual percentage of cells (and tiles) covered by the compared algorithms is shown in Table II. Note that, our algorithm maximizes both metrics.

Fig. 3(f) illustrates the impact of our differentiated service extension, where we assign a higher weight to the area $[0.35, 0.4] \times [0.4, 0.45]$, identified by the blue rectangle. Compared to Fig. 3(a) where the data has the same weight, we clearly observe that more data (more red) is collected within this rectangle than in Fig. 3(a).

TABLE III
COVERAGE STUDY WITH VARIABLE DATA SIZE

Algorithm	tile cover.	cell cover.
Name-based(2)	100%	95.41%
Name-based(1)	100%	93.70%
FIFO	85.09%	68.50%
Distance-based	93.86%	78.22%
Coverage-based	97.37%	88.45%

Next, we study the coverage performance of the algorithms with variable data sizes. In this experiment, we randomly



(a) Name-based(2) (b) Name-based(1) (c) FIFO

(d) Distance-based (e) Coverage-based (f) Name-based (weighted)

Fig. 3. Performance of Information Funnel.

assign data objects sizes ranging from 100KB to 200KB. Please note that in our simulation, we first generate the random data size and then run the simulation, which guarantees that all the algorithms run on the same data set. The coverage performance is shown in Table III. Note that, our algorithm maximizes both metrics.

The evaluation shows that named-data networking can be leveraged for efficient automatic coverage (and hence, information utility) maximization simply by giving data hierarchical names, where length of the common prefix grows with data similarity.

V. Conclusions

In this paper, we introduced the information funnel, a data collection scheme that leverages the ability to name data (as in named-data networking) to offer information-maximizing content delivery for resource constrained social sensing applications. Our evaluation shows that our scheme increases the information coverage compared with the state-of-the-art solutions, while offering a very low overhead.

Acknowledgement

This work was sponsored by the Army Research Laboratory (ARL), DTRA grant HDTRA1-10-1-0120, and NSF grants CNS 1040380 and CNS 09-05014, and was accomplished under Cooperative Agreement Number 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 ARL 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] CCNx prototype software. `http://www.ccnx.org`.

[2] Google galaxy nexus. `http://www.google.com/nexus`.

[3] C. Boldrini, M. Conti, and A. Passarella. ContentPlace: social-aware data dissemination in opportunistic networks. In *Proc. of MSWiM*, 2008.

[4] R. Cristescu, B. Beferull-Lozano, M. Vetterli, and R. Wattenhofer. Network correlated data gathering with explicit communication: Np-completeness and algorithms. *IEEE/ACM Trans. Netw.*, 14:41–54, 2006.

[5] W. Dron, A. Leung, M. Uddin, S. Wang, T. Abdelzaher, R. Govindan, and J. Hancock. Information-maximizing caching in ad hoc networks with named data networking. In *Network Science Workshop (NSW), 2013 IEEE 2nd*, pages 90–93. IEEE, 2013.

[6] W. Gao and G. Cao. User-centric data dissemination in disruption tolerant networks. In *Proc. of IEEE Infocom*, 2011.

[7] H. Gupta, V. Navda, S. Das, and V.Chowdhary. Efficient gathering of correlated data in sensor networks. In *Proc. of MobiHoc*, 2005.

[8] Y. Huang, Y. Gao, K. Nahrstedt, and W. He. Optimizing file retrieval in delay-tolerant content distribution community. In *Proc. of ICDCS*, pages 308–316, 2009.

[9] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *Proc. of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT '09, New York, NY, USA, 2009. ACM.

[10] Y. Jiang, X. Xu, P. Terlecky, T. Abdelzaher, A. Bar-Noy, and R. Govindan. Mediascope: selective on-demand media retrieval from mobile devices. In *Proceedings of the 12th international conference on Information processing in sensor networks*, pages 289–300. ACM, 2013.

[11] S. Kumar, L. Shi, N. Ahmed, S. Gil, D. Katabi, and D. Rus. Carspeak: a content-centric network for autonomous driving. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '12, pages 259–270, New York, NY, USA, 2012. ACM.

[12] V. Lenders, G. Karlsson, and M. May. Wireless ad hoc podcasting. In *Proc. of IEEE SECON*, pages 273–283, 2007.

[13] F. Li and J. Wu. MOPS: Providing content-based service in disruption-tolerant networks. In *Proc. of ICDCS*, 2009.

[14] C. Liu, K. Wu, , and J. Pei. An energy-efficient data collection framework for wireless sensor networks by exploiting spatiotemporal correlation. *IEEE Trans. Parallel Distrib. Syst.*, 18:1011–1023, 2007.

[15] J. Liu, M. Adler, D. Towsley, and C. Zhang. On optimal communication cost for gathering correlated data through wireless sensor networks. In *Proc. of MobiCom*, 2006.

[16] Y. Ma, Y. Guo, X. Tian, and M. Ghanem. Distributed clustering-based aggregation algorithm for spatial correlated sensor networks. *Sensors Journal, IEEE*, 11(3):641–648, 2011.

[17] E. Ngai, M. B. Srivastava, and J. Liu. Context-aware sensor data dissemination for mobile users in remote areas. In *INFOCOM, 2012 Proceedings IEEE*, pages 2711–2715. IEEE, 2012.

[18] S. Pattem, B. Krishnamachari, and R. Govindan. The impact of spatial correlation on routing with compression in wireless sensor networks. *ACM Trans. Sensor Networks*, 4:24–33, 2008.

[19] M. J. Pitkanen and J. Ott. Redundancy and distributed caching in mobile DTNs. In *MobiArch*, 2007.

[20] M. Uddin, H. Wang, F. Saremi, G.-J. Qi, T. Abdelzaher, and T. Huang. PhotoNet: a similarity-aware picture delivery service for situation awareness. In *Proc. of IEEE RTSS*, 2011.

[21] M. Vuran, O. Akan, and I. Akyildiz. Spatio-temporal correlation: theory and applications for wireless sensor networks. *Computer Networks*, 45:245–259, 2004.

[22] J. Wang, R. Wakikawa, and L. Zhang. Dmnd: Collecting data from mobiles using named data. In *Vehicular Networking Conference (VNC), 2010 IEEE*, pages 49–56. IEEE, 2010.

[23] S. Wang, S. Hu, S. Li, H. Liu, M. Uddin, and T. Abdelzaher. Minerva: Information-centric programming for social sensing. In *Proc. of IEEE ICCCN*, 2013.

[24] U. Weinsberg, Q. Li, N. Taft, A. Balachandran, V. Sekar, G. Iannaccone, and S. Seshan. Care: content aware redundancy elimination for challenged networks. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pages 127–132. ACM, 2012.

[25] X. Xu, J. Luo, and Q. Zhang. Delay tolerant event collection in sensor networks with mobile sink. In *Proc. of INFOCOM*, 2010.

[26] L. Yin and G. Cao. Supporting cooperative caching in ad hoc networks. *IEEE Trans. on Mobile Computing*, 5, 2011.

[27] E. Yoneki, P. Hui, S. Chan, and J. Crowcroft. A socio-aware overlay for publish/subscribe communication in delay tolerant networks. In *Proc. of MSWiM*, 2007.

[28] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang. T-drive: driving directions based on taxi trajectories. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 99–108. ACM, 2010.

[29] K. Yuan, B. Li, and B. Liang. A distributed framework for correlated data gathering in sensor networks. *IEEE Trans. Veh. Technol.*, 57:578–593, 2008.

[30] D. Zhao, H. Ma, and S. Tang. Coupon: Cooperatively building sensing maps in mobile opportunistic networks. In *Mobile Ad-Hoc and Sensor Systems (MASS), 2013 IEEE 10th International Conference on*, pages 295–303. IEEE, 2013.

[31] Z. Zhu and A. Afanasyev. Lets chronosync: Decentralized dataset state synchronization in named data networking. In *Proceedings of the 21st IEEE International Conference on Network Protocols (ICNP 2013)*, 2013.

APPENDIX

Lemma 1: Algorithm 1 always schedule the data objects with least common name prefix with respect to the data set on the receiver side plus the data set already scheduled on the sender side.

Proof: In Algorithm 1, the procedure MINCHILD always returns the data objects with the shortest common name prefix with respect to the data objects at the receiver side in each iteration. The procedure UPDATEOCCUPANCY guarantees that the occupancy tree of the receiver side is updated by adding the data object currently is scheduled. Therefore, the lemma holds. ■

Lemma 2: When data objects to be scheduled share the same length of the common name prefix with respect to the data set on the receiver side plus the data set already scheduled on the sender side, Algorithm 1 always populates the name tree in a balanced fashion to schedule the data object whose name prefix has the least occupancy.

Proof: This lemma is guaranteed to hold by the procedure LEASTOCCUPANCY in Algorithm 1. The input of this procedure is the output data set of MINCHILD such that data objects have the same length of common name prefix with respect to the data set on the receiver side union the data set already scheduled on the sender side. The output of this procedure is the data object residing at the least populated name prefix branch. Therefore, the lemma holds. ■

Theorem 1: By marginal utility comparison rules in Section III-C, Algorithm 1 returns the optimal prioritization order of the data objects at the sender side when data objects have the same size and weight. (This optimality holds *without assumption on the number of data objects transmitted in one transmission session.* In other words, for *any* k data objects transmitted in one transmission session Algorithm 1 is optimal, where k is no greater than the total number of data objects at the sender side.)

Proof: Since data objects are assumed to have the same size, in one transmission session, the number of data objects can be transmitted is the same for any prioritization. Let's denote this number n. Furthermore, the assumption that data objects have the same weight implies that the marginal information utility of the data objects only depends on the data names.

By Lemma 1, Lemma 2 and the marginal utility comparison rules in Section III-C, we know that Algorithm 1 prioritizes the data objects in the non-increasing order of the marginal utilities. We claim that:

Claim: Given n numbers $\{m_1, m_2, \cdots, m_n\}$ and two permutations P and Q. P permutes those numbers in the non-increasing order and Q permutes in an arbitrary order. For any $0 \leq k \leq n$, $\sum_{i=P_0}^{P_k} m_i \geq \sum_{i=Q_0}^{Q_k} m_i$, where P_i (Q_i *resp.*) means the i-th number based on the permutation P (Q *resp.*).

Proof of Claim: Suppose that $\sum_{i=P_0}^{P_k} m_i < \sum_{i=Q_0}^{Q_k} m_i$. Then there must exist some number m_t that in the first k elements by the Q permutation but not in that by the P permutation and $m_t > m_{P_j}$ for some $0 \leq j \leq k$. Therefore, we got a contradiction with that P permutes the numbers in the non-increasing order. Therefore, $\sum_{i=P_0}^{P_k} m_i \geq \sum_{i=Q_0}^{Q_k} m_i$.

The above claim actually proves that our prioritization is optimal, since it permutes data objects in the non-increasing order of information utility. And the optimality is guaranteed for any number k of data objects transmitted in one transmission session, where k is no greater than the total number of data objects at the sender side by the above claim. ■

Theorem 2: The time complexity of Algorithm 1 is $O(n \log n + H^2 n)$, where H is the height of the name tree composed of the names of data objects to be transmitted. When $H = O(1)$, the time complexity is $O(n \log n)$, the best complexity bound for sorting based on comparison.

Proof: The time complexity of MINCHILD and LEASTOCCUPANCY is $O(\log(\Delta))$, where Δ is the maximum number of children of every node in the name tree, *i.e.*, $\Delta := \max_{v \in tree} |v.child|$. The time complexity of UPDATEOCCUPANCY is $O(H)$, where H is the height of the name tree. For each level in the name tree, the time complexity of the while loop is $O(n(\log \Delta + H))$, thus the total time complexity of Algorithm 1 is $O(H \cdot n(\log \Delta + H)) = O(n \log n + H^2 n)$, since $n = O(\Delta^H)$, which completes the proof. ■

Consider the fact that $H << n$ in practical, we have the time complexity of Algorithm 1 is $O(n \log^2 n)$ if $H = O(\log n)$, or $O(n \log n)$ if $H = O(1)$, which means the time complexity of our algorithm is almost the same as the sorting algorithm.

Theorem 3: Compared with the optimal offline algorithm satisfying the hierarchical similarity property and deminishing return property, in one transmission session, the approximation ratio of Algorithm 1 in the general case is $\frac{N}{N+\delta}$, where N is the total number of packets transmitted in the transmission session by Algorithm 1 and $\delta = \lfloor \frac{L_{\max}}{L_{\min}} \rfloor$, $L_{\max}$ (*resp.* $L_{\min}$) is the size of the largest (*resp.* smallest) data object.

Proof: The optimality requires the two properties must be satisfied, which means the occupancy should be well balanced in the name tree. By Theorem 1, Algorithm 1 balances the occupancy in the name tree optimally in the uniform data size case, which can be generalized straightforward in the variable data size case. Thus, all the data objects transmitted by using Algorithm 1 should be transmitted using the optimal algorithm. Otherwise, the occupancy balancing is violated.

In the worst case, using Algorithm 1 we can waste $L_{\max} - \epsilon$ transmittable bytes by scheduling a largest data object in the end of the transmission session. However, we can use those bytes to transmit several small data objects. So the number of packets transmitted by the optimal algorithm is at most $N + \lfloor L_{\max}/L_{\min} \rfloor = N + \delta$, which proves the theorem. ■