Sensor Similarity Search in the Web of Things

*Cuong Truong, *Kay Römer, †Kai Chen

Institute of Computer Engineering, University of Lübeck, Lübeck, Germany
*{truong, roemer}@iti.uni-luebeck.de †adam808@zju.edu.cn

***Abstract*—An increasing number of sensors is being connected to the Internet and their output is published on the Web, resulting in the formation of a Web of Things (WoT) that will soon connect tens of Billions of devices. We propose sensor similarity search, where given a sensor, other sensors on the WoT are found that produced similar output in the past. At the heart of our approach is an algorithm that exploits fuzzy sets for efficiently computing a similarity score for a pair of sensors that is used to obtain a ranked list of matching sensors. Using sensor data sets from real deployments, we find that this approach results in a high accuracy.**

I. Introduction

A steadily increasing number of sensors worn by people (e.g., contained in mobile phones), embedded into the environment (e.g., sensor networks), and into objects (e.g., smart objects and appliances) are being connected to the Internet. This trend is leading to the formation of an Internet of Things (IoT) that is expected to interconnect Billions of devices by 2020 [1]. By publishing the resulting sensor data streams in the Web, novel real-world applications can be created by mashing up sensors and actuators with services and data available on the Web, leading to a Web of Things [2].

As in the traditional web, search will be a key service also in the WoT to enable users to find sensors with certain properties. Existing directories of online sensors such as Pachube[1], GSN [3], or Microsoft SensorMap [4] support search for sensors based on textual metadata that describes the sensors (e.g., type and location of a sensor, measurement unit, object to which the sensor is attached) and which is manually entered by the person deploying the sensor. Other users can then search for sensors with certain metadata by entering appropriate keywords.

Unfortunately, this approach does not work well in practice, as humans make mistakes when entering metadata, different users use different terms to describe the same concept, or important metadata is not entered at all. For example, in [5] a user study is described where 20 participants were asked to enter metadata for a wheather station sensor using a simple user interface. Those 20 persons made 45 mistakes in total.

There are several approaches to address this problem. Firstly, some metadata of a sensor such as sensor type can be stored on the sensor during production using so-called Transducer Electronic Data Sheets (TEDS) as defined by IEEE 1451, for example. However, most of the relevant metadata of a sensor depends on the deployment and use of the sensor (e.g., logical location of the sensor, object to which the sensor is attached) and cannot be provided by the producer of a sensor. Secondly, there are efforts to provide a standardized vocabulary to describe sensors such as SensorML or the Semantic Sensor Network Ontology (SSN). Unfortunately, these ontologies and their use are rather complex and end users likely won't be able to provide correct descriptions of sensors and their deployment context without help from experts.

Note that the same problem also applies to search for multimedia items on the web such as images and videos, which typically can only be found if appropriate textual descriptions are provided by users. One interesting alternative approach that avoids the use of metadata altogether is searching by example. For example, Google Images includes an option to find images that are similar to another image. There are even specialized search engines such as TinEye[2] which find images that are similar to a given image.

In this paper we adopt this search-by-example approach to sensors, i.e., a user provides a sensor, respectively a fraction of its past output as an example, and requests sensors that produced similar output in the past. This service could be used for different purposes. Firstly, it could be used to find places with similar physical properties. For example, if one wants to find places that have similar climatic conditions as a known place A, one could pick a temperature sensor that is known to be at place A, and then search for other temperature sensors with similar output. Secondly, it could be used to assist users with the formulation of a metadata description of a newly deployed sensor. A user would deploy a new sensor and then search for sensors with similar output, fetch the metadata of the found sensors, and reuse appropriate fractions of the metadata for the new sensor.

II. Requirements and Architecture

As we aim to apply sensor similarity search to the Internet of Things, the approach should be scalable to many sensors. This implies that the comparison of two sensors in order to decide if they are similar should be efficient. In particular, as many sensors have limited power supply, communication overhead with sensors should be minimized, which implies that sensors should compute a compact summary of their past output data that can be downloaded from the sensors

[1]https://pachube.com/

[2]www.tineye.com

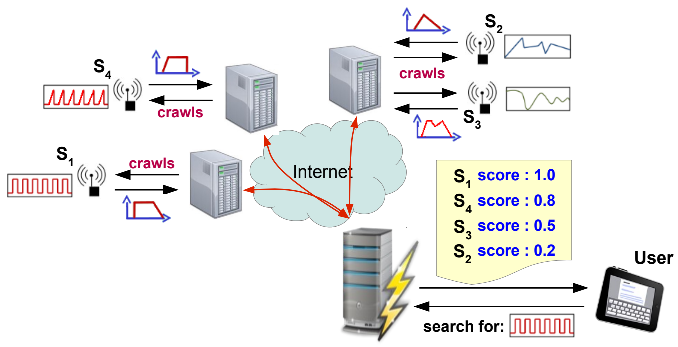

Fig. 1: Architecture of sensor similarity search.

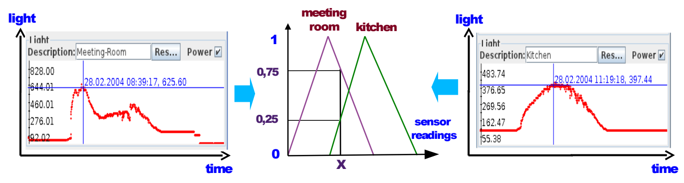

Fig. 2: Fuzzy-based sensor similarity computation.

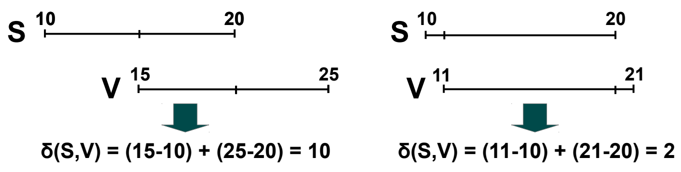
S 10 20
V 15 25
δ(S,V) = (15-10) + (25-20) = 10
S 10 20
V 11 21
δ(S,V) = (11-10) + (21-20) = 2

Fig. 3: Measurement Range Difference.

with little communication overhead and indexed by a search engine. Further, the comparison of the output of a sensor with those indexed summary data structures should be efficient such that search results are returned quickly to the user. Finally, this comparison function should be flexible such that sensors can be found that show similar trends in their output despite differences in the actual output time series.

Fig. 1 illustrates our approach. We represent the output of a sensor over a long time period by a fuzzy set which is computed by the sensor itself. Periodically the search engine crawls sensors to download and index those fuzzy sets in a distributed database system in the Internet. Each fuzzy set has a memory footprint of few tens of bytes and can thus be efficiently downloaded from the sensors. Note that each sensor gateway acts as a communication bridge between a local sensor network and the Internet.

To perform a search, the user specifies a time series of sensor values. This time series is compared to the indexed fuzzy sets and a similarity score is computed for each indexed sensor. The sensors with the highest similarity scores are presented to the user sorted by decreasing similarity score.

In the remainder of the paper we describe how fuzzy sets are extracted from sensor data, how similarity scores are computed, and evaluate the approach using two data sets obtained from real sensors. Finally, we discuss how we intend to extend this approach for scalable sensor similarity search in the Web of Things.

III. Comparing Sensors

In this section, we propose a method based on fuzzy sets to compute a similarity score for a pair of sensors based on their output. A fuzzy set [6] 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 $\mu_F(x), x \in U$ such that $\mu_F(x) \in [0,1]$. The closer the value of this function is to one for a given x, the more x belongs to the fuzzy set F. With this definition, an element $x \in U$ can be a member of more than one fuzzy set at a time, with different degrees of membership.

A. Approach

Fig. 2 illustrates our approach. We have the output measurements of two sensors located in two locations $kitchen$ and $meeting\_room$. Suppose that for each sensor we have built a fuzzy set from its measurements, i.e., we have $F_{kitchen} = \{(x, \mu_{kitchen}(x)) | x \in \mathbb{R}\}$ and $F_{meeting\_room} = \{(x, \mu_{meeting\_room}(x)) | x \in \mathbb{R}\}$. Note that we assume the set of real numbers $\mathbb{R}$ as the universe of discourse to represent scalar measurements of sensors. For the sake of illustration we assume the shape of the membership functions is a triangle. Now given a set of measurements $U_S \subset \mathbb{R}$ of a third sensor S, we want to compute scores of the similarity of S with the sensor in the kitchen and the sensor in the meeting room.

If we pick a sample measurement $x \in U_S$, the membership functions of $F_{kitchen}$ and $F_{meeting\_room}$ will tell us the degree of membership of x in the two fuzzy sets, which is $\mu_{meeting\_room}(x) = 0.75 > 0.25 = \mu_{kitchen}(x)$. We therefore say that the value x is more likely read by the sensor located in the meeting room. Based on this approach, we define the similarity score of the sensor S with respect to a sensor V as

$$\Phi_S(V) = \frac{1}{\delta(S,V)} \frac{1}{|U_S|} \sum_{x \in U_S} \mu_V(x) \qquad (1)$$

We call $\delta(S,V)$ the *measurement range difference* between two sensors S and V that is given by

$$\delta(S,V) = |q_1^S - q_1^V| + |q_3^S - q_3^V| \qquad (2)$$

where q_1^S, $q_3^S \in U_S$ and q_1^V, $q_3^S \in U_V$ are the first and third quartiles of the measurement sets of sensors S and V, respectively. We call $[q_1^S, q_3^S]$ and $[q_1^V, q_3^V]$ the measurement ranges of sensor S and V. Fig. 3 shows an example for measurement ranges of two sensors S and V. The small overlap in the left between the two ranges implies a large $\delta(S,V)$, wheras the big overlap in the right implies a small $\delta(S,V)$.

In summary, the similarity score is the higher, the more the measurement ranges overlap and the more the measurements of sensor S belong to the fuzzy set defined by the output of

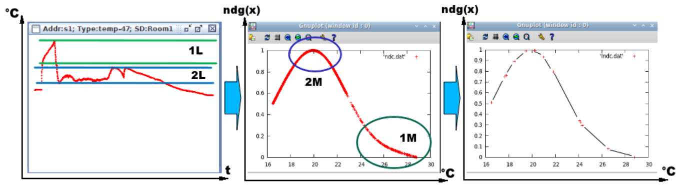

Fig. 4: Fuzzy set construction.

sensor V. Note that the similarity score of sensors with disjoint measurement ranges is zero.

B. Fuzzy Set Construction

We will elaborate in this subsection how a fuzzy set is constructed from a given set of sensor measurements. Consider Fig. 4. The left image shows the time series of measurements of a temperature sensor S over a time period Δt, and the graph in the middle shows the constructed fuzzy set. We denote F_S as the constructed fuzzy set and U_S as the set of measurements of S. We want to find a membership function $\mu_S(x)$ that represents the series of measurements of the sensor over time.

Considering an interval $\Delta x = [x - r,\ x + r] \subset [x^S_{min},\ x^S_{max}]$ for $r > 0$, we are interested in how many measurements $x \in U_S$ fall into $[x - r,\ x + r]$ over Δt because this captures the behaviour of the object that our sensor is measuring: does temperature tend to be within the range Δx? Put in another way, the density of the population of sensor measurements in Δx describes the likelihood of temperature to be within Δx. By letting $r \rightarrow 0$ and by sliding Δx over $[x^S_{min},\ x^S_{max}]$ we can calculate the likelihood for each temperature value x in the measurement range. $\mu_S(x)$ is then defined as this likelihood of $x \in [x^S_{min},\ x^S_{max}]$.

The work in [7] presents a way to compute how densely a data point is surrounded by other data points in close proximity. We borrow this idea to compute the density of sensor measurements within an interval $[x - r, x + r]$ around a measurement value x, and call it the *neighbor density* of x:

$$ndg(x) = \sum_{y \in U_S} e^{-\left[\frac{2 d_E(x,y)}{r}\right]^2} \tag{3}$$

where $d_E(x, y)$ is Euclidean distance between two values x and y. Due to the exponential function, measurement values which are outside of $[x - r, x + r]$ have little influence on $ndg(x)$. We then normalize $ndg(x)$ to values between 0 and 1 using min-max normalization. $\mu_S(x)$ is then defined as the normalized version of $ndg(x)$. We call this membership function the *neighbor density function*.

For a visual explanation of our approach, consider Fig. 4 again, which shows the neighbor density function of the temperature sensor in the middle. The peak in region "2M" results from a dense distribution of measurements within region "2L", while the low values in region "1M" are explained by a sparse distribution of measurements within region "1L".

C. Fuzzy Set Approximation

Since the storage overhead for a fuzzy set is proportional to the size of the measurement range of the sensor, it may be expensive to store a fuzzy rule, or more specifically the fuzzy set and its membership function. Furthermore, this cost is multiplied by the number of sensors which is expected to be large in the IoT vision. To reduce storage overhead, we propose to represent the neighbor density function by a set of line segments that approximate the curve of the function. An illustration is given in Fig. 4, where in the middle we have the neighbor density function whose linear approximation is shown in the right side of the figure.

To approximate the membership function $\mu_S(x)$ of a sensor S, we first define a derivation threshold d_{th}, compute $\mu_S(x)$'s first derivative d_1 at x_1 (x_1 is the second smallest value in the measurement range of S), and mark the point $A_1 := (x_1, \mu_S(x_1))$. We then iterate over points $(x_i, \mu_S(x_i))$ on the curve and compute $\mu_S(x)$'s first derivative d_i, until $d_i - d_1 > d_{th}$. We assign $x_2 := x_i$, $A_2 := (x_2, \mu_S(x_2))$, and store the line A_1A_2 as the approximation of $\mu_S(x)$ 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 $(x^S_{max}, \mu_S(x^S_{max}))$. The resulting set of line segments is the desired approximation of $\mu_S(x)$.

Due to the exponential weighting of distances in equation 3, fuzzy sets typically have a smooth curve and can be represented by few line segments. As each line segment (except the first one) can be defined by two integer values, the memory footprint of the fuzzy set is small and typically in the order of few tens of bytes.

D. Towards Scalable Similarity Search

As outlined in Sect. II, all fuzzy sets would be indexed in a data base. For a search operation, the output time series of a given sensor would be used to compute a similarity score for each fuzzy set. While the comparison of two sensors is efficient and takes in the order of few tens of microseconds even when implemented in Java (see Sect. IV), the search latency grows linearly in the number of sensors. In future work, we therefore explore efficient algorithms to speed up the search. Here we briefly discuss potential directions.

Similar to typical web search engines, it may be sufficient to provide a small list of N top-ranked sensors to the user instead of a complete list of all sensors. This can be exploited to reduce the computation overhead of similarity search.

A first, simple strategy is to parallelize search by distributing the fuzzy sets over multiple computers. Each search request would be forwarded to all computers, similarity scores would be computed in parallel, the top-N list of sensors would be returned and merged to obtain the final rank list.

To speed up computation of similarity scores on a single computer, an incremental approach can be used, by first computing an approximate similarity score according to equation 1 for a small subset of the measurements U_S of the given sensor, for example, by only using every 10th sample from the time series. In a second pass, more samples are added, say every 5th sample, and so on. This way, a first approximate search result can be very quickly presented to the user which is continuously refined the longer the user waits.

If we have computed the scores for the first N sensors and sorted them by decreasing score, we can ignore each further sensor with a score that is smaller than the score of the lowest-ranked of the N sensors. Note that the score computed according to equation 1 is additive and each summand is upper-bounded by $\frac{1}{|U_S|\delta(S,V)}$ as the fuzzy set function is upper-bounded by 1. When we have computed a partial sum s and k summands remain, we know that the final score cannot be larger than $s + \frac{k}{|U_S|\delta(S,V)}$. If that value is smaller than the score of the lowest-ranked of the N sensors, we can abort the computation of the score and ignore the sensor. Otherwise, if the sensor has a score that is larger then the score of the lowest-ranked of the N sensors, the new sensor is inserted into the list according to its score and the last sensor in the list is dropped. Note that the similarity score of sensors with disjoint measurement ranges is zero according to equation 1, which can be decided with a simple check without even computing the sum.

Finally, there are cases where for any given sensor S, the similarity score with a sensor V_1 is always smaller than the similarity score with a sensor V_2, i.e., $\forall S : \Phi_S(V_1) < \Phi_S(V_2)$. The resulting partial order $V_1 \prec V_2$ over sensors can be exploited to prune large numbers of sensors during search.

IV. Evaluation

In this section we evaluate the performance of our sensor similarity search. As a result of searching for a given sensor, a list of sensors ranked by decreasing similarity score is obtained. Similar sensors should be ranked highly (i.e., on top of the list), while dissimilar sensors should be ranked low (i.e., at the bottom of the list). Unfortunately, "similar" is highly subjective and depends on the perception of the user. One user may consider two sensors to be similar, while another user may not. Hence, it is difficult to obtain a ground truth for evaluation.

To resolve this issue, we manually group sensors based on their location as nearby sensors should produce similar output if the measured physical quantity has a low spatial variation. For example, all temperature sensors in a room should produce similar output and may thus form a group. This way, we obtain groups of sensors $G_1, G_2, \ldots$. We now pick a sensor s from a group G_i and search for similar sensors. We would expect that all sensors from the same group G_i are ranked highest. However, the result may be imperfect, i.e., sensors from G_i might be ranked lower than sensors from other groups. Therefore, we need a metric to quantify the accuracy of a rank list, which we describe next before we present the evaluation setup and results.

A. Degree of Ranking Accuracy

Figure 5 shows possible rank lists obtained as a result when searching for a sensor s from a group G_i. The check marks

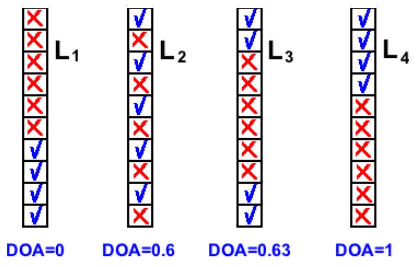

Fig. 5: Illustration of the DOA metric.

indicate matching sensors, i.e., sensors from the same group G_i, while crosses indicate non-matching sensor from other groups. The best possible result is list L_4 as all matching sensors are ranked highest. The worst result is list L_1.

We now define a metric that maps a rank list to a scaler value between 0 (worst result) and 1 (best result). 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. To normalize to the interval $[0, 1]$, we divide by the number of non-matching sensors. By subtracting the resulting value from 1, we obtain the desired metric. Thus, we define the degree of ranking accuracy (DOA) of a rank list L as follows:

$$DOA(L) = 1 - \frac{1}{C_L(N_L - C_L)} \times \sum_{i=1}^{N_L} e_L(i) \tag{4}$$

where N_L is the length of rank list L, C_L is the number of matching sensors in L, and $e_L(i)$ 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 $e_L(i) := 0$. Fig. 5 shows the value of the metric for different rank lists.

B. Experiment Setup

To evaluate our similarity search, we implement a simulation tool in Java that is able to replay recorded measurements of multiple sensor, execute search operations over these sensors, and compute the resulting ranking accuracy according to the above metric.

We use two data sets with recorded sensor values from real-world deployments for the evaluation. As described earlier, we group sensors in each of the data sets based on their location, such that sensors in a group should observe similar (but not identical) output.

The first is the NOAA data set[3] which contains the output of sensors monitoring ocean and atmosphere (e.g., barometric pressure, wind speed, air temperature, conductivity, water velocity) that are deployed along the coast lines of various places in North America. We use 23 barometric-pressure sensors from this data set and group them into 5 groups, namely Alaska (3

[3]http://tidesandcurrents.noaa.gov/gmap3

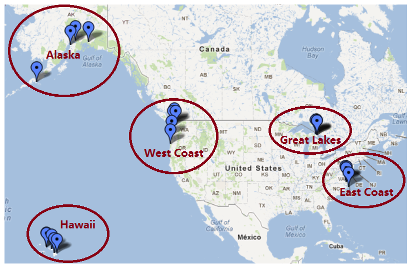

Fig. 6: Our selection of locations in the NOAA data set.

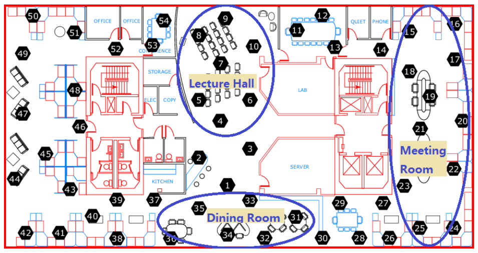

Fig. 7: Our selection of locations in the Intel Lab data set.

sensors), West-Coast, Great-Lakes, East-Coast, and Hawaii (5 sensors each) as shown in Fig. 6.

The second is the IntelLab data set[4] which contains recorded measurements of 54 sensor nodes equipped with four different sensors, namely temperature, light, battery voltage, and humidity (i.e., 216 sensors in total). These sensors were deployed in the Intel Berkeley Research Lab between February 28th and April 5th, 2004. We select a set of 12 humidity sensors and group them into three groups, namely Lecture-Hall (4 sensors), Dining-Room (4 sensors), and Meeting-Room (4 sensors) as shown in Fig. 7.

To perform the evaluation, we sequentially pick one sensor after another from each of the two test sets, search for sensors similar to that one, obtain a rank list, and compute the DOA value. For each sensor, we use the last 24 hours of data which is representative as the data tends to repeat every day. This approximately equals 1500 data points in the IntelLab data set and 200 data points in the NOAA data set.

C. Accuracy

Figs. 8 and 9 show the resulting DOA values when searching for each of the sensors in the NOAA and IntelLab data sets, respectively. Also, a box plot aggregating the results is shown.

As observed in the figures, our sensor similarity search obtains a high degree of accuracy as the average DOA is above 0.95 for both data sets. The boxplots show a stable performance of our approach with small first and third quartiles, i.e., 0.04 for NOAA and 0.07 for IntelLab. There are, however, a few outliers such as search trials number 10 and 23 in Fig. 8, and number 1 in Fig. 9. The reason for this is that even though sensors in each group are deployed close to each other, they may experience significant variations due to microclimates (in

[4]http://db.csail.mit.edu/labdata/labdata.html

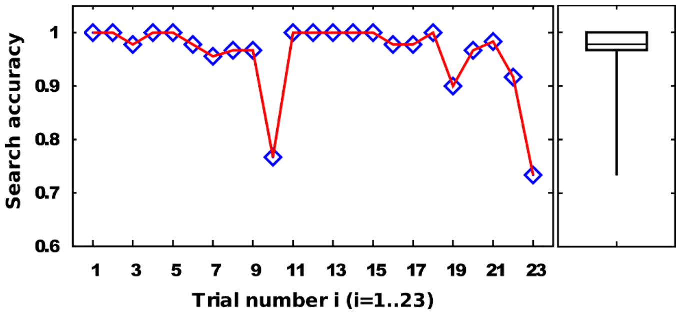

Fig. 8: Degree of accuracy (NOAA).

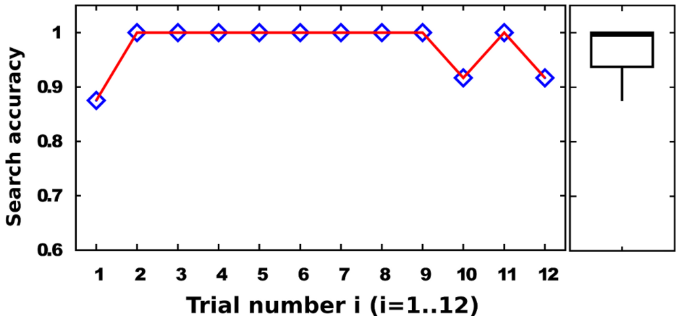

Fig. 9: Degree of accuracy (IntelLab).

the NOAA data set) or due to sensors be ing close to the heating or air conditioner (in the IntelLab data set).

D. Performance

We investigate the time needed to compute a similarity score for a pair of sensors as this is the fundamental operation performed by the search engine. We use the approach in [8] to minimize the impact of garbage collection and just-in-time compilation in the Java VM on computation time measurement.

For a similarity score computation, we obtained an average computation time of $150\mu s$ for the IntelLab data set, and $20\mu s$ for the NOAA data set. The difference stems from the fact that the number of measurements per day in NOAA is much smaller (200) than for the IntelLab data set (1500). That is, even with a brute force approach we can compare against 6666 to 50000 sensors per second. The computer used in our experiment has an Intel Core i5 CPU that runs at a clock rate of 2.4Ghz.

It is worth noting that, even though the number of measurements per day of NOAA sensors is much smaller than that of IntelLab sensors, the accuracy is similar for both data sets as can be seen in Figs. 8 and 9. This indicates that the incremental approach outlined in Sect. III-D may be very effective.

V. Related Work

Content-based similarity search has been used in traditional multimedia systems to search for images and videos that are similar to a given example image or video [9], [10]. However, these methods cannot be applied to sensors.

In the context of the IoT, a number of systems support search for sensors based on metadata, for example Pachube[5], GSN [3], or Microsoft SensorMap [4]. Our approach is complementary as it does not rely on metadata, but searches for sensors with similar output. However, similarity search could be integrated with search based on metadata, e.g., performing similarity search only for sensors with certain metadata, or first searching sensors based on metadata, and then searching for sensors similar to some of the found sensors.

In recent work, we also investigate the related problem of searching for sensors that output a given value at the time of the query [11]. However, in our current work we consider the statistical similarity of sensors over a longer period of time, which is a different problem.

The work in [12] identifies similarities between generated streams of neighbouring nodes in a sensor networks so that the streams can be aggrerated to save bandwidth. A data stream is divided into multiple sets and these sets are compared against the sets of another data stream to find similarites using the Jaccard similarity function. Our goal is, however, different as we do not seek for similar parts of two data streams, but we want to determine if the two entire streams are similar by using the fuzzy set representation of sensor data.

In [13], the authors propose a systematic design for a search application in the Web of Things. Their design is based on clustering of sensors with similar semantic descriptions. Our work aims to provide a fundamental service that determines if two sensors are similar based on their output data.

The work in [14] proposes to automatically create clusters of sensors that adhere to a semantic boundary (e.g., a room) based on computing the similarity between sensor outputs, rather than conventional networking metrics (e.g., connectivity). Their approach is expensive in terms of memory and computation overhead. They require storing all sensor measurements either at a central processing point or at sensors while we need to store only a much lighter approximation of a fuzzy set. Further, similarity computation using correlation coefficients is costly when compared to our approach. Their computation also requires two measurement sets to have the same number of samples while ours does not.

The so-called *context proximity* paradigm is discussed in [15] as a secure way for connecting and grouping sensors. The key idea is that sensors in close context proximity should perceive correlated patterns thus can be securely connected (e.g., two devices being shaken at similar patterns as they both are attached to the same walking person). Lester et al. exploit this idea in [16] using accelerometer sensor data to group sensors, that are worn by the same person, with high accuracy. Rather than grouping a small number of sensors, our focus is different as we aim at searching for sensors at a much larger scale, using a different technique, fuzzy sets, to model the behaviour of sensors from their outputs, and use these models to compute the similarity of different sensors.

[5]https://pachube.com/

VI. Conclusion and Outlook

We are witnessing the formation of a Web of Things, where the output of sensors connected to the Internet is published and mashed up with data and services on the Web to create novel real-world applications. A fundamental service in the resulting Web of Things is search for sensors. Instead of relying on manual annotations (which are often incorrect, inconsistent, or incomplete), we propose sensor similarity search, where based on the past output of a sensor, sensors with similar output are found. We designed an efficient approach to compute a similarity score for a pair of sensors. All sensors compute fuzzy sets that represent their past output using only few tens of bytes. These fuzzy sets are indexed in a data base. Given the output of another sensor, similarity scores are computed for each indexed sensor, sensors are ranked by this score and returned to the user. Using sensor data from two real-world deployments, we could show the high accuracy of our approach. Building upon those results, we will explore scalable search algorithms to support searching among large numbers of sensors in the Web of Things. Eventually, we also plan to perform a user study to assess the accuracy of our approach.

References

[1] 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.

[2] F. Mattern and C. Floerkemeier, *From the Internet of Computers to the Internet of Things*, ser. LNCS. Springer, 2010, vol. 6462, pp. 242–259.

[3] A. Salehi, M. Riahi, S. Michel, and K. Aberer, "Gsn, middleware for streaming world," *In Proc. 10th Int. Conf. on Mobile Data Management*, 2009.

[4] S. Nath, J. Liu, and F. Zhao, "Sensormap for wide-area sensor webs," *IEEE Computer*, 2008.

[5] A. Broering, F. Bache, T. Bartoschek, and C. P. van Elzakker, "The sid creator: A visual approach for integrating sensors with the sensor web," *14th AGILE Int. Conf. on Geographic Information Science, April 2011, Utrecht, Netherlands*.

[6] 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.

[7] S. L. Chiu, "Extracting fuzzy rules from data for function approximation and pattern classification," *Fuzzy Information Engineering A Guided Tour of Applications*, pp. 1–10, 1997.

[8] B. Boyer, "Robust java benchmarking: Part 1 and part 2," *IBM's developerWorks, Technical Library*, 2008.

[9] P. Darasb, T. Semertzidis, L. Makrisb, and M. G. Strintzisa, "Similarity content search in content centric networks," *Proc. Intl. Conf. on Multimedia, MM '10*.

[10] Z. Wang, M. D. Hoffman, P. R. Cook, and K. Li, "Vferret: Content-based similarity search tool for continuous archived video," *CARPE'06*.

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

[12] J. M. Bahi, A. Makhoul, and M. Medlej, "Data aggregation for periodic sensor networks using sets similarity functions," *IWCMC*, 2011.

[13] B. Christophe, V. Verdot, and V. Toubiana, "Searching the web of things," *IEEE Intl. Conf. Semantic Computing*, 2011.

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

[15] 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.

[16] 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.