NEST: Efficient Transport of Data Summaries over Named Data Networks

Karim Khalil, Azeem Aqil, Srikanth V. Krishnamurthy
University of California Riverside
{karimk, aaqil001, krish}@cs.ucr.edu

Tarek Abdelzaher
University of Illinois at Urbana Champaign
zaher@illinois.edu

Lance Kaplan
Army Research Lab
lance.m.kaplan.civ@mail.mil

***Abstract*—In many emerging data retrieval applications, in response to queries, consumers are interested in getting a summarized version of content quickly rather than retrieving all available data. Recently, Named Data Networks (NDN) have been considered for efficient transfer of summarized information, but the research is still in its infancy. In this paper, we propose *NEST*, a novel transport protocol for delivering extractive summaries of a dataset distributed across multiple producers over NDN. The goal is to exploit diversity in network conditions between a consumer and different producers towards delivering the consumer-specified summary while minimizing latency. *NEST* first creates a unified hierarchical representation of the available distributed content using state-of-the-art distributed clustering. Then, using this representation of the dataset, the protocol creates interest messages based on which consumers can opportunistically retrieve representative data objects from the best producers while adapting to dynamic network conditions by capitalizing on the flexibility offered by the NDN infrastructure. We implement *NEST* on the Mini-NDN network emulator and evaluate its performance using datasets collected from Twitter. Our experimental results show that *NEST* takes advantage of producer diversity achieving large latency reduction gains of up to 50% compared to baseline protocols.**

I. Introduction

With the emergence of Internet of Things (IoT) and dissemination of online social content, sources of various kinds continuously generate streams of data. The number of such sources is growing continuously, leading to an exponential growth in the available data for a consumer [1], [2]. In fact, consumers may experience a data deluge leading to information overload if they get all the data pertaining to a subject of interest [3]. One way to cope with this data deluge, is via data summarization services which enable clients (whether humans or computer systems) to retrieve summaries with user-specified granularity. These summaries can then be used in analysis and decision making processes.

To exemplify the above, consider a smart city scenario [4] wherein sensors continuously gather data about traffic conditions. On their path to the destination, smart cars contact road infrastructure hot-spots for updates. In this scenario, collected data may have significant redundancy, local data at different repositories have semantic overlap, and network conditions are diverse. Consumers are likely to be interested in receiving content with varying granularity of detail as quickly as possible. As a second example, consumers interested in getting a summary of top stories from a variety of news media may be interested in quickly retrieving only an overview, based on which they may then choose to get more details only on certain stories. Data summarization can be an effective solution in delivering the right level of detail to consumers. In general, summaries can either be processed content that provide a synopsis of the data, or a set of representative samples that sufficiently satisfy the consumer's need. In this paper, we focus on the latter.

There is a set of challenges that will need to be addressed in order to deliver summaries from a set of producers to consumers. First, retrieving summaries from different producers independently will create redundant content, thus wasting communication resources and defying the purpose of summarization. Solving this problem requires the efficient creation of a global representation of the content available at the different producers. Furthermore, the network must be able to match a consumer's request with what is available at the various producers and retrieve the proper summary. In many applications, consumers are not interested in the source of the content or where it is, but rather the content itself and how fast it can be retrieved. Finally, since the network conditions between the consumer and the plurality of producers can be diverse (e.g., varying bandwidth and link delay), the transport framework has to be intelligent to retrieve the content from the "best" producer, i.e., the content that fulfills the consumer's requirement with the best performance (e.g., minimum latency).

In this paper, we develop NDN-based Efficient Summary Transport (*NEST*), a transport protocol which efficiently transfers an extractive summary of a dataset distributed across multiple connected producers to the requesting consumers, with low latency. Our framework is developed on top of the Named Data Networks (NDN) infrastructure, an implementation of the Information Centric Networks (ICN) paradigm. NDN is a pull-based network architecture which supports the forwarding of content from producers to consumers using *hierarchical names*. It offers a way to seamlessly map content to interests and thus, we argue that it is natural to leverage its abilities towards achieving the efficient transport of content summaries from a diverse set of producers to consumers. *NEST* allows producers to converge to a *common namespace*, wherein objects that are very similar are *named* similarly, linking the summarization problem to NDN's name-based forwarding. *NEST* is designed as an end-to-end protocol that runs at the hosts and does not require changes to the underlying NDN infrastructure.

ISBN 978-3-903176-08-9 2018 IFIP

NEST first creates a global hierarchical representation of the dataset by synchronizing the producers' local datasets with minimal overhead. Subsequently, this representation is used to generate an ordered list of names that is sent to interested consumers to guide them in retrieving content summaries of varying granularity. In this list, object names are "constructed" such that, from a set of similar objects at different producers, an object is seamlessly returned from the producer with the most favorable network conditions for each request based on the ordered list, thereby achieving minimum latency. In building *NEST* we make the following key contributions:

- We develop a distributed synchronization algorithm that capitalizes on recent advances in distributed clustering to create a global view of the shared dataset, towards realizing summaries of varying granularity.
- We develop interest-name design rules that automatically and opportunistically adapt to varying network conditions to minimize latency in delivering summaries to consumers over the underlying NDN.
- We implement *NEST* on Mini-NDN, a network emulator for NDN. We then perform extensive evaluations using datasets collected from Twitter. Our results show that *NEST* exploits producer diversity to reduce latency by up to 50% compared to baseline summary transport strategies that retrieve specific data objects.

II. Background

ICN has become popular recently for presenting an alternative and future architecture for the Internet as it becomes more content-centric rather than host-centric, and NDN [5] is one typical implementation. In NDN, consumers send interest messages requesting specific content using hierarchical names, where one data message is returned for each interest message. The interest is generated by a consumer to indicate that she seeks to retrieve a matching object. Partial prefix matching is used when checking whether a named object matches an interest name. In addition, intermediate routers employ multi-path forwarding rules to pass interest messages to the next hop until it reaches producers of the named content. Producers then return data messages where the payload is the data object with a matching name. Data messages are forwarded along the *reverse paths* corresponding to that taken by the interest message. Whenever a router receives a data message, the entry for the corresponding interest is removed from its Pending Interest Table (PIT) after it is forwarded. Any subsequent data messages for the satisfied interest are suppressed (i.e., not forwarded). If caching is enabled, intermediate routers keep a copy of the data messages for a specified period of time, and return it for subsequent matching interests from any consumer. A key feature of the interest message format is the exclusion option; consumers use this optional field to specify object name suffixes that they do not want to retrieve for the given name prefix in the interest message.

In our work, we consider the problem of efficient transport of data summaries. For a given dataset $\mathcal{P}$, a summary is a data subset of $\mathcal{P}$ such that each data point in the summary represents a set of similar (we formally define similarity in Section III-A) data points in $\mathcal{P}$. Recent work [6] proposed

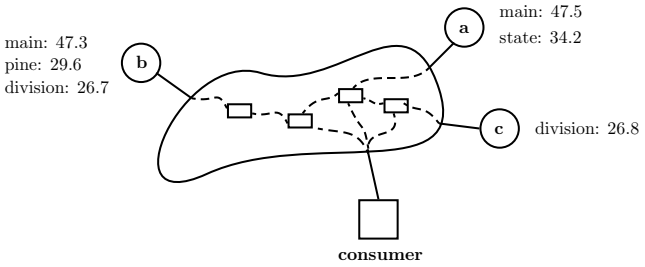


Fig. 1: An example network with three producers a, b and c. Shown are average speed measurements at each producer.

a summary transport protocol for NDN in which an ordered "names list" is created from a hierarchical tree representation of a dataset. The structure of the tree is such that data objects sharing a longer name prefix have more semantic overlap. Thus, when a summary of the content under the tree is requested, returning objects in a shortest-shared-prefix-first order minimizes information loss (relative to retrieving all data) over all different orders of a given summary size by reducing semantic redundancy. Here, as more data objects are transported according to this order, finer granularity details about the data are retrieved. However, only one producer was considered. In data summarization applications in which clients are interested in a summary of the dataset distributed across multiple repositories (producers), dynamic network conditions cause clients to experience very different network delays relative to the different producers from which the summaries are transferred. In addition, redundant content from different producers might be retrieved thereby causing the summaries to be of poor quality (unnecessary redundant content). Thus, when multiple data producers share similar data objects, opportunities to improve latency performance by retrieving *any* object from a set of similar objects are available. However, to exploit these opportunities, a flexible and adaptive data transport protocol is needed. The novelty of *NEST* is that it allows consumers to realize the advantage of *producer diversity* to retrieve summaries with minimum latency while requiring no changes to the underlying NDN architecture.

To illustrate producer diversity, consider an example in which a consumer is interested in a summary of average vehicle speeds in certain section of a city in a given period of time. Measurements are collected from various sensors into a set of three repositories (i.e., producers), named a, b and c, which are connected to the consumer as shown in Fig. 1. In this example, consider data from four streets: main, state, division and pine. Due to varying network conditions (e.g., varying wireless channel quality, network congestion, etc.), the delay in retrieval of data from different producers will be different. In particular, the connection to producer b is experiencing longer delays compared to other producers. Thus, to get a summary of the measurements quickly, the best strategy is to retrieve "main" and "state" from producer a, "division" from producer c and only "pine" from producer b. In other words, the consumer would better retrieve it from the producer with lower delay. The challenge, however, is to figure out which data objects to retrieve from which producer

in order to minimize latency while still fulfilling a notion of completeness of the collected summary. *NEST* offers an efficient solution for this problem be leveraging NDN.

III. System Design

NEST comprises two main functional components viz., (a) Producer Synchronization (ProdSync) and (b) Producer Diversity guided Summary Transport (*PDST*). The first component creates a hierarchical representation of the dataset shared by multiple producers, while the second component manages the transport of data objects between producers and consumers on the NDN. In the following, we discuss the design details of each of the two components.

A. Producer Synchronization

In many applications, data is collected from sensors and cached at a connected set of repositories (i.e., producers) for further processing and dissemination to consumers. Since, transporting objects from individual producers to consumers independently can result in performance penalties and wasteful transfers (e.g., duplicate or redundant data), summarization inherently requires co-ordination or more precisely synchronization between producers. Thus, the first challenge in efficiently delivering summaries from the set of producers (which is typically the order of tens) to consumers is to derive a global view of the available data. This global view necessarily describes the different groups of similar data points (i.e., clusters) as well as the relationship between them. For example, in Fig. 1, measurements from the same street are considered similar and thus are clustered together. Also data from all streets in each neighborhood can be grouped together when only information at a more abstract level is needed by the consumer. When such *hierarchical clustering representation* is available, it suffices to retrieve a representative of each cluster at the required level of detail to get a summary of the available data. This hierarchical representation naturally leads to hierarchical names for each data object based on which cluster it belongs to (similar to naming in Unix-like file systems).

For large datasets, it is not practical to transmit local datasets or large samples thereof to a centralized location to construct such a global hierarchical representation of the available data. In this context, the producer synchronization problem is how to efficiently create a unified view of the hierarchical names of data objects at all the producers. To this end, we develop ProdSync, an iterative distributed clustering algorithm in which producers exchange meta-data of local clusters and samples from their local datasets while incurring minimal communication overhead. This enables the construction of a global tree representation in which each producer maintains information about the position of their local data points in the tree.

To optimize the amount of data exchanged between producers (i.e., overhead) in each iteration of ProdSync, we employ a recently developed distributed clustering algorithm [7], which is based on the construction of ϵ-coresets [8]. This algorithm guarantees a bounded clustering cost relative to a centralized solution, at the minimum communication cost. It was later shown to be communication-optimal [9], where the communication-optimality metric used is the number of data points exchanged between the producers in the network. This metric is also correlated to the convergence time of the ProdSync algorithm; the more the messages exchanged between producers, the more time it takes for ProdSync to converge.

Algorithm 1 ProdSync

Input: Similarity threshold τ, set of producers $\mathcal{N}$
1: Initialize: $\mathcal{L} = r, \mathcal{N}^r = \mathcal{N}$
2: **repeat**
3: Pick an unprocessed tree node $l \in \mathcal{L}$
4: Select coordinator $n_l \in \mathcal{N}^l$
5: Each producer $i \in \mathcal{N}^l$ solves local clustering problem on $\mathcal{P}_i^l$
6: Producers exchange local clustering costs c_i^l
7: Coordinator collects samples $\mathcal{S}_i^l$ from all producers $i \in \mathcal{N}^l$
8: **if** $\max_{p,q \in \cup_i \mathcal{S}_i^l} d(p,q) < \tau$ **then**
9: Coordinator solves global clustering on $\cup_i \mathcal{S}_i^l$
10: **else**
11: l is a leaf node
12: **end if**
13: Coordinator delivers solutions $\mathcal{G}^l$ to all producers in $\mathcal{N}^l$
14: Producers send coordinator local tree info $\mathbf{u}_i^l$
15: Update $\mathcal{T}, \mathcal{N}^l, \mathcal{L}$: $\mathcal{L} \leftarrow g \ \forall g \in \mathcal{G}^l$
16: **until** all $l \in \mathcal{L}$ are processed
Output: Hierarchical tree representation $\mathcal{T}$

Notation: In the following, we introduce notation that will help in the description of ProdSync. Suppose we have a set of connected producers (repositories) $\mathcal{N}$, of size N. Let the global dataset be denoted by $\mathcal{P}$, where $\mathcal{P}_i \subset \mathcal{P}$ is the local subset corresponding to producer $i \in \mathcal{N}$. Let the distance measure between any pair of data points $p, q \in \mathcal{P}$ be given by $d(p,q)$.[1]

Let tree $\mathcal{T}$ be a hierarchical representation for the dataset $\mathcal{P}$, capturing similarities between data points in a hierarchical form. Suppose $\mathcal{L}$ is the set of tree nodes on $\mathcal{T}$, and let $r \in \mathcal{L}$ be the root node. Let $\mathcal{P}^l$ be a data subset of $\mathcal{P}$ holding data under subtree rooted at node l. We also define $\mathcal{P}_i^l = \mathcal{P}_i \cap \mathcal{P}^l$ and $\mathcal{N}^l = \{i \in \mathcal{N} : \mathcal{P}_i^l \neq \phi\}$. Note that $\mathcal{N}^r = \mathcal{N}$ and $\mathcal{P}^r = \mathcal{P}$. In each iteration l of ProdSync, each producer i solves an instance of k-means clustering and sends local information $\mathcal{S}_i^l$ (to be made precise) to a designated coordinator n_l, where the coordinator for tree node r (i.e., n_r), is called the root coordinator. In a clustering problem, the clustering cost is the sum of squared distances between each point in the dataset and its corresponding cluster center. We say that p and q are similar if $d(p,q) < \tau$, for a given threshold τ.

ProdSync details: The details of ProdSync are presented in Alg. 1. The algorithm iteratively clusters the dataset $\mathcal{P}$ shared across all producers $\mathcal{N}$ to generate the tree $\mathcal{T}$. To process a node $l \in \mathcal{L}$, a coordinator n_l is first selected[2] which later collects information about local clustering solutions from all producers in $\mathcal{N}^l$. In each iteration, the goal is to find a set of

[1] Vector space representation of data as well as similarity measures vary depending on application and data type. While we use specific representation and similarity measures in Section V, the effect of these on the clustering quality is of separate interest and is beyond the scope of this paper.

[2] We defer a discussion of how we implement coordinator selection to Section IV.

k cluster centers $\mathcal{G}_l$ at the coordinator, representing all data subsets $\mathcal{P}_i^l, i \in \mathcal{N}_l$. This is achieved by using an efficient distributed clustering algorithm [7], described briefly in the following.

Each producer first solves a k-means clustering problem on the local dataset $\mathcal{P}_i^l$ and then non-uniformly samples $\mathcal{P}_i^l$ based on the clustering costs c_i^l collected from all other producers $i \in \mathcal{N}^l$. In this sampling, a point with higher cost is sampled with higher probability. Each producer constructs its local portion of the ϵ-coreset, $\mathcal{S}_i^l$, which consists of the samples and their corresponding weights. Then, $\mathcal{S}_i^l$ are collected at the coordinator. A weighted k-means clustering problem is solved on the weighted samples $\cup_i \mathcal{S}_i^l$. The solution of the global clustering problem is then shared with producers in $\mathcal{N}^l$.

In ProdSync, a global clustering is solution is computed for a node l only when l is not a leaf node. We reach a leaf node in $\mathcal{T}$ (and hence stop further iterations of clustering on that node) when the diameter of the cluster is less than the threshold τ (condition on Line 8). In this case, each producer then updates the coordinator with cluster membership counts $\mathbf{u}_i^l$, which is a vector of size k. This information helps in creating the tree representation of the dataset as well as names for objects. Then, new nodes are added to the tree $\mathcal{T}$, one node representing each cluster in $\mathcal{G}_l$. Iteration stops when all nodes in $\mathcal{L}$ are processed.

Complexity analysis: At each iteration $l \in \mathcal{L}$, three rounds of message exchanges between the coordinator and other producers are required. First, the costs of local clustering solutions are collected by the coordinator and the sum cost is shared with all producers. Then, samples are sent to the coordinator and the solution $\mathcal{G}_l$ is returned to each producer. Finally, updates $\mathbf{u}_i^l$ are sent to the coordinator. The communication overhead is thus $O(N)$ per iteration.

To process a node $l \in \mathcal{L}$, each producer i solves an instance of the k-means clustering problem on the local dataset $\mathcal{P}_i$ (Steps 5 and 9). This problem is NP-hard [10]. However, there exist efficient approximations such as the Lloyd's algorithm [11], with time complexity $O(|\mathcal{P}_i|ksw)$, where s is the dimensionality of vectors representing the data points and w is the number of iterations needed for convergence. It was shown that, in practice, k-means converges in linear time with respect to the number of data points [12]. The number of nodes on the tree (i.e., $|\mathcal{L}|$), can vary between $O(\log_k |\mathcal{P}|)$ to $O(|\mathcal{P}|)$. In practice, we pipeline and parallelize the processing of tree nodes such that communication delay does not contribute a purely additive component to the total processing time. We study this in detail in Section V.

B. Producer Diversity guided Summary Transport

Given the hierarchical cluster representation of the data (as computed in Section III-A), we now need to decide the order in which data objects must be requested from these clusters. The intuition is that, in constructing a summary, we first want to have at least one representative of each cluster (e.g., a measurement of speed on each street), then get a second representative of each cluster (a second measurement from that street), and so on. If clusters are hierarchical, then we need one representative of each big cluster (say, street) before getting a representative of each sub-cluster (say city block on a street). As illustrated in the example in Section II, the choice of representative to retrieve entails a choice of producer, some being more accessible (better network conditions) than others. We want to retrieve representatives from more accessible producers. A challenge is thus to decide on a retrieval plan that minimizes latency.

In this section, we develop an efficient transport protocol, called Producer Diversity guided Summary Transport (*PDST*) that takes the output tree $\mathcal{T}$ from ProdSync, transforms it to a List of Ordered Names (LON) that is delivered to interested consumers. To construct the LON, $\mathcal{T}$ is parsed such that leaves are visited in certain order and a data point is chosen from visited leaf and then added to LON. The tree is traversed such that the marginal utility of retrieved data objects is maximized. Thus, as more items are retrieved as per the LON, a more fine-grained summary is obtained by the consumer. Consumers request summary data objects based on the LON, and *PDST* delivers data objects from producers to consumers with minimum transport latency, defined as follows.

Definition 1. *The transport latency $T(j)$ corresponding to a given interest message j is the total time delay between sending the interest message and the reception of a data message.*

Fix an interest message j and let $\mathcal{N}(j) \subset \mathcal{N}$ be the set of producers with data objects that can satisfy interest message j. Let $T_i(j)$ be the transport latency when data is returned from producer i. The objective of *PDST* is to minimize the latency in retrieving data objects, whenever matching data objects are available at multiple producers. In other words, PDST aims to achieve $T^*(j) = \min_{i \in \mathcal{N}(j)} T_i(j)$. While doing so, the protocol must adapt to dynamic network conditions, and specifically varying link delays.

Satisfying the minimum transport latency and adaptability to dynamic network conditions, are challenging problems for multiple reasons. First, it is undesirable that consumers maintain state information for all available producers (e.g., by keeping the history of received data). Moreover, explicitly and continuously measuring transport latency for objects from different producers will incur non-negligible overhead. The novelty of *PDST* is that it achieves the aforementioned goals by crafting interest messages in a format that capitalizes on the features of NDN. Specifically, *PDST* exploits NDN's forwarding characteristics viz., multi-path and partial prefix match forwarding. It also leverages the fact that intermediate routers suppress multiple data messages retrieved in response to a single interest message and only forward the first match. The main observation is that if $\mathcal{N}(j)$ always reflects producers that have similar data objects that satisfy j, NDN operations will automatically help achieve $T^*(j)$ without the need to explicitly measure network conditions. To this end, *PDST* does not require any modifications to NDN and is only run at the producers and consumers as an application.

PDST achieves minimum transport latency and adapt to varying network conditions using three different processing steps at the different participating network entities. Below, we discuss the different steps of *PDST* in detail before we

formalize our result.

1) Root-coordinator-side PDST: The root coordinator is tasked with generating the LON from $\mathcal{T}$. First, names for all objects are created. These names are then processed to identify producer diversity opportunities. Finally, the tree is traversed to generate the ordered list of names.

First, names of all data objects at the leaves of $\mathcal{T}$ are automatically created. In particular, during clustering, tree nodes are given labels (e.g., '0' for the left branch and '1' for the right branch when $k = 2$), and data objects at the leaves are named by concatenating label names from the root node to the leaf, similar to the work in [13], thereby constructing a name prefix. However, unlike the work in [13], the root coordinator n_r does not have actual data points from all other producers; rather, it has the *counts* of data objects under each leaf from each producer. This information is collected in Step 14 of Alg. 1. Thus, n_r can now generate names for all data objects at the leafs of $\mathcal{T}$.

Consider a tree $\mathcal{T}$ created using ProdSync with $k = 2$ for data at two producers a and b. Under some tree leaf l' with a name prefix $p = /t/0/0/1/0/1$, suppose producer a and producer b have two and three data objects, respectively. Here, t represents the *topic* at the root of the tree. Now, the root coordinator can simply name data objects under this leaf as $/t/0/0/1/0/1/a0$, $/t/0/0/1/0/1/a1$, $/t/0/0/1/0/1/b0$, $/t/0/0/1/0/1/b1$, $/t/0/0/1/0/1/b2$. Based on the user-specified and application-dependent similarity measure, objects under l' are deemed similar. At the same time, each of the producers a and b will fix some order for their local data objects, based on user-specified weights corresponding to each data object (e.g., content popularity, freshness, etc). Note that while each producer can rank order similar local data objects using user-specified weights, the relative ordering between similar data objects at different producers is not needed at the root coordinator. This is because our design gives priority to improving transport latency by retrieving the next (highest weight) data object (based on local ranking) from the producer with the best network conditions.

Denote any leaf of $\mathcal{T}$ with objects from multiple producers, as an *opportunity leaf*. The next step for the root coordinator is to process the data object names such that a special symbol (˜) is concatenated at the end of all object names under any tree leaf where data objects belonging to multiple producers exist. Thus, for leaf l', the object names will be processed to be $/t/0/0/1/0/1/a0$˜, $/t/0/0/1/0/1/a1$˜, $/t/0/0/1/0/1/b0$˜, $/t/0/0/1/0/1/b1$˜, $/t/0/0/1/0/1/b2$˜. This special symbol in the object names will later be used by the consumer to construct interest messages that allow retrieval of data objects from the producer with the minimum transport latency.

Given the hierarchical tree representation $\mathcal{T}$ created using ProdSync as described in Section III-A, the root coordinator can now transform $\mathcal{T}$ into an LON by traversing $\mathcal{T}$ from the root to the leaves and returning the name of the object at the leaf. During traversal the branches are selected such that an object with the shortest-shared-prefix, with respect to previously returned object names, is returned.

Finally, this processed list is sent to the consumers upon request. In particular, when a consumer requests a summary of a dataset under the prefix $/t$, the root coordinator will send a data message carrying the LON for data objects under the corresponding tree. Note that the LON is generally of much smaller size compared to data objects (e.g., in social media, a tweet could have an image or video object embedded in it).

2) Consumer-side PDST: Each consumer running the *NEST* application will first request the LON under some tree root $/t$. The consumer then sends interests for items in the LON in the given order. However, the interest names used will vary depending on whether the object belongs to an opportunity leaf. For such objects, a *producer diversity opportunity* exists and thus the consumer can take advantage of it. In particular, for any object name in the LON ending in ˜, the consumer sends the interest message with the partial name up to the prefix of the corresponding leaf in $\mathcal{T}$. For example, if the next data object name in the LON is $/t/0/0/1/0/1/b0$˜, the consumer sends the interest message $/t/0/0/1/0/1/$ instead.

This interest will be forwarded by the underlying NDN to all producers with data objects under the corresponding opportunity leaf. Thus, all producers will respond with data objects under the given leaf, and only one data message will be forwarded to the requesting consumer while other messages will not be forwarded. To avoid retrieving duplicate objects that were retrieved previously using the same partial name, the consumer employs the *exclude* option in the interest message. In particular, it includes the last component in the name of the objects retrieved previously under same leaf. For example, when an interest message is sent with the partial name $/t/0/0/1/0/1/$ and data object $/t/0/0/1/0/1/b0$ is retrieved, the next interest message for an object belonging to the same opportunity leaf will be $/t/0/0/1/0/1/(-b0)$.

One of the main advantages of crafting the interest messages as described above is that the framework automatically adapts to changing link delays. Thus, two consumers sending the same interest message will get potentially different (but semantically similar) data objects from different producers. Furthermore, as network conditions change over time, a consumer may get data objects from other producers because of latency advantages. Since *PDST* capitalizes on NDN forwarding rules, it also works when caching at intermediate routers is enabled without the need to modify the software they run. Specifically, caches will also use partial prefix matching and return objects with matching names. If no matches are found in the cache, the interest will be further forwarded.

3) Producer-side PDST: On the producer side, each producer maintains an ordering of the local data points based on user-specified and application-dependent weights. For example, in a social media application, the popularity of the content could be used as the weight. In a sensor network application, more recent events or measurements could be weighted highly if freshness is desired. Whenever the producer receives an interest message with a partial name, it responds with a data object from the subtree specified by the name prefix, returning the first object in order after the excluded objects. For example, if the interest message received by producer b is $/t/0/0/1/0/1/(-b0)$, then it returns object $/t/0/0/1/0/1/b1$. Note that the object name $b1$ might not be the actual object name at producer b, but rather a pointer

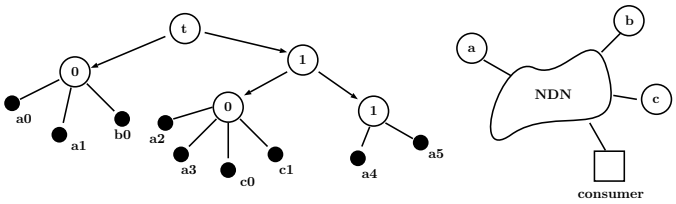


Fig. 2: Example network and tree with three producers (a,b,c) and a consumer.

	NEST's LON	Interest name sent	Data name received
1	$/t/0/a0^{\sim}$	$/t/0/$	$/t/0/b0$
2	$/t/1/0/a0^{\sim}$	$/t/1/0/$	$/t/1/0/c0$
3	$/t/0/a1^{\sim}$	$/t/0/(-b0)$	$/t/0/a0$
4	$/t/1/1/a4$	$/t/1/1/a4$	$/t/1/1/a4$
5	$/t/0/b0^{\sim}$	$/t/0/(-b0,-a0)$	$/t/0/a1$
6	$/t/1/0/a3^{\sim}$	$/t/1/0/(-c0)$	$/t/1/0/c1$
7	$/t/1/1/a5$	$/t/1/1/a5/$	$/t/1/1/a5$
8	$/t/1/0/c0^{\sim}$	$/t/1/0/(-c1)$	$/t/1/0/a2$
9	$/t/1/0/c1^{\sim}$	$/t/1/0/(-c1,-a1)$	$/t/1/0/a3$

TABLE I: Example of *PDST* operation.

to a data object under the given prefix which is second in order based on the weights. This order is maintained only locally by each producer.

In Table I, an example LON is shown. The corresponding network with one consumer and three producers, as well as the hierarchical data representation $\mathcal{T}$ are shown in Fig. 2. In the example network, the transport latency to producer c is the lowest, then to producer b, and then to producer a. The second column is generated by the root coordinator in *NEST* and represents the LON delivered to consumers requesting a summary of content under the prefix $/t$. In the third column, the interest names that the consumer uses in interest messages are shown. The names of the data objects received by the consumer in response, are shown in the last column. Note that the interest names in the third column are adapted based on names of data objects received so far (i.e., from previous rows), as listed in the last column. For example, consider row number 6. Here, the interest sent is for a data object that is under the prefix $/t/1/0/$. Since the consumer previously received the object $/t/1/0/c0$ (in row 2), it now includes $c0$ in the exclude field of the interest message. The NDN forwarding will pass the interest message $/t/1/0/(-c0)$ to all producers, but it will reach producer c first since it has the best network conditions with respect to the consumer. Now, producer c will check its local dataset for data objects under prefix $/t/1/0/$ and with rank order subsequent to object $c0$, returning object $/t/1/0/c1$. Data objects returned from other producers will then be suppressed by intermediate routers since the interest message would have been already satisfied by object $/t/1/0/c1$.

In the following, we formalize our main result. The proof is omitted for brevity.

Proposition 1. *Fix an interest message j. PDST achieves minimum latency $T^*(j)$.*

We note that Proposition 1 implies that *PDST* minimizes latency even if the link delay varies while the interest or data message has not been received at the destination.

Pipelining interests: PDST uses an adaptive pipelining window, which controls how many pending interests are allowed at any given time. In addition to being limited to a maximum size W, the window size is adapted based on the LON and the progress made thus far in processing the list. In particular, the consumer can send interests from the LON until a new entry requires sending a partial name which is *already* in use in a pending interest, or until the maximum window size is reached, whichever is smaller. This design prevents retrieval of duplicate objects, since names of previously retrieved objects are added to the exclude fields of subsequent interests, with the same partial names. We evaluate the choice of W in Section V. We also note that loss management is handled by the underlying NDN mechanisms through the use of timeout timers and retransmissions.

Caching: Before we conclude this section, we discuss how caching affects the performance of our system. As the number of consumers increase, it is expected that caches at intermediate routers will return data objects more often, improving latency performance with respect to a scenario wherein caching is disabled. This in turn could reduce the producer diversity opportunities that *NEST* tries to exploit to improve performance; the data is already cached en route. However, as will be shown in Section V, the marginal gain in latency reduction is large even when caching is enabled. In addition, the combined gain is substantial.

IV. Implementation

We implement *NEST* on Mini-NDN [14], an NDN network emulator based on the popular Mininet [15] virtual network environment. In Mini-NDN, a network topology is specified in which nodes are connected via links parameterized by link delay, bandwidth as well as loss percentage. Each node in the network is capable of running NDN applications, forwarding NDN packets according to the specified routing policy, as well as caching forwarded content.

Mini-NDN accomplishes these NDN functionalities by running an instance of Named Data Link State Routing Protocol (NLSR) [16] and NDN Forwarding Daemon (NFD) [17] on each instantiated node in the network. NLSR is a routing protocol responsible for populating NDN's Forwarding Information Base (FIB) while NFD is a network forwarder that is fully capable of forwarding NDN packets according to a diverse set of routing strategies.

Since Mini-NDN emulates the actual operations of NDN networks, one primary advantage is that the applications developed and tested on Mini-NDN can be readily operational on the NDN testbed [18] or other actual NDN networks.

A depiction of *NEST*'s different components is shown in Fig. 3. Each producer in the network runs the two functional components of *NEST* (ProdSync and *PDST*) simultaneously while the consumer runs *PDST*. First, producers in *NEST* run the "*NEST* Sync" application, which is responsible for implementing ProdSync and creating "*NEST* tree". In our implementation, we use k-means clustering with $k = 2$. The *NEST* tree is then passed to the "*NEST* Prod" application. In

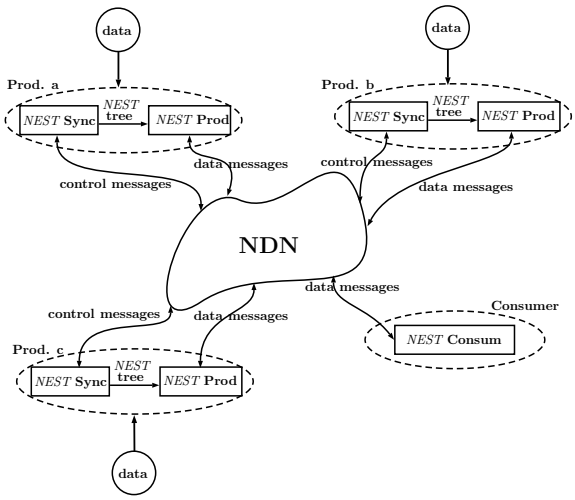


Fig. 3: A Network with three producers and a consumer running *NEST*. Each box represents an application running on producers or consumers.

this application, the tree is transformed to an LON which is then used to guide the transport of data items. Finally, the third application is the "*NEST* Consum" application running at each consumer. This component is responsible for sending interest messages that are responded to by the *NEST* Prod application running at each producer.

We implemented all the applications in Python. In the *NEST* Sync application, clustering information of every tree node is kept in a data structure that holds the state of the computations and data exchanged between the coordinator and non-coordinator producers. State and data are encoded into control messages, where an interest control message requests the start of computation or delivers the notification that a computation is completed, while the corresponding data control message delivers an acknowledgment or the requested data. On the other hand, in the *NEST* Consum application, the consumer implements a pipelining window of pending interests and a method to transform the LON to interest messages with partial names. *NEST* Prod implements a partial interest name match function to select messages to be sent to the consumers.

V. Evaluation Results

Setup: Our evaluations are based on a dataset collected from Twitter over a period of time from Dec 2016 to Mar 2017 using Twitter's streaming API and a set of search keywords for trending topics in politics, sports, and entertainment. Overall, we use a dataset of about 80K tweets in our evaluations.

In each experiment, we randomly distribute a sample of the dataset uniformly across the set of producers. We first pre-process the collected tweets to remove stop words, special characters, links and attachments, producing tokens. These tokens are then transformed to a high dimensional vector representation by computing the product of term frequency and inverse document frequency (*tf-idf*) [19], a popular method for text vectorization. We use the `sklearn` library [20] vectorizer to achieve this task.

TABLE II: ProdSync convergence time.

N	**3**	**5**	**7**	**9**
Time(s)	34	38	83	122

In Mini-Net, links connecting the producers and consumers are characterized by the link delay, the bandwidth, and the message loss rate. In our experiments, we fix the bandwidth and loss rates, and vary the link delays. We note that in Mini-Net, each host in the network runs the NDN stack and thus can be used as a producer, a consumer, and a forwarding switch, simultaneously. In addition, hosts have content stores and thus can cache data objects. In our experiments, we use a network topology similar to that used in the NDN testbed [18] and we have a varying number of producers consumers for different experiments as will be discussed in the following.

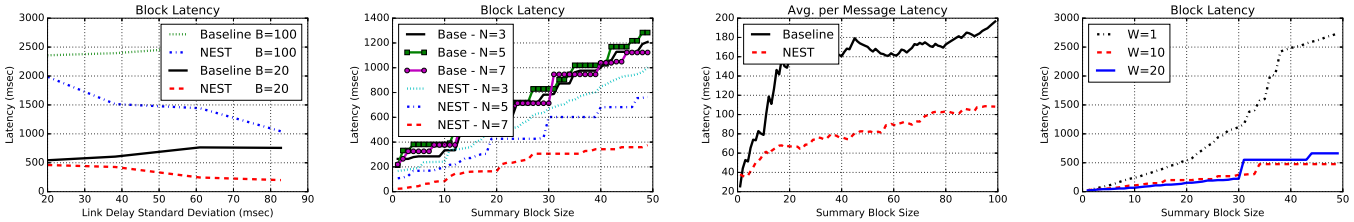
In the following, we define terms that we use in our evaluations. Let the *summary block* with size B be the number of data objects the consumer has to fetch in order to have a satisfactory summary. The *block latency* t_B is the delay from sending the interest message for the first data object in the block, until the successful reception of the data message corresponding to the last data object in the summary block. This quantity is directly proportional to the per interest latency defined in Section III-B.

We divide the evaluation results into two parts. In the first, we evaluate the performance of the ProdSync algorithm and quantify performance in terms of the convergence time. In the second, we focus on the latency performance of *PDST* and compare it to a baseline summary transport protocol with no producer diversity (i.e., a system in which the LON is used to retrieve data objects from specific producers, similar to the protocol in [6]).

A. Producer Sync

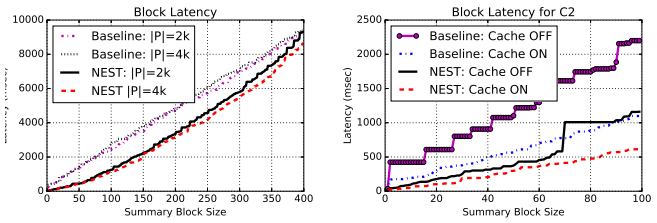
We consider networks with different numbers of producers and distribute a dataset of $4000N$ tweets uniformly at random over the N producers. We use Euclidean distance to measure similarity between different vectors representing tweets, and use a similarity threshold $\tau = 0.9$ as the stopping criterion for ProdSync. For the coordinator selection, in each iteration, we let the producer with the smallest ID perform the coordination tasks for the corresponding tree node. Producers are connected with link delays of 10 milliseconds.

We first evaluate ProdSync's convergence time. For scenarios with $N = 3, 5, 7, 9$ producers, we repeat the experiment 10 times and report the average convergence time in each case. Table II outlines the results. It can be seen that ProdSync's convergence time is approximately linear in the number of producers and the dataset size for $N > 3$. In many applications (such as traffic monitoring, news stories updates), major dataset changes happen on the order of hours. Thus, ProdSync provides a practical means for constructing the global representation of the distributed dataset as it can be run periodically at a rate that is faster than the rate of data evolution.



(a) Link delay variance. (b) Number of producers. (c) Per message latency. (d) Pipelining window sizes.

Fig. 4: Latency performance.



(a) Different sample size $|\mathcal{P}|$. (b) Caching.

Fig. 5: Latency performance.

B. Latency Performance

In this section, we evaluate the latency performance of *PDST* after the LON has been delivered to the consumers. We compare the performance to the baseline protocol.

1) Link delay variance: First, we vary link delay variance and measure the incurred latency. We fix the maximum pipelining window size to $W = 10$ and vary the link delays from the producers to the consumer in the range 5 to 200msec while maintaining a fixed average. Here, we consider a topology with 5 producers and 1 consumer, and we consider two different summary block sizes $B = \{20, 100\}$. As shown in Fig. 4a, the block latency improves as the link delay variance increases. This is because *NEST* effectively checks if similar objects exist at producers with better network conditions and fetches objects from those producers first. Essentially, objects with slow retrieval times are pushed to the end of retrieval order. Compared to the baseline system, block latency performance is improved by more than 40% when the link delay standard deviation is 50msec.

In Fig. 4b, we plot the block latency t_B for a varying B. We also show the performance for topologies with different number of producers. First, we observe that while the baseline system performance does not change when number of producers change, *NEST* fully utilizes producer diversity. In particular, as the number of producers increases, diversity improves and block latency decreases.

We also study the per message latency when $N = 5$, where W and link delays are chosen as in previous experiments. In Fig. 4c, we plot the average per message latency vs. different B. The figure shows that by taking advantage of producer diversity, the latency improvements can be as high as 50%. Note that as the block size increases for a fixed dataset size, this gain is expected to decrease. We study the effect of the ratio $\frac{B}{|\mathcal{P}|}$ in Section V-B3.

2) Pipelining: Next, we study the effect of the maximum pipelining window size W on the block latency. In Fig. 4d, there are five producers with link delays similar to those in the previous experiments. Note that *PDST*'s adaptive pipelining does not send new interest messages while pending interests with the same partial name exist, to avoid duplicate object retrievals. It is seen that pipelining improves block latency compared to a simple stop and wait approach ($W = 1$). In addition, consumers will experience more packet losses as W increases and thus more bandwidth wastage. The figure shows that diminishing gains result due to increasing W. We find that $W = 10$ achieves the best latency performance.

3) Impact of dataset size: Next, we study the effect of the dataset size on the block latency. It is expected that as the ratio $\frac{B}{|\mathcal{P}|}$ decreases, the latency gain of *NEST* increases. In other words, when the requested summary block size B is comparable to the dataset size, consumers may not be able to avoid fetching objects from producers with unfavorable network conditions. We fix five producers with link delays similar to previous experiments. Fig. 5a shows that, for a given block size B, the latency reduction gain is only slightly reduced when the sample size is halved. When $|\mathcal{P}| = 2000$, *NEST* can achieve a positive gain for summary block sizes up to 20% of the dataset, which is reasonable for applications in which consumers are interested only in data summaries of large datasets.

4) Caching: Finally, we study the performance of *NEST* when caching is enabled in the underlying NDN. In this experiment, the topology has two consumers and five producers with similar link delays as in previous experiments. Both consumers are using the same LON. We introduce a delay between the time each consumer starts fetching items to see the effect of caching. Caching allows intermediate nodes to temporarily store content that was previously forwarded to other hosts in the network. As seen in Fig. 5b, *NEST* yields latency performance improvements of about 50%. Compared to the baseline performance with caching disabled, the combined latency reduction gain is more than 70%.

VI. Related Work

There has been prior work on selectively sending a representative subset of data instead of the entire dataset [21], [22]. However, unlike our work, these efforts are application specific. Moreover, these approaches try to optimize for energy

efficiency in contrast to our goal of minimizing latency in retrieving the summary. Achieving our goal requires an approach that is much different from those proposed in these efforts.

More recently, multiple works considered optimizing latency in NDN [23]–[25]. In these works, architectural changes to NDN are proposed to improve the support for low latency applications such video conferencing [23]. In addition, multipath routing as well as network coding are employed [24], [25] to improve performance of video streaming. Unlike these approaches, *NEST* does not require changes to underlying NDN infrastructure and operates as an end-host application. Thus, we argue that it is much more general and easy to deploy.

On the other hand, distributed dataset synchronization was recently addressed [26]. The goal is to efficiently synchronize the state of a group of hosts for applications such as group text messaging. This is different from the problem we consider wherein we address producer synchronization, since we do not require all hosts to have the full dataset.

The closest works to ours are Espresso [13] and InfoMax [6]. The former creates a tree representation and object names from a given dataset for creating summaries, while the latter transforms the tree into an ordered list for transport. However, their model considers only a single producer. While we use a similar approach towards summarization, we address a different set of challenges wherein the dataset is distributed across multiple producers. In particular, transport performance is our primary issue of focus; this was not addressed in these works.

VII. Conclusion

In this paper, we target the problem of delivering a summary of a large dataset to consumers from a set of producers with low latency. Retrieval of such a summary of the dataset, is becoming popular in many emerging applications. We propose *NEST*, an efficient data summary transport protocol which leverages the NDN architecture towards achieving this goal. Our novel framework opportunistically fetches data from the producers with good network conditions relative to consumers after constructing a global view of the dataset shared between producers and establishing similarity relations between data points. Our experimental results show that large latency reduction gains can be achieved compared to baseline strategies that do not exploit producer diversity. The gains are especially noteworthy (up to 50%) when the number of producers and link delay variations are large.

Acknowledgment: This work was partially supported by the Army Research Laboratory 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 Army Research Laboratory 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. This work was also partially supported by the NSF CPS grant 1544969.

References

[1] L. Columbus, "Roundup of internet of things forecasts and market estimates, 2016," https://www.forbes.com/sites/louiscolumbus/2016/11/27/roundup-of-internet-of-things-forecasts-and-market-estimates-2016/#1b7ea093292d, 2016.

[2] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile Networks and Applications*, vol. 19, no. 2, pp. 171–209, 2014.

[3] B. Marr, "Big data overload: Why most companies can't deal with the data explosion," https://www.forbes.com/sites/bernardmarr/2016/04/28/big-data-overload-most-companies-cant-deal-with-the-data-explosion/#33cde7b06b0d, 2016.

[4] S. H. Bouk, S. H. Ahmed, D. Kim, and H. Song, "Named-data-networking-based its for smart cities," *IEEE Communications Magazine*, vol. 55, no. 1, pp. 105–111, 2017.

[5] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, P. Crowley, C. Papadopoulos, L. Wang, B. Zhang *et al.*, "Named data networking," *ACM SIGCOMM Computer Commun. Review*, vol. 44, no. 3, pp. 66–73, 2014.

[6] J. Lee, A. Kapoor, M. T. Al Amin, Z. Wang, Z. Zhang, R. Goyal, and T. Abdelzaher, "InfoMax: An information maximizing transport layer protocol for named data networks," in *IEEE 2015 24th Int. Conf. on Computer Commun. and Networks (ICCCN)*, 2015, pp. 1–10.

[7] M.-F. F. Balcan, S. Ehrlich, and Y. Liang, "Distributed k-means and k-median clustering on general topologies," in *Advances in Neural Information Processing Systems*, 2013, pp. 1995–2003.

[8] S. Har-Peled and S. Mazumdar, "On coresets for k-means and k-median clustering," in *Proceedings of the thirty-sixth annual ACM symposium on Theory of Computing*. ACM, 2004, pp. 291–300.

[9] J. Chen, H. Sun, D. Woodruff, and Q. Zhang, "Communication-optimal distributed clustering," in *Advances in Neural Information Processing Systems*, 2016, pp. 3727–3735.

[10] D. Aloise, A. Deshpande, P. Hansen, and P. Popat, "NP-hardness of euclidean sum-of-squares clustering," *Machine Learning*, vol. 75, no. 2, pp. 245–248, 2009.

[11] S. Lloyd, "Least squares quantization in PCM," *IEEE Trans. on Inf. Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[12] 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.

[13] J. Lee, M. T. Al Amin, and T. Abdelzaher, "Espresso: A data naming service for self-summarizing transport," in *2017 14th Annual IEEE Int. Conf. on Sensing, Commun., and Networking (SECON)*, 2017, pp. 1–9.

[14] "named-data/mini-ndn," https://github.com/named-data/mini-ndn.

[15] "Mininet," http://mininet.org/.

[16] "NLSR - named data link state routing protocol," http://named-data.net/doc/NLSR/current.

[17] "NFD - named data networking forwarding daemon," https://named-data.net/doc/NFD/current.

[18] "NDN testbed," https://named-data.net/ndn-testbed/.

[19] J. Ramos *et al.*, "Using tf-idf to determine word relevance in document queries," in *Proceedings of the first Instructional Conf. on Machine Learning*, vol. 242, 2003, pp. 133–142.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[21] H. Gupta, V. Navda, S. Das, and V. Chowdhary, "Efficient gathering of correlated data in sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 4, no. 1, p. 4, 2008.

[22] Y. Ma, Y. Guo, X. Tian, and M. Ghanem, "Distributed clustering-based aggregation algorithm for spatial correlated sensor networks," *IEEE Sensors Journal*, vol. 11, no. 3, pp. 641–648, 2011.

[23] M. Almishari, P. Gasti, N. Nathan, and G. Tsudik, "Optimizing bi-directional low-latency communication in Named Data Networking," *SIGCOMM Computer Commun. Review*, vol. 44, no. 1, pp. 13–19, 2013.

[24] S. de Arco, J. Eduardo, E. Bourtsoulatze, N. Thomos, and T. Braun, "Adaptive video streaming with network coding enabled named data networking," *IEEE Trans. on Multimedia*, 2017.

[25] K. Matsuzono, H. Asaeda, and T. Turletti, "Low latency low loss streaming using in-network coding and caching," in *IEEE INFOCOM*, 2017.

[26] Z. Zhu and A. Afanasyev, "Let's Chronosync: Decentralized dataset state synchronization in named data networking," in *2013 21st IEEE Int. Conf. on Network Protocols (ICNP)*, 2013, pp. 1–10.