WoTSF: A Framework for Searching in the Web of Things

Mina Younan
Faculty of Computers and Information
Minia University
El-Minia, Egypt
mina.younan@mu.edu.eg

Sherif Khattab
Faculty of Computers and Information
Cairo University
Giza, Egypt
s.khattab@fci-cu.edu.eg

Reem Bahgat
Faculty of Computers and Information
Cairo University
Giza, Egypt
r.bahgat@fci-cu.edu.eg

ABSTRACT
A key challenge in the emerging Web of Things (WoT) paradigm is how the human users and machines look for meaningful and readable information in huge and dynamic datasets in real-time, whereby the datasets are presented in different formats. This paper presents a technique to construct efficient, hierarchical web indices that are efficiently kept up-to-date. Also, a framework for searching in the WoT, namely WoTSF, is proposed and experimentally evaluated using a prototype. The proposed framework was shown to present a tradeoff between search speed and result accuracy as compared to the Dyser WoT search engine.

CCS Concepts
• Information systems~Web indexing

• Computer systems organization~Sensor networks

Keywords
Internet of Things (IoT); Web of Things (WoT); Web Search; Web Indexing.

1. INTRODUCTION
Augmenting everyday objects (e.g., light bulbs, curtains, and appliances) with embedded computers or visual markers allows things and information about them to be accessible digitally (i.e., through the Web or mobile phones). The augmented objects become internet's interface to the physical world [1] [2] [3]. In the Internet of Things (IoT), sensors and actuators that are connected to smart things (SThs) monitor and control their surrounding environment. They produce raw data about changes in states and events in the environment.

The Web of things (WoT) [4] uses existing web tools to present IoT data in high-level states as final conclusions about objects and events. That is, WoT integrates sensors and actuators not only to the internet (network layer) but also to the web (application layer) to provide users with an easy interface to present information about physical objects in a readable form [5]. Searching for SThs and Entities of Interest (EoIs) is an essential service for almost all of the WoT applications [6][7][8][9][10] because users of the WoT are more interested in high-level states especially about EoIs [3]. WoT search engines, such as Dyser [3] [11], crawl SThs and EoIs pages and build an index of all possible states and probabilities of those states in order to answer queries.

The current trend in searching for SThs and EoIs in the WoT is to build loosely coupled systems [12], so that anyone can rebuild his own search engine for his own network [3]. But still, there is no general search engine for the IoT and the WoT [13] [14]. Many challenges face the WoT searching service in general [1][7][15][13]. Whereas a huge number of heterogeneous connected devices, there is no standardized naming for SThs' attributes during the device registration process. Moreover, SThs have dynamic information, such as readings and locations of movable objects on which sensors and actuators are attached. Even some of the static STh information (e.g., logical path) is not considered as STh attributes although they are needed in the search process. For example, the Dyser WoT search engine [3] [8] needs to answers queries such as "*find rooms at building X, occupancy:empty*". Because WoT pages host dynamic parts (i.e., pages parts are coded using AJAX), some search crawlers cannot crawl them [16]. Finally, STh state naming is not standardized.

The main contribution of this paper is a technique to build a compact and accurate index and to efficiently keep it up-to-date. Many WoT networks contain similar types of sensors; for example, smart buildings that have an occupancy sensor in each room. Using a master index that indexes only the existence of an empty room in the building instead of indexing each sensor value reduces the index size and directs the fine-grained room search only into buildings that have empty rooms with high probability.

This paper also presents a framework for searching in the WoT (called WoTSF) that addresses the challenges mentioned above. WoTSF builds high-level indices for each STh and EoI type in all WoT networks using a certain aggregation function (e.g., count or maximum). The master indices are created from local indices of distributed local search engines. A prototype of WoTSF was implemented and evaluated in comparison to Dyser [3] using three assessment criteria: index size, processing time and the accuracy of the results.

The remainder of the paper is organized as follows. In Section 2, related work of search engines in the IoT and the WoT is presented. Section 3 describes the proposed WoTSF architecture. In Section 4, the implementation of WoTSF is described followed by experimental evaluation. Finally, conclusions and future work are presented in Section 6.

2. RELATED WORK
In this section, we spotlight the gap between current WoT search engines and the requirements of WoT search.

Conference'10, Month 1–2, 2010, City, State, Country.

DOI: http://dx.doi.org/10.1145/12345.67890

2.1 Real-time web search engines

A web search engine, like Google, is an application for retrieving relevant information to a user query from the domain of the World Wide Web (WWW) [14]. It matches the user query with static content of web pages that are crawled and indexed in a distributed database. Indexing keywords of the crawled web pages is an essential component of the web search. Result ranking depends on keywords matching score. In general, web search engines are document centric.

Search engines of social networks, like Twitter and Facebook, are server centric. They update states of their users providing real-time search for messages, comments and reviews, where users can search for messages, comments, etc., using keywords, and results are returned in real-time. Updates in user states are visible in real-time to all users [3].

2.2 IoT and WoT search engines

Searching the WoT is a special type of web search and network search [14]. Like web search, WoT search uses the static parts of the user query (e.g., find sensor of model X). Like network search, WoT search matches search queries against a dynamic information space.

Dyser [3][11] is a content-based search engine for the WoT. It works in a similar way to traditional search engines (e.g., Google), whereby the index of Dyser is built upon keywords, and users use simple and structured query language to search for entities. Dyser can search for SThs and EoIs in real-time. In Dyser, queries about states of SThs and EoIs are answered at the time of the query. Dyser does not index historical data; it indexes page keywords (static and quasi-dynamic information) and prediction models in order to predict future states of SThs and EoIs. Dyser searches for entities in two steps. First, it searches on Google for entity pages using a *magic* string. Second, it searches in its internal indices for entities that match static part of the query (i.e., the keywords). Like Dyser, we propose the WoTSF that contacts local WoT Search Engines (WoTSEs) that can answer specific parts of the query (sub-queries), and then merges the results to be returned as a ranked list.

Shodan [17] indexes website HTTP and FTP banners of devices (e.g., desktops, routers, and IP cameras) connected to the internet. It helps users to find a specific device has a specific content in its banner. For example, '*Server: SQ-WEBCAM*' searches for a web cam. Shodan is a keyword based search engine. It searches for devices not EoIs.

SenseWeb [18] is a spatial search engine in the WoT for retrieving last-seen locations of SThs. Thingful [19] is a search engine for finding devices, datasets and real-time data sources by geo-location across many IoT networks. It has a unique geographical index of connected objects. Its query is split in two parts, *what* and *where,* to filter and speed up the search process.

2.3 Searching in the WoT

Christophe et al. [8] discuss searching in the WoT focusing on two points: (1) requester (machine or user), to select the best search algorithm, and (2) type of results (e.g., searching for a similar object or getting certain decision depending on the states of different objects) benefiting from the semantic web. Implementing semantic web solves problems of non-standardized naming. Two scenarios are discussed for searching in private and public WoT, which stimulate the need for a general search engine for the WoT.

Mayer et al. [20] present *DiscoWoT*, an extensible discovery service for solving the problem of using multiple formats in WoT applications. The discovery service is based on multiple discovery strategies. Using this service, clients semantically identify SThs using their URL. This service could be implemented with the crawler of the WoT search engine.

Mayer et al. [7] propose a web-based infrastructure for the WoT in order to facilitate the integration of SThs and interaction of human users and other SThs (i.e., a look-up service). They identify four types of queries to search for SThs resources; in the case of dynamic integration of SThs, implementing '*Request For Query (RFQ)*' enhances the search result.

Zhang et al. [21] present a framework for a distributed range-query search, which consists of three modules: (1) reporter, (2) indexer, and (3) query executor. The framework handles queries such as searching for sensors that read values in a range. Sensors send only abnormal readings to reduce the frequencies of data migration, and consequently reducing the frequency of updating the index of ranges. They focus on speeding up search for the special range queries, regardless if the search is implemented internally in a specific network or as a global search engine. The range-query framework proposed in [21] reduces the index size by reducing data migration between SThs. This query type is considered by our proposed WoTSF.

Truong et al. [6] [22] propose to search for sensor similarity and sensor content in the WoT. The sensor has three types of information: (1) static (e.g. sensor type), (2) quasi-dynamic (e.g., location), and (3) dynamic (e.g., measurement value). Similarity depends on STh types and historical data. But content-based search is another type of sensor queries for finding sensors whose values fall in a certain value range (similar to [21]).

A case study for crawling the WoT has been discussed [23], whereby the search engine spiders crawl RESTful services [24] instead of dynamic parts (i.e., parts coded with AJAX) in STh pages following Google recommended optimizations [25]. The WoTSF uses datasets generated by the testbed in [1] for building local indices for individual WoTSEs, which are built based on the Dyser search engine [3]. Crawling WoT systems for building general indices is done similar to Google search engine, whereby WoT homepages host a server-root file to direct search engine spiders [25]. The WoTSF indexes and downloads this file containing the required information about the WoT.

To sum up, most of above mentioned search systems support a keyword-based search, which will be difficult due to the challenges of WoT search. None of them discussed issues related to crawling and indexing data from different WoT networks. Using semantic web for enhancing search results is out of the scope of this paper but will be addressed in the future.

3. THE WOTSF ARCHITECTURE

Searching for high-level states is desirable for human users more than searching for raw sensory data [6]. Indeed, sensory data are huge and have a very short life-span [13] as sensors monitor daily life events. Users are interested in current states based on the sensory data. Because IoT and WoT systems are dynamic and real-time, the time that a search query takes before it returns the response to the user may be more than the time between state changes, leaving the query response stale and inaccurate. The solution in this case is to speed up the search process by filtering queries using prediction models and keeping indices up-to-date by building distributed high-level indices for different types of queries. In what follows, we present motivating scenario and describe the architecture of the proposed WoT search framework, which aims at speeding up the search process.

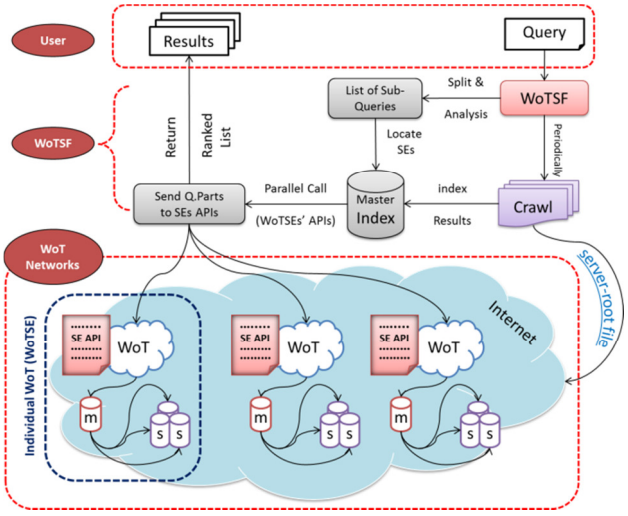


Figure 1. The architecture and query life-cycle in the proposed WoT Search Framework (WoTSF).

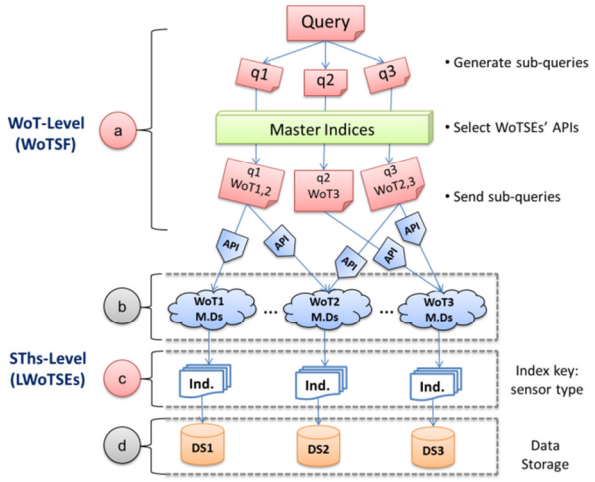


Figure 2. Index structure and query processing in the WoTSF. Two levels of indices are used: master indices and SThs-level indices.

3.1 Motivating scenario

Suppose that, there are three WoT networks that monitor three different smart hotels, where each WoT network has its own search engine. The first WoT network uses micro-format embedded in the EoIs pages, the second uses microdata, and the third implements AJAX for updating its pages in real-time, storing its data in RDF files. A user wants to search for a quiet and sun-lit room in the three hotels. The user searches each individual search engine and ranks the results. According to this scenario, there is a need to build a general search engine for the three systems, so that the user searches this global search engines instead of searching each local engine.

3.2 Query processing

The architecture of the WoTSF is shown in Figure 1. The main elements of the architecture are a global search engine (WoTSF) and one or more individual local search engines (WoTSEs). In this figure, *m* represents the index of a local search engine and *s* represents data storage in the WoT. The WoTSF crawls server-root files of the WoT networks and indexes general information about the WoT networks, such as URL of WoTSEs' APIs, list of sensor types and all possible states, and high-level dynamic information using a certain aggregation function (e.g., average). WoTSF works as a middle-layer between the user and the WoTSEs, whereby each WoTSE works on its local indices that relate to a specific WoT network. When WoTSF receives a query, it analyzes it generating a list of sub-queries to be pushed into a filtered list of WoTSEs' APIs, such as shown in Figure 2 (a). Each individual WoTSE builds multiple-indicies (Figure 2 (b,c)) to serve different queries quickly connecting them to SThs/EoIs datasets, such as shown in Figure 2 (d). Search results are ranked then returned to the query initiator (WoTSF user).

3.3 Two-level index structure

Building general indices for a massively increasing number of sensors requires the indices to be updated at a potentially very high rate. Distributing the indices, whereby each WoTSE handles its own indices, reduces the frequencies of updates in the higher-level indices. The hierarchical structure of indices is shown in Figure 3. The individual WoTSEs are located in WoT networks where datasets of sensors are located. For each STh the WoTSE gets the aggregation or summary of its readings for each unit of time (e.g., day, hour); the same thing is done for each EoI using previously calculated summaries of its SThs. The WoTSF calculates the aggregation for each building per EoI state. The WoTSF builds high-level indices about all WoT networks, called master indices or WoT-level indices, such as shown in Figure 1 and Figure 2 (a). In smart buildings domain, master indices could be called building-level indices. Individual WoTSEs execute sub-queries on their local indices, which are called secondary indices or STh-level indices.

The idea of building high-level indices using aggregation functions on EoI states has the following benefits: (1) distributed queries and distributed indices [26], (2) up-to-date indices, and (3) scalability to different WoT systems. As a result of benefits 1 and 2, WoTSF makes a balance between *search speed* and *accuracy of results*. Search speed produces results in less time (i.e., saving in network overhead) by getting results from master indices only. Result accuracy depends on updates from the individual indices. However, the two-level index produces additional delay due to network overhead but saves time in crawling, parsing, and indexing. Instead of re-indexing the whole WoT, each local WoTSE re-indexes only the new data either by applying the pull method (in the case of periodic index updates) or using the push method [1] [3] [23] (whereby sensors send only abnormal changes keeping top-k values up-to-date [21]. Keeping WoTSE indices using the pull approach decreases the complexity of data indexing and increases data accuracy [3]. Updating data will be for a few number of devices that sense changes in their states.

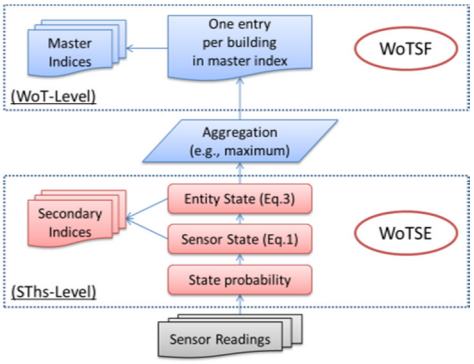


Figure 3. Hierarchical structure of WoTSF and WoTSE indices

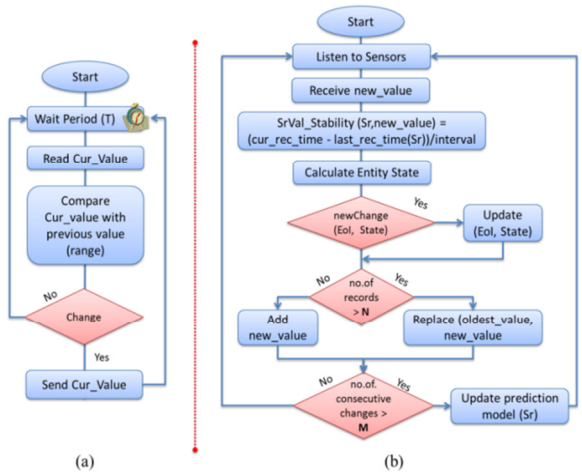


Figure 4. Flow charts for sensor and gateway process life-cycle. (a) Sensor periodically sends changes in its readings to its gateway (b) gateway listens to its sensors and builds their prediction models.

To sum up, the proposed two-level index structure makes a balance between data analysis in the search process and data migration and communication in the indexing process [21] [26].

4. WOTSF IMPLEMENTATION

A prototype of the WoTSF framework was implemented using C# and Web Services Applications (WCF) for communication between components. The main processes of the search engine are: crawling, indexing, searching, and ranking. The implementation of these components is described in the following subsections.

4.1 Crawling

A flowchart of the sensor process life-cycle is shown in Figure 4 (a), where the sensor sends only abnormal readings to be indexed by its gateway (push method). Historical data are stored in a local database (datasets in Figure 2 (d)), where the database servers serve queries consisting of aggregation functions (e.g., average and count). The flowchart for handling an incoming message from a sensor holding abnormal data is shown in Figure 4 (b). The gateway listens to its sensors and updates EoI states according to sensor values. If the number of records of the latest sensor values is greater than a threshold, the oldest value is replaced by the newest one; otherwise the gateway adds the sensor value to the list. Updating EoI prediction model depends on a number of consecutive state changes. Extracting the prediction model type (e.g., Single Period Prediction Model (SPPM) [3]) from the sensors' historical readings is out of this paper's scope.

4.2 Indexing

The WoTSF prototype produces indices like Dyser [3], a set of quadruple prediction records for each sensor in the time unit (e.g., 7-Days). Because this prototype works on a dataset generated by the testbed in [23], sensors may produce different values (readings) or change their states frequently in less than a minute (i.e., each sensor has more than one reading in the same time unit (e.g., a day)). For simplicity, we use the k^{th} percentile (e.g., 50^{th} percentile) for representing each sensor by a single value in the time unit. Resulting values were then used for generating prediction models in that time unit. For example, on Sunday a temperature sensor reads a set of values {5, 20, 21, 20, 20, 21, 23, 20, 21, 23}, Suppose that the temperature sensor has three possible states 'cold', 'warm', and 'hot', following the rules {if (value≤18) then 'cold', and if (18<value≤27) then 'warm', else 'hot'}. Using the 50^{th} percentile, the sensor could be represented by the value 20, thus the temperature sensor on Sunday could be represented by the state 'warm' with probability '50%'.

Like what is done with Dyser [3], we suppose that gateways (base-stations) in each WoT network generate prediction models from the historical data of their STHs and EoIs. Then, the search engine spider expands those models according to their periods. A general index similar to Dyser index (DSE index) for all WoT networks is shown in Figure 5 (a). On top of these quadruple prediction records, high-level indices are generated, such as shown in Figure 5 (b). An aggregation function is used for making a conclusion about a certain type of EoI states, for example the probability of having an empty room in a certain smart building. In this implementation, the WoTSF prototype generates a general index for STHs of type 'occupancy' in WoT networks, where each WoT network has a number of STHs. WoTSF uses the aggregation function 'maximum' to index information about probability of having an empty room in each WoT network according to the prediction model type (7-Days). When multiple EoIs in one building have the same aggregated value on one day, multiple records for the building are stored as shown in Figure 5 (b); otherwise, zero or one record is stored per day for each building.

4.3 Searching

WoTSF supports two modes for searching. In the case of *speed search* (searching in high-level indices), the WoTSF looks up its indices and ranks EoIs by their probabilities. But, in the case of *accurate results*, the WoTSF performs a deep search using WoTSEs' APIs then ranks results according to sensor probabilities. The WoTSF filters buildings using the static part of the query (i.e., type of buildings and location) selecting the appropriate indices, then searches for the dynamic part. However, the search in WoTSF and in Dyser are different.

For example, using Dyser, when the user selects a search criteria and probability>=50%, Dyser searches locally in its

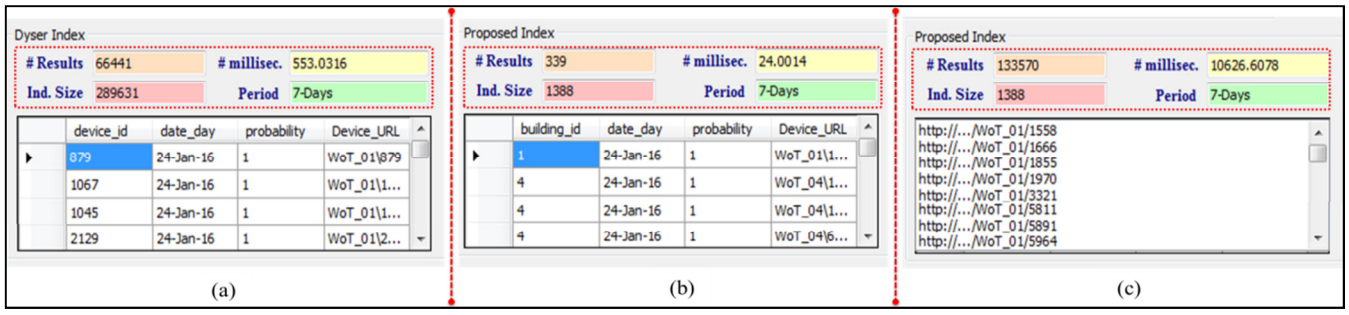


Figure 5. Search analysis: (a) searching on Dyser index (b) searching on Building-Level index (WoTSF) (c) WoTSF runs on Building-Level index and recursively calls appropriate WoTSEs' APIs returning a ranked list of results.

master index and returns a ranked list of direct STh URLs, which have probability >=50%, such as shown in Figure 5 (a). For WoTSF, the search process depends on the level of search, Building-level or STh-level. When the user sends the same query on Building-level, WoTSF first selects appropriate indices according to the static part of the query, then the WoTSF searches for buildings that score higher probabilities, and finally returns a list of results containing a variety of buildings, such as shown in Figure 5 (b). In other words, if a single building has more than one empty room, the WoTSF will return only one record per building according to the aggregation function that is used during the indexing process (e.g., maximum). When selecting STh-level, the WoTSF calls WoTSE APIs in parallel after filtering WoT networks using its high-level indices and ranks WoTSE results, as shown in Figure 5 (c). In this case, WoTSF results are more fine-grained. TheWoTSF prototype supports asking for first-*k* results as well, to make the WoTSF stop after getting the first *k* results. The WoTSF prototype implements auto suggestions, like Dyser, for solving the problem of using structured queries [3].

4.4 Ranking

We define Sensor State *SS(i)* as the probability that a sensor is in a certain state *i* (e.g., warm). *SS(i)* is evaluated as a constant value, *K,* (e.g., 0.5) multiplied by 1 if the sensor is in the desired state*, i*, and by 0 otherwise plus (*1-K*) multiplied by the probability of being in state *i.* All possible states are indexed according to a certain prediction model, whereby the probability value ranges from zero to one. Thus, *SS(i)* is evaluated as follows:

$$SS(i) = K * CS(i) + (1 - K)\,(P_i) \quad (1)$$

where *CS(i)* is 1 if current state is *i* and 0 otherwise and P_i is the probability of being in state *i*. We define Entity State *ES(i)* as the probability that an entity is in state *i.* Because the *ES(i)* may depend on more than one sensor state, the *ES(i)* is measured as the average of the corresponding sensors' states, where all STHs are independent and affect EoI state by the same degree such as follows:

$$ES(i) = \frac{1}{n} \sum_{j=1}^{n} SS(j) \quad (2)$$

where *n* is the total number of sensors on which the entity state depends, and $SS(j)$ is the individual sensor states. The ranking process is based on the entity state. An example of searching for a quiet room consequently searches for a temperature sensor in a warm state and for a loudness sensor in a low state. Suppose that K=60%, if the indexed values about the temperature sensor indicate that the current state is warm (i.e., CS(warm) = 1) but expected to stay in warm with probability 0.3, then the sensor state SS(warm) = (0.6 * 1) + (0.4* 0.3), which equals 72%. Assuming that the loudness sensor has SS(low) = 85%, then using Eq. 2, the probability of the room being in the quiet state is 78.5%.

But when sensor states affect *ES(i)* with different probabilities, then *ES(i)* is evaluated such as follows:

$$ES(i) = \sum_{j=1}^{n} D(j) * SS(j) \quad (3)$$

where *D(j)* is the impact factor of sensor j in *ES(i)*. Suppose that probability of having a quiet room is evaluated by 70% * SS(low) + 30% * SS(warm), then using Eq. 3, the probability of the room being in the quiet state is 81.1%.

5. EXPERIMENTAL EVALUATION

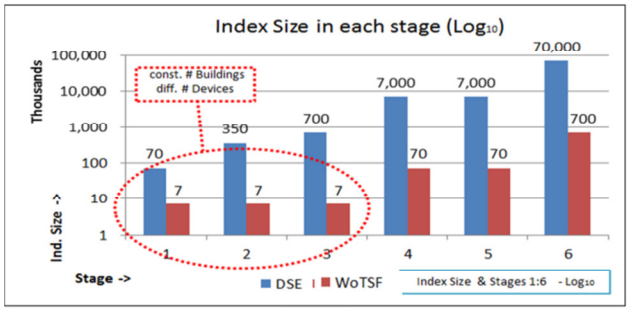
The framework evaluation was done on the integrated WoT testbed [1]. The structured queries that Dyser (DSE) can answer have two parts: static and dynamic. The WoTSF filters indices on which search will be executed returning a ranked list of the predicted EoIs that should be evaluated after that on the dynamic part using one of the two search modes mentioned above.

WoTSF was evaluated on indices that are built using real datasets and random synthesized values. When the user selects 'Real', a list of registered sensors in the WoT dataset appear, so that the user can select a sensor type and limit the list of devices. When the user selects 'Random', a sensor type should be selected and the number of buildings (WoT networks) and number of devices in each building should be determined. The WoTSF protoype generates a random number of days of the week (0:7) for each sensor, consequently each sensor has a random number of quadruple records.

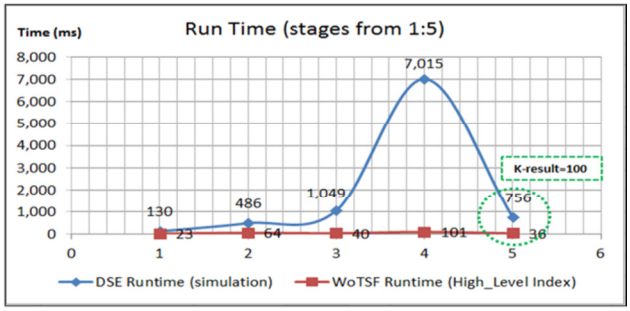
The main criteria of assessing the WoTSF are: (1) index size, (2) processing time (i.e., time consumed for crawling, indexing, searching, and ranking), and (3) result consistency and accuracy (i.e., number of intended and accurate results).

Model:	7-Days		Query:	hotel, room:empty @Egypt-Cairo				WoTSF-Index (HL):	Building-Level
Stage	**#Devices**	**#Buildings**	**#Records**	**DSE_Index**	**WoTSF_Index**	**DSE_Results**	**WoTSF_Results**	**DSE_S.run-time (ms)**	**WoTSF_S.run-time (ms)**
1	10	1000	70,000	70,000	7,000	9,939	1,000	130	23
2	50	1000	350,000	350,000	7,000	49,730	1,000	486	64
3	100	1000	700,000	700,000	7,000	99,545	1,000	1,049	40
4	100	10000	7,000,000	7,000,000	70,000	995,060	10,000	7,015	101
5	100	10000	7,000,000	7,000,000	70,000	100	100	756	36
6	100	100000	70,000,000	70,000,000	700,000				

(a)



(b)



(c)

Figure 6. WoTSF Evaluation: (a) DSE and WoTSF index sizes and processing times for different cases (stages) of number of devices and buildings. (b) Comparison of index sizes (log-scale). (c) Processing time.

5.1 Index size

Figure 6 (a) shows the number of devices per building, buildings, and index sizes of DSE and WoTSF (master index) at different stages, whereby each stage has a different number of buildings, devices per building, or both. The DSE index size is the product of the number of buildings, the number of devices per building, and the number of time units used in the prediction model (7 days in the table). The WoTSF master index size is only the product of the number of buildings and the number of time units, as each building is summarized by one value for each time unit. Figure 6 (b) shows that the index size of the WoTSF is less than the index size of the DSE, and it also indicates that in case of a constant number of WoT networks (e.g., buildings), increasing the number of devices of the same type makes DSE index size increases, while the WoTSF index does not change.

5.2 Processing time

WoTSF builds its master indices, depending on a server-root file that directs the crawling process, by downloading and indexing specific data about the WoT. That is, in WoTSF the STh pages are not accessed but only the root file. As a result, the number of recursive crawling processes was reduced as well as the time consumed for parsing and indexing WoT pages. Dyser [3] [11] implements an extensible service [20] for parsing multiple formats (e.g., microdata) by which WoT networks data are written, which consumes additional time.

The time consumed for searching in the WoTSF depends on the search mode. The WoTSF gives the users the ability to select between (1) searching in the high-level indices only to reduce network overhead (*speed search*) and (2) using high-level indices for filtering the search scope and selecting appropriate WoTSEs' APIs that are called recursively (*accurate results*).

The chart shown in Figure 6 (c) shows the relation between the processing time of DSE and WoTSF on the same query ('hotel, room:empty @Egypt-Cairo') at the different stages listed in Figure 6 (a). The time consumed by WoTSF was less than DSE due to the smaller index size, the faster crawling, and most significantly the use of the speed search mode in WoTSF. In stage 5, whereby the search results were limited to the first 100 results, DSE stoped after getting 100 SThs that have probability>50%, reducing the effect of network overhead and reducing the gap between DSE and WoTSF.

5.3 Result accuracy

The WoTSF architecture increases the ability of keeping indices up-to-date, whereby each WoTSE sends only aggregated information to the master index. But, for the sake of measuring result accuracy, we assume that both DSE and WoTSF have up-to-date indices (but of different sizes as indicated in 5.1). Table 1 shows two buildings and the sensor state *SS(occupied) in each* room. Table 2 indicates that when the WoTSF searched only in its high-level indices, the result accuracy was less than DSE (2 *vs.*6 for all results, 2 *vs.* 4 for first-4 results, and 2 *vs.* 3 for results with probability>= 50%).

5.4 Summary

From Table 3, we argue that WoTSF benefits from the distributed query processing and distributed indices [26], and the idea of extensible discovery services [20] for parsing different formats by leaving each WoTSE to parse the formats used in its own network. Consequently, WoTSF reduces the time consumed for crawling, parsing, and indexing.

Table 1. Each Building represents a single WoT network and hosts an occupancy sensor in each room

Building ID	Room ID: Prob.	Building ID	Room ID: Prob.
100	101:0.8	200	201:0.3
	102:0.4		202:0.5
	103:0.7		203:0.4

Table 2. WoTSF and DSE search results (list of rooms) according to different query types.

Search Engine	Number of results: Correct number of results		
	all	*first-4*	Probability >= 50%
WoTSF	2:6	2:4	2:3
DSE	6:6	4:4	3:3

Table 3. A qualitative comparison between WoTSF and DSE.

Criteria	DSE	WoTSF
Main Idea	uses Google SE in its internal search process	works on top of Dyser
Index Size	one record per device	Nearly, one record per WoT network
Granularity of master index	Device (STh) level	Network (e.g., building) level
Case: Speed Search	▪ All available results ▪ More time for ranking. ▪ Accuracy of results depends on indices update.	▪ Results are top values per WoT network. ▪ Faster search ▪ Indices more up-to-date
Case: Accurate Results	▪ Time: searching and ranking time only. ▪ Indices: up-to-date (based on predictions). ▪ Accuracy: high	▪ Time: searching and ranking time + network overheads. ▪ Indices: up-to-date. ▪ Accuracy: high
Pros	▪ Less network overheads. ▪ More accurate results (returns all matching results in each WoT).	▪ Small and semi-dynamic indices. ▪ Less time consumed in crawling, parsing, indexing, searching, and ranking.
Cons	▪ Larger indices. ▪ Harder to keep indices up-to-date. ▪ More time for crawling, parsing, indexing, searching, and ranking.	▪ Tradeoff between search speed and result accuracy

6. CONCLUSION AND FUTURE WORK

Human users prefer searching for high-level states of SThs and EoIs in real-time and in the future. We have proposed a search framework, namely WoTSF, that runs on top of the Dyser search engine [3] for speeding the search process, reducing index size, and efficiently keeping indices up-to-date. In WoTSF, individual distributed search engines (WoTSEs) handle the indexing and crawling of their networks and present aggregate information to the global search engine, whereas a master index

maintains summary information about the distributed search engines. WoTSF reduces the time of the crawling process by limiting it to per WoT-network instead of per device.

WoTSF has limitations. Because only aggregate information is stored at the master index, WoTSF has to call appropriate WoTSEs' APIs in parallel to retrieve fine-grained accurate results. This step would increase the search time due to the network overhead. Further performance analysis can be planned in the future to study this matter to reduce the network overhead. Using semantic technology will be helpful for interoperability. Dyser uses Google for searching for the static part of the query; taking benefit of integrating existing search engines is also a subject of future work. The WoTSF expands the quadruple prediction of Dyser according to the prediction model type. The prediction model type limits the interval between subsequent page crawls (e.g., 7-days). Scheduling the crawling processes to balance network overhead and result accuracy is a subject of future work.

7. REFERENCES

[1] M. Younan, S. Khattab, and R. Bahgat, "An Integrated Testbed Environment for the Web of Things," in The 11th International Conference on Networking and Services (ICNS 2015), ISBN: 978-1-61208-404-6, Rome, Italy, May, 2015, pp. 69-78.

[2] C. Pfister, "Getting Started with the Internet of Things," First Edition ed., B. Jepson, Ed. United States of America.: O'Reilly Media, Inc., May 2011.

[3] B. Ostermaier, K. Romery, F. Mattern, M. Fahrmairz, and W. Kellererz, "A Real-Time Search Engine for the Web of Things," in The 2nd IEEE International Conference on the Internet of Things (IoT), Tokyo,Japan, November. 2010, pp. 1-8.

[4] D. Guinard and V. Trifa, "From the Internet of Things to Web of Things," in MEAP - Building the Web of Things. Manning, 2015, ch. 1, pp. 1-28.

[5] L. Mainetti, V. Mighali, and L. Patrono, "A Software Architecture Enabling the Web of Things," IEEE Internet of Things Journal, DOI: 10.1109/JIOT.2015.2477467, vol. 2, no. 6, pp. 445-454, 2015.

[6] C. Truong, "Routing and Sensor Search in the Internet of Things," PhD Thesis, Institute of Computer Engineering, University of Lubeck, Hanoi, Vietnam, 2014.

[7] S. Mayer, D. Guinard, and V. Trifa, "Searching in a Web-based Infrastructure for Smart Things," in the 3rd IEEE International Conference on .,the Internet of Things (IoT 2012), , Wuxi, China, October 2012, pp. 119-126.

[8] B. Christophe, V. Verdot, and V. Toubiana, "Searching the 'Web of Things'," in the 5th IEEE International Conference on Semantic Computing (ICSC 2011), Stanford University, Palo Alto, CA, USA, 2011, pp. 308-315.

[9] S. K. Datta and C. Bonnet, "Search engine based resource discovery framework for Internet of Things," in the 4th IEEE Global Conference on Consumer Electronics (GCCE 2015), Osaka, Oct. 2015, pp. 83-85.

[10] A. Shemshadi,L. Yao, Y. Qin, Q. Z. Sheng, and Y. Zhang, "ECS: A Framework for Diversified and Relevant Search in the Internet of Things," in Web Information Systems Engineering (WISE 2015), J. W. e. al., Ed. Springer International Publishing Switzerland, 2015, ch. 30, pp. 448-462.

[11] K. Römer, B. Ostermaier, F. Mattern, M. Fahrmair, and W. Kellerer, "Real-Time Search for Real-World Entities: A Survey," in Proceedings of the IEEE Vol. 98, No. 11, SPECIAL ISSUE: Sensor Network Applications, ISSN: 0018-9219, Francisco, November 2010, pp. 1887-1902.

[12] D. Pfisterer, K. Romer, D. Bimschas, H. Hasemann, M. Hauswirth, M. Karnstedt, O. Kleine, A. Kroller, M. Leggieri, R. Mietz, M. Pagel, A. Passant, R. Richardson, and C. Truong, "SPITFIRE: Towards a Semantic Web of Things," Communications Magazine, IEEE, vol. 49, no. 11, pp. 40-48, Nov. 2011, http://dx.doi.org/10.1109/MCOM.2011.6069708.

[13] X. Jin, D. Zhang, Q. Zou, G. Ji, and X. Qian, "Where Searching Will Go in Internet of Things?," in IEEE , 2011.

[14] M. Uddin, "Real-Time Search in Large Networks and Clouds," KTH, School of Electrical Engineering Licentiate Thesis ISBN 978-91-7501-879-9, ISSN 1653-5146, September 2013.

[15] D. Guinard, "A Web of Things Application Architecture - Integrating the Real-World into the Web," PhD Thesis, Computer Science, Eidgenössische Technische Hochschule ETH Zürich, Zürich, 2011.

[16] P. Suganthan G C, "AJAX Crawler," in Data Science & Engineering (ICDSE), International Conference on. IEEE, Cochin, Kerala, July 2012, pp. 27-30.

[17] (2015, Jan.) shodan search engine. [Online]. www.shodanhq.com

[18] A. Santanche, S. Nath, J. Liu, B. Priyantha, and F. Zhao, "SenseWeb: Browsing the Physical World in Real Time," in Demo Abstract, ACM/IEEE IPSN06, Nashville, TN, 2006, pp. 1-2.

[19] umbrellium ltd. (2015, Oct.) Thingful. [Online]. https://thingful.net

[20] S. Mayer, D. Guinard, "An Extensible Discovery Service for Smart Things," in Proceedings of the 2nd International Workshop on the Web of Things (WoT 2011), ACM, San Francisco, CA, USA, June, 2011, pp. 7-12.

[21] C. Zhang, T. Zhang, and M.Wang, "A Distributed Range Query Framework for the Internet of Things," in the 18th IEEE International Conference on Intelligence in Next Generation Networks (ICIN 2015), ISBN 978-1-4799-1866-9, Paris, France, February, 2015, pp. 83-88.

[22] C. Truong, K. Romer, and K. Chen, "Sensor Similarity Search in the Web of Things," in World of Wireless, Mobile and Multimedia Networks (WoWMoM), IEEE International Symposium, San Francisco, CA, June 2012, pp. 1-6.

[23] M. Younan, S. Khattab, and R. Bahgat, "Evaluation of an Integrated Testbed Environment for The Web of Things," in International Journal On Advances in Intelligent Systems (IntSys 2015), vol. 8, no. 3&4, pp. 467-482, Nov. 2015.

[24] Dr. M. Elkstein. (2014, Nov.) Learn REST: A Tutorial. [Online]. http://rest.elkstein.org/2008/02/what-is-rest.html

[25] Google. (2010, Jan.) Search Engine Optimization (SEO) - Starter Guide.

[26] M. Hammoud, D. A. Rabbou, R. Nouri, S. beheshti, and S. Sakr, "DREAM: Distributed RDF Engine with Adaptive Query Planner and Minimal Communication," in the 41st International Conference on Very Large Data Bases, Kohala Coast, Hawaii., 2015, pp. 1-12.