Data Acquisition for Real-time Decision-making under Freshness Constraints

Shaohan Hu*, Shuochao Yao*, Haiming Jin*, Yiran Zhao*, Yitao Hu†, Xiaochen Liu†,
Nooreddin Naghibolhosseini‡, Shen Li*, Akash Kapoor*, William Dron§, Lu Su¶,
Amotz Bar-Noy‡, Pedro Szekely†, Ramesh Govindan†, Reginald Hobbs‖, Tarek F. Abdelzaher*
*University of Illinois at Urbana-Champaign, †University of Southern California, ‡City University of New York,
§Raytheon BBN Technologies, ¶State University of New York at Buffalo, ‖US Army Research Laboratory
Email: {shu17,syao9,hjin8,zhao97,shenli3,akapoor5,zaher}@illinois.edu, {yitaoh,liu851,ramesh}@usc.edu, pszekely@isi.edu,
{nnaghibolhosseini@gc,amotz@sci.brooklyn}.cuny.edu, wdron@bbn.com, lusu@buffalo.edu, reginald.l.hobbs2.civ@mail.mil

***Abstract*—The paper describes a novel algorithm for timely sensor data retrieval in resource-poor environments under freshness constraints. Consider a civil unrest, national security, or disaster management scenario, where a dynamic situation evolves and a decision-maker must decide on a course of action in view of latest data. Since the situation changes, so is the best course of action. The scenario offers two interesting constraints. First, one should be able to successfully compute the course of action within some appropriate time window, which we call the *decision deadline*. Second, at the time the course of action is computed, the data it is based on must be fresh (i.e., within some corresponding validity interval). We call it the *freshness constraint*. These constraints create an interesting novel problem of timely data retrieval. We address this problem in resource-scarce environments, where network resource limitations require that data objects (e.g., pictures and other sensor measurements pertinent to the decision) generally remain at the sources. Hence, one must decide on (i) which objects to retrieve and (ii) in what order, such that the cost of deciding on a valid course of action is minimized while meeting data freshness and decision deadline constraints. Such an algorithm is reported in this paper. The algorithm is shown in simulation to reduce the cost of data retrieval compared to a host of baselines that consider time or resource constraints. It is applied in the context of minimizing cost of finding unobstructed routes between specified locations in a disaster zone by retrieving data on the health of individual route segments.**

I. Introduction

This paper addresses the problem of timely data retrieval in resource-poor environments under data freshness constraints. The problem is motivated by disaster response and tactical mission scenarios, where network resources are severely limited. We assume that a decision maker must make decisions on specific courses of action among multiple alternatives. Each alternative requires that multiple conditions be satisfied for this alternative to be a valid choice under the circumstances. Hence, the decision maker must query sources and other sensors to understand the current conditions and make each decision accordingly. Had resources been sufficient, this would have been trivial. However, in resource poor environment, retrieving the requisite data to evaluate a condition takes time and consumes some amount of resources (i.e., has cost). Moreover, conditions change dynamically. Hence, if a decision takes too long, conditions may change, calling for a re-evaluation of the course of action, thereby further delaying the decision. The problem addressed is one of making timely decisions at minimum cost while satisfying freshness constraints of the underlying data.

To give a concrete example, consider a disaster management scenario where an active threat is spreading, while members of a rescue team must find their way to a set of destinations that need help. The team leader must decide on a valid route to each destination such that help can be sent along. Making the wrong decision will cause back-tracking and delay, which may jeopardize mission success. The set of alternative routes to a given destination constitutes the set of alternative courses of action that the corresponding decision must choose from. For a given route to be valid, each segment on the route must be obstruction-free. This creates multiple conditions (one per segment) that must be jointly satisfied. However, the state of various road segments is not always available or not always fresh. Just because a segment was available an hour ago does not mean it is available now. In a weather-related disaster, a tree might fall across it, it might get flooded, or a major accident might render it blocked. Hence, a sufficiently recent condition for each segment must be retrieved. This requirement gives rise to data freshness constraints. Different segments might have different freshness requirements. For example, the state of segments that are far enough from the threat might change more slowly than that of segments in harm's way. Also, once a segment becomes unavailable, its state persists for a duration that depends on the type of damage incurred. Car accidents might clear in hours, but a collapsed building may block a road for days. The rescue team in question may be sharing scarce network resources with other teams. Hence, data pertinent to the aforementioned decisions must be retrieved at minimum cost.

The paper offers a general formulation to the above problem. An algorithm is developed that minimizes the cost of decisions while reducing the chance of timeouts and ensuring the freshness of data. The algorithms is shown in simulation to outperform several baselines for data retrieval such as retrieving the lowest cost object first.

The novelty of this work lies in exploiting the *structure of the decision* to significantly reduce data retrieval cost. There are two aspects to the exploited decision structure. First, the decision considers alternative courses of action. Hence, the algorithm has a degree of freedom in deciding which alternative to evaluate first. For example, an alternative that is more likely to be valid (and can be evaluated at a lower cost) is a good start. Second, evaluating an alternative entails retrieval and evaluation of multiple conditions that must all be satisfied for the alternative to be deemed a valid course of action. The algorithm again has a choice, deciding which condition to evaluate first. In particular, evaluating the most risky condition first (i.e., the one most likely not to be satisfied) might be better

1052-8725/15 $31.00 © 2015 IEEE
DOI 10.1109/RTSS.2015.25

in that it minimizes effort wasted on evaluating ultimately invalid alternatives. The above considerations must be balanced against freshness requirements. Specifically, retrieving data objects with short freshness intervals first is a bad idea because by the time all other pieces are retrieved, the freshness intervals of these objects might expire, forcing repeat retrieval. Instead, it is better to retrieve data objects with the longest freshness interval first. The above insights give rise to an optimization problem solved and evaluated in this paper, demonstrating promising results.

The rest of this paper is organized as follows. We give overall problem description and system design in Section II, and then present detailed analyses and algorithms in Section III. In Section IV we describe our implementation of an end-to-end route planning system. Evaluation of our proposed algorithms and system is presented in Section V. We survey related work in Section VI, and finally conclude in Section VII.

II. Overview

We aim to design a sensing system specifically tailored for resource constrained dynamic environments, such as disaster response and recovery. Under resource limitations (e.g., network bandwidth, node battery power, etc), data stays at sources, not to needlessly use the network. Only upon explicit requests do sensors take/share measurements of the environments and transmit back the sensory data or results. Under such settings, retrieval of sensor data from sources needs to be carried out with care such that the minimum amount of resources is consumed.

Adding to the complexity of the problem, the environment that is of interest is dynamic at different timescales. For example, in a politically unstable region, every now and then roads are taken up by protesters or rioters, preventing safe passage. As the crowds move around, the traffic blockage situation evolves with time; road information obtained that is old may no longer reflect the actual states of the environment. As another example, an earthquake and its aftershocks can affect a region differently under different situations: If a segment of a road gets blocked because it is flooded due to damage to its sewage system, then perhaps in a day or two it will get repaired and return to functional status. If a bridge over a river collapses during the earthquake, then it might take months before it can be repaired or an new one constructed. Therefore, data retrieved from remote sources may have very different *freshness* characteristics. In carrying out sensing tasks to help with decision making, we need to take into consideration such data freshness characteristics in order to avoid reaching invalid or inconsistent decisions made from partially stale sensor data.

The goal is then to fetch data from sources in a way that minimizes system resource consumption while reaching answers using data items all within their freshness intervals. In general, a decision-maker's information need is formed into a query, which is then translated into requirements for a set of relevant data objects. A retrieved object can be subjected to a test that evaluates a condition on the object. For example, an audio clip from an road acoustic sensor might be used to determine if there is currently a convoy passing by; an image of the inside of an emergency shelter can be inspected to determine if the occupancy is reaching its limit. The goal of the decision-maker is to find the best course of action. Each possible course of action requires multiple conditions to be evaluated to determine if this course of action is valid. It is our objective to compute a *good* data retrieval plan that makes efficient use of network resources to help choose the right course of action from sufficiently fresh data.

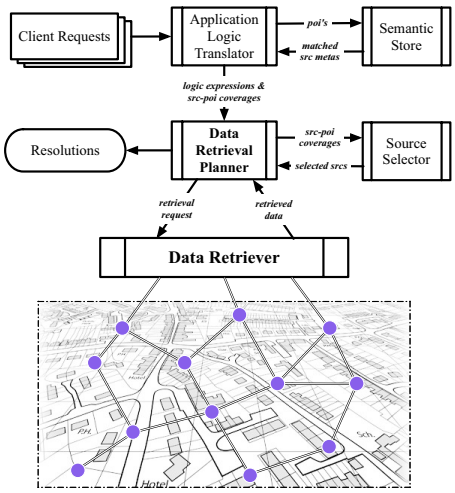


Fig. 1: System Design

We use network bandwidth consumption (i.e., the size in bytes of the retrieved data) as the cost measure for supporting the decision-making task. We do, however, want to point out that the algorithms developed in our paper can be used on any cost metric definition as long as it is additive. For example, in a scenario where energy is the critical resource, we can easily redefine the cost metric to optimize energy consumption.

Fig. 1 shows the architecture of our resource constrained sensing system. Requests from decision makers upon arrival are converted and encoded to boolean logic expressions, where the alternative courses of action are represented by a disjunction (OR) of terms, whereas the conditions that need to be satisfied for a course of action to be valid are represented by a conjunction of variables (AND). A directory, called semantic store is assumed to exist that knows which data source has what information. Given the menu of available information objects at different sources, a source selection module [1] is used to select a suitable source to contact for each needed object. A data retrieval planner then computes an optimal retrieval order. The latter module is the focus of this paper.

III. Problem Description & Solution

In this section, we describe the sensing problem and discuss the detailed algorithms we develop.

A. Problem Description

For ease of discussion, we introduce several key notations first. In making a decision, let $\{a_i\}$ be the set of alternative courses of action decided on, and $\{t_{i_j}\}$ the set of conditions for a particular a_i to be valid. Each condition t_{i_j} has cost c_{i_j} (e.g. data retrieval bandwidth cost), latency l_{i_j} (e.g. data retrieval delay), probability of being satisfied p_{i_j}, and freshness interval d_{i_j}, after which it needs to be reevaluated.

As previously discussed, for a decision maker a decision is made by choosing one among multiple courses of

action, each of which consists of multiple conditions that must simultaneously hold. To evaluate a condition, some data must be retrieved over the resource-poor network, and in a timely fashion due to environment dynamics, as different conditions have corresponding freshness intervals. At the time the decision is made all underlying data must be fresh, which means if the retrieval order was $O = \langle t_{i_1} t_{i_2} \dots t_{i_n} \rangle$ then the freshness interval d_{i_j} of any of the retrieved data item t_{i_j} is greater than the sum of the retrieval latencies of the data item itself and all subsequent ones in the retrieval order:

$$d_{i_j} > \sum_{s=j}^{n} l_{i_s}.$$

Given this setting our goal is then to minimize expected decision cost.

Sequential processing is generally more cost efficient than its parallel counterpart, as it can minimize the probability of unnecessary data fetches caused by parallel retrievals, which can otherwise be avoided by evaluation short-circuiting in sequential processing. Therefore, towards the goal of minimizing the total cost, we want to plan the evaluation of conditions in a sequential fashion whenever plausible. However, this might not always be possible, for example when data items associated with an request all have rather short freshness intervals and sequential processing would always lead to old data items expiring before the whole request can be resolved. In situations as such, we would need to make our solution capable of adapting parallel processing s.t. the request *can* at least be resolved in time, because otherwise talking about cost optimization is of little value.

Furthermore, for the post-disaster scenarios we consider, oftentimes *a* valid course of action does exist for the decision making task—for example, when looking for a route for a medical team to go from location A to B after an earthquake, many or even most of the candidate routes might be blocked, but some *will* be in reasonable condition—the focal point here is that we want to find *a* positive resolution as soon as possible with minimal resource consumption. Having said that, we make the following observations. i) Making a choice among multiple courses of action is accomplished by finding *a* single valid one, and ii) Once we've started verifying the validity of a particular course of action, there is no reason to turn to another one before we are finished with the current one (i.e., verified that either all its conditions hold or at least one fails) under the goal of minimizing cost.

The above discussions serve as a guideline of how we go about solving our problem at hand. In order to devise an actual solution, we need to answer the following questions,

1) Among all courses of action, which one should we examine first,
2) For verifying the validity of a particular course of action, how should we plan the evaluation of its conditions, and
3) If sequential processing does not suffice, how should we employ parallel processing to try to avoid causing freshness interval violations.

Before diving into the detailed algorithm that answers the above questions, we give a rough solution sketch here: We first pick the course of action that has the highest valid probability per unit (expected) cost. We then schedule its conditions in an EDF-inspired sequential retrieval order. If verifying them all in that order does not lead to freshness constraint violation, we rearrage the order to try to decrease the expected cost; otherwise, we increase the level of parallel retrievals in order to try to avoid the freshness deadline violation. If this course of action is valid, we are done; otherwise we move onto next best course of action in the request.

B. Algorithms

As our ultimate goal is to minimize the excepted cost of carrying out a sensing task, we first need to rank all the alternatives (courses of action) according to their cost effectivenesses, computed as the validity probability per unit cost. This should feel natural as, for example, the more probable a candidate route is in good condition and the lower the cost is for verifying its condition, the sooner we should examine it for a route finding task.

Prior work [2], [3] from the theory community have shown that for a particular action a_i, the most cost effective processing order is then computed by always picking first the condition with the highest probability of short-circuiting its siblings per unit cost. If we use $t_{i_u} \succ_t t_{i_v}$ to denote that condition t_{i_u} should precede t_{i_v} in the optimal order, we have

$$t_{i_u} \succ_t t_{i_v} \Leftrightarrow \frac{1 - p_{i_u}}{c_{i_u}} > \frac{1 - p_{i_v}}{c_{i_v}}.$$

With the order $\langle t_{i_{o0}}, t_{i_{o1}}, t_{i_{o2}}, \dots \rangle$, we can compute a_i's expected cost as

$$c_i = c_{i_{o0}} + p_{i_{o0}}(c_{i_{o1}} + p_{i_{o1}}(c_{i_{o2}} + p_{i_{o2}}(\dots))),$$

and its validity probability as

$$p_i = \prod_j p_{i_j}.$$

Therefore, the order of the courses of action is constructed as

$$a_i \succ_a a_j \Leftrightarrow \frac{p_i}{c_i} > \frac{p_j}{c_j},$$

similar to the order of conditions, with the only difference of the short-circuiting probability being the validity probability, as opposed to the failure probability.

Having established the order $\langle a_{o0}, a_{o1}, a_{o2}, \dots \rangle$, we can then start processing the request by following that order. For a particular action a_i, we want to find out its validity status with the least cost, but at the same time without violating any of its component conditions' freshness constraints. Thus we proceed as follows. Inspired by EDF, we first order the a_i's conditions according to their freshness intervals, latest first

$$t_{i_u} \succ_d t_{i_v} \Leftrightarrow d_{i_u} > d_{i_v},$$

which we call a_i's *Latest Deadline First* (*LDF*) order.

Theorem 1. *If LDF order cannot avoid data freshness violation, no sequential order can.*

Proof: We use proof by contradiction. For retrieving all conditions for some action a_i, let's suppose its LDF order

$$O = \langle t_{i_1} t_{i_2} \dots t_{i_n} \rangle$$

causes data freshness constraint violation(s), i.e., for some condition t_{i_f}, its freshness interval is shorter than the sum

of the retrieval delays of itself and of all subsequent ones in the LDF order,

$$d_{i_f} < \sum_{k=f}^{n} l_{i_k}. \tag{1}$$

We assume, for contradiction, that there exists a different (from LDF) sequential order O' that causes no data freshness constraint violations. So in O', t_{i_f} must *not* still be the front of all elements of the set $S_f = \{t_{i_k} | f \leq k \leq n\}$, which means some other $t_{i_s} \in S_f$ must be the new front of S_f in order O'. Since t_{i_s} is behind t_{i_f} in O, so we have

$$d_{i_s} < d_{i_f}. \tag{2}$$

Chaining the Inequalities (1) and (2) together, we have

$$d_{i_s} < \sum_{k=f}^{n} l_{i_k},$$

which means that test t_{i_s} will miss its freshness deadline in O'. Contradiction reached. ■

Now, assuming we were to retrieve *all* tests' data items of this action by following the LDF order, we can check to see if any freshness constraint violation would've occurred by the end of the retrievals. If not, it means this course of action *can* be checked by sequential processing, and we might be able to further decrease its expected cost; otherwise, it *cannot* be checked by sequentially retrieving its data items, and we will need to add parallel processing in order to decrease the total retrieval latency and make the action resolvable without violating any data item's freshness constraints. We next discuss each of the two cases in detail.

Algorithm 1 When a_i's LDF order already satisfies freshness constraints, rearrange the LDF order to minimize the resolution cost

Input: Action a_i's conditions $\{t_{i_j}\}$, the corresponding costs $\{c_{i_j}\}$, retrieval latencies $\{l_{i_j}\}$, probabilities of being true $\{p_{i_j}\}$, and freshness intervals $\{d_{i_j}\}$
Output: The retrieval order

$Q_c \leftarrow \emptyset$, $Q_d \leftarrow$ LDF order
$L \leftarrow \{t_{i_j}\}$ sorted in descending order of $\frac{1-p_{i_j}}{c_{i_j}}$
while $Q_d \neq \emptyset$ **do**
  for t_l in L **do**
    $Q_H \leftarrow Q_c + \langle t_l \rangle + Q_d \setminus \langle t_l \rangle$
    if Q_H meets freshness constraints **then**
      $Q_d \leftarrow Q_d \setminus \langle t_{i_j} \rangle$
      $Q_c \leftarrow Q_c + \langle t_{i_j} \rangle$
      break
    end if
  end for
end while
return Q_c

1) Cost Minimization: If a_i's LDF retrieval order does not violate any test t_{i_j}'s freshness constraint, we can possibly rearrange it in order to decrease the expected resolution cost. Note that if a_i is valid (i.e. all its conditions are true), we wouldn't be able to decrease the cost, as all data items do need to be retrieved for verifications. However, if it turns out that a_i actually is invalid (i.e., one or more of its conditions are false), then we want to detect this failure with as little cost as possible. Therefore, the rearrangement proceeds as follows: We initialize $Q_d =$ LDF order and $Q_c = \emptyset$. Then we compute the order L for all of a_i's tests according to their per unit cost failure probabilities $\frac{1-p_{i_j}}{c_{i_j}}$ (in descending order). We then pick L's first condition $t_{i_j} \notin Q_c$ and check if the hypothetical order

$$Q_c + \langle t_{i_j} \rangle + Q_d \setminus \langle t_{i_j} \rangle$$

would violate any freshness constraint (where $+$ denotes concatenation). If not, we remove t_{i_j} from the LDF order Q_d

$$Q_d = Q_d \setminus \langle t_{i_j} \rangle,$$

and append it to the end of Q_c

$$Q_c = Q_c + \langle t_{i_j} \rangle.$$

We terminate when $Q_d == \emptyset$. This whole process essentially tries to make failures appear as soon as possible (in the sense that least amount of cost has been incurred) if there *is* a failure, without violating any freshness deadline. The algorithm pseudo-code is shown in Alg. 1.

The computational complexity is dominated by the nested loops and the check for freshness constraint violations for each test, of order $O(n^3)$ where $n = |a_i|$.

Algorithm 2 When an action's LDF order does not meet freshness constraint, transform the LDF order to add parallel retrievals in order to eliminate constraint violations

Input: Action a_i's conditions $\{t_{i_j}\}$, the corresponding costs $\{c_{i_j}\}$, retrieval latencies $\{l_{i_j}\}$, probabilities of being true $\{p_{i_j}\}$, and freshness intervals $\{d_{i_j}\}$
Output: The retrieval order

$Q_d \leftarrow$ LDF order, $S_p \leftarrow \emptyset$
while $|Q_d| > 0$ **do**
  $t_e \leftarrow$ end element of Q_d
  $Q_d \leftarrow Q_d \setminus \langle t_e \rangle$
  $S_p \leftarrow S_p \cup \{t_e\}$
  if $Q_d + S_p$ meets freshness constraints **then**
    return $Q_d + S_p$
  end if
end while
return NULL

2) Freshness Constraint Violation Avoidance: On the other hand, if a_i's LDF retrieval order does violate some condition t_{i_j}'s freshness constraint, then we want to add parallelism to the sequential order to try to eliminate the violation. Keeping the parallelism at the end is beneficial in terms of preventing cost from unnecessary increase as any failed condition before the end will still be able to short-circuit the paralleled one. It is quite straightforward to carry out the transformation: We prepare queue $Q_d =$ LDF order and set $S_p = \emptyset$. We take Q_d's end condition t_{i_j} and check if the order

$$Q_d \setminus \langle t_{i_j} \rangle + S_p \cup \{t_{i_j}\}$$

would violate any freshness deadline, where all data items in the set are to be retrieved in parallel. If yes, we remove t_{i_j} from Q_d

$$Q_d = Q_d \setminus \langle t_{i_j} \rangle,$$

and add it to S_p

$$S_p = S_p \cup \{t_{i_j}\},$$

and move on to the new end item in Q_d and continue the process; otherwise, we terminate the transformation as we have successfully eliminated the freshness constraint violation. The algorithm pseudo-code is shown in Alg. 2.

The computational complexity is dominated by the for-loop and the check for freshness constraint violations, of the order $O(n^2)$ where $n = |a_i|$.

Algorithm 3 vLDF — Resolution algorithm for answering a sensing request

Input: The courses of action $\{a_i\}$, and each a_i's conditions $\{t_{i_j}\}$, the corresponding costs $\{c_{i_j}\}$, retrieval latencies $\{l_{i_j}\}$, and success probabilities $\{p_{i_j}\}$, and freshness interval $\{d_{i_j}\}$
Output: Request resolution result

$L_a \leftarrow \{a_i\}$ sorted in descending order of $\frac{p_i}{c_i}$, $\#_{fail} \leftarrow 0$
for a_i in L_a **do**
  $Q_c \leftarrow \emptyset$, $Q_d \leftarrow$ LDF order, $S_p \leftarrow \emptyset$
  $L \leftarrow \{t_{i_j}\}$ sorted in descending order of $\frac{1-p_{i_j}}{c_{i_j}}$
  while $Q_d \neq \emptyset$ **do**
    for t_l in L **do**
      $T_d \leftarrow Q_c + Q_d$'s degree of freshness violations
      $Q_H \leftarrow Q_c + \langle t_l \rangle + Q_d \setminus \langle t_l \rangle$
      $T_H \leftarrow Q_H$'s degree of freshness violations
      if $T_H \leq T_d$ **then**
        $Q_d \leftarrow Q_d \setminus \langle t_{i_j} \rangle$, $Q_c \leftarrow Q_c + \langle t_{i_j} \rangle$
        break
      end if
    end for
  end while
  while $|Q_c| > 0$ **do**
    $t_e \leftarrow$ end element of Q_c
    $Q_c \leftarrow Q_c \setminus \langle t_e \rangle$, $S_p \leftarrow S_p \cup \{t_e\}$
    if $Q_c + S_p$ meets freshness deadlines **then**
      Proceed with resolving a_i by following $Q_c + S_p$
      if a_i succeeds **then**
        return a_i as an successful result
      else
        $\#_{fail} \leftarrow \#_{fail} + 1$
        break
      end if
    end if
  end while
end for
if $\#_{fail} == |L_a|$ **then**
  return request resolves to failure
else
  signal freshness deadline violation unavoidable
end if

Finally, in designing a solution that handles both the cost minimization and deadline violation avoidance, we combine the above two techniques into an unified algorithm. The basic idea is as follows: First we carry out cost-saving rearrangement to the LDF order as much as possible, given that i) if there was no freshness constraint violations before, we do not introduce one now, and ii) if there were, we do not worsen the violation degree (e.g., if there was a deadline miss by 3 minutes, we cannot carry out order rearrangement that increases the miss to 4 minutes). After the rearrangement, we carry out parallel transformation at the end of the retrieval order just as previously described. We collect all above discussions and present the complete sensing algorithm in Alg. 3. We call our algorithm *Variational Latest Deadline First* (vLDF).

Given our separate complexity analyses for Alg. 1 and 2, it is easy to see that our unified algorithm Alg. 3 runs in $O(mn^3)$ time, where m is the number of alternative courses of action in the request, and n the number of conditions in an action.

IV. Implementation

In order to test our system in a realistic setting, we implement a route planning system that targets post-disaster scenarios, for disaster response teams. In our normal everyday life, we can easily use Google Maps to compute routes by specifying source and destination locations. But after a natural disaster, routes returned by Google Maps or any other traditional route planning service might not suffice as roads might be blocked or damaged, and the entire environment is dynamic, with various aspects changing over time (e.g., bridges might collapse, roads might get flooded or blocked, people might setup temporary camps and move around). Therefore, verifications (e.g. visually via pictures taken of the roads) are needed to make sure if a route is in reasonable condition for vehicles to pass, and they need to be carried out in a timely fashion s.t. results do not become stale as the environment changes. The emergency networks set up by first responder teams are likely of very low bandwidth, and are shared by multiple response teams (e.g. infrastructure, medical, etc), thus it is key no single request exhausts the network resource.

Our post-disaster route planing system consists of the following components,

- Router: We use open-source router code Gosmore [4] to compute routes on Open Street Map [5] data. We modify the Gosmore code so that it computes, for specified source and destination locations, any number of candidate routes the user desires.
- Meta Store: A database that hosts the meta data (location, time, size, etc) of pictures taken by remote camera phones. Note that due to goal of minimizing network resource consumption, photos themselves stay on the phones, and only meta data are sent to the central semantic store. For fast indexing and searching, we use ElasticSearch engine [6], one of the most popular enterprise search engines, as the back-end.
- Camera Pipeline: The subsystem that takes care of picture taking and meta data extraction on the remote phones, meta data transmission to the meta store, and sending the actual images upon explicit requests. We use the Medusa/MediaScope systems [7], [8] for this component.
- Source Selector: A road segment can potentially have multiple pictures that can be retrieved for checking its condition. In order to minimize the transmission cost, we perform source selection computation [1] to reduce the set of relevant pictures.
- Request Resolver: The component that uses our algorithms discussed in Section. III as the underlying engine to plan for optimal request resolution strategies.
- User Interface: A web interface where a user can specify the source and destination locations (by either clicking on the map, or using natural language, which would then go through an NLP processor for location name extraction, and Google geocoding [9] for conversion to lat-lon coordinates), provide human judgments to retrieved images (by specifying if they reflect road segments being in good or bad conditions), and see the final routing result. AJAX is used to pass information between the client web front-end and the server back-end algorithms for iterative interactions and updates.

Upon receiving the source-destination input, the system

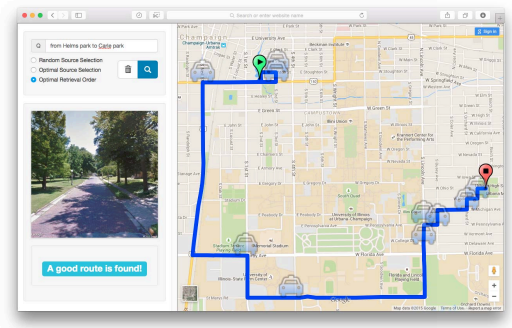


Fig. 2: Screenshot of the web front-end interface of our implemented route planning system.

computes multiple (>10) candidate routes and extracts all the unique road segments of all the routes. For each road segment, a geo-polygon is computed and then used to query the semantic store for matched meta data. The source selection module then stripes the set of all relevant meta data down to a minimal-cost set that still covers all the relevant road segments. With this set of meta data, our algorithm computes the optimal retrieval order, and iteratively carries out image retrieval and order update according to user input to resolve the routing request. A screenshot of the system finding a good route for a user specified source-destination location pair is shown in Fig. 2. The car icons indicate the locations where images have been retrieved and approved by the user; the result route is highlighted by the blue polyline on the map.

V. Evaluation

We take two complementing approaches in evaluating our proposed solutions. On one hand we design and carry out extensive simulations that explore how various problem and system aspects affect the behavior of our proposed algorithm. On the other hand we specify concrete application scenarios and demonstrate the effectiveness of our algorithm and system through actual resolutions of route finding requests.

We compare our algorithm (*vLDF*) to the following three baseline methods in the evaluation.

- *Lowest Cost Source First* (*LCF*) — Data retrievals follow the request's data items' costs, in ascending order.
- *Short-circuiting Benefit* (*SCB*) — Data retrieval order is computed according to the short-circuiting benefit (i.e., short-circuit probability per unit cost), as described by Casanova et al. [3].
- *Probability based Prediction* (*PbP*) — We assign to each test a predicted value based on its success probability (i.e., $v(t_{i_j}) = \mathbb{1}_{p_{i_j} \geq 0.5}$). Then each action a_i's value is computed according to its conditions' predicted values: $v(a_i) = \prod_j v(t_{i_j})$. a_i's cost is then the sum of its conditions' costs if $v(a_i) == 1$, and the smallest cost of its 0-valued conditions otherwise. The actual retrieval order is as follows, all 1-valued actions are ordered before 0-valued ones; actions of the same predicted value are ordered according to their predicted costs (ascending); for a 1-valued action, its conditions are ordered according to their costs (ascending); for a 0-valued action, all its 0-valued conditions are placed before 1-valued ones, and conditions of the same value are ordered according to their costs (ascending). After the retrieval of each data item, the request and the retrieval order is updated according to the actual value newly fetched.

As none of the above baseline methods take into consideration data freshness, so upon encountering data expiry, they simply refetch the data if their respective updated retrieval orders dictate so. We set each request to timeout when its lapse has exceeded the sum of all its data items' freshness deadlines, as a way to prevent infinite expiry-refetch loops.

A. Algorithm Behaviors

We first introduce our simulation settings, and then go through each of the set of experiments.

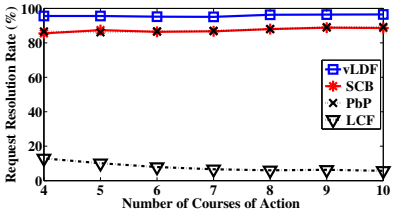
We experiment with the scenario involving data items of 2 different freshness deadline levels, tight and relaxed, and set a ratio parameter (tights' percentage, from 40% to 100%, default at 70%) to vary the mixture of the 2 types of data items. We experiment with different numbers of alternative courses of action per request (from 4 to 10, default at 8), and different numbers of conditions per action (from 4 to 10, default at 6), indicating the complexity of each alternative for the application requests. All data items' sizes range from 2 MB to 5 MB (default at 3450 KB)[1]. Due to our target post-disaster setting, we experiment with limited network bandwidths from 3.5 KBps to 6.5 KBps (default at 5 KBps), simulating a slow emergency network set up by first responder teams[2]. Conditions' average success probabilities range from 45% to 95% (default at 75%).

Regarding general network topologies, rather than committing to a particular structure, we use a single parameter α to indicate the network's common bottleneck ratios, which is defined as follows: When performing concurrent retrieval of multiple data items, the α portion of each item's transmission time will be spent in a bottleneck, where different data items would queue up and transmit in sequential order; the $1 - \alpha$ portions, on the other hand, proceed in parallel, thus the total delay incurred by those portions are just the maximum among all data items being concurrently fetched. We experiment with α ranging from 0 to 100% (default to 50%). In terms of each data item's individual transmission delay, because of the fluctuations caused by various internal and external factors, predicted transmission delay will never be perfectly accurate. Thus for simulating the actual transmission delay in a noisy network, we add a Gaussian noise to the predicted value, with the mean ranging between ± 3 minutes (default at 0-mean) and standard deviation 0 to 6 minutes (default at 1).

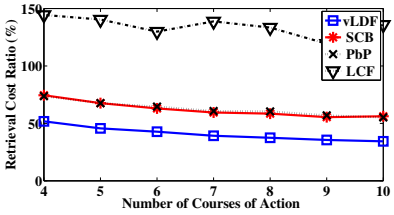
We carry out our simulation experiments by tuning one parameter at a time and fixing all the rest to their default values. We use two metrics for performance measures:

- *Request Resolution Ratio* — The percentage of the number of resolved requests over the total number of requests attempted (1000 randomly generated for each parameter setting). The higher the resolution ratio is, the better.
- *Retrieval Cost Ratio* — The percentage of the total cost of all retrieved data items over the total cost of all relevant data items, averaged over all resolved requests from the 1000 runs. The lower the cost ratio is, the better.

[1]These sizes roughly correspond to image sizes of today's mobile phones.
[2]These bandwidths roughly correspond to that of military ad-hoc communication networks under similar settings.

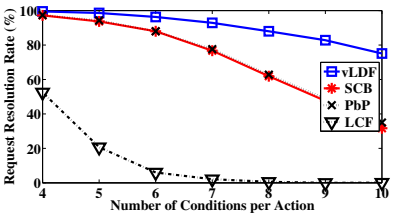


(a) Request resolution ratio

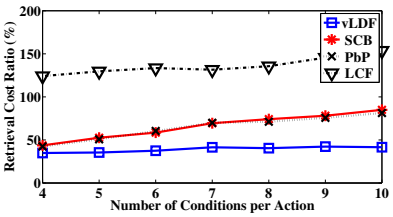


(b) Retrieval cost ratio

Fig. 3: # Courses of action per request

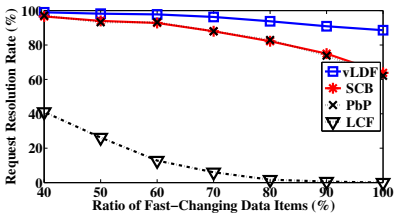


(a) Request resolution ratio

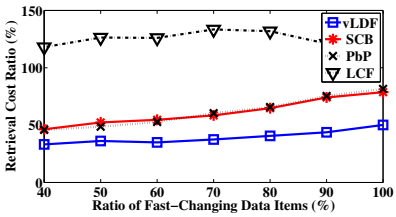


(b) Retrieval cost ratio

Fig. 4: # Conditions per action



(a) Request resolution ratio



(b) Retrieval cost ratio

Fig. 5: Fast-changing data portion

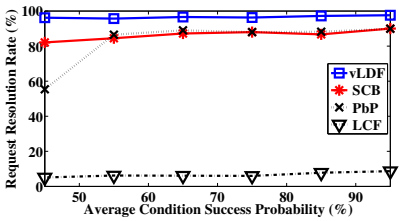
We first look at how the number of courses of action within a request affects data retrieval algorithms' behaviors. This in practice corresponds to the number of different alternatives an application request can be satisfied. Take the post-disaster route planning scenario as an example, we can easily draw parallels between actions $\Leftrightarrow$ routes, and conditions $\Leftrightarrow$ road segments. We experiment with 4 to 10 actions per request, and show results in Fig. 3. As seen in Fig. 3a our vLDF scheme achieves the highest request resolution ratios compared to the baseline methods. The SCB (short-circuiting benefit) and PbP (probability based prediction) show very similar performance (which is the case through all simulation experiments), while the simplest LCF (least cost first) scheme achieves the worst performance (which also is the case through all simulation experiments). This general performance comparison is reasonable because our vLDF scheme considers success probability, cost, and freshness deadlines; SCB and PbP both fail to take into consideration the freshness deadlines; and LCF only looks at the costs. We see that LCF's retrieval cost ratios are normally beyond 100%; this is because it needs to deal with data items' expiries and carry out refetches, thus greatly increasing the network resource consumption. We also observe the trend of more courses of action lead to lower retrieval cost ratios, as more actions can get short-circuited by a succeeding one.

We next look at how the number of conditions per action (which can be a measure of the complexity of each request's resolution alternative is) affects performance of the various schemes. Results are shown in Fig. 4. As seen, having more conditions to check for each resolution alternative makes it harder for all schemes to resolves requests, evidenced by the declining resolution rates. But our vLDF scheme shows the slowest dipping trend. On the other hand, for the requests vLDF *is* able to resolve, the retrieval cost ratio is not affected much, compared the the other methods who clearly need to consume more network bandwidth for their query resolutions.

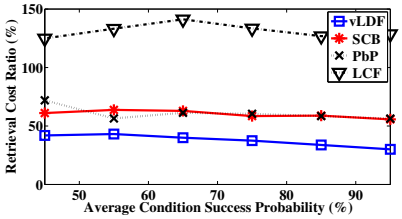
As previously discussed, we have conditions generally belonging to two different categories, namely slow changing and fast changing. Therefore, we also experiment with how the mixture ratio of the two categories affects the behaviors of all the approaches. The results are shown in Fig. 5. As seen, as the proportion of fast-changing data items increases, all schemes gradually show degraded performance, as it is generally harder to resolve requests in time without data freshness violation when we have to deal with more objects that intrinsically changes states fast.

Since our vLDF algorithm, as well as the PbP and SCB baselines, exploits conditions' logic relations, it would be interesting to see how the performance of each of the scheme is affected by different success probability settings. For sensing tasks, we can think of these probabilities as representing a measure of how good of a prior knowledge we already have of the target environment: good and clear knowledge (e.g., probabilities close to 100% certainty) would lead to algorithms more often making the correct guesses and picking the more optimal data items to retrieve to short-circuit other data items; on the other hand if the prior knowledge is ambiguous (e.g., probabilities close to 50%), algorithms will make more wrong guesses. Our simulation results are shown in Fig. 6. The trend of change is rather subtle, but is still visible: generally the closer all conditions' success probabilities are to certainty (i.e., 100%), the higher percentage of requests that can be resolved. This is because as the success probabilities become more certain, all schemes (except for LCF) make the right guesses more frequently, and thus resolve the requests more efficiently. For the case of LCF, the slight increasing resolution ratio comes from the fact that higher certainty (for success) leads to sooner valid course of action being found. Given the above discussion on request resolution ratio, the retrieval cost ratio results shown in Fig. 6b should be clear as well.

Next up, we experiment with varying requests' data item sizes as well as the network bandwidth, and examine how the various schemes' performance changes. Results are shown in Fig. 7 and 8. As can be seen, the two sets of Figures appear to be mirror images of each other. This makes sense as both of them directly influence the retrieval delays during request resolutions; and the retrieval delays directly affect the various schemes' behaviors. Increasing data item sizes therefore has a similar effect as decreasing network bandwidth. Let's focus our attention on the network bandwidth experiment result as shown in Fig. 8: Higher transmission speed lead to more requests being resolved and at the same time less network resource consumed. With more resource available on

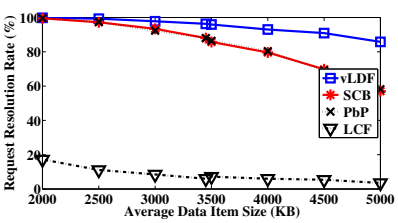


(a) Request resolution ratio

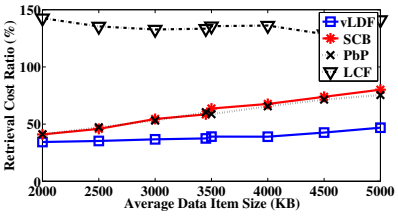


(b) Retrieval cost ratio

Fig. 6: Prob. of conditions being true

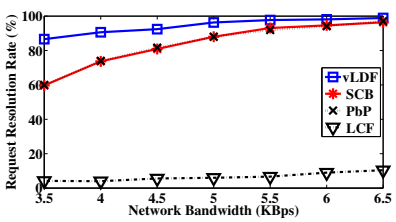


(a) Request resolution ratio

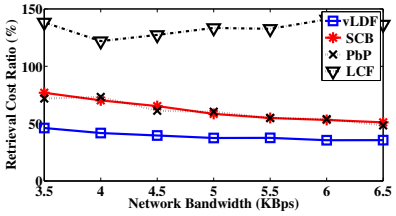


(b) Retrieval cost ratio

Fig. 7: Average data size



(a) Request resolution ratio



(b) Retrieval cost ratio

Fig. 8: Network bandwidth

the network, requests tend to get resolved faster with less freshness violations. Thus the retrieval cost ratios generally decreases for all schemes, except for the LCF approach, which shows fluctuating (but still all poor) performance.

For the general network topology, we use the parameter α to indicate the level of shared bottleneck in the network, as also previously discussed when we introduce experiment settings. Basically, a higher α value indicates an more severe bottleneck shared by all nodes within the network; parallel data retrievals from nodes will be queued up to become sequential processing more severely. Results are shown in Fig. 9. Since all three baseline methods have no way of incorporating parallel retrievals, changing α value has no obvious effect on them. For our scheme vLDF on the other hand, a trend of decreasing proportion of requests being resolved can be observed as the level of shared network bottleneck increases. But even then, we still see a clear advantage margin of our algorithm over any other baseline methods. We can also see a slight hint of increase in the network resource consumption of our algorithm from Fig. 9b.

Lastly, we look at how network fluctuations (as represented by Gaussian noise added onto estimated transmission delays) affect the performance of the various algorithms. This set of experiments are interesting because network transmission delays need to be estimated through the running of our vLDF algorithm: If the actual transmission takes longer to finish than what's estimated, we expect performance degradation; on the other hand if the transmission finishes sooner than expected, then data retrieval plan computed by vLDF would have been too conservative, in the sense that too much caution would have been taken to prevent freshness deadline violations, and cost minimization could have been carried out more aggressively. These intuitions are confirmed by results shown in Fig. 10: the advantage margin of vLDF over SCB and PbP are at its largest when the mean of the network fluctuation is at 0, and shrinks if the mean moves to both the negative and positive directions. The general trend that underestimating transmission time (positive fluctuation mean) leads to poorer performance is understandable, as longer actual transmission time will lead to more data freshness violations and thus potentially more request resolution timeouts and more data refetches.

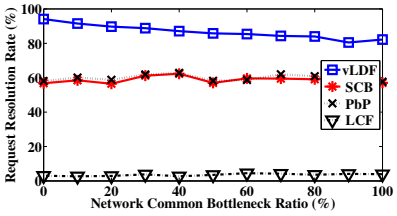
In addition to the mean, we also experiment with various levels of network fluctuation standard deviations. Results are shown in Fig. 11. We can observe for our vLDF algorithm that, as the standard deviation increases, the proportion of requests that can get resolved decreases and the retrieval cost ratio slightly increases. These are reasonable because of the higher level of unexpectedness of the network transmission delays caused by the larger standard deviations. All other baseline methods do not show clear trend of changes, because the mean of the fluctuation is set at 0 for this set of experiments, thus positive and negative effects tend to cancel out over time.

B. Route Finding Application

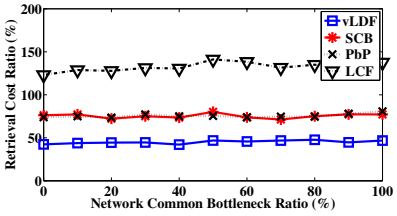
After rather abstract simulation experiments, we now look at a few concrete instances of the route planning application running using our implemented system. As it is difficult to find actual post-disaster scenarios to test our systems in, we take the following steps as a way to emulate the disaster settings: We imagine a chaotic environment in the Urbana-Champaign IL region. We crawl Google Maps Street View images, and Instagram's Urbana-Champaign traffic accidents and road blocks images (all geotagged). We insert all image meta data to the meta store, and use the Emane/Shim [10] network simulator to intercept all image requests (where link bandwidth is set at 5KBps). Each image's probability of showing a road segment of being in good condition is set to be reversely proportional to the road segment's speed limit (accidents tend to happen on high speed roads rather than residential roads). Freshness intervals are also set reversely proportional to speed limit, as whatever abnormalities on higher speed roads tend to get cleared more quickly. We experiment with two different underlying data retrieval algorithms: our vLDF, and the PbP baseline. The end to end query resolution image retrieval cost and latency comparisons are shown in Table I. As seen, vLDF consistently over-performs the baseline. Fig. 12 shows the actual route finding results from a particular run. The PbP scheme tests Route I and II first before testing Route III, where our algorithm vLDF selects Route III to check first.

VI. Related Work

The recent proliferation of increasingly capable and affordable sensing devices has given rise to a broad spectrum

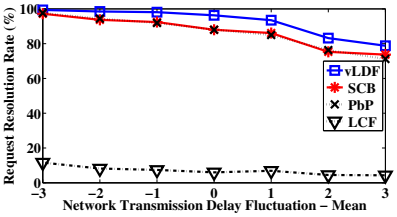


(a) Request resolution ratio

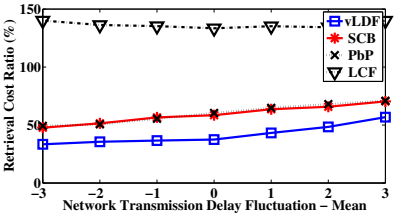


(b) Retrieval cost ratio

Fig. 9: Network common bottleneck ratio

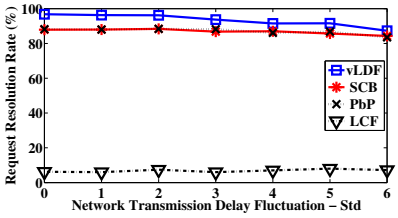


(a) Request resolution ratio

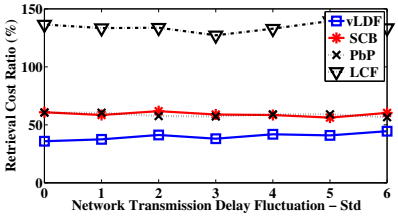


(b) Retrieval cost ratio

Fig. 10: Network delay noise mean



(a) Request resolution ratio



(b) Retrieval cost ratio

Fig. 11: Network delay noise variance

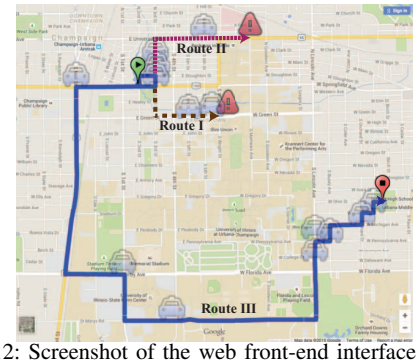


Fig. 12: Screenshot of the web front-end interface of our implemented route planning system as used to perform route planning task between a specified source-destination location pair

vLDF Cost (KB)	*PbP* Cost (KB)	*vLDF* Time (s)	*PbP* Time (s)
516	685	164	255
343	598	150	206
319	485	160	248
506	1093	165	372
524	1042	175	206

TABLE I: Results from 5 runs route planning

of sensing systems [11]–[18]. For example, BikeNet [11] is a sensor network for bikers to share data and map regions. CarTel [12] is a mobile sensing system for automobiles, where data can be collected, processed, and visualized. Coric et al. [13] design a crowdsensing system that helps to identify legal parking spaces. MaWi [14] is an indoor localization system with improved accuracy. Hu et al. [17] design and implement the SmartRoad system that takes advantage of vehicular sensing data to automatically detect and recognize traffic lights and stop signs. Given the richness of distributed sensing systems, various techniques [19]–[24] have been proposed for data clean-up and fact-finding.

A major challenge in the design of distributed sensing systems stem from the constraint of system resources. Significant studies have thus been conducted to tackle this problem. For example, Breadcrumb [25] is an automatic and reliable sensor network for the firefighting situations. In the SensorFly [26] project, low cost mobile sensing devices are utilized to build an indoor emergency response system. PhotoNet [27] provides a post-disaster picture collection and delivery service for situation awareness purpose. The sensing system proposed in this paper complements prior work by exploring the logic relations among sensory data to improve the communication efficiency of the system.

The problem of communication efficiency has also been studied in some existing work. For example, time-series prediction techniques are used to reduce communication burden without compromising user-specified accuracy requirements in wireless sensor networks [28]. Regression models are used to estimate predictability and redundancy relationships among sensors for efficient sensor retrievals [29]. MediaScope [8] is a mobile sensing system with various algorithms designed to help with timely retrievals of remote media contents (e.g. photos on participants' phones) upon requests of multiple types (nearest-neighbor, spanners, etc). Gu et al. [30] design inference-based algorithms for data extrapolation for disaster response applications. Minerva [31] and Information Funnel [32] explore data prioritization techniques based on redundancy or similarity measures for information maximization. Data aggregation techniques are also studied to reduce network transmission and improve classification tasks' accuracies [33]–[35]. Different from the above work, our system is designed to deal with the sensing scenarios where data items bear not only logic relations but also freshness deadlines.

The optimization of boolean predicate evaluation, as a theory problem, has been studied extensively. Greiner et al. [2] analyze and give theoretical results for the various subspaces of the general PAOTR (probabilistic and-or tree resolution) problem. Luby et al. [36] propose a set of tools for analyzing the probability an and-or tree evaluates to $true$. Casanova et al. [3] give various heuristic-based algorithms for the general NP-hard PAOTR problem and show performance comparisons. While borrowing some ideas from the above theoretical studies, our work also goes beyond them to consider much more practical problem settings including concurrent requests, deadline constraints, and the existence of partial retrieval sets.

Similar to this paper, some recent work, such as Carlog [37], ACE [38], and Hu et al. [39], also studied the optimization techniques for query request. Specifically, Carlog explores latency optimization for vehicular sensing applications, ACE aims at energy efficiency for continuous mobile sensing

applications, and and Hu et al. focus on cost optimization under query-specific deadline constraints. However, none of them takes into consideration the deadlines of data items which are of practical importance in many sensing applications. In contrast, in this paper we proposed a generalized framework that leverages the logic relations among sensors to optimize the utilization of system resources under the deadline constraints of both query and sensory data. Bearing these advantageous properties, the proposed system can be applied to the full spectrum of application domains.

VII. Conclusions

In this paper we target resource constrained dynamic environments (e.g. post disaster) and develop data retrieval algorithms for crowd-sensing applications. By incorporating classic results from the real-time community, and at the same time exploiting logical dependencies among data items, our algorithms offer considerable reduction in the underlying network bandwidth consumption, while at the same time respecting the timeliness requirement under the environment dynamics. Results from extensive simulations show that our algorithms outperform several baselines by significant margins. We also implement a post-disaster route finding system and demonstrate the advantage of our algorithms through realistic application scenarios.

Acknowledgment

Research reported in this paper was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement W911NF-09-2-0053, DTRA grant HDTRA1-10-1-0120, and NSF grants NSF CNS 13-29886, NSF CNS 13-45266, and NSF CNS 13-20209. 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.

References

[1] A. Bar-Noy, M. P. Johnson, N. Naghibolhosseini, D. Rawitz, and S. Shamoun, "The price of incorrectly aggregating coverage values in sensor selection," in *DCoSS*, 2015.

[2] R. Greiner, R. Hayward, M. Jankowska, and M. Molloy, "Finding optimal satisficing strategies for and-or trees," *Artificial Intelligence*, vol. 170, no. 1, pp. 19–58, 2006.

[3] H. Casanova, L. Lim, Y. Robert, F. Vivien, and D. Zaidouni, "Cost-optimal execution of boolean query trees with shared streams," in *IPDPS*, 2014.

[4] Gosmore, "http://wiki.openstreetmap.org/wiki/Gosmore," 2015.

[5] OpenStreetMap, "http://www.openstreetmap.org," 2015.

[6] ElasticSearch, "https://www.elastic.co," 2015.

[7] M.-R. Ra, B. Liu, T. F. La Porta, and R. Govindan, "Medusa: A programming framework for crowd-sensing applications," in *MobiSys*, 2012.

[8] Y. Jiang, X. Xu, P. Terlecky, T. Abdelzaher, A. Bar-Noy, and R. Govindan, "Mediascope: Selective on-demand media retrieval from mobile devices," in *IPSN*, 2013.

[9] Google Maps Geocoding API, "https://developers.google.com/maps/documentation/geocoding/," 2015.

[10] W. Dron, A. Leung, J. Hancock, M. Aguirre, Thapa, and R. Walsh, "Core shim design document," in *NS-CTA Technical Report*, 2014.

[11] S. B. Eisenman, E. Miluzzo, N. D. Lane, R. A. Peterson, G.-S. Ahn, and A. T. Campbell, "Bikenet: A mobile sensing system for cyclist experience mapping," *TOSN*, vol. 6, no. 1, p. 6, 2009.

[12] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, and S. Madden, "Cartel: A distributed mobile sensor computing system," in *SenSys*, 2006.

[13] V. Coric and M. Gruteser, "Crowdsensing maps of on-street parking spaces," in *IEEE DCOSS*, 2013.

[14] C. Zhang, J. Luo, and J. Wu, "A dual-sensor enabled indoor localization system with crowdsensing spot survey," in *IEEE DCOSS*, 2014.

[15] S. Hu, L. Su, H. Liu, H. Wang, and T. F. Abdelzaher, "Smartroad: a crowd-sourced traffic regulator detection and identification system," in *IPSN*, 2013.

[16] S. Hu, H. Liu, L. Su, H. Wang, T. F. Abdelzaher, P. Hui, W. Zheng, Z. Xie, J. Stankovic *et al.*, "Towards automatic phone-to-phone communication for vehicular networking applications," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 1752–1760.

[17] S. Hu, L. Su, H. Liu, H. Wang, and T. F. Abdelzaher, "Smartroad: Smartphone-based crowd sensing for traffic regulator detection and identification," *ACM Transactions on Sensor Networks (TOSN)*, vol. 11, no. 4, pp. 55:1–55:27, Jul. 2015.

[18] S. Hu, L. Su, S. Li, S. Wang, C. Pan, S. Gu, T. Amin, H. Liu, S. Nath, R. R. Choudhury, and T. Abdelzaher, "Experiences with enav: A low-power vehicular navigation system," in *UbiComp*. ACM, 2015.

[19] D. Wang, L. Kaplan, H. Le, and T. Abdelzaher, "On truth discovery in social sensing: A maximum likelihood estimation approach," in *IPSN*, 2012.

[20] Q. Li, Y. Li, J. Gao, B. Zhao, W. Fan, and J. Han, "Resolving conflicts in heterogeneous data by truth discovery and source reliability estimation," in *SIGMOD*, 2014.

[21] S. Wang, D. Wang, L. Su, L. Kaplan, and T. F. Abdelzaher, "Towards cyber-physical systems in social spaces: The data reliability challenge," in *RTSS*. IEEE, 2014, pp. 74–85.

[22] C. Meng, W. Jiang, Y. Li, J. Gao, L. Su, H. Ding, and Y. Cheng, "Truth discovery on crowd sensing of correlated entities," in *SenSys*, 2015.

[23] C. Miao, W. Jiang, L. Su, Y. Li, S. Guo, Z. Qin, H. Xiao, J. Gao, and K. Ren, "Cloud-enabled privacy-preserving truth discovery in crowd sensing systems," in *SenSys*, 2015.

[24] S. Wang, L. Su, S. Li, S. Hu, T. Amin, H. Wang, S. Yao, L. Kaplan, and T. Abdelzaher, "Scalable social sensing of interdependent phenomena," in *IPSN*, 2015.

[25] H. Liu, J. Li, Z. Xie, S. Lin, K. Whitehouse, J. A. Stankovic, and D. Siu, "Automatic and robust breadcrumb system deployment for indoor firefighter applications," in *MobiSys*, 2010.

[26] A. Purohit, Z. Sun, F. Mokaya, and P. Zhang, "Sensorfly: Controlled-mobile sensing platform for indoor emergency response applications," in *IPSN*, 2011.

[27] M. Y. S. Uddin, H. Wang, F. Saremi, G.-J. Qi, T. Abdelzaher, and T. Huang, "Photonet: a similarity-aware picture delivery service for situation awareness," in *RTSS*, 2011.

[28] Y.-A. L. Borgne, S. Santini, and G. Bontempi, "Adaptive model selection for time series prediction in wireless sensor networks," *Signal Processing*, vol. 87, no. 12, pp. 3010 – 3020, 2007.

[29] C. C. Aggarwal, A. Bar-Noy, and S. Shamoun, "On sensor selection in linked information networks," in *DCoSS*, 2011.

[30] S. Gu, C. Pan, H. Liu, S. Li, S. Hu, L. Su, S. Wang, D. Wang, T. Amin, R. Govindan *et al.*, "Data extrapolation in social sensing for disaster response," in *DCoSS*, 2014.

[31] S. Wang, S. Hu, S. Li, H. Liu, M. Y. S. Uddin, and T. Abdelzaher, "Minerva: Information-centric programming for social sensing," in *ICCCN*, 2013.

[32] S. Wang, T. Abdelzaher, S. Gajendran, A. Herga, S. Kulkarni, S. Li, H. Liu, C. Suresh, A. Sreenath, H. Wang *et al.*, "The information funnel: Exploiting named data for information-maximizing data collection," in *DCoSS*, 2014.

[33] L. Su, J. Gao, Y. Yang, T. F. Abdelzaher, B. Ding, and J. Han, "Hierarchical aggregate classification with limited supervision for data reduction in wireless sensor networks," in *SenSys*, 2011.

[34] L. Su, S. Hu, S. Li, F. Liang, J. Gao, T. F. Abdelzaher, and J. Han, "Quality of information based data selection and transmission in wireless sensor networks." in *RTSS*, 2012, pp. 327–338.

[35] L. Su, Q. Li, S. Hu, S. Wang, J. Gao, H. Liu, T. F. Abdelzaher, J. Han, X. Liu, Y. Gao *et al.*, "Generalized decision aggregation in distributed sensing systems," in *RTSS*, 2014.

[36] M. G. Luby, M. Mitzenmacher, and M. A. Shokrollahi, "Analysis of random processes via and-or tree evaluation," in *SODA*, 1998.

[37] Y. Jiang, H. Qiu, M. McCartney, W. G. J. Halfond, F. Bai, D. Grimm, and R. Govindan, "Carlog: A platform for flexible and efficient automotive sensing," in *SenSys*, 2014.

[38] S. Nath, "Ace: Exploiting correlation for energy-efficient and continuous context sensing," in *MobiSys*, 2012.

[39] S. Hu, S. Li, S. Yao, L. Su, R. Govindan, R. Hobbs, and T. Abdelzaher, "On exploiting logical dependencies for minimizing additive cost metrics in resource-limited crowdsensing," in *DCoSS*, 2015.