Pervasive Edge Data Sharing in MANET

Xintong Song*, Yaodong Huang†, Qian Zhou†, Fan Ye†, Yuanyuan Yang† and Xiaoming Li*
*School of Electronics Engineering and Computer Science, Peking University, Beijing
{songxintong, lxm}@pku.edu.cn
†Department of Electrical and Computer Engineering, Stony Brook University, New York
{yaodong.huang, qian.zhou, fan.ye, yuanyuan.yang}@stonybrook.edu

***Abstract*—The proliferation and daily congregation of modern mobile devices have created abundant opportunities for peer edge devices to share valuable data with each other. The short contact durations, relatively small sharing sizes, and uncertain data availability, demand agile, light weight peer based data sharing. In this paper, we propose Peer Data Sharing (PDS) that enables edge devices to discover which data exist in nearby peers, and retrieve interested data robustly and efficiently. We use Wi-Fi ad hoc network as an example to demonstrate that PDS can leverage different underlying network and link technologies with proper adaption. Extensive evaluations show that PDS discovers and retrieves almost 100% data in tens of seconds, and remains robust despite wireless contention, simultaneous consumer requests and user mobility.**

I. Introduction

The proliferation of modern sensor-rich mobile devices (e.g., smartphones) and opportunistic congregation of users have created novel opportunities for peer data sharing. Many times spontaneous, agile data exchange among nearby users is desired. For example, during large outdoor events (e.g., music festivals, university commencements), smartphones carried by people can capture diverse data, including human activities, their locations, and image/video clips. When shared among peer devices, such data can help people avoid food stands of long lines, discover interesting souvenirs and artifacts, or enjoy images, video clips of special, memorable moments.

Peer data sharing in such pervasive edge environments has some unique characteristics. Each user may possess certain data and need data by others. However, which devices are around, and what kinds of data they carry, occur opportunistically and cannot be foretold. The limited durations (e.g., a few to tens of minutes) devices are in proximity, and the modest amount yet unforeseeable kinds of data, stipulate fast, light weight discovery and exchange on a *peer basis*. This decentralized sharing differs from most crowdsensing [1] applications where a central backend receives data from all devices and then distributes among them.

We propose *Peer Data Sharing (PDS)* that enables mobile devices to quickly discover what data exist in nearby peers and retrieve desired data from possibly multiple devices. PDS discovers all existing data and retrieve required data faithfully across opportunistically gathered peers, under limited wireless bandwidth and potentially frequent message losses, with low overhead and latency. PDS differs from existing data discovery and sharing work in mobile ad hoc networks [2]–[4]. It adopts a content centric design [5], [6] where data are self-contained entities that can be referenced, stored and accessed independently from their original producers. Thus data can be widely cached at and retrieved from any willing and capable nodes. The consumer can retrieve data from a close by cached copy, or different chunks from multiple cached copies, to reduce latency and aggregate bandwidth. PDS also differs from existing content centric networks [5], [6] in terms of wireless medium and network scale, which will be discussed in Section V.

PDS is designed at application level above the network stack. It can leverage different underlying network (e.g., IP) and link technologies (Wi-Fi infrastructure/ad hoc mode, Bluetooth, ZigBee, Wi-Fi Direct [7], D2D [8], etc.) with proper adaption. In this paper, we use Wi-Fi ad hoc mode as an example and demonstrate how to improve PDS performance by adapting it to this particular link technology.

We make the following contributions: 1) We devise robust and efficient pervasive data discovery (PDD) and retrieval (PDR) mechanisms. PDD returns all data existence information faithfully, despite dynamic changes in both device and data sets, while PDR retrieves different portions of data from multiple cached copies robustly and efficiently. 2) Dispte that PDS can leverage different underlying network and link technologies, we develop mechanisms to combat the consequence of possibly heavy contention losses in Wi-Fi ad hoc network specifically. 3) We perform large scale simulation to evaluate our proposal. The results show that consumers can discover almost 100% data in several seconds time.

The rest of this paper is organized as follows: In Section II, we present the general design of PDS. Then in Section III, we discuss how to adapt PDS to Wi-Fi ad hoc network. We present simulation based evaluation results in Section IV. Finally we compare related work in Section V and conclude our work in Section VII.

II. Peer Data Sharing

A. Assumptions and Goals

We make the following assumptions: the environment is uncertain and dynamic. Which devices are in proximity and what data they possess, are opportunistic and not known beforehand. Although users are free to move in/out any time, many of them stay for extended periods of time from a few to tens of minutes. The geographical area where users congregate (e.g., restaurants, parks, airports) and thus the network size are usually limited. Devices have reasonable storage (e.g., 16GB or higher). The amount and duration of data exchange are usually moderate (e.g., a few MBs and minutes). To enable opportunistic caching, we assume nodes will overhear transmitted frames whenever possible and act on the content. PDS does not assume any specific radio technology. Devices can connect to each other through different technologies (e.g., Wi-Fi ad hoc, Wi-Fi Direct [7], D2D [8], Bluetooth, etc.). However, for different technologies certain adaption is needed. In this paper we use multi-hop Wi-Fi ad hoc network as an example to demonstrate

The work in this paper was supported in part by the grant from US National Science Foundation under grant number CSR-1513719.

such adaption. All devices are cooperative and play by the rules. Only publicly sharable data are exchanged and we do not consider security or privacy issues in this work.

Each device can be a *consumer* that requests desired data, or a *producer* that provides them (either generated locally or cached). We focus on a typical scenario where the consumer needs one large, possibly popular data item (e.g., a video clip) consisting of many small *chunks*. A large data item can be consistently divided into small chunks when generated by the producer. Such dividing allows each chunk to be stored and accessed independently, which avoid excessive resource consumption on each node that holding the data item. As a result, chunks of popular data items can be available from multiple nearby devices.

Due to the uncertainty, a consumer has to discover what data exist in nearby devices. The Peer Data Discovery (PDD) provides such a "menu" of available data as completely and faithfully as possible. Peer Data Retrieval (PDR) should return at least one copy of each requested data item/chunk. Both are best effort: occasionally missing existing or reporting disappeared data is allowed because applications are not mission critical.

B. Data Descriptors, Metadata and Queries

When generating a new data item, a node creates and associates to it a ***data descriptor*** (i.e., metadata) consisting of multiple ***attributes*** each having a name and taking a certain value of some primitive type (e.g., string, integer, float, Unix time). For example, an NO_x pollutant sample may have *data type* of `NO`$_x$, *time* of the sample generation at `2016-01-01 08:00:00`, and *location* the GPS coordinates of the sample. To avoid conflicts, a *namespace* where the data type is defined (e.g., `environment monitoring`) can be added.

Each descriptor is a ***metadata entry*** that indicates the *potential* availability of the corresponding data item/chunk. Thus all such entries together describe what data may exist in the environment. Because metadata entries have small sizes and are frequently requested by many consumers, they are widely cached. Any node receiving, relaying or overhearing metadata entries will cache them to serve potential future requests. If a metadata entry is cached by a node without respective payload, an expiration time is added to this entry. Upon expiration, the node removes the entry if it does not yet have the payload. These simple rules make metadata and corresponding data roughly synchronized in the network.

A consumer sends ***queries*** to specify desired data and retrieve them from other devices. A query consists of a collection of ***predicates*** specifying desired values on attributes using a relation (e.g., $=$, $>$, $\in$, etc.) to a value or value range. Queries can be specified for data items, chunks and metadata. The retrieval of them follows similar query-response mechanisms, to be presented in Section II-C and II-D.

C. Peer Data Discovery (PDD)

Peer Data Sharing (PDS) consists of two components: ***Peer Data Discovery (PDD)*** and ***Peer Data Retrieval (PDR)***. They share similar message formats, processing procedures and routing mechanism. We present PDD in this section, and introduce PDR design in Section II-D focusing on its differences.

PDD collects metadata through multi-round requests. In each round the consumer sends a query message requesting metadata,

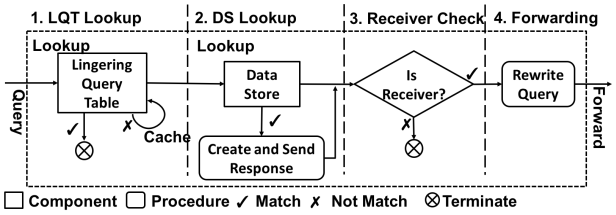

Fig. 1. PDD Query Processing: When receiving a query, a node should perform 4 steps: Lingering Query Table Lookup, Data Store Lookup, Receiver Check and Forwarding.

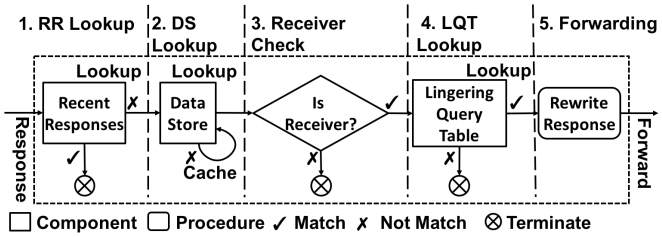

Fig. 2. PDD Response Processing: When receiving a response, a node should perform 5 steps: Recent Responses Lookup, Data Store Lookup, Receiver Check, Lingering Query Table Lookup and Forwarding.

and waits for response messages carrying metadata entries to return. Each node receiving that query should reply all the metadata entries it holds back to the consumer. The consumer dynamically decides whether and when to start a new round, or terminate the data discovery if it determines that almost all data entries are returned.

A metadata query contains the namespace (set to `system`), data type (set to `metadata` since metadata is also a type of data), a globally unique query ID to detect redundant copies, an expiration time beyond which the query is removed, the ID of the node transmitting the query (at the current hop) for returning the response, an optional list of receiver IDs of the intended next hop receivers (when not all neighbors), and an optional set of filters that further narrow down the query based on attributes of interested metadata entries. A response contains a namespace (`system`), data type (`metadata`), an optional set of attributes corresponding to filters in the query, a random thus globally unique response ID to detect redundant copies, a list of receiver IDs of the intended next hop receivers, and metadata entries as the payload.

1) Query Processing: Figure 1 show how PDD processes incoming queries. A node first examines whether the query has been received before (**LQT Lookup**). A redundant copy should be discarded. Otherwise, the new query is inserted into the LQT. Then the node examines whether it has matching data in its Data Store (**DS Lookup**). Since metadata are requested, it creates and sends a response message that contains all its metadata entries. Next, the node examines the receiver list of the query. If it is one of the intended receivers, the node should continue to forward the query (**Receiver Check**). Before relaying the query, it updates the receiver ID list with intended next hop receivers, and changes the sender ID to that of its own (**Forwarding**).

2) Response Processing: Figure 2 show how PDD processes returning responses. A node first examines whether the response has been received before (**RR Lookup**) from other neighbors (e.g., overheard). To enable opportunistic caching, new metadata entries in the response will be added to the node's data store (**DS Lookup**). Then the node examines the receiver list of the response to see whether it should relay the response (**Receiver Check**). This ensures only nodes on the right path (e.g., reverse)

will relay the response. Before relaying the response, it finds unexpired matching lingering queries in LQT (**LQT Lookup**), and sets the Receiver IDs to neighbors who transmitted these queries, then sends the message (**Forwarding**).

3) Multi-round Discovery: Depending on network quality and node mobility, messages can get lost and nodes can be temporary disconnected. Thus the above single round method may fail to discover some data items. PDD adopts a multi-round discovery algorithm to obtain as many metadata entries as possible.

The consumer makes two decisions: when the current round is finished, and whether to start the next round. Upon each response, it computes the ratio of number of responses received within a recent time window $\mathtt{T}$ to that since sending the query. If the ratio is less than a threshold $\mathtt{T}_r$, the current round is considered (almost) finished. As time goes, less and less responses return. Thus the rule detects the "diminishing" of this trend. It then computes the proportion of new metadata entries received in this round compared to all received, including previous rounds. If the proportion is greater than a threshold $\mathtt{T}_d$, showing many new entries are received in the current round and more might be out there, the consumer starts a new round.

D. Peer Data Retrieval (PDR)

After nearby data are discovered, a consumer can retrieve interested data items. Data items are usually much larger than metadata, and might be divided into many chunks and distributed carried by different nodes. PDR has two phases: ***chunk distribution information (CDI) retrieval*** and ***chunk retrieval***. In phase 1, the consumer requests the large data item's CDI, which describes where the nearest copy of each chunk can be found. The CDI is built on demand by propagating a query in the network and soliciting responses. In phase 2 the consumer requests and retrieves each chunk from its nearest provider.

1) Chunk Distribution Information Retrieval: When chunk distribution information does not exist or outdated, CDI retrieval is conducted in manner similar to PDD. We focus on the differences: the query specifies namespace "`system`", data type "`cdi`" and "`descriptor`" whose value is the requested data item's metadata, which includes possibly its unique name. A node creates a response if its Data Store (DS) has chunks or unexpired CDI entries of the requested data item. An entry contains a ***chunk id***, a ***hop count*** to the nearest chunk copy, and a ***neighbor id*** via which the copy can be retrieved. When a chunk can be retrieved with the same least hop count via multiple neighbors, a CDI entry is created for each neighbor. If a node does not have the chunk in its Data Store, the respective CDI entry is removed after an expiration time.

A CDI response has namespace "`system`", data type "`cdi`", the same "`descriptor`," and a list of ChunkId-HopCount pairs each indicating which chunk can be retrieved at the specified hop count from the transmitting node. Upon a response, a node creates a new CDI entry for each received ChunkId-HopCount pair, with ***hop count*** $= HopCount + 1$, and ***neighbor id*** set to the transmitting neighbor. Responses will return to the consumer along reverse paths of query propagation. Eventually CDI entries are created on demand at each node, indicating which neighbors have the shortest paths to which chunks.

2) Recursive Chunk Retrieval: Since one large data item may have many chunks, the consumer sends multiple chunk queries, each requesting a subset of the chunks and directed at a different neighbor closest to those chunks. A node receiving a chunk query will reply requested chunks that it holds, and further divides the subset of remaining chunks into multiple sub-queries, each directed at a different neighbor. This recursive query division allows simultaneous requests of different chunks from different (and nearest) neighbors, both aggregating the bandwidth and reducing latency. Given CDI entries, each chunk should always be retrieved from the neighbor with the least hop count. When multiple such neighbors exist for one chunk, any one is fine.

III. Adaption to Multi-Hop Wi-Fi Ad Hoc Network

Although PDS are designed to be able to leverage different underlying network and link technologies, certain proper adaption for each technology is necessary in order to achieve reasonable performance. In this section, we present how PDS is adapted to multi-hop Wi-Fi Ad Hoc Network as an example. For simplicity in enabling overhearing, all messages are sent by UDP broadcast.

As we will show in Section IV, the above baseline design suffers high data loss due to contentions in wireless broadcasts because mechanisms (e.g., RTS/CTS, ack/retransmission) for robust unicast are absent.

We apply three techniques and compare how they can improve PDS performance: random back off, acknowledgement and retransmission, redundancy detection. They can be applied individually or in combination.

A. Random Back Off.

In random back off, a node waits for a certain back off time before sending the message. The back off time is randomly picked within the range from 0 to `MaxBackoffTime`. The back off separates transmission attempts of neighboring nodes, so as to reduce the chance of simultaneous transmission and thus collision. The cost is increased latency, which we will evaluate in Section IV.

B. Ack/Retransmission.

Ack/retransmission is used for messages with receivers specified and within one hop only. Because queries are flooded, a node can hear multiple broadcasts of the query from neighbors. The chance of losing all of them is small. After sending a response, a node waits for the ack from intended receivers. A receiver should send back an ack, including the ID of the response and its own ID, so that the sender knows which receiver has received which response. Upon a `RetrTimeout`, if the sender has not received acks from all intended receivers, it broadcasts the response again with receiver IDs from those not yet ack'd only. This is repeated until a `MaxRetrTime` threshold is reached.

C. Redundancy Detection

To avoid receiving redundant entries, the consumer applies a redundancy detection technique. It appends to the query a Bloom filter including entries already received. [1] Bloom

[1] We have compared histogram, wavelet [9] and Bloom filter and found Bloom filter has the highest compression ratio for discrete and unrelated individual items like metadata entries. We do not elaborate due to space limit.

filter [10] is a space-efficient data structure representing a set of elements and widely used to test whether a given element is in the set. Upon receiving a query, the Bloom filter is cached together with the lingering query.

Nodes *rewrite* response and query messages en-route to avoid redundancy. When sending back or relaying a response, a node should test each metadata entry against the Bloom filter in the matching query. It should send back only those not included in the Bloom filter (thus not yet received by the consumer); it also inserts them in the Bloom filter of the lingering query in LQT, so such entries from other nodes will not be relayed again. When propagating a query, the node should rewrite the query by inserting into its Bloom filter new entries that it just sent in response. It updates the query in LQT and forwards the updated one. Thus downstream nodes will not send the same entries.

We also tried erasure coding [11] that can tolerate the loss of up to m out of $k+m$ data blocks. However, we find it has little effect: it chops the response into k and then adding m more encoded blocks, thus increasing the transmission attempts and collision chances multiple folds.

IV. Performance Evaluation

A. Methodology

We implement PDS in NS-3 [12], a widely used network simulator that can accurately mimic lower layer stack behavior such as 802.11 MAC. Nodes are configured with 802.11b protocols and 1Mbps broadcast data rate. We distribute 100 nodes as a 10 by 10 grid at proper neighboring distances such that each node can communicate directly with its 8 surrounding neighbors. For experiments with only one consumer, it is at the center of the grid. For multi consumer scenarios, the consumers are randomly selected from a 5 by 5 sub-grid at the center. The size of each metadata entry is 30 bytes, enough to cover the most common data type, time and location attributes.

We use several metrics to quantify the performance of PDS: ***recall*** is the fraction of distinct metadata entries received by the consumer, ***latency*** is the time from the consumer sending the query to the arrival of the last returned metadata entry, and ***p%-latency*** is the time till receiving p% of all returned metadata entries; ***message overhead*** is the number of bytes of all messages. We distribute metadata entries among all nodes uniform randomly at the beginning of simulation.

We prepend letter 'B', 'A' and 'R' to PDS to indicate whether random back off, ack/retransmission and redundancy detection is used respectively. Moreover, we use letter 'S' to indicate single round PDS and 'M' for multi round. E.g., SBA-PDS is for single round PDS with both random back off and ack/retransmission.

Several factors impact the performance: ***metadata amount*** is the number of different metadata entries; ***total metadata amount*** is that of all entries including redundant copies; ***redundancy*** is the number of copies of each entry. We also evaluate how parameters in different variants (e.g., `MaxBackoffTime`, `MaxRetrTime`, `RetrTimeout`, T_d) affect their performance. Unless specified, results are averaged over 5 runs.

B. Single Round Pervasive Data Discovery (S-PDD)

First we study how the metadata amount and redundancy impact the performance of the baseline single round pervasive data discovery (S-PDD). Figure 3 shows that the recall decreases in general when data amount increases. We find a saturation point of around 10,000 total metadata amount, beyond which the recall becomes much lower. E.g., with one copy, the recall remains around 0.4, then starts to decrease obviously beyond 10,000 distinct entries; with two copies, it remains around 0.55 before 5,000 distinct entries (10,000 total entries). Similar observations can be made from other curves. The latency increases as more distinct entries or redundant copies exist since more time is needed to deliver them, also it plateaus around 1.5-2s beyond 10,000 entries, and the message overhead increases almost linearly with more entries or copies. Those figures are omitted due to the space limitation.

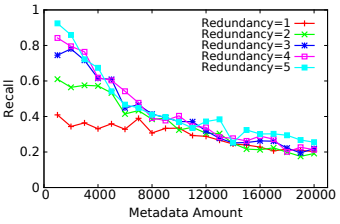

Fig. 3. Recall of S-PDD. It is pretty low (less than 0.5) in most cases. The saturation point is around 10,000 total metadata amount.

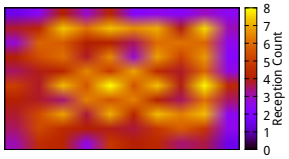

Fig. 4. Heatmap of reception count of S-PDD where roughly half of the neighbors lose the message everywhere.

To verify our conjecture that heavy contention caused the poor recall, we draw a heatmap (Figure 4) showing the *reception count* of how many neighbors successfully receive a message broadcast from each node, with 5,000 metadata amount and 1 redundancy. Most nodes (except those on edge) have 8 neighbors. Thus a reception count of close to 8 is desired. We can see that most nodes have the count around 4, meaning roughly half of the neighbors lose the message. Thus at each hop, the neighbor on the reverse path has about 50% chance of losing the message, which explains the poor recall.

Next we study how to achieve high recall while maintaining reasonable latency and overhead. We use 5,000 distinct entries as the normal load and those beyond 10,000 for stress tests. Redundancy is always set to 1 since each entry initially has only copy one on its original producer.

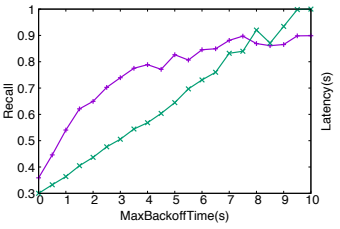

Fig. 5. Recall of SB-PDD keeps growing until `MaxBackoffTime` reaches 5s. Latency of SB-PDD keeps growing linearly.

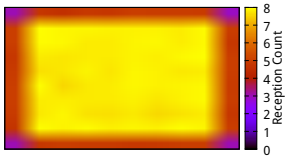

Fig. 6. Heatmap of reception count of SB-PDD with nearly perfect reception count everywhere.

1) SB-PDD: Random Back Off: We increase `MaxBackoffTime` and evaluate how it affects the performance. Figure 5 shows steady recall improvement to 0.8 until the `MaxBackoffTime` reaches 5s, beyond which the recall increases marginally. The latency increases linearly with larger `MaxBackoffTime`. At `MaxBackoffTime` of 5s, the latency is around 30s, much higher than S-PDD ($\sim 2s$). Given the average back off time of 2.5s, the roughly 10-hop back and forth path between the consumer and a farthest node requires 25s. The message overhead (figure omitted) is about 3 times of that of S-PDD (1.57MB vs. 0.42MB at 5,000 entries),

part of which is simply because that messages travel longer hops and reach the consumer with much higher chances.

We also examine the heatmap (Figure 6) and finds that most nodes (except those on edge) have 8 or close to 8 reception counts. This shows that random back off is effective to space transmissions among neighbors such that the contentions become very few. Overall SB-PDD increases recall but at the cost of much higher latency.

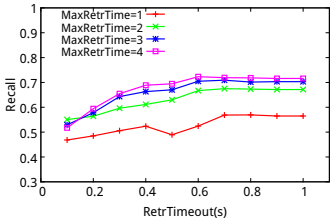

Fig. 7. Recall of SA-PDD improves as `RetrTimeout` and `MaxRetrTime` increase.

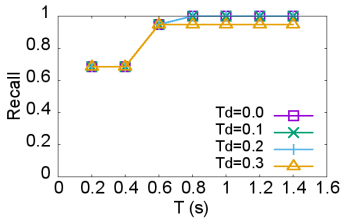

Fig. 8. Recall of M-PDD becomes stable when `T` reaches 1.0s.

2) SA-PDD: Ack/Retransmission: We evaluate the impact of `RetrTimeout` and `MaxRetrTime` on the performance of SA-PDD. Figure 7 shows that as `RetrTimeout` or `MaxRetrTime` increases, recall improves. This is because longer `RetrTimeout` allows more time for ack to return. Thus the sender does not prematurely retransmit and this reduces contention. More retries directly improve the chances of reception. However, the benefits have diminishing returns: beyond 0.6s `RetrTimeout` or 4 `MaxRetrTime` the gain is marginal.

From the above, we decide to use 0.6s `RetrTimeout` and 4 `MaxRetrTime` as the best combination (giving 0.7 recall). SA-SPPS achieves slightly less recall (72.3% vs. 82.7%) but significantly less latency (2.89s vs. 35.13s) and message overhead (0.69MB vs. 1.57MB) comparing to SB-PDD.

3) SBA-PDD: Random Back Off and Ack/Retransmission: We perform extensive evaluation on all combinations of random back off, ack/retransmission and erasure coding. We find that mechanisms enabling erasure coding always have worse performance (lower recall, larger latency and message overhead) compared to those with the same configuration only disabling erasure coding. Among all the combinations, SBA-PDD achieves the best performance with 96.5% recall, 33.3s latency and 2.18MB message overhead.

We conduct stress test on SBA-PDD to evaluate its robustness to network saturation. When metadata amount increases from 5,000 to 15,000/20,000 (50%/100% beyond saturation), recall of SBA-PDD drops from 96.5% to 83.8%/66.5%. The latency stays roughly the same (27.5s/29.9s vs. 28.8s). However, message overhead grows up to 4/8 times (9.17MB/16.85MB vs. 2.18MB) despite metadata amount only grows to 3/4 times. It shows that SBA-PDD is not very robust against saturation.

C. Multi-round Pervasive Data Discovery (M-PDD)

We perform extensive evaluation with different values of the three parameters that affect the number of rounds and their durations, and find that $T_d = T_r = 0$, and $T = 2s$ give the best performance. Using the above parameters, we evaluate all combinations of random back off, ack/retransmission and redundancy detection on baseline M-PDD (Table I). Due to multi-round, many of them achieve recalls over 90%, some

TABLE I
M-PDD NORMAL TEST

-PDD	MR	MBR	MAR	MBAR
Recall (%)	100.0	88.5	**100.0**	99.2
Latency (s)	18.7	37.2	**11.7**	32.7
95%-latency (s)	14.4	29.7	**8.9**	27.4
Message Overhead (MB)	9.10	2.80	**5.56**	2.72

TABLE II
MAR-PDD STRESS TEST

Data Amount	5,000	10,000	15,000	20,000
Recall (%)	100.0	100.0	100.0	100.0
Latency (s)	10.9	18.1	21.9	25.5
Message Overhead (MB)	5.05	9.76	15.83	22.79

even achieve 100%. Among all variants, MAR-PDD achieves the highest recall (100%) and the smallest latency (11.7s), at a lower overhead (5.56MB). It also beats the performance of the best single round variant SBA-PDD (96.5% recall and 36.8s latency).

We also find that 95%-latency for MAR-PDD is only 8.9s, indicating that among all metadata entries collected 95% of them are received within the first 76% time. Similar observations can be made in other variants. This means applications can probably start whatever they need to do next with most of returned metadata entries at 70%-80% latency time, without needing to wait for the long tail of the last 5% entries.

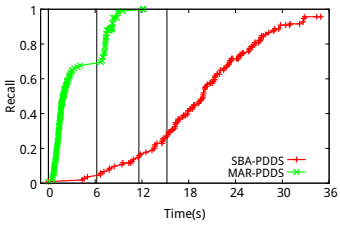

Fig. 9. Recall of SBA-PDD and MAR-PDD change over time with 5,000 metadata amount. The starting point of each round of MAR-PDD are indicated with black vertical lines.

Fig. 10. Latency and message overhead of PDR with multiple consumers.

To understand why MAR-PDD beats SBA-PDD on both recall and latency, we plot how recall changes over time for both of them with 5,000 metadata amount, and mark the beginning of each round for MAR-PDD with a vertical line (Figure 9). We find that although MAR-PDD achieves less recall in the first round (0.7 vs. 0.95), it has much shorter first round (2.5s vs. 35s) because there is no back off. Thus it quickly starts subsequent rounds, getting over 95% and finally 100% in the second and third rounds. Its 4 rounds combined take less than a third of the latency of SBA-PDD.

MAR-PDD achieves high recall because: 1) There are less and less metadata entries to collect in subsequent rounds. Redundancy detection filters out already received entries. 2) Metadata entries lost in previous rounds leave cached copies along the return path, and more copies are created progressively closer to the consumer. Thus it takes much less hops to retrieve them.

Table II shows the stress test results on MAR-PDD. As metadata amount increases from 5,000 to 20,000, recall remains at 100%, with sub-linear latency and roughly linear message

TABLE III
SEQUENTIAL MULTIPLE CONSUMER TEST

-th Consumer	1	2	3	4	5
Recall (%)	100.0	100.0	99.6	100.0	100.0
Latency (s)	9.0	10.5	7.1	4.5	0.2
Message Overhead (MB)	5.05	4.71	6.07	6.96	0.002

overhead increases. This shows the robustness of MAR-PDD against network saturation.

D. Multiple Consumers

We evaluate MAR-PDD when there are multiple consumers sending queries sequentially. The intervals between consecutive consumers' requests are set large enough for the previous data discovery to be completed. Table III shows the results. We find that all consumers achieve nearly 100% recall, and latency becomes smaller for later consumers: 9-11s for the first two, 7.1s, 4.5s for the third and fourth. This is due to the overhearing and caching: more redundant copies are created and closer to consumers. So a consumer has more cached entries and needs to collect less entries from closer copies. The last one takes only 0.2s because it has already cached more than 95% entries even before sending its own query.

E. PDR Performance

We also evaluate the performance of PDR when there are multiple consumers. Recall in our experiments is always 100%. Figure 10 shows that, from the 1st consumer to the 5th, latency of sequential consumers decreases from 46.1s to 38.1s, while message overhead decreases from 54.22MB to 23.11MB. The significant overhead drop is because more copies of chunks are cached during previous retrieval, thus the average hop each chunk is transmitted become much smaller. Closer chunk distance also decreases latency. Chunks from different directions eventually have to wait for the consumer to receive, thus the drop in latency is somewhat limited.

V. RELATED WORK

PDS differs from existing data discovery and sharing work in mobile ad hoc networks [2]–[4]. They are mostly designed for traditional endpoint based networks, where data are bond to specific nodes with certain network address. Existing endpoint based ad hoc routing protocols [13], [14] focus on finding one path to a specific destination address. PDS adopts a content centric design where routing entries are for data instead of addresses. Data are cached opportunistically by any capable and willing nodes. Thus consumers do not need to know or care at which addresses the data exist, as long as existing data are discovered and at least one copy (probably the nearest one) is retrieved.

Information centric networks [5], [6] have been studied extensively. PDS shares similar query-response processing to Content Centric Network (CCN) [5] and Named Data Network (NDN) [6]. Due to differences in wireless medium, network scale, PDS differs from them in important aspects: 1) Both CCN and NDN are initially intended for wired networks, whereas PDS leverages the broadcast wireless medium to reduce message overheads and enable opportunistic overhearing. 2) Bandwidth is a scarce resource in shared wireless medium. In CCN/NDN, each Interest is removed upon the return of any matching Data, and Interest/Data are delivered as-is. While PDS uses lingering queries each can guide the return of many response messages to avoid repeating a query many times.

VI. DISUCSSION AND FUTURE WORKS

Caching and reusing of overheard data is one of the key advantages of PDS' content centric design. In our current implementation, we assume devices have enough space to cache everything they overheard. However, in reality storage space of devices is limited. Thus proper caching strategy are needed to decide which data item or chunk should be cached/replaced, which becomes one of the future works of this paper.

In this paper, we evaluate PDS with NS-3 based simulation without nodes' mobility. This is based on the observation from real world that although people are moving around from time to time, the chance of network topology changing a lot during several seconds is very low. To make this work more realistic, building smartphone based prototype and evaluate it in dynamic scenarios in the real world becomes another future work of this paper.

VII. CONCLUSIONS

In this paper, we propose content centric data discovery and retrieval among peer edge devices, which is fundamental to many novel applications where opportunistically congregated devices need to share each other's sensing data. We demonstrate how our proposal can leverage different underlying network and link technologies by adapting it to Wi-Fi ad hoc network. Evaluations show almost 100% data retrieval in short time under multiple consumers and real world mobile scenarios.

REFERENCES

[1] R. K. Ganti, F. Ye, and H. Lei, "Mobile crowdsensing: current state and future challenges," *Communications Magazine, IEEE*, vol. 49, no. 11, pp. 32–39, 2011.

[2] A. N. Mian, R. Baldoni, and R. Beraldi, "A survey of service discovery protocols in multihop mobile ad hoc networks," *IEEE Pervasive computing*, vol. 8, no. 1, pp. 66–74, 2009.

[3] F. Sailhan and V. Issarny, "Scalable service discovery for manet," in *Third IEEE International Conference on Pervasive Computing and Communications*. IEEE, 2005, pp. 235–244.

[4] G. Ding and B. Bhargava, "Peer-to-peer file-sharing over mobile ad hoc networks," in *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*. IEEE, 2004, pp. 104–108.

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

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

[7] "Wi-Fi Direct," http://www.wi-fi.org/discover-wi-fi/wi-fi-direct.

[8] A. Asadi, Q. Wang, and V. Mancuso, "A survey on device-to-device communication in cellular networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 1801–1819, 2014.

[9] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine, "Synopses for massive data: Samples, histograms, wavelets, sketches," *Foundations and Trends in Databases*, vol. 4, no. 1–3, pp. 1–294, 2012.

[10] B. Chazelle, J. Kilian, R. Rubinfeld, and A. Tal, "The bloomier filter: an efficient data structure for static support lookup tables," in *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2004, pp. 30–39.

[11] H. Weatherspoon and J. D. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in *Peer-to-Peer Systems*. Springer, 2002, pp. 328–337.

[12] "NS-3," https://www.nsnam.org/.

[13] "RFC 3561: Ad hoc On-Demand Distance Vector (AODV) Routing," http://www.rfc-base.org/rfc-3561.html.

[14] "RFC 4728: The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4," http://www.rfc-base.org/rfc-4728.html.