Content-Based Sensor Search for the Web of Things

Cuong Truong, Kay Römer
Institute of Computer Engineering, University of Lübeck, Germany
{truong, roemer}@iti.uni-luebeck.de

***Abstract*—In the emerging Web of Things (WoT), the real-time state of real-world objects is published by Internet-connected sensors using Web technologies. A key service in the WoT will be a search engine that allows searching for real-world objects with a certain state. We propose content-based sensor search, where given a search query that is composed of a range of sensor values, sensors that have been reading values in that range at the time of the query are found. In this paper, we present an approach based on fuzzy sets to realize scalable content-based sensor search in the WoT. Using sensor data sets obtained from real deployments, we find that our approach results in a low communication overhead. To demonstrate the practical feasibility of our approach, we implemented a prototypical search engine that enables the user to search for sensors that are available on the Web.**

I. Introduction

We are witnessing the formation of a Web of Things (WoT) where sensor- and actuator-augmented real-world objects are being connected to the Internet using standard communication protocols, allowing them to publish their real-time state on the Web, making them observable and controllable with minimal delay using state-of-the-art Web technologies. For example, services like Xively[1] and Microsoft SenseWeb [1] offer APIs for publishing structured sensor data that can be viewed with a Web browser, or ThingSpeak[2] allows users to interact with actuators (e.g., switch a light on/off) using Twitter messages. The WoT, therefore, extends the traditonal document-centric Web to be a universal interface for the real world by giving real-world entities a Web representation that can be accessed using lightweight APIs (e.g., based on the REST principles).

As in the traditional Web, we believe that a key service in the WoT will be a search engine that allows to search for real-world entities with a certain state. Those states are observed by sensors and change very frequently. This dynamic nature of the states of real-world entities makes current search engines designed for the traditional Web not suitable for the WoT, as they are usually designed for relatively static content and meta-data tags of Web documents (e.g., Web pages, video and audio titles and tags, etc). Moreover, the anticipated huge number of sensors to-be-connected to the WoT [2] implies that a search engine that needs to communicate with all available sensors in the WoT when processing a user's search query is impratical due to the tremendous communication overhead incurred.

To reduce communication overhead, we propose to construct a *prediction model* from past data of each sensor in the WoT, and index this model in a distributed database system in the Internet, that can be accessed by the search engine with low overhead. These prediction models, in response to a search query, will be used to estimate the probability that a given sensor has recently been producing sensor measurements that match the query. The search engine computes a *rank list* of sensors sorted in descending order of these probabilities and contacts top-ranked sensors to check if their recent output actually matches the query, until a sufficient number of matches has been found, and presents these matches to the user as search result. This way we can drastically reduce not only communication, but also time overhead to process the query. To adapt to the frequent changes of sensor data, we propose to periodically construct a new prediction model of an indexed sensor. The newly constructed prediction model is then merged with the existing one in such a way that recent changes in sensor data are reflected, and at the same time, the information from past sensor data is also preserved according to a certain *fading factor*. We call this process the *adaptation process*.

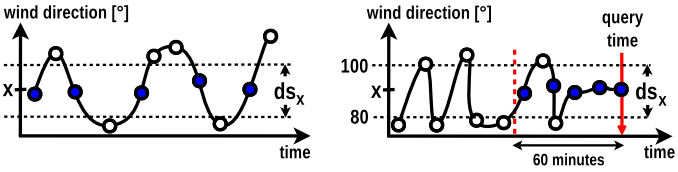

Fig. 1: Measurements of wind direction.

A further challenge for content-based search is that sensor measurements usually fluctuate over time as the monitored physical processes change often quickly but marginally. Hence, searching for sensors based on a given single value at an instant of time is of limited benefit. Fig. 1-right illustrates this with a wind sensor that measures the wind direction (measurements falling within [80°, 100°] are depicted as filled circles, whereas ones do not are not filled). Due to the nature of the wind, direction measurements may change rapidly over time, making it hard to tell which direction the wind is blowing by just considering the latest sensor measurement. However, if we consider sensor measurements during the last 1 hour, statistics show that the wind has mainly blown towards the North as measurements fell mostly within 80° and 100°. Thus, a human would say that the wind has been blowing north (for the past hour). Inspired by this observation, we propose content-based search for sensors that *have been producing sensor measurements in a certain range of values for a certain amount of time prior to the query*. In particular, the majority of the sensor measurements in that time interval should fall into the desired value range.

The contribution of this paper is three-fold: (i) we propose content-based search for the WoT and present an architecture to realize this service; (ii) we design a light-weight prediction model based on fuzzy logic that estimates the probability that a sensor is matching a search query; (iii) we evaluate our approach using data sets from real-world deployments and find that our approach has low communication overhead and is computationally efficient.

[1]https://xively.com
[2]https://www.thingspeak.com

978-1-4799-1353-4/13/$31.00 ©2013 IEEE

The novelty of our work, firstly, lies in supporting search based on raw sensor data which requires no domain expertise for deriving high-level states from raw sensor data and gives users the flexibility of defining their own application-specific search. Secondly, we addresse the severe resource limitations of sensor node platforms. These nodes often have to run for months to years on a single battery, therefore power-intensive wireless communication has to be kept to the absolute minimum. In addition, these platforms are typically based on a simple microcontroller with very constrained computational and storage capabilities. Our approach based on fuzzy sets meets these requirements, and at the same time, thirdly, deals with the imperfections of sensor data time series captured by low-cost hardware as well as, fourthly, scales to a very large number of sensors in a future WoT.

II. Related Work

Existing approaches for searching sensors in the WoT are mainly based on textual descriptions of sensors or similarity between sensor data streams. Examples of the former are Snoogle [3], Microsearch [4], Xively, and SenseWeb [1], that allow the user to find real world objects by posing a query that contains a list of keywords such as "room" or "book". Examples of the latter are [5], [6], [7], and [8], that find sensors that produce sensor data streams similar to a given sensor data stream. Althouth these works consider fundamentally different problems than content-based search, they are related to our work in that they also search for sensors in the WoT.

The systems mentioned above cannot be used to search for states of real world entities, e.g., "hot", "empty", which are dynamic and depend on the recent measurements of embedded sensors. For example, we could find a "room" but cannot find a "room" that is currently "empty". Our previous work in [9] and [10] addresses this issue by allowing a user to search for sensors that output a given value at the query time. The key idea there is to exploit the periodicities in sensor output (e.g., a meeting room is occupied every Monday from 8 to 10), or correlations between sensors (e.g., parking spots close to the entrance of a building are often all occupied, whereas spots further away are often free) to build prediction models that predict which sensors would output the sought value at the time of the query.

The limitation of our previous work is two-fold: (i) It assumes that there is a class of sensors that exhibits a high degree of periodicity (e.g., people-centric sensors) within a considered time window. For example, the measurements of ocupancy sensors monitoring a lecture hall exhibit a dominant periodic pattern of the lecture hall being "occupied" from Moday to Friday and being "free" in the weekend; and (ii) The "sensor output value" considered is not a raw sensor value but a high-level state derived from raw sensor measurements (e.g., "free" or "occupied" correspond to sensor measurements being lower than or higher than a certain threshold, respectively). This approach does not only require proper domain expertise to derive those states from raw data, but also confines the search engine to specific applications.

Our approach is different as we support search for raw sensor data which requires no domain expertise and gives users the flexibility of defining their own application-specific search. Furthermore, we make no assumption regarding periodicity in sensor data.

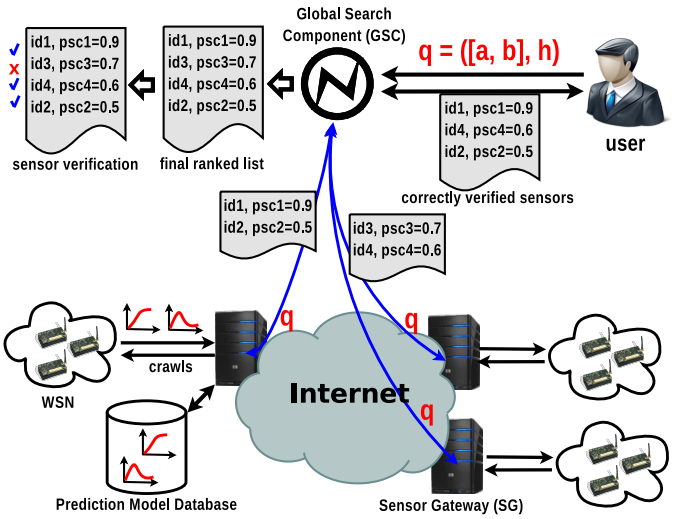

Fig. 2: Content-based Sensor Search: Architecture.

III. Architecture

In this section, we describe the architecture of our approach for addressing the content-based sensor search problem (see Fig. 2). We have multiple *sensor gateways (SG)* distributed across the Internet, each is responsible for one or more sensor networks. Each SG is also associated with a *prediction model database*. The network of SGs forms a distributed prediction model database system in the Internet.

We construct, from the output measurements of a sensor over a time window w, a *prediction model* which is computed by the sensor node itself. Periodically each w time units, the SG crawls sensors to download and index those prediction models in the distributed prediction model database. Each prediction model has a memory footprint of few tens of bytes and can thus be efficiently downloaded from the sensors.

To perform a content-based search, the user specifies a *search query* $q = ([a, b], h)$, which means that the user wants to search for sensors in the WoT that have been reading values in the range $[a, b]$ over the time window $[t_q - h, t_q]$ (t_q is the time at which q is submitted), and hands q to the *global search component (GSC)* via a user interface (GUI). As all SGs register with the GSC, the latter will forward q to relevant SGs, where q is used to compute a prediction score psc for each indexed sensor, which is an estimated probability that the sensor matches q. After prediction, a SG forms a list of sensors ranked by the psc and sends the list to the GSC.

After receiving all partial rank lists from SGs, the GSC merges them together to form the *final rank list* that is sorted in descending order of psc. The GSC, then, processes the final rank list from top to bottom to verify if sensors are actually matching q by communicating the sensors through the SGs and asking if they have been reading in the range $[a, b]$ over the time interval $[t_q - h, t_q]$. The verification process stops when a k actual matching sensors have been found. The GSC presents these k matching sensors, together with their prediction score, to the user as the search result.

The proposed architecture scales to large networks in terms of computation and storage overhead as prediction models are computed by sensor nodes and are stored in distributed SGs,

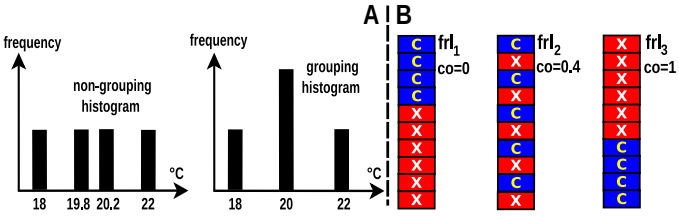

Fig. 3: A: Temperature histogram; B: Comm. overhead

and the resolution of a search query is performed in parallel by mulitple SGs.

IV. Content-based Search: A Fuzzy-based Approach

Fuzzy set theory [11] deals with uncertainties using approximate rather than fixed and exact reasoning, to arrive at definite decisions. The imperfectness and non-uniformity of raw sensor data, thus, makes fuzzy sets a natural approach for our problem. A fuzzy set is an extension of a crisp set which allows partial membership rather than only binary membership. A fuzzy set F on a universe of discourse U is defined by its membership function $mf_F(x), x \in U$ such that $mf_F(x) \in [0, 1]$. The closer the value of this function is to 1 for a given x, the more x belongs to the fuzzy set F. With this definition, an element $x \in U$ can be *member of more than one* fuzzy set at a time, with *different degrees* of membership.

The design of our prediction model is based on this key concept of fuzzy set theory. In this work, we present the time-independent prediction model (TIPM), which consists of two components, namely *sensor measurement density* and *sensor measurement stability*. For the sake of explanation, we consider a temperature sensor S monitoring a room, whose time series of measurements over the time period $[t_c - w, t_c]$ is $ts_S = S(t_1), S(t_2), .., S(t_n)$, where $S(t_i)$ is S's measurement at time t_i, t_c is the construction time of the model, and w is a given time window. We denote x^S_{min} and $x^S_{max} \in ts_S$ as the minimum and maximum values among the measurements of sensor S over w, respectively.

A. Sensor Measurement Density

To estimate the probability that a given value is currently read by a sensor, a histogram of the past sensor measurement distribution could be used. For example, by looking at the histogram of measurements of a temperature sensor monitoring a room shown in the right of Fig. 3-A, we notice that the room temperature was mainly 20°C during last 24 hours. Since we are predicting if S is reading a value x, it can be assumed that the more times S read x in the past, the higher the probability it is reading x at the current moment. Thus, a histogram of every temperature readings of S in the last w time units helps predicting current readings of S.

In reality, however, a simple histogram over sensor measurements is of limited benefit. The reason is that sensor measurements are inherently imperfect due to the presence of jitter and noise. Moreover, we, as human beings, usually are not able to differentiate between slight changes in the state of a physical entity. For example, temperature values of 19.8° and 20.2° are, for many practical purposes, indistinguishable from 20°C by a human user (who is, for example, searching a well-tempered room).

Fig. 3-A illustrates this issue. In the left of Fig. 3-A, the frequency of 18°, 19.8°, 20.2°, and 22° are the same, which means all those temperature values would be predicted with the same probability. However, if we group temperature measurements that are close to each other and compute a histogram value for the group as in the right of Fig. 3-A where 19.8° and 20.2° are grouped, we can say that a temperature value close to 20°C is being read now with high probability as the frequency of values in the small range around 20° is twice as high as the frequency of the small ranges around 18° and 22°.

Inspired by this observation, our approach is not to count the exact number of occurences of distinct sensor measurements x, but to consider the *density* of sensor measurements *surrounding* x. Considering a real value $r_d \in (0, \frac{x^S_{max} - x^S_{min}}{2}]$ and the interval $dd_x = [x - r_d, x + r_d] \subset [x^S_{min}, x^S_{max}]$, the size of the population of sensor measurements in dd_x is proportional to the probability that x or sensor measurements y that are r_d-close to x (i.e., $|x - y| \leq r_d$) will be read by S in the future. By sliding dd_x over $[x^S_{min}, x^S_{max}]$ we can calculate this density for each sensor measurement x in the measurement range $[x^S_{min}, x^S_{max}]$.

We evaluate the density of sensor measurements within dd_x around a measurement x during w by calculating the distances between x and every sensor measurement $y \in [x^S_{min}, x^S_{max}]$, computing these distances' weight as the exponential decay function (i.e., values with larger distances have an exponentially smaller weight), and sum up the computed weights. We then normalize the sum to $[0, 1]$ by dividing it by the number of sensor measurements $|ts_S|$. Formally, the density md^w_S for a sensor measurement x is given by

$$md^w_S(x) = \frac{1}{|ts_S|} \sum_{i=1}^{|ts_S|} e^{-\frac{|x - S(t_i)|}{r_d}} \quad (1)$$

Due to the exponential function, sensor measurements which are outside of $[x - r_d, x + r_d]$ have little influence on $md^w_S(x)$. We call $md^w_S(x)$ the *sensor measurement density* function. Note that, $md^w_S(x)$ is exactly the histogram value of x when $r_d = 0$. However, as mentioned above, r_d should be greater than zero to reflect human perception as well as the jitter and noise of raw sensor data.

B. Sensor Measurement Stability

We consider search queries of the form $q = ([a, b], h)$, which implies searching for sensors whose readings fall within $[a, b]$ over the time interval $[t_q - h, t_q]$. This reflects human perception that a physical entity is at a certain state, meaning it has been in that state for a certain period of time. A change of the perceived state of the entity normally corresponds to a significant change of measured sensor values from one range to another distinct or overlapping range. For example, in Fig. 1-right, if you are standing at the northern part of a hill and feeling that the wind blows towards you, that means the wind has been lately blowing north. In other words, the wind direction measurements *almost continuously* and *closely one after the other* fall in [80°, 100°] in the last 60 minutes. Although you may notice that there are points in time when the wind did not blow north, *"almost"* all of the time during the last hour it did. We call this the *stability* of the wind direction, or more general, the stability of a monitored physical process. It can be assumed that the more stable a physical process was in the past, the higher probability that it is stable now. Thus,

information of how stable the physical process was in a range $[a, b]$ in the past w time units helps predicting its stability in $[a, b]$ over the time interval $[t_q - h, t_q]$.

We formally express stability by considering our sensor S. We model a certain "perceived temperature" x of the room as the range $ds_x = [x - r_s, x + r_s]$, where $r_s \in [0, \frac{x^S_{max} - x^S_{min}}{2}]$. We are interested in how continuous in time are the temperature measurements that fall within $[x - r_s, x + r_s]$ during w. The closer in time those measurements are sampled one after another, the more stable we consider that room temperature is at value x during w. An illustration of stability is given in Fig. 1, where the distribution of 5 sensor measurements in Fig. 1-right is considered more stable than the distribution of other 5 sensor measurements in Fig. 1-left.

We denote the time series of temperature measurements that fall in ds_x during w as

$$ts^x_S = S(t_1), S(t_2), ..., S(t_n) | S(t_j) \in ts_S, j = 1..n \quad (2)$$

where $n = |ts^x_S|$ is the number of temperature measurements found between $[x-r_s, x+r_s]$, t_j is the time at which $S(t_j)$ was sampled. Note that the set of $\{t_j\}$ is a subset of the set of time stamps $\{t_i | i = 1..|ts_S|\}$, and we assume that $\forall j, t_j < t_{j+1}$.

We compute the stability ms^w_S of sensor measurements around x during w by summing up the exponentially weighted distances of adjacent timestamps t_j and t_{j+1} (i.e., larger distances have an exponentially smaller weight). This way, distributions like the one given in Fig. 1-right result in a much larger stability value than the one in Fig. 1-left, due to the exponential weighting. We then normalize the sum to [0, 1] by dividing it by the number of evaluated distances. Formally we obtain

$$ms^w_S(x) = \frac{1}{n-1} \sum_{j=1}^{n-1} e^{-\frac{t_{j+1} - t_j}{t_n - t_1}} \quad (3)$$

By sliding ds_x over $[x^S_{min}, x^S_{max}]$, we can calculate a stability value for every sensor measurement $x \in [x^S_{min}, x^S_{max}]$. We call $ms^w_S(x)$ the *sensor measurement stability* function.

C. TIPM Construction and Query Resolution

Our prediction model is based on the density and stability of sensor values as introduced in the previous sections. In particular, the higher the product of the density and stability values of x, the greater the probability that S is currently reading x. Formally, we obtain the prediction model function $pm^w_S(x)$ for the time series ts_S during w:

$$pm^w_S(x) = md^w_S(x) \times ms^w_S(x) \quad (4)$$

With this TIPM defined, if we search for sensors that are reading x, those sensors S that have highest values of $pm^w_S(x)$ will be ranked highest. If we search for sensors that are reading either x or y, those S that have highest values of $pm^w_S(x) + pm^w_S(y)$ will be ranked highest. In general, if we search for sensors reading a value in the range $[a, b]$, those S that have highest values of $\sum_{x \in [a,b]} pm^w_S(x)$ will be ranked highest. To avoid performing this summation for each query, we precompute and store the cummulative distribution function of $pm^w_S(x)$:

$$cdfpm^w_S(x) = \sum_{x^S_{min} \leq y \leq x} pm^w_S(y) \quad (5)$$

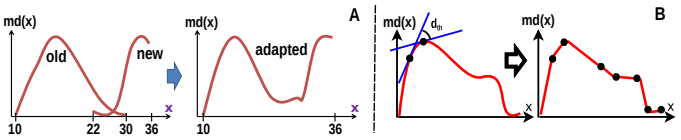

Fig. 4: TIPM Adaptation and Approximation.

For query resolution, once a search query is directed to an SG, all indexed sensors V with $x^V_{max} < a < b$ and $a < b < x^V_{min}$ are ruled out to narrow down the search space. For the remaining sensors, we calculate for each sensor V its prediction score as the sum of the estimated probabilities for every $x \in [a, b]$ that x is being read at the query time. Since we already have the precomputed distribution function (Eq. 5), the prediction score psc^q_V is obtained by simply substracting the values of the cummulative fuction for b and a, and normalizing the value to the range $[0, 1]$ by dividing it by the total cummulative probability of V:

$$psc^q_V = \frac{cdfpm^w_V(b) - cdfpm^w_V(a)}{cdfpm^w_V(x^V_{max}) - cdfpm^w_V(x^V_{min})} \quad (6)$$

The SG, then, sends the resulted rank list of sensors V and their psc^q_V to the GSC for merging and query verification.

D. TIPM Adaptation

The TIPM of an indexed sensor should be regularly updated to reflect significant changes in the sensor's ouput. However, as we are using past information of the monitored physical entity to predict its current behaviour, it is desired that this information is not totally discarded, but gradually fades away as time goes by. If we look at Eq. 1 and Eq. 3 again, we can see that the density and stability functions are additively constructed over the sensor's measurement range. Thus, we employ the exponentially weighted moving average technique to implement the adaptation process. A function is adapted by combining the old and the new versions of the function over the union of the two measurement ranges. Formally we have

$$md^w_S(x) = (1 - \alpha) \times md^w_S(x)^{old} + \alpha \times md^w_S(x)^{new} \quad (7)$$

$$ms^w_S(x) = (1 - \alpha) \times ms^w_S(x)^{old} + \alpha \times ms^w_S(x)^{new} \quad (8)$$

where the old and new indicators represent old and new density and stability functions of sensor S, respectively. The factor α is a *fading factor* assigned to the old and new functions of S's prediction model. Through α we can control how "important" are the recent measurements of sensor S in comparison with S's measurements in the more distant past.

As time goes by, the resulting union of measurement ranges describes the value domain whereas the adapted function predicts the sensor output. For example, in Fig. 4-A, the combined value domain of the density function of the sensor in consideration is $[10, 36]$, which is the union of the two value domains $[10, 30]$ and $[22, 36]$.

E. TIPM Approximation

Since the storage overhead for a TIPM of sensor S is proportional to the size of its time series ts_S, communication and storage of TIPM may be expensive. We observe that the curves of the component functions (i.e., density and stability) of TIPM are typically smooth due to the exponential weighting.

Thus, we propose to represent these component functions by a set of line segments that approximate their curves. As each line segment can be defined by two float values, the memory footprint is small and typically in the order of few tens of bytes. We then perform Eq. 4 and Eq. 5 based on the approximated density and stability functions.

We use the density function $md_S^w(x)$ for explanation (the same principle applies for the stability function as well). The illustration is given in Fig. 4-B, where in the left we have the TIPM's density function whose linear approximation is shown in the right side of the figure. We first define a derivative threshold d_{th}, compute $md_S^w(x)$'s first discrete derivative d_1 at x_1 (x_1 is the second smallest value in the measurement range of S), and mark the point $A_1 := (x_1, md_S^w(x_1))$. We then iterate over points $(x_i, md_S^w(x_i))$ on the curve and compute $md_S^w(x)$'s first discrete derivative d_i, until $d_i - d_1 > d_{th}$. We assign $x_2 := x_i$, $A_2 := (x_2, md_S^w(x_2))$, and store the line A_1A_2 as the approximation of $md_S^w(x)$ for the interval $[x_1, x_2]$. After that, we assign $A_1 := A_2$ and $d_1 := d_i$, and continue to iterate over points on the curve in the same fashion until we reach the point $(x_{max}^S, md_S^w(x_{max}^S))$. The resulting set of line segments is the desired approximation of $md_S^w(x)$.

V. Evaluation

In this section we evaluate the performance of our content-based search engine in terms of communication overhead using simulation. Note that the search results are always accurate, as the search engine verifies that sensors actually match the search query. Therefore, a rank list only influences the communication overhead but not the accuracy.

A. Communication Overhead of a Rank List

Since sensor verification requires communication between the GSC and the sensor nodes, the optimal rank list would have all matching sensors ranked on top as that would minimize the number of sensors that must be verified until a k matching sensors are found, thus minimizing communication overhead. Fig. 3-B shows possible final rank lists obtained after q is evaluated. The character "C" indicates matching sensors, while character "X" indicates non-matching ones. The best possible result is list frl_1 as all matching sensors are ranked highest. The worst result is list frl_3.

We now define a metric that maps a rank list to a scalar value between 0 (best result) and 1 (worst result). This value is proportional to the number of sensors that need to be verified and thus communication overhead. For each matching sensor, we compute the ranking error, i.e., the number of non-matching sensors ranked higher. We then compute the average ranking error of all matching sensors, which equals 0 in the best case, and equals the number of non-matching sensors in the worst case, and divide it by the number of non-matching sensors to obtain the desired metric. We define the *communication overhead* as follows:

$$co(rl_q) = \frac{1}{m(rl_q)\left[|rl| - m(rl_q)\right]} \times \sum_{i=1}^{|rl|} e_{rl_q}(i) \qquad (9)$$

where $|rl_q|$ is the length of rank list rl_q, $m(rl_q)$ is the number of matching sensors in rl_q, and $e_{rl_q}(i)$ is the ranking error of a matching sensor at rank i, i.e., the number of non-matching sensors ranking higher than i. If i is a non-matching sensor, then $e_{rl_q}(i) := 0$.

B. Simulation Setup

We use several sensor data sets that were recorded from different real world sensor network deployments for our evaluation. For each data set, we select the size of w for constructing TIPM to be 24 hours. This is a natural choice as the state of physical entities tends to repeat day after day. We simulate the process of a human posing search queries by a Poisson process with $\lambda = 144$ over 24 hours (i.e., on average 1 query each 10 minutes).

When there is a query $q = ([a, b], h)$, the range $[a, b]$ is randomized within $[A, B]$ where A and B are the minimum and maximum values of the sensor readings of all sensors in the data set, respectively. The time interval h is fixed at 60 minutes. The query time t_q is randomized between $[0, w]$. For each search query, we obtain a rank list whose communication overhead is determined by Eq. 9. The average of the communication overhead of all queries posed within a day, i.e., 24 hours, is used as the communication overhead of our search engine during that day. At the end of each simulated day, sensors compute the density and stability functions of their prediction model and send them to the SG for adaptation and indexing. The simulation is run until a sensor of the data set has reached its last measurement. We repeat the simulation for 100 times and compute the average as well as the standard deviation of the communication overhead of our search engine.

We use 4 data sets for our evaluation. They are IntelLab[3] containing 59 humidity and temperature sensors, NOAA[4] containing 80 air and water temperature sensors, MavPad[5] containing 23 motion and humidity sensors. The last data set is the combination of these three data sets. Evaluation on this combined data set gives us an idea of how our search engine would perform in real life where sensors have different types and diverse deployment contexts.

C. Tunable Parameters

The fading factor α is used to control the importance that we assign to recent sensor data with respect to the data from the more distant past. We conducted evaluations with varying values of α to investigate the effect of this parameter on the average communication overhead.

As the result for a search query q, we receive a rank list rl_q sorted by prediction scores psc (see Eq. 6). We denote psc_{min} as the minimum psc of all sensors in rl_q. We would expect that the larger the number of sensors available in the IoT, the more likely sensors exist that match a query, and the more likely we obtain a rank list with a large psc_{min} value. However, in our experiments we only have a limited number of sensors (162 sensors). To get a feeling of how the TIPM would perform with a larger number of sensors (and thus with a higher psc_{min}), we also evaluate the communication overhead of those rl_q whose psc_{min} is greater than a threshold value. We performed evaluations with different threshold values and found that average communication overhead decreases when we increase the threshold value (simulating an increasing number of sensors in the IoT).

D. Numerical Results

In the following, we discuss the evaluation results in terms of average communication overhead (ACO) for different values

[3] http://db.csail.mit.edu/labdata/labdata.html
[4] http://tidesandcurrents.noaa.gov/gmap3
[5] http://ailab.wsu.edu/mavhome/index.html

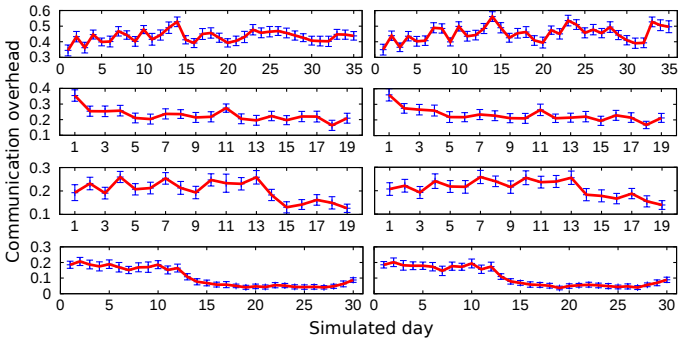

Fig. 5: Communication overhead with $\alpha = 0.3$ (left column) and $\alpha = 0.8$ (right column), for MavPad, IntelLab, Combined, NOAA data sets (from top to bottom)

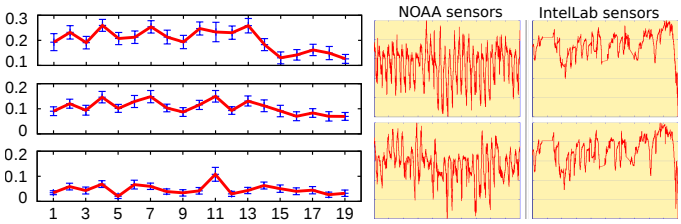

Fig. 6: Left: ACO for thresholds on psc_{min}: 0, 0.2, 0.5 (from top to bottom); Right: Heterogeneity in time series.

of α and psc_{min}.

Fig. 5 shows the ACO for 4 data sets with two different values of α ($psc_{min} = 0$). From top to bottom are the results for MavPad, IntelLab, Combined, and Noaa data sets. In the left and right columns are the results for $\alpha = 0.3$ and $\alpha = 0.8$, respectively. We can see the common trend that the ACO decreases over time, which reflects the effect of our TIPM and adaptation process. The best case is the results for NOAA data set where the ACO from day 15 on is around 0.05, which is very low. The worst case is the MavPad data set with the ACO of about 0.45, which is still low.

The reason for the ACO difference among data sets is, probably, due to the nature of sensor data in different data sets. For example, the Intel Lab is a small, closed, and mostly static environment, so we expect a low level of heterogeneity among the time series of sensors in the IntelLab data set. In contrast, weather sensors of the NOAA data set are spread over a large geographical area along the coast line in the northern part of America, resulting in a high level of heterogeneity in their time series. Thus, we expect that searching in the NOAA data set would more likely result in rank lists containing at least on sensor with high psc value. Fig. 6-right illustrates the low level of heterogeneity in the time series of 2 humidity sensors in the IntelLab data set while the opposite is observed in the time series of 2 air temperature sensors from the NOAA data set.

A closer look at the results reveals that $\alpha = 0.3$ results in slightly lower overall ACO than does $\alpha = 0.8$. This confirms our approach that not only recent historical sensor data but also data from the more distant past should be used for prediction (via the adaptation process).

Fig. 6-left shows how the choice of the threshold on

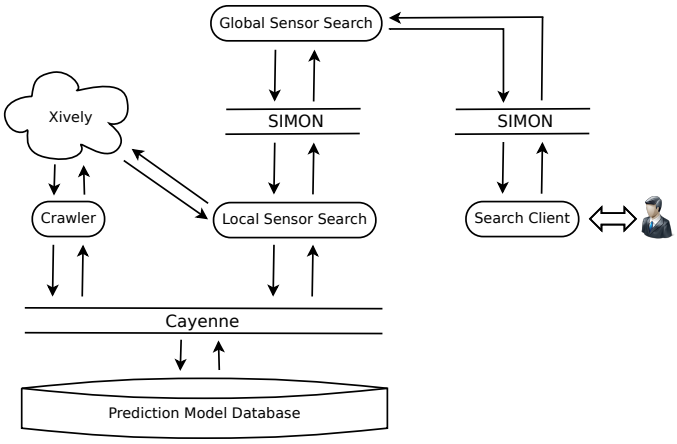

Fig. 7: Content-based sensor search engine: Component layout.

psc_{min} affects ACO. Due to limited space, we only present here the results for the combined data set with different threshold for psc_{min} ($\alpha = 0.3$). For other data sets, similar results were obtained. From top to bottom are results for psc_{min}=0, psc_{min}=0.2, and psc_{min}=0.5. We see that the ACO decreases when we increase the threshold value for psc_{min}, which confirms that the proposed TIPM incurs low communication overhead. This hints that the communication overhead decreases as the number of sensors in the IoT increases. The reason is that in a real IoT, sensors usually have highly different types and diverse deployment contexts, thus resulting in highly heterogenous time series of sensor data. Therefore, the probability to obtain rank lists with small psc_{min} decreases with increasing number of sensors.

VI. Prototypical Search Engine

To demonstrate the practical feasibility of our content-based sensor search algorithm, we implemented (using the Java programming language) a prototypical search engine that enables users to search for sensors that are available on Xively[6]. Xively is an IoT-focused cloud service that allows users to connect sensors to and publish sensor measurements on the Web, and to build their own IoT applications by mashing up these sensors, sensor measurements, Web data and services using the Xively API. In our current implementation, all components reside at and run on the same computer that has 4 GB RAM and an Intel Core 2 Quad Q8200 CPU. However, they can be as well placed on different computers across the Internet by simply modifying a config file without touching the code.

A. Component Layout

Fig. 7 shows the components of the search engine and the communication among them. Periodically, the Crawler requests Xively to discover new sensors, download sensor data for both new and indexed sensors, construct prediction models from this data, index, and update them in the prediction model database. The Local Sensor Search (LSS) is responsible for resolving search queries, building partial rank lists, and verifying if sensors on Xively actually match search queries as requested by the GSS.

The Prediction Model Database (PMDB) organizes storage and access for data generated by the Crawler and LSS. In order

[6]https://xively.com

Fig. 8: Content-based sensor search engine: Screenshot.

to avoid dependence on any particular database engine, we use Cayenne[7] as an abstraction layer for managing and accessing a database. Cayenne is an ORM (Object Relational Mapping) framework that allows for seamlessly binding database schemas directly to Java objects, managing atomic database transactions (e.g., commit and rollbacks, joins, sequences, etc), thus enabling programmers to work only with Java objects abstracted from a database. The underlying database engine that we use for PMDB is MySQL. At the time of writing, our PMDB contains already more than 20000 sensors (i.e., their URL, meta-information, and a TIPM).

The Search Client (SC) provides users with a GUI for creating content-based sensor search queries and for presenting the users the search results.

The Global Sensor Search (GSS) is responsible for accepting search queries from SC, forwarding them to LSS, and forwading the resulting rank list from LSS to SC. The communication between GSS and SC as well as between GSS and LSS is implemented using SIMON[8], a tool that provides remote method invocations. With SIMON, we can access Java objects that are located in another JVM on the same computer or on another server computer somewhere on the Internet.

B. Demonstration

Fig. 8 shows a screenshot of our content-based sensor search engine in action. To execute a search, we create a content-based search query by specifying $a = 10$, $b = 20$, and $h = 60$ for 60 minutes, and click the "Start" button. After few seconds, the final rank list of sensors is displayed in the "Results" window. For each matching sensor, its measurements are plotted and its URL and metadata are shown. As we observe in the figure, the found sensors match our specified search query.

[7]http://cayenne.apache.org

[8]http://dev.root1.de/projects/2/wiki

VII. Conclusion and Future Work

We developed a novel search service for finding sensors in the Web of Things. The novelty of our search service lies in supporting search based on raw sensor data which requires no domain expertise for deriving high-level states from raw sensor data and gives users the flexibility of defining their own application-specific search. The proposed hierachical architecture scales well with the number of sensors in the IoT as computations and storage are distributed over multiple SGs. Moreover, since only compact prediction models are periodically downloaded from sensor nodes, wireless communication is minimized which is crucial for maximizing life time of battery-powered sensor nodes. At the same time, the fuzzy approach addresses the imperfections of sensor data obtained from low-cost sensor hardware. We implemented a prototypical content-based sensor search engine that enables users to search for sensors that are available on the Web, demonstrating the practical feasibility of the proposed search service.

As future work, our search service will be combined with other complementary sensor search systems (see Sec. II) to use "context" information such as type, location, and deployment context of sensors to provide users with better search results. We also plan to further improve the scability of our search service by improving the indexing and lookup algorithms for TIPMs.

VIII. Acknowledgement

This work has been partially supported by the European Union under contract number ICT-2009-258885 (SPITFIRE). The prototypical search engine was implemented by our colleague Kris Erik Schwerdt[9] at the University of Lübeck, Germany.

References

[1] A. Kansal, S. Nath, J. Liu, and F. Zhao, "Senseweb: An infrastructure for shared sensing," *IEEE Multimedia*, 2007.

[2] D. Evans, "The internet of things: How the next evolution of the internet is changing everything," *White Paper, Cisco Internet Business Solutions Group*, April, 2011.

[3] H. Wang, C. C. Tan, and Q. Li, "Snoogle: A search engine for pervasive environments," *IEEE Trans. on Parallel and Distributed Systems*, 2010.

[4] C. C. Tan, B. Sheng, H. Wang, and Q. Li, "Microsearch: When search engines meet small devices," *Pervasive*, 2008.

[5] L. E. Holmquist, F. Mattern, B. Schiele, P. Alahuhta, M. Beigl, and H.-W. Gellersen, "Smart-its friends: A technique for users to easily establish connections between smart artefacts," *Proc. Ubicomp*, 2001.

[6] J. Lester, B. Hannaford, and G. Borriello, ""are you with me?" - using accelerometers to determine if two devices are carried by the same person," *In Proc. 2nd Int. Conf. Pervasive Computing*, 2004.

[7] M. Gauger, O. Saukh, M. Handte, and P. J. Marron, "Sensor-based culstering for indoor applications," *SECON'08*, 2008.

[8] C. Truong, K. Römer, and K. Chen, "Fuzzy-based sensor search in the web of things," *Proc. Intl. Conf. on Internet of Things*, 2012.

[9] B. M. Elahi, K. Römer, B. Ostermaier, M. Fahrmair, and W. Kellerer, "Sensor ranking: A primitive for efficient content-based sensor search," *IPSN '09, San Francisco, CA, USA*.

[10] D. Pfisterer et al, "Spitfire: Towards a semantic web of things," *IEEE Communications Magazine*, Nov 2011.

[11] L. A. Zadeh, "Outline of a new approach to the analysis of complex systems and decision processes," *IEEE Trans. on Sys., Man and Cybern.*, vol. SMC-3, 1973.

[9]kriserik@quantentunnel.de