View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by UCL Discovery

Large-Scale Indexing, Discovery and Ranking for the Internet of Things (IoT)

YASMIN FATHY, Institution for Communication Systems (ICS), University of Surrey, United Kingdom
PAYAM BARNAGHI, Institution for Communication Systems (ICS), University of Surrey, United Kingdom
RAHIM TAFAZOLLI, Institution for Communication Systems (ICS), University of Surrey, United Kingdom

Network-enabled sensing and actuation devices are key enablers to connect real-world objects to the cyber world. The Internet of Things (IoT) consists of the network-enabled devices and communication technologies that allow connectivity and integration of physical objects (Things) into the digital world (Internet). Enormous amounts of dynamic IoT data are collected from Internet-connected devices. IoT data is usually multi-variant streams that are heterogeneous, sporadic, multi-modal and spatio-temporal. IoT data can be disseminated with different granularities and have diverse structures, types and qualities. Dealing with the data deluge from heterogeneous IoT resources and services imposes new challenges on indexing, discovery and ranking mechanisms that will allow building applications that require on-line access and retrieval of ad-hoc IoT data. However, the existing IoT data indexing and discovery approaches are complex or centralised which hinders their scalability. The primary objective of this paper is to provide a holistic overview of the state-of-the-art on indexing, discovery and ranking of IoT data. The paper aims to pave the way for researchers to design, develop, implement and evaluate techniques and approaches for on-line large-scale distributed IoT applications and services.

CCS Concepts: • **General and reference → Surveys and overview** ; • **Computer systems organization → Distributed architectures**; **Sensor networks** ; **Sensors and actuators**; • **Information systems → Data mining**; • **Networks → Application layer protocols**;

Additional Key Words and Phrases: Internet of Things (IoT), Wireless Sensor Network (WSN), large-scale data, indexing, discovery, ranking

ACM Reference Format:
Yasmin Fathy, Payam Barnaghi, and Rahim Tafazolli. 2010. Large-Scale Indexing, Discovery and Ranking for the Internet of Things (IoT). *ACM Trans. Web* 9, 4, Article 39 (March 2010), 67 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

The Internet of Things (IoT) has emerged as a paradigm that allows a large number of physical objects and devices with identifying, sensing and network capabilities (Things) to seamlessly communicate and interact with one another and with other resources (e.g. devices, services) in the network (Internet). The Web of Things (WoT) is envisioned as an evolutionary paradigm of IoT, which leverages the Web standards and technologies to fully integrate IoT objects with other virtual objects on the Web. The core concept of IoT/WoT is not new, but it is an evolution of various

Authors' addresses: Yasmin Fathy, Institution for Communication Systems (ICS), University of Surrey, James Clerk Maxwell Building, Stag Hill Campus, Guildford, Surrey, GU2 7XH, United Kingdom, y.fathy@surrey.ac.uk; Payam Barnaghi, Institution for Communication Systems (ICS), University of Surrey, United Kingdom, p.barnaghi@surrey.ac.uk; Rahim Tafazolli, Institution for Communication Systems (ICS), University of Surrey, United Kingdom, r.tafazolli@surrey.ac.uk.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2009 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.
1559-1131/2010/3-ART39 $15.00
https://doi.org/0000001.0000001

https://mc.manuscriptcentral.com/csur

data processing, Internet networking, and communication technologies. For example, different identification schemes such as Radio Frequency IDentification (RFID) tags and Internet Protocol (IP) addresses are used to identify things on the Internet. Wired/Wireless Sensor and Actuator Networks (WSANs) create a network of sensors that captures the state of the environment or objects and a network of actuators that is able to perform actions, which can affect the state of the environment or the objects.

The IoT resources (e.g. sensory devices) are heterogeneous and often deployed in distributed and dynamic environments over a large (dense or sparse) geographical area. Data provided by different IoT resources is continuously generated as streaming data that is usually multi-modal (e.g. light, temperature, sound) with different forms such as textual, numerical, streaming and multimedia data. The generated data can be represented in various formats (e.g. Comma-Separated Values (CSV) and JavaScript Object Notation (JSON)) along with different levels of granularities. The data can be diverse (e.g. data quality varies with different resources) with common locations and time dependencies [Barnaghi et al. 2013b]. The availability of IoT resources and their attributes can also vary over time [Barnaghi et al. 2013a].

It is expected that nearly 50 billion network-enabled devices will be connected to the Internet by 2020 [Ericsson 2011]. Ideally, published data from such devices should be available upon requests (queries) by authorised users (human or machine users) from any location and at anytime to respond to higher-level applications and service requests. Large-scale sensor networks can enable the development of a wide range of powerful applications that allow autonomous communication and exchanging of data among devices and services. For example, Intelligent Transport Systems (ITSs) apply data mining and knowledge discovery algorithms and techniques on sensor data collected from various resources (e.g. cameras, parking, and inertial sensors) to derive higher-level abstractions (e.g. high traffic, available parking spots). The IoT enables creating environmental metering and monitoring applications. For instance, smart metering allows getting automated readings for a better, more frequent and up-to-date understanding of energy usage [Khan et al. 2012]. Benefits provided by IoT technologies also empowers applications in agriculture sector such as smart farming solutions where data collected from different sensors can be analysed and processed for smart irrigation scheduling for various crops. Furthermore, IoT makes it possible for developing healthcare applications based on activity tracking (e.g. sleep, blood pressure, and heart rate) and reporting anomalies to aid independent living [Atzori et al. 2010]. IoT enables solutions that have also been applied to various smart city applications. For example, interested readers can refer to 101 smart city scenarios described at: http://www.ict-citypulse.eu/scenarios/.

Large deployments of IoT resources will not often run for a single application and will not only respond to a single request. To this end, the underlying IoT sensors and services require data mining and knowledge discovery algorithms for the development of large scale multi-purpose IoT applications. For this purpose, indexing, discovery and ranking methods and approaches for large-scale distributed and dynamic IoT networks have a potential impact on efficient access and use of available IoT resources and their published data.

There are several advancements in networking and communication domains such as high-speed networks in both wired and wireless communications, information-centric networking and others. To this end, Internet architecture has become more adaptable and efficient to allow flexible and adaptable communications for growing volumes of data. However, the current information access and retrieval solutions on the Web are far from ideal. Most of the existing solutions for the Web indexing and discovery are designed based on text-analysis (text/keyword based) and exploitation of links between different documents/data resources on the Internet and are not suitable for large-scale, dynamic IoT data networks [Fathy et al. 2016]. IoT data discovery requires ad-hoc and

Machine-to-Machine (M2M) search and discovery of the data (in contrast to Web search which is mainly triggered by human supplied keywords).

Some works in IoT domain are demonstrated by Wolfram Data Drop[1], Thingful[2], and Google's recent project Brillo[3] that provide information search and discovery approaches for IoT resources. However, most of the existing solutions for IoT do not provide specialised and distributed mechanisms for automated crawling, (on-line) indexing and ranking for large-scale and uncoordinated networks of IoT heterogeneous and distributed resources and data [Perera et al. 2013; Polytarchos et al. 2011]. Novel distributed, efficient and scalable methods and solutions are required to provide on-line indexing, discovery and ranking for large-scale IoT data and/or resources.

This paper explores the state-of-the-art of multi-modal and heterogeneous IoT data processing with a focus on IoT search elements (indexing, discovery and ranking). To provide the reader with a better understanding of essential processes that are required for designing and building IoT applications, we first discuss the process chain from connecting IoT resources to a network, up to the search and discovery of the resources and/or their published data for end-users through Web-based applications. To facilitate our discussion of the state-of-the-art on IoT search elements, different methods and approaches of each element: indexing, ranking and discovery are discussed and classified into three distinct perspectives for IoT/WoT applications: data, resource, and higher-level abstractions. The classifications are summarised in Table 1.

Table 1. Classifications of search elements for IoT/WoT applications

Perspective	Explanation	Example
Data	refers to IoT data value (sensor measurement/observation) that is published by IoT resource (e.g. a sensor)	Retrieve device observation/ measurement
Resource	refers to an IoT resource. It can be (networked-) sensors, devices or services	Access a device or a service by its unique identifier
Higher-level Abstractions	refers to inferring information and insights from published raw data by several IoT resources	Detect events/activity/pattern (e.g. cold weather, high traffic)

To the best of our knowledge, no existing surveys have covered the workflow of indexing, discovery and ranking for the IoT and search approaches from the data, resource, and higher-level abstractions perspectives. Overall, existing studies and surveys do not provide a concise classification and overview of different search techniques for IoT [Zhou et al. 2016]. For example, [Zhang et al. 2011] propose different measurement dimensions that are categorised into information aggregation, index, and query for developing an IoT search engine. However, the work does not provide a classification of the search techniques. A recent classification of the search techniques for WoT from three different perspectives (basic principles, data/knowledge representation, contents being searched) is discussed in [Zhou et al. 2016]. However, there are some overlaps between the three perspectives. Therefore, a general classification of search approaches for IoT/WoT applications is required. It is also worth mentioning that privacy and security are required for most of the IoT applications, but the detailed discussion on these issues are out of the scope of this survey. In summary, the main contributions of this paper are:

- The process chain starts with gathering and collecting observation and measurement data from different IoT resources, up to the search and discovery of the resources and their data

[1]http://datadrop.wolframcloud.com/
[2]http://www.thingful.net/
[3]http://developers.google.com/brillo/

for end-users is defined and explained. The essential and optional processes of this process chain for designing and building IoT applications are described and discussed.

- A holistic and clear overview of the state-of-the-art on indexing, discovery and ranking approaches and solutions for IoT are presented from three different perspectives: data, resource, and higher-level abstractions.

Table 2 provides a list for the definition of some common and general terms used in the paper.

Table 2. Definition of terms

Term	Definition
Internet of Things (IoT)	is a term to describe the communication and interactions of network-enabled and sensory devices (i.e. Things) in the digital world (i.e. Internet) to expose their functionalities and underlying services.
Web of Things (WoT)	is a term to describe data exchanging and interactions of the Internet-enabled devices (i.e. Things) on the Web using Web technologies to build Web applications and services on the top of the IoT.
IoT Resource	refers to any resource (e.g. service, ubiquitous device) in the WSN/Internet that can publish and share its data and services on the network.
Context-aware/ Situation-aware Services	are services that extract information from one or more IoT resources to capture specific event, situation, place or object to describe an environment at anytime and anywhere.
Quality of Service (QoS)	is the network's ability to achieve the desired level of performance regarding a set of performance measurements (e.g. latency, bandwidth, delay, and throughput).
Contextual Information	refers to the information which is related to observation and measurement data published by sensory devices such as battery status, device's reliability and availability.
Wireless Sensor and Actuator Network (WSAN)	is a network of distributed sensor and actuator nodes in which sensor nodes capture the state of environment or objects and actuator nodes can perform actions that affect the state of the environment or the objects.
IoT Middleware	provides a unified interface to access heterogeneous IoT services and devices with different data models or application layer protocols.
Big Data	is a term to describe large volumes of data that have different structure, velocity, granularity and qualities. The nature of big data imposes challenges in storing and managing it in conventional Database Management Systems (DBMS) and also processing it using conventional methods and tools.
Data Analytics	is a generic term that refers to applying analysis and processing methodologies to infer information and insights from data by extracting numeric values, patterns and/or events.
Discovery Service	is a service to find and discover data providers (resources) that can answer a requested query given the key search attributes (e.g. type, location and time).
Resource Discovery	is also known as a network discovery that refers to crawling, finding, and allowing IoT resources to be found/discovered automatically or manually.
Data Aggregation	is to fuse data using different techniques (e.g. average of sensor values) to a sink node (i.e. base station).
Data Integration	refers to integrating heterogeneous data from different IoT resources into one system (e.g. an existing Web application) by addressing the mismatching among various resources that use different protocols and their data models.
Data Interoperability	is to use common models to describe heterogeneous IoT data resources and services and/or allow communicating and exchanging data between different resources that might have different application layer protocols without integrating them into one system.
Data Abstraction	is about obtaining low granular expressive and meaningful data (summary form; weather is windy) from higher granular raw data.
Data Representation	is to represent IoT data in a human understandable and/or machine interpretable format.

Page 5 of 67 Computing Surveys

Data Publication	is about publishing data from heterogeneous IoT data sources. The publication might involve publishing the meta-data and/or annotation of the data semantically.
Data Dissemination	is to allow sensor nodes to communicate with a sink node (i.e. a base station) or other interested nodes in the network to share their collected data or to seek information from other nodes in the network.
Indexing	is about indexing and sorting IoT data/resources to allow fast access, retrieval and search processes for the resources and their data.
Data Ranking	is about prioritising IoT resources and services based on several criteria (functional (e.g. network latency) or non-functional (e.g. data quality and trust)).
Data Query	is a term that refers to retrieving information and data from resources. Query is often composed of type, location and time to construct an expression (e.g. temperature in London with freshness ≤ 5 seconds).
Meta-data	is enriched data to describe other data and make it more expressive and meaningful. In IoT, meta-data often describes the specifications and capabilities of the sensors (resources) (e.g. accuracy, calibration, source state, battery life and deployment location) and their measurement and observation data (e.g. time-stamp, data quality, values and unit).
Resource Description Framework (RDF)	is a World Wide Web Consortium (W3C) recommendation that is widely used to describe Web resources (as a meta-data data model).
Simple Protocol and RDF Query Language (SPARQL)	is a query language to query (search) and retrieve data that is stored in the RDF format.

The rest of the paper is organised as follows: Section 2 states the key components and design requirements for IoT environments and applications. Section 3 explains IoT data models and representation. Section 4 presents how IoT data can be aggregated and abstracted. Section 5 describes IoT data publication and provisioning. The state-of-the-art for indexing and discovery for distributed IoT resources and analysis of existing solutions are presented in Section 6 and ranking approaches are discussed in Section 7. Section 8 provides an overall analysis and discusses the outlook and applicability of indexing and discovery solutions to IoT frameworks. Section 9 draws conclusions and discusses the areas for further research.

2 DESIGN REQUIREMENTS

The Internet of Things (IoT) involves a wide range of different technologies including RFID technologies, communication protocols, data mining and machine learning to allow users to query measurement and observation data. This section discusses a general model for indexing, ranking and discovery of network-enabled devices to make their measurements and underlying services searchable and discoverable (see Fig. 1). [Barnaghi et al. 2013a] describe the process chain from collecting real-world observation and measurement data up to making the data accessible on the Web. However, indexing and ranking processes are not included in the process chain. [Abu-Elkheir et al. 2013] also explain life-cycle for IoT data starting from data production to data querying and analysis. The life-cycle does not cover indexing and ranking. Similar to [Barnaghi et al. 2013a] and [Abu-Elkheir et al. 2013] and based on the literature review we conducted, we propose a model that includes indexing and ranking for IoT data/resources.

Fig. 1 depicts the process chain of gathering and collecting observation and measurement data from different IoT devices and services. The solid lines represent the essential processes for building IoT applications that allow searching for IoT data (e.g. temperature data in a specific location). The dashed lines represent the optional processes; however, the overall workflow depends on the IoT applications. For example, road traffic monitoring applications require the aggregation of data from different types of sources to get vehicle traffic flows, bicyclists, and pedestrian counts. The

https://mc.manuscriptcentral.com/csur

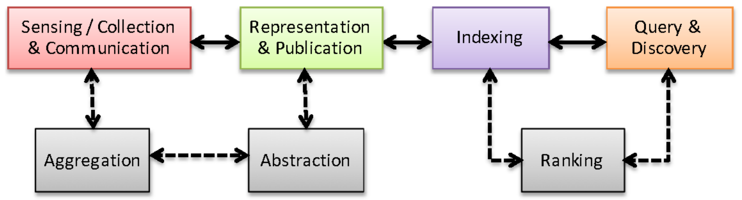
Sensing / Collection & Communication
Representation & Publication
Indexing
Query & Discovery
Aggregation
Abstraction
Ranking

Fig. 1. The process chain - from connecting IoT resources to making them discoverable and searchable

data is then abstracted to inform users about high/low traffic level and recommend routes. Smart home applications require data abstraction to detect events (e.g. fire) at home. Other applications do not require to aggregate and/or abstract the data, and the applications need to access (raw) sensor data of a single sensor (e.g. temperature sensor reading in a certain room). Searching for patterns (e.g. temperature more than 24°C in a certain location) or answering complex queries (e.g. get all temperature sensors whose locations are at most 2 km from my current location, get all sensors that have observed extremely low visibility within the last 30 minutes) require complex analysis and discovery of IoT data and resources. In applications where users are interested in getting measurement values (e.g. humidity level) in a specific location, and there are many humidity sensor devices in the same location, users may need a ranking mechanism for the results. The following lists the key design considerations for IoT data indexing and discovery.

(1) **Sensing/Collection**: includes interactions and communications between different sensory devices on a network and the necessary mechanisms that allow seamless communication with these embedded devices. Sensory devices can register to the network or the network itself has to crawl and detect them automatically.
(2) **Aggregation**: data from multiple, distributed and pervasive sources is combined and aggregated before being transmitted to a base station (e.g. a sink node or a gateway) for further processing. The key challenge in data aggregation is that data from different sources could have different data models and various granularities.
(3) **Abstraction**: is to create a higher-level description of the raw data that suffices to infer information and insights from the data such as patterns and events. Transmitting abstracted data instead of all raw data typically reduces communication and power consumption in in-network processing.
(4) **Representation**: IoT data can be individual observation and measurement data, or it can be time-series represented as streaming data. The data attributes are usually numerical, however, they can also be categorical (e.g. sensor type). The data can be stored in different formats such as Comma-Separated Values (CSV), JavaScript Object Notation (JSON) and others. IoT data should be represented in human-readable/machine-understandable formats.
(5) **Publication**: IoT data usually requires to be accessed and integrated with other real-world data; it needs to be published in human-readable/machine-understandable formats. It should also be published and stored in distributed locations (e.g. repositories or cloud services). The publication process might involve semantic annotation or interpretation of multiple sources to express and represent the data in a formal language and allow interoperability between the sources. Interoperability between multiple sources provides high-level semantic reasoning for low-level sensor data. It is noteworthy that deciding whether IoT data should be published with/without abstraction and/or aggregation is often application dependent. For example, it is required to process all raw data in healthcare applications because every single

data point can be significant. Whereas, data should be aggregated in traffic applications to summarise the data (e.g. high traffic, low traffic) and reduce the computational, storage and communication overhead within the network.

(6) **Indexing**: indexing large volumes of heterogeneous and dynamic IoT data/sources requires distributed, efficient and scalable mechanisms that can provide a fast access and retrieval to data in order to respond to user queries. A decision on how often these indexes should be updated and re-arranged while data streams are continuously published is crucial to enable on-line indexing. With on-line indexing, building indexing structures is incremental with the goal to update the indexes continuously with the new connected resources and the data that becomes available on the network, without re-building the entire indexing structure.

(7) **Discovery**: data discovery is about accessing specific sources to get the requested data and analyse it. On-line data discovery brings another dimension to play; access and analysis of on-line data from different sources, while other data is continuously published, is a challenging task. The accessing process is based on requested queries to find (search) data sources, patterns or events. Collaborative services are used on the obtained data from different sources to make analysis and provide intelligent decisions. Data can also be stored in repositories for temporary (short-term) or archived for long-term use. Data discovery could be limited by a time interval (time between two consecutive data points) in the processing of data for disaster monitoring.

(8) **Ranking**: ranking IoT resources requires the prioritisation of several criteria (e.g. data quality, devices availability, efficient energy and network bandwidth and latency), especially if there is the same type of services (e.g. temperature) at the same location. The ranking is a multi-objective decision-making process in which different criteria should be considered depending on the requirements and the network/device status. For instance, healthcare applications require trust and high-quality data. Emergency cases require transmitting and processing data with low latency to provide on-line command and control mechanisms.

(9) **Query**: end-users (i.e. machines or human users) in IoT applications may discover and query a certain data type at a particular location and at a specific time (e.g. current temperature in Guildford). A query can be composed of type, location, and time attributes (i.e. “exact query”). For instance, a query can be expressed as getting the temperature value (i.e. type) in Guildford (i.e. location) now (e.g. freshness ≤ 5 seconds). Other possible types of queries are proximity, range and composite queries [Barnaghi et al. 2013b]. Accessing location data to find a source of a particular type (e.g. temperature) needs to be handled by a discovery mechanism.

It is worth mentioning that deciding which data should be collected/crawled, how the data is represented and published and how users can interact with different applications are often application dependent. Indexing could be for a resource (i.e. sensory devices) or data (data published by resources). Indexing the resources allows on-line access to the resources and their published data at anytime.

Similar to the architecture for publishing and storing sensor data that has been discussed in [Barnaghi et al. 2013b], a refined model for indexing, discovery and ranking of distributed IoT data is shown in Fig. 2. We define Discovery Services (DSs) as services that allow on-line accessing, searching and analysing the on-line IoT data or the historical data that is stored and archived in information repositories. The main responsibility of Gateways (GWs) is to provide effective and efficient communications between upper layers and sensory devices. At gateway level, a resource discovery service is required which is a service that allows network-enabled devices sharing their data and services, to attach/detach to a network seamlessly with minimal effort [Guinard et al.

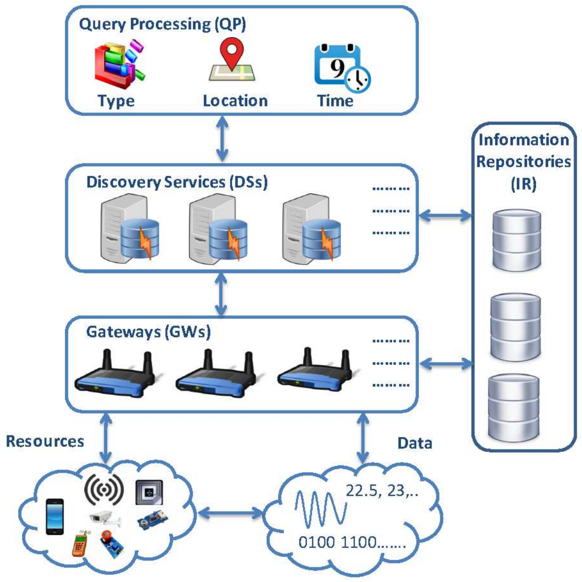

Fig. 2. A higher-level model for on-line IoT data analytics

2010]. Resource discovery plays a vital role in IoT applications to map physical objects from real-world into the digital world and provide a smart control and access to sensor nodes. Sensor nodes which are sometimes called "motes" [Levis and Culler 2002] contain one or more sensors that are attached to a board to provide processing, communication and storage capabilities. However, the motes usually operate with limited resources (Appendix A provides more details about common constrained operating systems that can run on motes). A mote can perform simple tasks such as detecting, recognition or monitoring, as well as complex tasks such as detecting certain movements or tracking objects, wherein coordination between different sensor nodes is essential [Römer et al. 2002]. Therefore, any solution/mechanism for IoT needs to consider the resource limitations.

DSs are responsible for distributing data among different repositories for long-term storage and on-line processing and mining for streaming data. Each DS holds a cache to store the most recently/frequently used or important data. Data could be accessed by DS, GW or be published directly to Information Repository (IR). Query Processing (QP) receives user queries and forwards the queries to DSs. DSs locate a set of related GWs that might have a resource that provides a response to the requested query or DS can search data that is stored in IRs. Fig. 3 depicts different possible scenarios where users may interact with IoT applications through DS. Users can discover different types of IoT data and services (as stated earlier in Section 1): a resource (e.g. sensor, service, device), data (e.g. sensor measurement/observation) and higher-level abstractions to obtain answers to complex queries that require collaborative analysis for different sources for advanced applications. We describe IoT data models and representation and various forms that the IoT data can be described before discussing how different attributes can be extracted, indexed and searched.

3 DATA MODELLING FOR THE IOT

This section describes what makes IoT data special by explaining the intrinsic characteristics of IoT data with some examples. It then explains how IoT data and resources can be queried. A discussion on how different types of IoT data can be represented is finally presented.

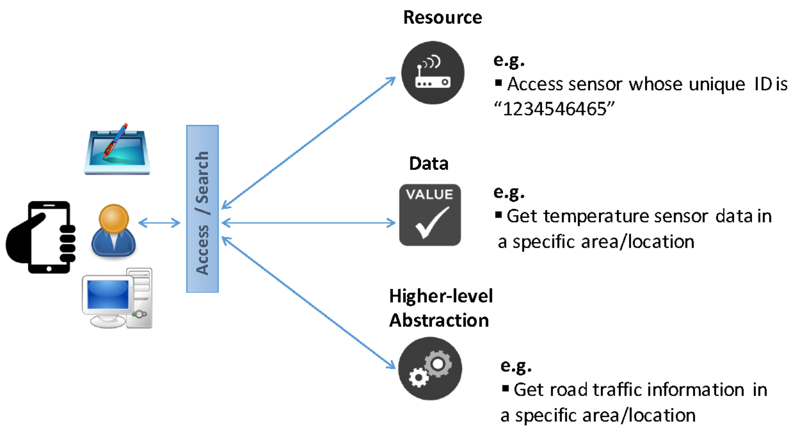
Resource
e.g.
▪ Access sensor whose unique ID is "1234546465"
Data
VALUE
e.g.
▪ Get temperature sensor data in a specific area/location
Higher-level Abstraction
e.g.
▪ Get road traffic information in a specific area/location
Access / Search

Fig. 3. How users may interact/discover IoT resources and services

3.1 What Makes IoT Data Special?

IoT and WSN data are interrelated. IoT data is collected from wired/wireless networks of smart things (devices, sensors and services). The data from the connected things is continuously generated as data streams. The massive data streams are temporally ordered and changing over time. IoT data streams are different from conventional data streams [Barnaghi et al. 2013b] as they have spatio-temporal dependency (time and location dependent). They are often generated with meta-data from large volumes of heterogeneous distributed resources; and consequently IoT data streams have a wide variety of representations. IoT data is also a type of big data that is usually collected from resources that are deployed in different geographical areas [Barnaghi et al. 2013a]. The characteristics of big data have been described using the five V's: Volume, Velocity, Variety, Veracity, and Value [Demchenko et al. 2013]. IoT data does not have only big data characteristics but also has distribution, spatio-temporality and dynamicity characteristics. The data is often generated in very dynamic and volatile environments [Barnaghi et al. 2013a]. The following explains the characteristics of IoT data:

- **Distribution**: refers to how IoT resources are arranged or spread out over geographical areas. There are multiple types of IoT resources ranging from Global Positioning System (GPS)-enabled vehicles for traffic control applications to metering (e.g. water and energy) and environmental (e.g. temperature and humidity) sensors for smart building management applications and others. The devices are deployed in distributed geographical locations and different environments.
- **Volume**: refers to the available amount of data for IoT consumers (users and applications). A massive number of sensor and actuator devices produce enormous volumes of data. For example, IBM estimates that car sensors produce 1.3 gigabytes every hour[4]. As mentioned earlier, it is foreseeable that the number of connected devices to the Internet will reach to 50 billion devices. As IoT grows, tremendous volumes of IoT data will be generated. To this end, more data will be produced at extremely large scale.
- **Velocity**: refers to the speed of data collection from IoT resources. Multitudes of resources produce data at different rates. For example, GPS-enabled vehicles in road networks generate data at higher rates (every few seconds or few minutes) for traffic management [Qin et al.

[4]http://www.ibmbigdatahub.com/blog/big-data-wheels

2016]. Weather sensors often produce readings at low rates (e.g. one reading per hour) for weather monitoring. However, some sensors such as Tekscan sensors have scan rate up to 1 million elements per second[5]. Dealing with the data that arrives at different rates is a challenging task [Qin et al. 2016].

- **Variety (Heterogeneity)**: refers to the variety of data (e.g. text, numeric, audio) captured by IoT resources. Data is generated in a wide range of formats and with various modalities by different devices. The data could be structured (e.g. tables, records), semi-structured (e.g. eXtensible Markup Language (XML)) or unstructured (text expressed in human language such as audio/video) [Chen et al. 2014]. The devices are built by different manufacturers and communicate via various protocols for different purposes. The aggregation of services produced by IoT resources has led to having diversity, multi-modality and heterogeneity in the data [Qin et al. 2016].
- **Veracity**: refers to the quality of IoT data and resources (e.g. accuracy, uncertainty and resource availability). The availability of a resource might change over time due to battery life, mobility or other issues. Distributed IoT resources can produce duplicated, missing or incomplete sensor readings due to network outages. For example, in traffic management applications, every vehicle can be identified by a unique RFID tag and vehicle's location can be captured by GPS. In this case, the data is redundant if multiple RFID readers read the same tag, and the data is uncertain when RFID readers report non-existing IDs or fail to read the tags.
- **Value**: refers to the usefulness of the data collected from sensory devices. IoT consumers are usually interested in leveraging information and insights from massive observations of raw data to find, extract and capture higher-level abstractions (e.g. events and patterns). For example, detecting "door closed" or "door open" events in smart home applications and detecting abrupt change in temperature in fire alarm systems.
- **Spatio-temporality**: refers to the spatial and temporal features of IoT data and resources. IoT devices often capture data with their current location and the time the data was taken. For example, in real-time railway applications[6], GPS unit is attached to every train to enable finding the departures or arrivals from any station and at anytime. The applications need to track the location and time of trains to provide users with reliable and accurate real-time information.
- **Dynamicity**: refers to the changes of IoT data and/or resources over time. IoT data is continuously updated. For example, in real-time tracking and monitoring applications (e.g. real-time traffic), the locations of the objects can change continuously over time, and as a result, their observations change to reflect the real-world. Overall, in Wireless Sensor and Actuator Networks (WSANs), many resources are connected and communicated on the network; however, the connection between resources and the network can be sporadic. Therefore, IoT data is often generated in dynamic and volatile environments.

3.2 Queries

As stated earlier, IoT data has particular characteristics such as spatio-temporality, heterogeneity to name but a few, and the data can be generated in highly distributed and dynamic environments. These characteristics affect how the data can be used and queried from different sources. To this end, distributed and scalable indexing, discovery, and ranking are key enablers to allow efficient distributed queries [Barnaghi et al. 2013a]. The query is often composed of a set of data attributes

[5]http://www.tekscan.com/support/faqs/what-are-sensors-sampling-rates
[6]http://www.realtimetrains.co.uk/

Page 11 of 67 Computing Surveys

such as type, location, time and other meta-data attributes (e.g. data quality) that describe the observation and measurement data. Different types of queries are summarised in [Barnaghi et al. 2013b]. For example, an approximate query is to provide an estimated answer to the query (e.g. temperature in London with data quality $\geqslant 70\%$). A range query is another type of queries, where a range of values of an attribute (e.g. location range) is requested (e.g. get the temperature at a given latitude $(50 - 51)$ and longitude $(0.10 - 0.12)$ within the last 5 seconds). These queries rely on data published from resources. However, IoT consumers can also query a resource (e.g. by its identifier) or higher-level abstractions (e.g. traffic in Oxford street) where data from different sources is integrated and processed. To better allow querying of IoT data and resources, different query languages and methods exist based on the way the data is represented. For instance, RDF Query Language (SPARQL) [Prud'Hommeaux et al. 2008] is used to query IoT resources whose data and other meta-data attributes are structured in Resource Description Framework (RDF) [Klyne and Carroll 2006] format. The mechanism of querying IoT data (e.g. query languages) is affected by its representation (e.g. numerical/symbolic and discrete/continuous) and formats (e.g. JSON, XML). Detailed discussion about different data representation models in the IoT is included in the next section. Although there is work on standardising query languages for querying IoT data, the existing IoT data indexing and discovery approaches define their own mechanism of querying IoT data (Section 6 includes more details).

3.3 Data Representation in the IoT

IoT data can have different representations and can be stored in various formats. Data representation in IoT is driven by the characteristics of the data collected from various IoT resources. Furthermore, IoT data might have different types that can be broadly categorised into three groups: numerical vs. symbolic, discrete vs. continuous and static vs. streaming. These categories and their explanations are summarised in Table 3. It is worth noting that IoT data can be a mix of several categories. For example, GPS data for tracking public transport or package tracking is continuous numerical data. The following discusses data representation concerning the characteristics and various formats of IoT data.

Table 3. Data classification

Data Category	**Explanation**
Numerical vs. Symbolic	• Numerical refers to data that has numerical values (e.g. $10°C$ for temperature, 50 db (decibels) for sound). • Symbolic refers to data that has string/text values (i.e. a set of characters). For example, events observed by sensors are symbolic, such as "Cold Temperature" where numeric readings of sensor data should be within a particular range (e.g. $(0 - 5)°C$). Some sensor data attributes are also symbolic such as sensor type and description.
Discrete vs. Continuous	• Discrete refers to data with a finite or a limited set of values. Discrete data can be numeric. For instance, relative humidity sensor has a single value between $(0-100)\%$. Discrete data can also be symbolic (categorical). For example, presence detector has a single value from a set of possible values (presence or absence of humans) for monitoring light (turning lights on/off) in a room. • Continuous refers to data with an infinite set of values such as continuous observation and measurement from sensory devices (e.g. data provided by a dynamic accelerometer for continuous monitoring of acceleration of movable objects).

https://mc.manuscriptcentral.com/csur

Static vs. Streaming	• Static refers to data that does not change. For example, the location of a static sensor. • Streaming refers to data that changes over time such as real-time data. For example, the data collected from mobile sensors is data streams which have different sensor readings over time due to varying sensor's location.

3.3.1 Data Classification.

There are different categories of IoT data; discrete or continuous (i.e. sensor readings are a series of values over a continuum) values [Cooper and James 2009] or generated (synthesis) data based on predictive models such as [Jones and Thornton 2000] and [Schuol and Abbaspour 2007]. IoT data is sometimes a mix of discrete/continuous and symbolic attributes/variables (e.g. sensor location, sensor type) [Harada 2002]. IoT data can be represented as a series of numerical measurements, or the data could also be text from micro-blogging and social media. Understanding different IoT data models and structures is beneficial for designing and developing IoT applications and environments.

Real-world data from sensory devices is commonly time-series data which is represented as streaming data [Lin et al. 2003]. Data stream is a term to define an infinite sequence of temporal data that is generated at different rates over time (as stated in explanation of velocity characteristic of IoT data in Section 3.1). The data streams can be subject to concept drift. Data drift happens when unbounded data streams do not have the same distribution over time [Yang and Fong 2013]. Time-series data is a sequence of data points in which each point p is represented by $p = (v, t)$, where v is a real-value and t is the time on which v is obtained [Zoumpatianos et al. 2014]. It is worth noting that data streams could be either constant or variable in terms of time between consecutive data streams [Kranen and Seidl 2009]. When a data stream item arrives, it can be processed and analysed on-line and often stored for short-term or long-term in repositories based on the application requirements. More concretely, there are two possible approaches for processing data streams; either processing all M streams with length L that have arrived so far, or a subsequence (sampling) of the current arrived streams (e.g. last m streams with length l, where $m \in M$ and $l << L$) [Guha et al. 2003; Keogh and Lin 2005]. The latter is called "sliding window" data streams.

The inherent features and characteristics of big data in IoT (as stated earlier in Section 3.1) preclude loading, managing and storing it in a traditional Database Management System (DBMS) which is feasible for static more than sporadic and dynamic data [Babcock et al. 2002]. The dependency between data streams imposes a challenge in analysing them on-line with minimal processing and communication overhead as well as storage. Higher-level representation of high granular raw data is often used as an alternative for processing continuous data streams. The higher-level representation requires reducing raw data space and possibly classifying it. Similarly, this is typically necessarily for any Web search engine that deals with large-scale documents and Web pages, in which each document should be represented by a fixed-length vector and usually unsupervised techniques are applied to classify them.

Overall IoT data could be simple (e.g. air pollution and humidity) or more complex (e.g. patterns and events). IoT data itself is small because it is often numeric values. However, meta-data (e.g. data quality, source state, location and battery life) is usually larger than the data [Barnaghi et al. 2013b]. For example, a temperature sensor captures 15$^{\circ}C$ as a temperature of a room. The numeric value represents the data. However, meta-data could be a description of the specification, identity, location, manufacturer and battery life of the sensor, data unit (Fahrenheit $^{\circ}F$ or Celsius $^{\circ}C$) and data owner. However, meta-data could be smaller than the data itself if the data is conventional multimedia data (e.g. video and audio). Meta-data typically makes IoT data more expressive and meaningful.

3.3.2 *Data Formats.*

IoT data can be represented and stored in different formats. The data can be stored in a text format as Comma-Separated Values (CSV) (tabular) files [Huang et al. 2014], JavaScript syntax: JavaScript Object Notation (JSON) format or in a hierarchical structure using eXtensible Markup Language (XML) format such as SensorML [Botts and Robin 2007]. However, models such as SensorML are too complex and have a high overhead to provide representations for large-scale IoT data [Barnaghi et al. 2013b]. CSV and XML formats are often used to integrate different IoT data sources into Web applications, and JSON is used in Representational State Transfer (REST)-based applications [Piyare and Lee 2013]. However, these various formats do not have "explanation" added to the data (e.g. meta-data) or a formal query language. RDF is one of the most widely used frameworks to describe Web resources by adding semantic meaning to their data model (meta-data data model), and it is a building block for standard Semantic Web. RDF represents semantic data as triples (subject, property, object); the subject has a property that links it to a value or another object. RDF provides a single semantic model for data that can be represented in various formats, known as serializations; RDF/XML [Beckett and McBride 2004], RDF/JSON [Davis et al. 2013]. The Simple Protocol and RDF Query Language (SPARQL) allows querying data that is stored in the RDF format. Other alternative RDF serialisations include N-Triples [Beckett and Barstow 2001], Notation-3 (N3) [Berners-Lee and Connolly 2011] and Turtle [Beckett et al. 2011]. They represent data as triples, but they are different in their level of expressivity and are designed for Web applications [Su et al. 2015]. Recently, efficient XML Interchange (EXI) is recommended for IoT applications [Schneider et al. 2011] due to its optimisation in converting XML messages into binary form to allow transmission with low bandwidth.

Linked Data [Bizer et al. 2009] allows data items to be linked to concepts that are defined in commonly used ontologies or vocabularies [Barnaghi et al. 2013b]. Linked Data also allows links between sources based on their meta-data and related attributes and provides links between different RDF data. The SSN ontology has been proposed by the W3C's[7] Semantic Sensor Network Incubator group (SSN-XG) to provide a formal description of sensor capabilities and its measurement and observation data [Compton et al. 2012]. The Web Ontology Language (OWL)[8] is a semantic Web language that allows publishing ontologies on the Web. Ontology Web Language for Services (OWL-S)[9] is a semantic language for describing Web services. OWL-S provides a set of standard vocabulary that can be used with OWL to create semantic service description. Sensor Markup Language (SENML) [Jennings et al. 2015] is also designed specifically for resource-constrained devices; it defines a data model for transmitting simple sensor measurements into application layer protocols requests such as HTTP or Constrained Application Protocol (CoAP) [Bormann et al. 2012].

The data can be represented and processed as individual items or streams. However, some applications may use aggregation and abstraction to store higher-level representations such as patterns and/or events. In this case, the discovery can also be performed based on the higher-level abstractions.

[7]World Wide Web Consortium (W3C)
[8]http://www.w3.org/OWL/
[9]http://www.w3.org/Submission/OWL-S/

4 DATA AGGREGATION AND ABSTRACTION

This section describes IoT data preparation including data pre-processing to filter unwanted data and dimensionality reduction to reduce the size of the original data. It then explains how the prepared data can be summarised using data aggregation, integration and abstraction techniques.

4.1 Data Preparation

Collected raw data from various sensory devices could be noisy, redundant or come at sporadic rates and with different distributions; data pre-processing is often a key requirement. Pre-processing techniques can be either mathematical function (minimum, maximum, range) in the time-domain, wavelets and Fourier transformation in the frequency domain or Discrete Wavelet Transform (DWT) and others in the discrete domain. Interested readers can refer to the survey of [Figo et al. 2010] for a detailed discussion about different pre-processing techniques. After pre-processing of the raw data, dimensionality reduction techniques can be applied to obtain low granularity data that retains the most relevant information of the original form [Ganz et al. 2015].

4.2 Dimensionality Reduction

To handle large amounts of data collected from various IoT resources, different dimensionality reduction techniques can be applied to obtain a lower dimensional representation of the original data such as Singular Value Decomposition (SVD) [Golub and Reinsch 1970], Piecewise approximation approaches [Keogh et al. 2001b; Yi and Faloutsos 2000], symbolic models [Lin et al. 2003] and several other solutions. Interested readers can refer to Appendix B for more details about different dimensionality reduction techniques.

After the pre-processing and dimensionality reduction of raw IoT data that is collected from various sources and sensory devices, the data can be aggregated into information that allows fast search and query for higher-level abstraction of events and patterns, to the compression of data for transmission over the network. Data summarisation is a transformation of higher granular data into a compact description with a goal of retaining enough information about the raw data. The data can be summarised by applying aggregation, integration and/or abstraction processes to extract knowledge and create meaningful insights.

4.3 Data Aggregation

Data aggregation is part of "data fusion" [Rajagopalan and Varshney 2006]. Data fusion is a general term that refers to integration and combination of data from distributed sources while data aggregation gathers measurement and observation values from multiple heterogeneous data sources by removing redundant data and/or eliminating multiple transmissions to provide fused information with reduced energy and communication overhead [Krishnamachari et al. 2002; Rajagopalan and Varshney 2006]. For example, temperature measurements from multiple data sources are aggregated together (e.g. averaging temperature values of different sources in the same location). However, aggregation could also be between different types of sources. For instance, creating a comprehensive city traffic application is based on getting vehicle traffic flows, bicyclists and pedestrian counts using different IoT data resources.

Aggregation techniques can be applied in-network wherein an aggregation node receives a packet (data) from different sources and aggregates or merges the data into one outgoing packet. In-network aggregation can be combined with data reduction when the aggregation node receives data (packets) from sensory devices that have the same type (e.g. it receives packets from different temperature sensors, and it aggregates them into one outgoing packet that contains their average). The aggregation can also be without reduction by merging received data from sensory devices

Page 15 of 67 Computing Surveys

that have different types (e.g. air pollution and humidity) into one packet (instead of forwarding two packets) without any processing (e.g. average, maximum, minimum) of the data [Fasolo et al. 2007]. In-network aggregation can be quite complex if it requires co-ordination between the various distributed nodes in the network.

[Krishnamachari et al. 2002] propose a data-centric routing mechanism for aggregating sensor data in a network. The mechanism applies simple operations (e.g. min, max) on sensor data. The experiments were conducted on 100 sensors, and the number of sinks is varied from 1 to 15. However, there is high traffic in the centralised approach if a large number of sensors produce data at high rate and scalability becomes an issue. [Rooshenas et al. 2010] present a distributed aggregation mechanism on intermediate nodes based on Principal Component Analysis (PCA) to optimise packet transmission to a base station. The intermediate nodes are selected with a constraint that at most two hops are required to connect a node to a base station. Other different aggregation approaches also exist. For example, aggregating nodes with their neighbours or aggregating correlated nodes before transmission. Neighbouring nodes are often highly correlated, so transmitting the difference between their measurements is often enough to represent the data. In typical WSN, gateways are responsible for aggregating data received from nodes (i.e. sources). [Troubleyn et al. 2014] propose to broadcast in-network aggregation that reduces energy consumption by roughly 27% in full mesh (i.e. multipoint-to-multipoint) WSN. In their approach, a set of packets is transmitted at once if a certain high degree of aggregation reaches. However, if the number of connected nodes is increasing, there will be an increase in traffic, consequently, scalability and latency become an issue.

4.4 Data Integration

Integrating data from different sensory devices is different from aggregating them. Data integration is to connect and combine heterogeneous data sources in a system/an application (e.g. an existing Web application) [Barnaghi et al. 2012b], while data aggregation includes data from different sources in a summarised form. Integration of heterogeneous data sources requires interoperability among them. Interoperability allows different IoT and Web data sources to convey the unambiguous meaning of data between them. Interoperability enables loose coupling between the various sources while integration enables tight coupling [Kotis and Katasonov 2012] in which mismatching between different sources protocols have to be addressed to allow communication between different devices. To this end, the Open Geospatial Consortium (OGC) proposes Sensor Web Enablement (SWE) standard to allow exchanging and accessing data from heterogeneous sensors through Web service interfaces and communication protocols. Also, to provide integration and interoperability between the sensor data, OGC provides Sensor Model Language (SensorML) as a common standard model to describe their sensor observation and measurement data [Botts et al. 2008]. Semantic Sensor Web framework is proposed by [Sheth et al. 2008] to enable interoperability by annotating heterogeneous sensor data semantically. [Barnaghi et al. 2013b] also propose a model to describe annotated data streams semantically. The data is annotated using a pre-defined template, and this facilitates the interoperability between different data models. The sensor data is then clustered (by k−means) to be distributed among repositories for answering different types of SPARQL queries. It is worth noting that the way the data is aggregated and/or integrated from various sources has an effect on how the data is abstracted for conceptualising the real-world.

4.5 Data Abstraction

Data abstraction is a transformation of lower-level raw data (e.g. 2°C) into a higher-level information that describes patterns or events (e.g. cold weather). [Sigg et al. 2012] define abstraction as the amount of processing that can be applied to the data with an intention to raise the abstraction

https://mc.manuscriptcentral.com/csur

level. [Ganz et al. 2015] explain the abstraction as the derivation from raw sensor data into valuable information (e.g. high traffic, windy weather, light is on). It is worth noting that attaining high descriptive data by selecting a set of the most relevant/informative features (i.e. feature vectors) to describe the original data is called "feature extraction". The data could also be classified and grouped based on a set of features, events, or patterns using clustering algorithms (e.g. $k-$means).

[Chen and Kotz 2002] propose an abstraction approach to aggregate raw data into an abstract graph to support developing context-aware applications. [Sigg et al. 2012] apply different levels of context abstraction on raw sensor data to study the effect of the quality of abstraction levels, raw input data as well as the order of context operations (acquisition, interpretation and prediction) on the accuracy of context prediction. [Ganz et al. 2015] apply a two-level abstraction: lower-level on raw data and higher-level (semantic abstraction) to obtain meaningful abstractions by applying machine-learning techniques and reasoning mechanisms on the lower-level abstracted data. Inferring higher-level abstraction and representation from raw sensor data allows searching for events and patterns. Overall aggregation and abstraction approaches are based on various applications scenarios (e.g. e-health, smart home, disaster and environment monitoring).

It is worth mentioning that aggregation, abstraction and full interoperability between all different types of heterogeneous IoT resources (e.g. sensory devices, Web services) are still far from ideal to subsume their heterogeneity in different data models and various protocols [Barnaghi et al. 2012b; Desai et al. 2015; Miorandi et al. 2012; Zeng et al. 2011]. IoT data is generated by different devices and resources in physical and cyber domains. The way that the data is published and provisioned has an impact on crawling, accessing and indexing of the data. The following discusses publication and provisioning of IoT data.

5 PUBLICATION AND PROVISIONING

This section introduces how IoT data can be published. It then compares a range of available IoT protocols and discusses their features that drive them to be considered desirable for a broad range of IoT applications. The section also includes the essential comparison criteria on which protocol(s) should be selected for a certain application or a use case. The advantages, disadvantages of these protocols and example of applications where these protocols are used, are summarised at the end of the section.

IoT data is collected and published with different streaming qualities and various granularities. Data providers have different ways of publishing the data to make it available, understandable and accessible to the public. The collected data is stored in a central data repository or the Cloud. The data can be accessed directly from the data source (sensory device) or via an Application Programming Interface (API)/Web service. The way the data source is connected to the Internet to publish its data is based on the application requirements (e.g. real-time communication, interoperability, Quality of Service (QoS), client/server or publish/subscribe, message queuing or data-centric). The data could be published as raw data or with meta-data to facilitate annotating the data semantically and enable interoperability with other data sources. Due to different data models from heterogeneous IoT resources, it is still a challenging task to have (dynamic) automated semantic interoperability of IoT data [Barnaghi et al. 2013a]. Heterogeneity in the IoT can be handled at different levels. Heterogeneity can be at the data level such that data is generated in a wide range of formats and with various modalities by different devices. The data can be structured (e.g. tables, records) or semi-structured (e.g. XML) (as stated earlier in Sections 3.1 and 4.4). Heterogeneity can also be at the communication level such that IoT resources (e.g. sensors) communicate via various protocols for different purposes.

Publishing IoT data and services on the Web is only one component to build IoT applications. Another dominant factor is that building IoT environments and applications requires a reference architecture that considers a network model to allow heterogeneous IoT resources to connect directly or through a gateway to the Internet forming Machine-to-Machine (M2M) networks for publishing, managing, and processing the IoT data and actuation commands. Regardless of the way the IoT resources are connected to the Internet, their collected data is stored/archived into a central repository or in a distributed Cloud [Karagiannis et al. 2015]. The data could also be processed and analysed close to where it is collected (at the edge of networks - fog computing) such as in Smart and Connected Vehicle (SCV) applications [Bonomi et al. 2012]. There is no standard or unified architecture that can be deemed as a blueprint for all concrete IoT development and deployment. To this end, several standards organisations are involved in developing and promoting unified and consolidated standards to offer connectivity for IoT devices. For example, the Institute of Electrical and Electronics Engineers (IEEE) provides standardisations for the physical, Media Access Control (MAC) and application layers such as IEEE 802.15.4 over Low-power, Wireless Personal Area Network (LoWPAN) to meet the new requirements thrown up by IoT (e.g. low power and low data-rate). A detailed discussion about standardisation of the ISO/OSI and TCP/IP stacks for the IoT is outside the scope of this article, and the reader is referred to [Al-Fuqaha et al. 2015; Bartoli et al. 2011; Hui and Culler 2008; Palattella et al. 2013].

The Internet Engineering Task Force (IETF)[10] has three main working groups that allow native connectivity between network-enabled devices and the network (e.g. Internet); 6LoWPAN, Routing Over Low Power and Lossy Networks (ROLL) and Constrained Restful Environments (CoRE) working groups. The 6LoWPAN working group provides an adaptation layer to provide an efficient IPv6 networking through incorporating IEEE 802.15.4 over LoWPAN into (6LoWPAN) [Vasseur and Dunkels 2008] that is suitable for resource-constrained devices and sensor networks. ROLL provides IPv6 routing solutions for Low Power and Lossy Networks (LLNs). Moreover, CoRE provides a standard framework to integrate IP-based constrained devices into the Internet. Constrained Application Protocol (CoAP) [Bormann et al. 2012]- an IoT application layer protocol has been developed as a part of the framework to support Web-based applications. For a detailed discussion on IETF standardisations in the IoT, the reader is referred to [Ishaq et al. 2013; Sheng et al. 2013]. Moreover, ulPv6 [Durvy et al. 2008] is a smaller IPv6 stack which is a further step to enable communication interoperability between IPV6-enabled sensor devices and other IPv6-enabled devices on the Internet.

The Internet Protocol for Smart Objects (IPSO) Alliance[11] also promotes the use of Internet Protocol (IP) for the networking of embedded sensor and actuator (smart) devices to transmit their observation and measurement data in the IoT domain. IPSO also complements the effort of other standards organisations such as IETF, IEEE, the Industrial Internet Consortium (IIC)[12], the Open Connectivity Foundation (OCF)[13](formerly Open Interconnect Consortium (OIC)), World Wide Web Consortium (W3C)[14] and others by documenting and running interoperability tests of different IP-based standards released by these standards organisations [Ko et al. 2011]. Interoperability between different implementations of CoAP on various platforms has been performed by the European Telecommunications Standards Institute (ETSI) [Ishaq et al. 2013].

6LoWPAN and the smaller IPv6 stack (ulPv6) allow communications at lower layers, but different application layer protocols still hinder the seamless interconnection and data exchange between sensory devices. Some application layer protocols are designed to be suitable for resource-constrained IoT devices with a limited power, memory and processing capabilities such as CoAP, Message

[10]http://www.ietf.org/
[11]http://www.ipso-alliance.org/
[12]http://www.iiconsortium.org/
[13]http://openconnectivity.org/
[14]http://www.w3.org/WoT/IG/

Queueing Telemetry Transport (MQTT) [Hunkeler et al. 2008; Locke 2010], Advanced Message Queuing Protocol (AMQP) [Vinoski 2006], IETF's eXtensible Messaging and Presence Protocol (XMPP) [Saint-Andre 2011] and Data Distribution Service (DDS) for real-time systems [OMG 2006]. We emphasise that the interoperability between application layer protocols is required, but the interoperability support for other lower protocol layers of the TCP/IP protocol stack is indispensable [Zeng et al. 2011]. However, it is still hard to have one design for the interoperability between all protocol layers that can fit for all various Internet-connected devices. We summarise the differences between the protocols mentioned above in Table 4. Our comparison criteria are: transport layer protocol (e.g. TCP, UDP), interoperability between other protocols, QoS for message delivery (delivery notification) and messaging exchange pattern (e.g. request/response).

Table 4. Characteristics of different application layer protocols

Protocol	Transport	AllowInteroperability	Delivery Notification	Messaging Pattern
CoAP	UDP	Yes [Palattella et al. 2013; Shelby et al. 2014]	• Confirmable • Non-confirmable • Acknowledgement • Reset [Shelby et al. 2014]	• Synchronisation (REST-based) request/response • Asynchronisation responses (with observer option)
MQTT	TCP	Yes (MQTT-SN [Stanford-Clark and Truong 2013])	• Exactly once • At least once • At most once [Locke 2010]	Publish/subscribe
XMPP	TCP	Yes [Saint-Andre 2011]	No delivery [Karagiannis et al. 2015]	• Asynchronous publish/subscribe • Request/response
AMQP	TCP	Yes [Vinoski 2006]	• Exactly once • At least once • At most once [AMQP Working Group and others 2012]	Asynchronous publish/subscribe
DDS	• TCP • UDP	Yes (DDSI) [Corsaro and Schmidt 2012]	It has nearly 23 QoS (e.g. topic, history, durability) [Al-Fuqaha et al. 2015; Corsaro and Schmidt 2012]	Publish/subscribe (multicast)

The REpresentational State Transfer (REST)-based services usually work over HyperText Transfer Protocol (HTTP). HTTP is an application layer protocol that is built on top of TCP. REST is an architecture design that allows developing Web services (RESTful) that communicate and access resources (elements) over HTTP by their Uniform Resource Identifier (URIs). CoAP supports a subset of HTTP functions; it uses HTTP commands (GET, POST, PUT, and DELETE) for interacting with resources. It is easily integrated into the Web; proxies are used to map HTTP into CoAP, and it can also create and support some semantic descriptions [Palattella et al. 2013]. However, there is a complexity for mapping the protocols [Kirsche and Klauck 2012]. CoAP supports a synchronous request/response approach. However, topic publication/subscription scheme is more suitable for IoT devices and applications [Kirsche and Klauck 2012]. CoAP is based on REST-style architecture and uses User Datagram Protocol (UDP) as an alternative transport layer protocol for TCP in HTTP. CoAP is a lightweight protocol with lower overhead (smaller packets than HTTP) that is accordant to IoT constrained devices with UDP communication and networks with limited resources. It supports asynchronous responses with a resource observation option; wherein clients request to observe sources and keep being informed by their changes [Bormann et al. 2012].

MQTT is a publish/subscribe messaging protocol which is developed by Organization for the Advancement of Structured Information Standards (OASIS)[15]. Unlike CoAP, which is based on UDP and supports request/response scheme, MQTT is TCP-based and provides de-coupling between publisher as data producer and subscriber as data consumer by supporting a message bus (one-to-many messaging distribution exchange over TCP/IP). It is also a client-server based, wherein a central broker (server/gateway) receives subscriptions from clients for certain sources/events (topics). MQTT targets machine-to-machine (M2M) communications. MQTT is also a simple and light-weight protocol with low data overhead that can be used in constrained networks. For example, MQTT's header is only 2 bytes which in comparison to AMQP which has header size 8 bytes and CoAP which has header 4 bytes, is much lower [Al-Fuqaha et al. 2015]. Unlike HTTP, which has no Quality of Service (QoS) for message delivery, MQTT supports three different kinds of QoS [Locke 2010]. The message can be delivered exactly once (message is sent only once, and duplicated messages are neglected), at least once (message is being sent until an acknowledgement is received) and at most once (message is sent once, and no response/acknowledgement is expected). The main drawback of MQTT is that MQTT clients must support TCP to connect to MQTTP broker. MQTT clients should also have an open connection to broker all the time, and MQTT broker also becomes a single point of failure. When a network has limited resources (e.g. Wireless Sensor Networks) or packet loss rate is high (high communication is required), the open connection becomes a bottleneck. MQTT does not directly support non-TCP/IP sensors and embedded devices. However, an extension of the MQTT which is called MQTT for Sensor Networks (MQTT-SN)[16] supports non-TCP/IP devices. MQTT-SN also allows interoperability between different devices [Stanford-Clark and Truong 2013].

XMPP is an open instant messaging (near real-time) and streaming eXtensible Markup Language (XML) protocol that has been proposed by IETF. It was developed essentially to allow communication between people through messages. It is TCP-based similar to MQTT; it supports asynchronous publish/subscribe approach that is suitable for the IoT and also supports request/response approach similar to CoAP. However, it has no QoS message delivery; it only depends on TCP's reliability that impedes it to support M2M communications. Decentralisation is the key advantage of XMPP [Saint-Andre 2011]; there is no central XMPP server, but individuals can deploy their XMPP servers and have control over it. Different clients with various server architectures can also communicate with each other [Mascetti et al. 2011]. For example, different servers in the Jappix[17] project, which is a social networking system, are connected and communicate through XMPP. Google has stopped supporting XMPP that was used in Google Talk and Hangout (which has replaced Google Talk). Intuitively, XMPP supports client-server and server-server communications. However, it is a text-based message exchange (XML format) which might have an overhead to interpret and parse XML [Karagiannis et al. 2015].

AMQP is a messaging queue-based protocol which is developed by OASIS. It is MQTT-like; it is based on asynchronous publish/subscribe approach. It has a message/exchange mechanism between queues to support one-to-one and one-to-many communications. Also, it relies on TCP and also has three different kinds of QoS similar to MQTT: at least once, at most once and exactly once for delivering the messages. AMQP supports routing producer-consumer, wherein consumer creates a message buffer (queue), and producer pushes the messages into the buffer.

DDS is a data-centric publish/subscribe standard that allows distributed, dynamic, scalable and real-time systems [Corsaro and Schmidt 2012]. It allows exchanging messages between publisher

[15]http://www.oasis-open.org/
[16]http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf
[17]http://jappix.org/

and subscribers. It allows detecting dynamic changes in meta-events; it provides content-based subscriptions. DDS is used in ProRail (a large-scale rail network management system with a reliable, real-time and fault-tolerance) and Volkswagen smart cars and other systems[18]. However, DSS does not support interoperability between different vendors/implementations. To allow interoperability, Data Distribution Service Interoperability (DDSI) Wire Protocol is developed [Management 2009]. It is argued in [Baldoni et al. 2011] that DDS should be deployed with strict settings to enable its powerful QoS and also the detection of real-time events from multiple heterogeneous sources. The high overhead of DDS's IP multicast is another drawback for deploying DDS on mobile nodes and wide-scale wireless networks [David et al. 2013; Esposito 2011].

Table 5. Advantage and disadvantage of different application layer protocols

Protocol	Advantage	Disadvantage	Application
CoAP	• Easily integrated with the Web [Palattella et al. 2013] • Stateless HTTP (easily HTTP mapping) [Shelby et al. 2014] • Light weight • Lower overhead [Shelby et al. 2014] • Multi-cast support [Shelby et al. 2014]	• Lack of topic publication/subscription approach [Kirsche and Klauck 2012] • Complexity for mapping protocols (application protocol) [Kirsche and Klauck 2012]	Integrated with HTTP and RESTful applications [Bormann et al. 2012]
MQTT	• Extremely Light weight [Hunkeler et al. 2008] • Subscription scheme by topic name [Locke 2010] • Can be used in low-bandwidth/unreliable network [Locke 2010]	• Centralised broker can be a point of failure (client connections with the broker are open all the time) [Hunkeler et al. 2008] • Clients have to support TCP/IP [Hunkeler et al. 2008]	For example: used in Facebook Messenger [Zhang 2011]
XMPP	• Persistent connection [Saint-Andre 2011] • Decentralisation (No central XMPP server) [Saint-Andre 2011] • Allow servers with different architectures to communicate [Saint-Andre 2011]	• No QoS [Karagiannis et al. 2015] • Streaming XML has overhead [Karagiannis et al. 2015]	For example: used in • Jappix project • Google Talk[19]
AMQP	• Store-and-forward capabilities [AMQP Working Group and others 2012]	• Low success rate with low bandwidth [Johnsen et al. 2013]	For example: used in JP-Morgan [O'Hara 2007]
DDS	• Suitable for real-time IoT [Corsaro and Schmidt 2012] • Has powerful QoS [Corsaro and Schmidt 2012] • Scalable, extensible and efficient standard [Corsaro and Schmidt 2012]	• Support IP multicast [David et al. 2013; Esposito 2011] • QoS polices are only applied in strict DDS environment [Baldoni et al. 2011] • Events are originated per source in a real-time not multiple sources [Baldoni et al. 2011]	For example: used in • ProRail • Volkswagen smart cars

The protocols mentioned above have different purposes. For example, while AMQP is a middleware-based protocol to build application and Web services, XMPP is used to access and communicate with applications that could be built using AMQP. MQTT and CoAP are used to collect data (resource discovery) from IoT devices or from services that are formed based on AMQP. More than one protocol could be utilised in one framework based on use cases and scenarios. However, many other protocols and technologies can be utilised in IoT. The key issue is to select the primary protocols that have best-fit in the target application or scenario. It is a challenge to determine the prominent protocols that are perfectly suitable for IoT domain. [Desai et al. 2015] propose Semantic Gateway

[18]http://portals.omg.org/dds/who-is-using-dds-2/
[19]Google has stopped supporting XMPP and Google Talk is replaced by Google hangout

as Service (SGS) to solve interoperability issue between different IoT devices that support different protocols. SGS supports multi-protocol proxy architecture and W3C's SSN ontology [Compton et al. 2012] to create interoperability and semantic reasoning between messages that are sent by different messaging protocols such as XMPP, CoAP and MQTT. The problem of creating bridging concept between protocols is discussed by [Waher 2015]. For example, if a device A supports CoAP and another device B supports XMPP: interoperability between different protocols is an issue. To resolve this, a pair of mappings (translation) between request/response of CoAP and XMPP can allow communication between these devices. However, protocol semantics add another dimension to play; MQTT supports only publish/subscribe semantics while CoAP supports observer and request/response patterns similar to XMPP.

Table 5 shows a comparison between the aforementioned protocols and their suitability for different application(s). When an IoT system/environment is deployed, different protocols can be used based on the target applications and scenarios. The communication and computational capabilities of the devices connected to the network should be taken into consideration while selecting the IoT protocols. To conclude, both of CoAP and MQTT are designed to be light-weight protocols and are suitable for networks of low-power and constrained devices. However, CoAP offers more control to give commands for a device. For example, CoAP can be used in IoT smart home applications to allow users to control their home appliances (e.g. air conditioning) via their smartphones. On the other hand, MQTT is suitable for battery-powered devices. It is also adequate for applications where interaction between devices might be required (readings from one device can control another device). For example, switching on/off air conditioning in a room is based on the temperature of the room obtained from a thermometer. MQTT is also suitable for applications that allow users to subscribe to a particular topic (e.g. a thermometer sensor publishes temperature data to a topic called "temp" and consumers subscribe to "temp" to receive updates about any changes in the temperature). AMQP has a store and forward feature which guarantees reliability to deliver messages over the network. For example, National Science Foundation (NSF)[20] uses sensor nets with AMQP support to collect readings from ocean platforms and allow dissemination of the readings over the network through publish-subscribe. XMPP is mainly used for wide range of chat applications. DDS enables real-time sharing and discovery of publishers and subscribers in scalable networks. For example, DDS can be used in distributed power management systems for smart energy where real-time sharing of data is required with low latency.

IoT resources can publish their data into a repository or store it temporarily on a gateway. Indexing mechanisms can then index the resources or the attributes of their published data. For example, a search query for traffic data in a certain street can return the number of cars and their velocity, or it can return link to resources that can provide this information (similar to most of the search engines on the Web that acts as information locators). Here we mainly discuss the indexing and discovery mechanisms based on IoT resources, the attributes of their published data, and higher-level abstractions. The gateway solutions and discovery at the resource/device level are discussed in Appendix C.

6 INDEXING AND DISCOVERY

It is impractical to scan all the data of all data resources to respond to user queries in a distributed system. Similar to Web search engines where Web pages are indexed to allow fast data retrieval, IoT resources and their measurement and observation data can be indexed. Indexing IoT data and resources can not be separated from data discovery. While indexing organises IoT resources and data to enable efficient search, discovery uses indexing to support query and search processes to

[20] http://oceanobservatories.org/

provide higher-level access to the resources and their data. Queries can be answered using either exact or approximate search; wherein it returns an exact result similar to full scan for all the data of all data resources in the former and relative similarity (closest match) of the requested query in the latter [Keogh and Ratanamahatana 2005]. As stated earlier in Section 3.2, there are three main types of queries: data, resource and higher-level abstractions. The following discusses indexing and discovery approaches based on these types of queries.

6.1 Data

We refer to data indexing as an approach to organise IoT data to enable fast search and retrieval of the data without identifying the data source. Data indexing can be categorised into spatial, thematic (e.g. XML) and time-series data indexing.

6.1.1 Spatial Data.

Spatial approaches enable indexing data based on its spatial features. The spatial (also called geographic) feature is often described by latitude, longitude and altitude coordinates. Some approaches have been proposed to map spatial data with two or more dimensions into a one-dimensional string (key) using space-filling curves (e.g. Z-order, Hilbert, Peano) and allow querying the data based on the constructed keys. Interested readers can refer to a detailed discussion about various space filling curves in [Lawder and King 2000; Mokbel et al. 2002]. [Zhou et al. 2014] propose a framework that supports spatial indexing for geographical values of collected observation and measurement data from sensory devices. The work is based on encoding the locations of measurement and observation data using geo-hash (Z-order curve). Geo-hash is a geocoding algorithm that is based on interleaving bits to convert spatial coordinates (longitude and latitude) into a single string. The framework has a Web-based query interface with spatial feature matching mechanism to match between requested user queries and indexed data. Users can search for data within a specific area (location) or within a certain distance from a given location. Geospatial Cyberinfrastructure for Environmental Sensing (GeoCENS) is another example for building indexing structure based on space filling approach [Liang and Huang 2013]. GeoCENS converts geographical attributes (longitude and latitude) of sensor data into a one-dimensional string (called quad-key) using Peano space-filling curve. GeoCENS provides an exact query search mechanism for data published from sensors based on its quad-keys. However, the approach does not support efficient processing of queries if a large number of sensors publish data at high rates. Spatial data can be indexed using tree-based methods such as R-tree, B-tree and kd-tree. Interested readers can refer to the survey of [Diallo et al. 2012; Gani et al. 2016] for detailed discussion about data indexing using different tree structures.

Other spatial solutions allow indexing data based on not only the spatial features but also the temporal ones. For example, [Zhong et al. 2013] propose a distributed spatio-temporal indexing called "VegaIndexer" for large volumes of sensor data to allow exact query search. The basic idea depends on having a global index for locating a block of information and a local index to select a specific feature object within the selected block. [Barnaghi et al. 2013b] annotate sensor data semantically to allow creating spatio-temporal indexing. The spatial characteristic of the data is described using geo-hash, while the temporal feature and the type of the data are encoded using MD5 digest algorithm. Singular Value Decomposition (SVD) is applied to geo-hash vectors to reduce dimensionality before applying $k-$means clustering algorithm to distribute data among repositories and allow data querying. Each data item is represented as a long string; the geo-hash tag of a location and the MD5 digest of time and type values. Answering approximate queries is based on a predictive method that is used to select the repository that might have the requested hash string. Then, a string similarity method is used to find the best match for a requested query in the selected repository. However, the indexing approach lacks an update mechanism, and the scalability is an issue because SVD is computationally intensive and is not tailored for large-scale data that can not fit in memory (disk-resident) [Ding et al. 2008].

6.1.2 Thematic Data.
Thematic indexing (based on specific terms, fields or patterns) has been proposed for indexing data given its terms or particular attributes. For example, a framework has been proposed in [Lunardi et al. 2015] to allow a set of various devices to register to a middleware. The framework provides an indexing method based on selected XML fields from the meta-data description of different connected devices in the network. The approach offers an updating method that uses subscribe/notify communication pattern between connected devices and the network. The framework also receives notification updates with new values of XML data fields. The data indexing can be updated within the proposed framework and the indexing mechanism allows querying specific XML fields. However, the indexing is not scalable because it is centralised and the time for answering queries increases with the increase in the number of connected devices. Similarly, [Aberer et al. 2007] propose Global Sensor Network (GSN) to allow registering sensors with their meta-data in an XML structure. Although the query is exact text-based search, this approach tends to have ambiguous descriptions of sensors because users add description terms for sensors manually.

[Ledlie et al. 2005] construct a distributed indexing structure for sensor data based on the descriptions of meta-data attributes in XML format. The approach constructs a single indexing structure for each attribute to respond to exact user queries. However, using simple XML attributes to index sensor data and having separate indexing structure for each attribute are not sufficient. This is because sensor data often does not have the same attributes and the indexing structure can not answer multi-attribute queries. [Harth and Decker 2005] present another indexing approach using RDF to describe sensor data. Each data element is stored persistently using RDF. RDF represents data as triples (subject, property, object) (RDF triple has been explained earlier in Section 3.3.2). The authors extend the triples into quads in which context information is added to the triples to represent the source of the data (e.g. URI), where RDF quads are indexed using B+ tree [Comer 1979]. The nodes in the tree are represented by (key, value) pairs. The key is a concatenated string of subject, property, object and context and the value refers to where the data resides on disk. The approach supports exact fast search (using concatenated keys). However, it assumes that the stored data is not updated. Also, the constructed index structure can only answer simple exact queries and does not support answering complex queries [Aggarwal et al. 2013].

[Wang et al. 2014] propose a Continuous Range Index (CR-index) tree-based method for indexing observation data based on their type attribute (e.g. temperature and oxygen saturation) and value ranges. The method constructs a compact indexing scheme in which a collection of observation and measurement data items are grouped into boundary blocks based on their value ranges $[min, max]$ (i.e. interval blocks). The constructed indexes enable answering value-range queries, however, this method can index only data with a single dimension [Wang et al. 2016]

6.1.3 Time-series Data.
As mentioned earlier in Section 3.3, IoT data can be represented as time-series data. [Agrawal et al. 1993] propose a Discrete Fourier Transform (DFT)-based feature indexing approach (called F-index) by first selecting f_c which is a cut-off frequency. It is a set of the first f features of every time-series sequence (i.e. f number of DFT coefficients) to represent data sequences from high-dimensional time domain to lower-dimensional frequency space. The resultant data sequences representation are subsequently indexed by R^*-tree [Beckmann et al. 1990] to answer similarity queries over time sequences. However, DFT lacks spatial aspect that is often essential attribute for IoT data as mentioned earlier. Determining f_c value is also not an easy task. Although, the paper argues that

the proposed approach guarantees the same Euclidean distance between data sequences in both frequency and time domains, f_c should have a value only between (1 and 3) to ensure representing data sequences with no false dismissal to answer queries.

Time-series data can be represented in symbolic forms using symbolic-based methods such as SAX (Appendix B provides more details about SAX). [Shieh and Keogh 2008] propose an updated version of SAX, called iSAX (indexable Symbolic Aggregation approXimation) to construct an indexing scheme that supports both fast exact search and fast approximate search of one terabyte time-series data. iSAX has an advantage over SAX; it allows representing SAX symbols with different cardinalities (i.e. number of bits) within the same word [Shieh and Keogh 2008]. iSAX utilises Windows NTFS file system for disk access and builds a hierarchical tree-based index structure. In the tree structure, each node can be root, internal, or a leaf. The node represents an iSAX word with at most two child nodes (iSAX words) except the leaf nodes. Furthermore, iSAX represents time-series data by depicting the data using a higher granularity at first level of the tree, where the new arrival data is then represented by a lower granularity in lower levels of the tree based on their representations. However, similar to SAX, determining iSAX parameters relies heavily on the data. Moreover, once the root's and the child nodes' representations are constructed, it is not possible to update them [Camerra et al. 2014], which is a constraint; especially if we consider using iSAX in indexing on-line time-series data which requires the indexing mechanism to be continuously updated with no prior knowledge of the data size. Intuitively, iSAX does not allow the child nodes to be represented by a higher cardinality once they are created. Terminal (leaf) nodes initially have a single child that maps to a file whose location on disk is hashed. Once the number of time-series data elements that can be represented by the same iSAX word increases more than a threshold th in that file, another child node (file) is created to avoid overflow at the same level of the tree structure.

An adaptive iSAX- and tree-based indexing mechanism to answer approximate queries has been proposed by [Zoumpatianos et al. 2014]. It is claimed that this approach outperforms other approaches such as iSAX 2.0 [Camerra et al. 2010]; the cost of index construction is shifted from initialisation time to query time. This shift is performed by creating and refining indexes while responding to queries. There are also some other existing works to extend iSAX (Appendix D provides more details about different extensions of SAX-based approaches). However, iSAX extensions are still inefficient in providing on-line indexing. Similar to SAX, iSAX strongly assumes that raw data has a Gaussian distribution and uses a z-normalisation in which the magnitude of the data is lost. However, IoT data does not necessarily follow Gaussian distribution; data distribution might change over time due to the nature of the observed phenomenon and/or concept drifts. iSAX also does not allow the structured tree's root to be modified by a higher cardinality once it is created. However, IoT data is dynamic and it may require changes in the constructed tree structure (Section 3.1 provides more details about IoT special characteristics).

Discovery techniques for time-series data can be exact without false dismissal (the ability to retrieve all qualified sequences that match requested query) or approximate (false dismissal might occur) [Keogh et al. 2001b]. Selecting the similarity measure and the time-series complexity are the main complications of similarity search [Keogh 1997]. Euclidean distance is a similarity measure between two time-series sequences. Euclidean distance assumes that two sequences (S_1, S_2) are similar if each point in S_1 is mapped to its counterpart point in S_2. However, if both sequences are identical, and one of them is slightly shifted, Euclidean distance fails to detect their identical nature [Salvador and Chan 2007]. Therefore, Euclidean distance has been shown to be a volatile distance measure for this type of applications because it is easily affected by any small shifts along time-axis [Keogh and Pazzani 2000].

Dynamic Time Warping (DTW) [Berndt and Clifford 1994] is an alternative similarity measure. It is a dynamic programming technique that has been used to warp time-series sequence by shifting points in the sequence to align two sequences in time domain. The shifting approach allows better distance calculation to find their similarities. There is no difference between using DTW or Euclidean distance as a similarity measure in large datasets. Large datasets do not often require aligning time-series sequences to compute their similarities because similarities can often be in their nearest neighbours [Shieh and Keogh 2008]. Moreover, DTW outperforms Euclidean distance in small datasets [Ding et al. 2008]. However, the principal issue that confines using DTW as a similarity measure is its complexity [Keogh and Kasetty 2003]. There are some other similarity measures, however, DTW has been shown to be the most appropriate similarity measure between time-series data [Ding et al. 2008]. [Rakthanmanon et al. 2013] show that Euclidean distance is a one-to-one comparison (i.e. point-to-point), in which two sequences are identical if and only if both have the same length. The complexity of Euclidean distance is $O(n)$, where n is the length of the sequence. On the other hand, DTW compares one-to-many (i.e. one point from one sequence could map to many points in the other one). However, it does not necessarily mean that DTW compares sequences with different length [Ratanamahatana and Keogh 2005; Salvador and Chan 2007]. Its complexity is $O(nm)$, in which n and m are the lengths of two sequences. Wrapping window is a value to constrain points from both sequences to be mapped to a specific width window; other constraints are also discussed in [Berndt and Clifford 1994; Ding et al. 2008]. Euclidean distance is a special case of DTW in which it has a zero warping window size [Ratanamahatana and Keogh 2005].

DTW can be further speeded up to mitigate its complexity by using smaller warping windows [Ratanamahatana and Keogh 2005]. For example, [Salvador and Chan 2007] propose an enhanced DTW called "FastDTW" that has linear complexity and space $O(n)$ comparing to standard DTW which has $O(nm)$ complexity. The basic idea is to speed-up DTW calculations by reducing the number of data points that should be visited to find warp path and carrying out DTW on an abstracted representation of the data recursively. [Keogh and Pazzani 2000] also propose Piecewise Dynamic Time Warping (PDTW) by applying DTW on reduced time-series data that is approximated by Piecewise Aggregate Approximation (PAA) to speed-up and overcome the complexity of DTW. Similarity measure comparison has been discussed in details in [Ding et al. 2008]. Moreover, [Rakthanmanon et al. 2013] use DTW to provide exact search for arbitrary length queries. They present a search suite (UCR) that comprises optimising the normalisation step of time-series data and provides a better approach for obtaining lower bounds to mine enormous volumes of time-series sequences.

Other time-series algorithms are proposed to allow analysing data that is available continuously based on classification methods and tree structures to construct hierarchical indexing. For example, [Grass and Zilberstein 1996] propose an algorithm to provide answers at anytime, given interruptions. The accuracy of the output improves with processing time. However, the approach does not provide an incremental learning to keep the learning process updated by any new arrival data stream. [Seidl et al. 2009] present Bayes tree indexing based on aggregating hierarchical Gaussian mixture in the form of a tree to represent the entire dataset. The approach supports both incremental learning and anytime classification of new arrival data streams. The latter allows fast access and search for data with a good level of accuracy. The approach also allows approximate query based on density estimation and relies on a supervised classifier; wherein data is labelled. However, Gaussian mixture models are not suitable for multi-feature indexing (a separate index structure is constructed for each feature). The number of mixture components also need to be known in advance before constructing the models which make the models less flexible.

Overall, IoT indexing and discovery require distributed, efficient and scalable methods and solutions that can support discovery and indexing of dynamic and real-time multi-feature data that is continuously published by different resources.

6.2 Resource

We refer to resource indexing as an approach to organise IoT resources to facilitate querying or finding a specific resource or a resource that can answer user queries. IoT resources often offer service descriptions while publishing their data. However, resource descriptions can be enriched by applying machine learning techniques to represent the resources in latent semantic space. To this end, indexing can be constructed based on their latent factors. Indexing by Latent Dirichlet Allocation (LDI) is introduced in [Wang et al. 2015b]. Latent Dirichlet Allocation (LDA) [Blei et al. 2003] is a probabilistic topic modelling to analyse and classify a collection of documents into latent topics where topics have certain probabilities to generate particular words. LDI relies on LDA to represent a document in a latent topic space. LDI then indexes documents related to the requested queries based on the similarity between the queries and the documents in the topic space. This classification of documents does not require labelled training data.

Semantic indexing based on the description of IoT resources has been proposed by [Cassar et al. 2010]. The authors describe IoT resources and services semantically using Ontology Web Language for Services (i.e. OWL-S) and then use LDA to index IoT resources based on their semantic service descriptions to answer approximate user queries based on a semantic search. The latent approaches are mainly tailored for indexing textual data. [Wang et al. 2015a] propose geo-spatial indexing to locate a gateway that might have a connected resource that has an approximate answer for a given query. The spatial attributes of IoT resources are described using geo-hashing. The approach can answer semantic queries using SPARQL query language. The index structure supports updating operations (i.e. connecting or disconnecting a resource) within each gateway without updating the entire indexing structure. However, the approach supports only static locations for connected IoT resources. Some other approaches such as Linked Sensor Middleware (LSM) [Le-Phuoc et al. 2011] offer limited functionalities [Perera et al. 2013] for search based on logical queries. For example, LSM indexes and links data sources based on their semantic description, however, querying the resources relies on selecting an approximate area on a map or based on a sensor type. LSM also assumes that data from resources is static and is not susceptible to frequent changes.

Spatial indexing of IoT resources which is based on their geographical locations has been proposed by [Hoseinitabatabaei et al. 2014]. The proposed approach is a distributed hierarchical indexing for IoT resources. Each resource is connected to a gateway, and consequently, each gateway constructs a GMM model to represent its connected resources. The model is updated by a short-term process by calculating the variations based on Variation Compensation Vectors (VCV) or a long-term one if the current model is no longer sufficient and need to be substituted to represent the current data. The model finds a resource that has an answer for a given query. However, the indexing approach has two main shortcomings; it assumes that constructed GMM models are initially trained on labelled data, and if there were many types of services (e.g. air pollution, humidity, temperature), each data type requires an individual constructed GMM model at each gateway. The hierarchical architecture of this approach might have Single Point Of Failure (SPOF), in which if a top node fails the entire service halts. Although the approach has a query processing mechanism, it only allows exact search (e.g. temperature in a specific location (e.g. longitude: -101.70593, latitude: 57.45899)); it does not support approximate search or searching for patterns. In addition, all gateways are assumed to be homogeneous (have the same type of services) which is not the case in most of IoT environments. Another distributed spatial mechanism for indexing IoT resources has been proposed in [Fathy et al. 2016]. The indexing structure is built by clustering different resources based on their spatial features. A tree-like structure is then constructed per cluster in which each branch represents a type of resource (e.g. temperature, humidity sensors). The indexing mechanism supports an adaptive process for updating indexing with minimal cost. However, the approach is limited to a predefined

Page 27 of 67 Computing Surveys

set of resource types and it only supports exact search queries of multi-dimension attributes (i.e. exact locations and types).

[Meliou et al. 2009] propose multi-dimensional tree approach to index sensor nodes using a probabilistic model to answer approximate queries. The indexing method creates a tree-structure where each node has a constructed GMM model. The models of child nodes are aggregated at their parent node. However, the way the indexes are constructed does not support spatial queries [Mohamed et al. 2011] and the query statements are received to all sensors, and this has a high overhead for large-scale distributed IoT networks.

Another approach is searching for IoT resources according to their unique identifiers. There are different Object Identifier (OID) schemes that can be used as a unique identifier for IoT resources. For example, Electronic Product Code (EPC) is used as a unique identifier for physical objects. EPCs can be encoded on RFID tags, while Ubiquitous Code (ucode) is another approach for identifying physical objects based on Ubiquitous ID (UID) architecture [Abowd and Mynatt 2000]. EPCglobal [Traub et al. 2005][21] is a consortium that has provided EPC as a universal identifier for physical objects. EPCglobal has three main components; EPC, EPC Discovery Service (EPCDS) and EPC Information Service (EPCIS). EPCIS enables access to EPC-related data and events. EPCDS links between user queries and EPCIS; EPCDS receives user queries to obtain information about an exact EPC object or events associated with specific objects. EPCDS provides URLs (links) to EPCIS servers using Object Naming Service (ONS) to access objects related to the query. The static identifier for querying objects and centralised query processing are the main limitations of EPCGlobal.

Distributed Hash Table (DHT)-based overlay networks for discovery services of IoT resources have been discussed in [Paganelli and Parlanti 2012]. An overlay network is a network of connected nodes; wherein each node has a particular view of other nodes in the network. DHT is a distributed data structure which allows nodes in an overlay network to be defined by a (key, value) pair, where DHT provides a lookup for value by object's unique identifier (key). DHT offers two main operations: put (key, value) and value = get (key) to store and retrieve the data object associated with a given key. DHT allows publishing data from different nodes in the network and can employ efficient routing requests to find the owner (node) of a given key. DHT supports only exact match for a given key and can not handle complex queries [Paganelli and Parlanti 2012].

A distributed discovery server architecture of EPCglobal based on DHT is proposed in [Manzanares-Lopez et al. 2011]. However, this approach supports exact query for only EPC code as an object identifier and lacks support for complex queries [Paganelli and Parlanti 2012]. [Paganelli and Parlanti 2012] propose a distributed data discovery service approach and an indexing mechanism on top of a DHT framework. Unlike EPCglobal that supports single attribute (object identifier) query, this method supports exact query for specific objects, multi-attribute and range queries (approximate queries) as well as a flexible identification scheme. The approach maps multi-dimensional data space into a single one, and Prefix Hash Tree (PHT) [Ramabhadran et al. 2004] on top of DHT is subsequently used to construct the indexing structure. PHT is a binary tree in which a prefix identifies each node of the tree, and the data is stored at the leaf nodes. Answering queries requires sequential traversal of the tree structure down to leaves whose prefixes overlap with the queries. When the number of connected devices in IoT environment grows, data generated by the devices might frequently be changed, and the rates of data update and query access become an issue [Sekine and Sezaki 2008]. Interested readers can refer to detailed discussions in [Evdokimov et al. 2010; Paganelli and Parlanti 2012] on distributed EPCGlobal and DHT-based discovery for IoT resources.

Most of the existing discovery services in the IoT are centralised [Polytarchos et al. 2011] or have limited functionalities. IoT data and environments require distributed, dynamic and scalable

[21] http://www.gs1.org/epcglobal

https://mc.manuscriptcentral.com/csur

methods to discover different IoT resources and enable on-line access to different types of queries of the resources and their published data.

6.3 Higher-level Abstractions

Indexing resources and/or their published data allows searching and detecting higher-level abstractions such as events, activities or patterns. As stated earlier in Section 3.3, IoT data can be represented in a symbolic form. [Bhattacharya et al. 2007] derive high-level semantic events (e.g. finding weather pattern HCHC, H (Hot) and C (Cold)) from low-level sensor data. The work is based on transforming raw sensor readings into symbolic states (e.g. C for temperatures $\leq 25°C$ and H for temperatures $> 25°C$). Model-based Index STructure (MIST) indexing is then proposed to support searching for events. MIST is an in-network, hierarchical and distributed tree-based indexing structure that builds local Hidden Markov Model (HMM) for each node to capture the hidden states and derive the semantic meaning of sensor data. Child models are then aggregated into their parent based on spatial correlations between the models. The model allows answering different approximate semantic queries: range query (e.g. return all sensors that observe specific pattern with probability > threshold), top-q query (e.g. get the sensor that has most likelihood to observe a given pattern) and others. Although a bottom-up aggregation of child models into their parent is used to preserve the correctness of indexing, domain knowledge is required to infer correspondence between states. Therefore, this approach is not suitable to provide on-line indexing for IoT where data and its distribution might be changing over time, and new states need to be inferred from the data.

A SAX-based approach for sensor data (SensorSAX) to find exact patterns has been proposed by [Ganz et al. 2013]. The work is an enhancement of SAX approach to change adaptively the window size based on the spread of data values (i.e. using standard deviation criterion). The proposed approach converts raw sensor data into symbolic representation and infers higher-level abstractions (e.g. dark room or warm temperature).

Geographic Hash Table (GHT) is proposed by [Ratnasamy et al. 2002]. GHT uses a hashing function where the key is an event name (e.g. high temperature), and its value is the location of a node that has that key. GHT only supports exact queries. Overall, GHT has two main drawbacks: it only supports binary events (i.e. event occurs or does not occur), and it groups nodes with the same event type (i.e. key) together even if they are far away. Distributed Index for Features in Sensor networks (DIFS) is an extension of GHT [Greenstein et al. 2003]. DIFS builds the indexes in a tree-based structure where each node in the tree stores a range of values in a certain geographical area to detect higher-level events (e.g. hot regions). The tree structure allows answering a range of queries. DIFS indexes are constructed based on one attribute (one type of service). DIFS assumes that the distribution of values in each node is uniform. However, IoT data distribution could change over time, therefore constructing and updating DIFS are costly. This is because DIFS is based on GHT in which every node in the tree should be aware of the boundary of the entire geographical area [Wu and Li 2009], and each node in DIFS tree structure can have one or more parents even if the parents might be located far away [Demirbas and Lu 2007].

[Li et al. 2003] propose Distributed Index for Multi-dimensional data (DIM), which is a tree-based indexing approach. DIM is based on dividing the network field into different geographical zones where each zone corresponds to a node in the tree, and each node represents a range of values such that the tree root represents the entire range. DIM allows answering multi-dimensional range queries. However, the routing algorithm to answer user queries is computationally expensive which hinders its scalability within large-scale networks [Bharambe et al. 2004].

It is expected that many users can request several queries simultaneously while new data is continuously created; however, updating the indexes and answering queries need to be completed in parallel. Although the adaptive indexing approach gradually builds parts of the index, it lacks scalability. While data scales and more data and queries are received, the updates for the indexing happen more often. As a result, more queries have to be processed against more data. In other word, while data scales, the overall cost of query processing increases.

6.4 Summary

Different index and discovery approaches are discussed above based on the types of queries. Some indexing approaches such as thematic (e.g. XML fields), time-based (e.g. time-series) and others are based on data indexing without identifying the data source. Other indexing methods rely on the semantic description or geographical (spatial) locations of resources for constructing the indexes for different resources. Indexing for higher-level abstractions is used to discover and infer information from IoT resources and their data. Some solutions find a symbolic pattern such as SAX-based approaches, and others discover events using the spatial feature of resources/data. Given the three most important types of queries, taxonomy of indexing approaches is summarised in Fig. 4 and Table 6 shows a classification of discovery methods.

Table 6. Discovery classification approaches

Search/ Discovery	**Data**	**Resource**	**Higher-level Abstractions**
Exact	[Aberer et al. 2007; Harth and Decker 2005; Ledlie et al. 2005; Liang and Huang 2013; Lunardi et al. 2015; Rakthanmanon et al. 2013; Zhong et al. 2013]	[Fathy et al. 2016; Hoseinitabatabaei et al. 2014; Manzanares-Lopez et al. 2011; Paganelli and Parlanti 2012; Traub et al. 2005]	[Ganz et al. 2013; Ratnasamy et al. 2002; Traub et al. 2005]
Approximate	[Agrawal et al. 1993; Barnaghi et al. 2013b; Berndt and Clifford 1994; Salvador and Chan 2007; Seidl et al. 2009; Shieh and Keogh 2008; Wang et al. 2014; Zhou et al. 2014]	[Cassar et al. 2010; Le-Phuoc et al. 2011; Meliou et al. 2009; Paganelli and Parlanti 2012; Wang et al. 2015a]	[Bhattacharya et al. 2007; Greenstein et al. 2003; Li et al. 2003]

Overall, the existing approaches for indexing and discovery of IoT data/resources are either centralised or do not provide efficient update mechanisms to allow on-line indexing and discovery. IoT environment is dynamic, in which many resources join the network and others become unavailable for various reasons (e.g. mobility, source state, battery life). The time it takes to create and update indexes, as well as processing and responding to users' queries should be minimal. IoT requires on-line, distributed, scalable and efficient indexing, discovery and query of resources and their data. The nature of IoT data (spatio-temporal and high dimensionality and often high diversity) imposes challenges in data classification and analysis that are offered by conventional machine learning algorithms (Appendix E provides an experiment on high diverse real-world IoT data). IoT methods and solutions should allow machines and human users to interact with discovery services and find the requested data automatically. It is expected that large dynamic data is needed to be processed and analysed even before users finish typing their queries [Ostermaier et al. 2010]. Discovery should be on-line without prior knowledge of the full sequence of data with consideration of integration of newly available data into the training model. Indexing and ranking mechanisms are key enablers for efficient data discovery.

7 RANKING

The prime goal of ranking is to prioritise and order resources and services by selecting the most suitable ones among them based on user queries and requirements. On the Web, Google uses

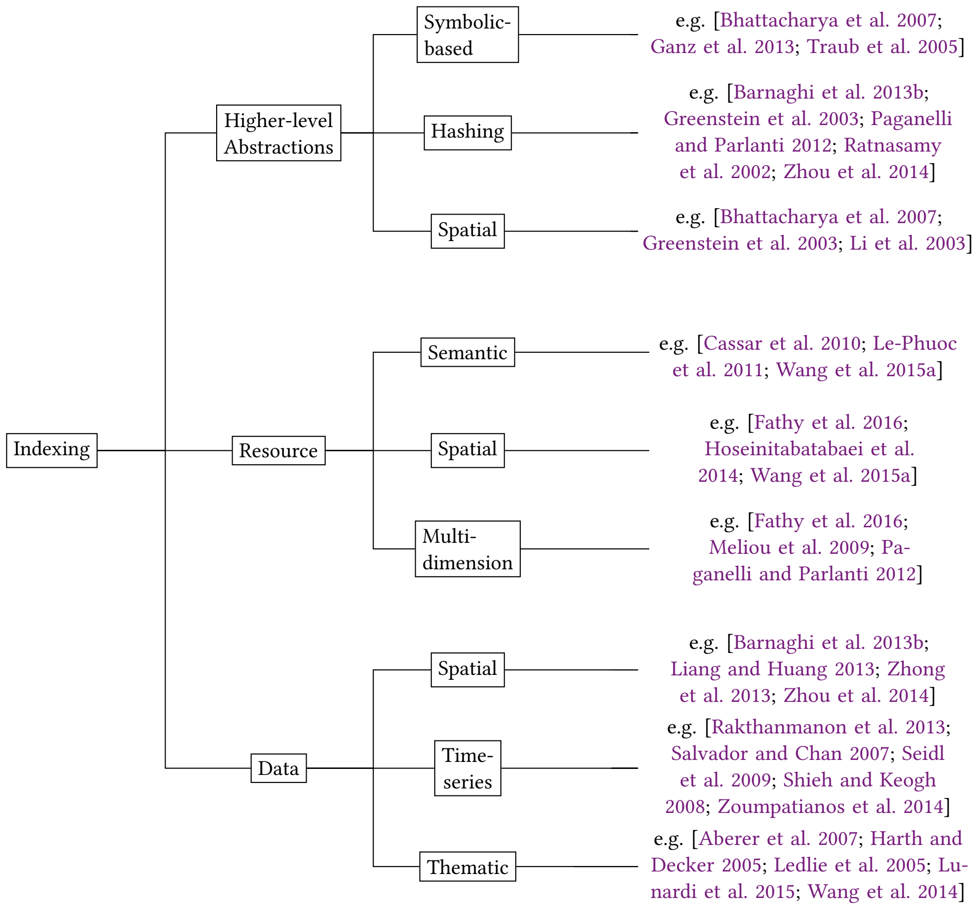
Indexing
Higher-level Abstractions
Resource
Data
Symbolic-based
Hashing
Spatial
Semantic
Spatial
Multi-dimension
Spatial
Time-series
Thematic
e.g. [Bhattacharya et al. 2007; Ganz et al. 2013; Traub et al. 2005]
e.g. [Barnaghi et al. 2013b; Greenstein et al. 2003; Paganelli and Parlanti 2012; Ratnasamy et al. 2002; Zhou et al. 2014]
e.g. [Bhattacharya et al. 2007; Greenstein et al. 2003; Li et al. 2003]
e.g. [Cassar et al. 2010; Le-Phuoc et al. 2011; Wang et al. 2015a]
e.g. [Fathy et al. 2016; Hoseinitabatabaei et al. 2014; Wang et al. 2015a]
e.g. [Fathy et al. 2016; Meliou et al. 2009; Paganelli and Parlanti 2012]
e.g. [Barnaghi et al. 2013b; Liang and Huang 2013; Zhong et al. 2013; Zhou et al. 2014]
e.g. [Rakthanmanon et al. 2013; Salvador and Chan 2007; Seidl et al. 2009; Shieh and Keogh 2008; Zoumpatianos et al. 2014]
e.g. [Aberer et al. 2007; Harth and Decker 2005; Ledlie et al. 2005; Lunardi et al. 2015; Wang et al. 2014]

Fig. 4. Taxonomy of indexing approaches

PageRank [Brin and Page 2012] to rank the results of search queries (i.e. Web pages). The work is based on links between different Web pages. Hypertext Induced Topic Selection (HITS) is another ranking approach [Kleinberg 1999]. HITS ranks search results based on the linking between Web pages that can provide more information for requested queries. Both of the approaches are based on connections and links between different sources and are mainly tailored for textual data. In IoT, resources can be related to each other based on their type of spatial features. However, each resource can have many features, as well as, different observation and measurement data. In this case, IoT ranking should be a multi-objective decision-making process in which various criteria should be considered depending on the application domain.

[Thirumuruganathan et al. 2013] present an algorithm called "RANK-est" to provide top k queries from Web databases (e.g. Amazon) through a search interface. Overall searching on Web databases provides the top k information related to a given query. The authors assume that every database record is represented as a tuple (a tuple contains multi-attributes and their values). The authors also focus on using a static ranking function. However, there are two types of ranking function; static (i.e. query-independent) and dynamic (i.e. query-dependent). The static ranking function assigns a static score for each item in the database. The static ranking function could be observable which can be queried or unobservable (i.e. proprietary) which cannot be queried. In the former, users request a query with a set of desired attributes. For instance, searching for an application on Apple store with a set of specified criteria (e.g. category and price). Another example is to find books on Amazon, within a price range (the price for every book is displayed in the query result). In proprietary ranking, users could also search for books, sorted by best-sellers, but they do not know the actual revenue for each seller. On the other hand, query-dependent is a ranking function in which various query models are used for different queries. In particular, query results are ranked based on the closeness of the match between the attributes of a query and every tuple in the database. [Thirumuruganathan et al. 2013] conducted their experiments on Amazon DVD and book items using a static ranking function which can not be extended to other application scenarios.

[Guinard et al. 2010] propose a ranking service for sorting IoT services based on their types (e.g. temperature, health), their multi-dimensional attributes (e.g. service's location) and/or the Quality of Service (QoS) (e.g. latency). The ranking service also performs a set of chained ranking strategies for multi-criteria evaluation of various parameters with different weights that are determined in a given query (e.g. 50% for location, 40% for service type and 10% for network latency). However, the ranking approach depends on simple terms for describing the services which are typically suitable only for IoT simple scenarios and applications [Guinard 2011; He and Da Xu 2014].

[Yuen and Wang 2014] rank sensor services based on two categories of QoS in WSN: network-based (latency, bandwidth, delay, reliability and throughput) and sensor-based (accuracy, trust and cost). The ranking relies on an objective function that is a weighted sum of different QoS parameters assigned by the user. However, weighting these parameters is not dependent on joint comparisons, but instead it relies on pairwise comparisons. [Yau and Yin 2011] present QoS-based service ranking approach that takes one step further by selecting and ranking available services based on users' QoS requirements instead of QoS of the services themselves. Users have to determine the importance of a set of QoS requirements such as reliability, delay and others before searching for a service. The ranking approach is not only based on the importance of the QoS requirements, but also on users' confidence for assigning values to each QoS's attributes.

[Niu et al. 2014] also propose a ranking approach for WSN services using QoS with an incorporation of user feedback/rating (i.e. Quality of Experience (QoE)) for different services within a time interval (time threshold). However, the proposed approach assumes that the information of user assessment for various services and QoS are available at the services/sensors level which is not a valid assumption in dynamic WSN where there is no control on when users can access and rate the services. Also, the approach does not consider heterogeneity between services where different services have different characteristics [Wang et al. 2015c]. The work presented in [Xu et al. 2007] combines both QoS information published by service providers and the reputation scores (users' feedback) regarding the services' performance for selection and ranking the services that match user requirements. The approach also relies on storing reputation scores for all services which might lead to a storage problem [Paradesi et al. 2009].

[Elahi et al. 2009] propose a sensor ranking mechanism that is based on predictive models to estimate the probabilities of content-based sensors that match requested queries. The approach can only rank results of simple queries about some higher-level states (e.g. occupancy; free or occupied) of an object (e.g. room). Another method based on semantic prediction models is proposed in [Mietz et al. 2013] for ranking IoT resources. The models are constructed using RDF to describe states that sensors can measure (e.g. warm temperature) and states can be queried using SPARQL query language. The query result is a ranked list of resources based on their probabilities to infer a requested state. The approach assumes that a set of states is defined in advance which hinders updating the models with new sensors that might have new states.

CASSARAM (context-aware sensor search, selection and ranking model) for IoT domain is proposed in [Perera et al. 2013] to rank sensors based on contextual information. CASSARAM allows users to query (search) sensor data based on their priorities such as reliability, availability, battery life and query results are subsequently ranked according to these parameters. CASSARAM relies on Comparative Priority-based Heuristic Filtering (CPHF) algorithm to provide an efficient and fast ranking mechanism. Data model in CASSARAM is extended for describing sensors using SSN ontology to allow semantic queries in [Perera et al. 2014]. However, answering queries is only efficient with a few number of sensors.

The latent factors approach using LDA is incorporated from topic modelling domain to rank sensor services. For example, [Cassar et al. 2014] utilise Ontology Web Language for Services (OWL-S) to describe sensor services semantically and apply LDA (a generative probabilistic unsupervised machine-learning technique) to map service descriptions into latent factors (topics). A ranking process is performed on latent factors based on their similarities with user queries. Recently, a ranking approach based on collecting contextual information from sensors services and the semantic description of the services to minimise the cost for their accesses to answer user queries is proposed in [Wang et al. 2015c]. The authors discuss four main requirements for ranking mechanism; ranking should be on-line, distributed, simple, efficient and have independent complexity from WSN and minimal energy consumption. However, the efficiency of the proposed approach is based on the assumption that the WSN has a limited number of sensors.

Quality of Information (QoI) is identified by some quality attributes such as accuracy and completeness. QoI has been used to characterise and rank data and information collected from sensor networks. For example, [Klein and Lehner 2009] propose a data quality model for sensor data streams to control QoI during data query process. The model considers data quality attributes such as accuracy, confidence, completeness, and timeliness. To guarantee efficient QoI over data streams, the model uses jumping data quality windows approach that considers splitting streaming data into consecutive non-overlapping fixed-length time windows. The QoI attributes are measured for each window. The main shortcoming of this model is that the accuracy measurement for each window is mainly based on sensors' precision provided by their manufacturer. In this case, a sensor might be considered producing accurate values even if it has some faults or failures (e.g. calibration error or freezing) [Asmare and McCann 2014].

[Bisdikian et al. 2013] combines QoI and Value of Information (VoI). VoI represents the utility of information gathered from sensor data in the application-specific context such as trust level of sensor devices and usefulness of the data collected from sensor devices. The work uses analytic hierarchy multi-criteria process based on the application-specific requirements to drive different weights for attributes for ranking collected data and information. On the other hand, [Lin et al. 2014] present a Max-Significance-Min-Redundancy metric approach to identify the QoI for each source. The metric is based on two main quality attributes: source significance (i.e. to what extent a source contributes to a classification task) and source redundancy (i.e. information overlap between sources) for incremental selection and ranking of different information sources. The ranking has an objective to maximise the significance of information gathered from selected sources. However, selecting sources that can maximise the significance of information might cause redundancy in which information gathered from new sources might be relevant to some previously selected sources. To this end, the approach decreases the redundancy of information sources while selecting the information sources that maximise the significance. However, the main shortcoming of this approach is that ranking proceeds by selecting one source at a time which is not suitable for a large number of connected sources in large-scale IoT networks. Furthermore, it is worth mentioning that utilising QoI for ranking IoT data is highly dependent on the properties of the collected data

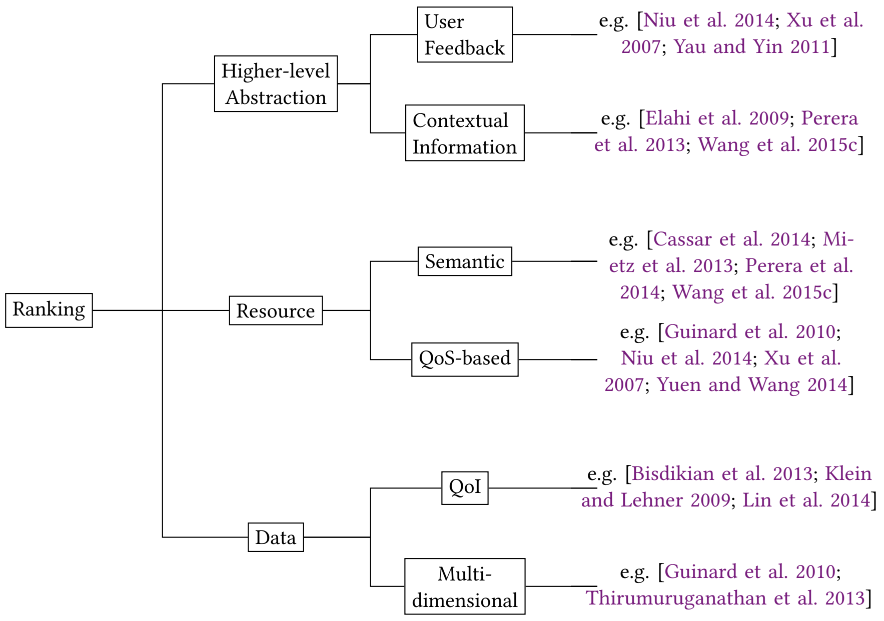
Ranking
Higher-level Abstraction
User Feedback
e.g. [Niu et al. 2014; Xu et al. 2007; Yau and Yin 2011]
Contextual Information
e.g. [Elahi et al. 2009; Perera et al. 2013; Wang et al. 2015c]
Resource
Semantic
e.g. [Cassar et al. 2014; Mietz et al. 2013; Perera et al. 2014; Wang et al. 2015c]
QoS-based
e.g. [Guinard et al. 2010; Niu et al. 2014; Xu et al. 2007; Yuen and Wang 2014]
Data
QoI
e.g. [Bisdikian et al. 2013; Klein and Lehner 2009; Lin et al. 2014]
Multi-dimensional
e.g. [Guinard et al. 2010; Thirumuruganathan et al. 2013]

Fig. 5. Taxonomy of ranking approaches

and how these properties can comply with the application requirements [Bisdikian et al. 2009]. Taxonomy of ranking approaches is summarised in Fig. 5

Overall ranking the available resources and their published data and services is often dependent on user needs. The ranking also depends on characteristics of a network of devices such as battery levels, network delay, latency, and bandwidth. Dependability, availability, reliability and the quality of information are also among the factors that can determine the ranking score of IoT resources and their data.

8 ANALYSIS AND DISCUSSION

This section analyses indexing, discovery and ranking approaches for the IoT and discusses the areas for further research. Indexing and ranking of heterogeneous IoT resources are key enablers for data discovery and search to provide fast access to resources, on-line retrieval and analysis of their published data. Different indexing methods and solutions are discussed to enable resource, data and higher-level abstractions queries. Resource indexing methods could be based on the semantic description of the services provided by the resources (e.g. [Cassar et al. 2010]), their deployment locations (spatial feature) (e.g. [Fathy et al. 2016; Hoseinitabatabaei et al. 2014]) or other features. Indexing to enable query of the data published by IoT resources could be thematic (e.g. [Lunardi et al. 2015]) where indexing can be constructed based on terms such as XML field given that the data is stored in XML format. The indexing could also be built according to the spatial and temporal features of the data (e.g. [Zhong et al. 2013]). Furthermore, indexing can be constructed for querying higher-level abstractions to find patterns such as "dark room" using methods such as SAX (e.g. [Ganz et al. 2013]). The indexing could also rely on a hashing function to find events such as "high temperature" (e.g. [Paganelli and Parlanti 2012]).

Extensions of SAX for indexing time-series data are discussed in various work. For example, [Shieh and Keogh 2008] index 100M time series of length 256 (random walk) and [Camerra et al. 2010] index 1000M (one billion) time-series of length 256 (Random Walk). However, SAX is originally designed for representing only uni-variant data (data with the same type), and this hinders representing service type, time, and location. To represent different kinds of services, SAX could be extended to describe service types with a specific prefix character within the SAX symbols. For example, TXXX represents temperature and HXXX represents humidity, where XXX represents the standard SAX symbols. However, this requires that a set of prefix letters to be known in advance and excluded from the alphabet set that is used in SAX representation. Using a specific set of letters hinders extending the type of services if a new type of resource/data is added. Time might be concatenated with SAX symbols. However, there is a high complexity to build such structure. It is noteworthy that two IoT data streams with various value ranges could have the same representation due to normalisation step in the construction of SAX structure. One solution to this problem is to associate coefficient for each SAX representation to allow comparison between different streams. However, a modified pattern creation method such as SAX is still not suitable for IoT applications that receive data on-line from different resources due to its normalisation for input time series and lack of support for temporal and spatial features of IoT data and resources.

Relying on a textual description of the services offered by different resources is also impractical for two reasons. The first reason is that if the descriptive text is added manually by individuals (e.g. device owners) as in [Aberer et al. 2007], text tends to be inaccurate or ambiguous which affects the accuracy. A possible solution to this problem is to provide a resource description template to individuals while registering their resources. However, this brings into view the second problem; the queries are text-based and not suitable for large volumes of numerical data. Moreover, in most of the cases, the indexing is centralised as in [Lunardi et al. 2015]. Distributed approaches for indexing resources have been discussed such as [Hoseinitabatabaei et al. 2014; Paganelli and Parlanti 2012]. However, the approaches have limited functionalities (e.g. support simple queries) or have unrealistic assumptions (e.g. resources should have the same type of service (e.g. temperature) to connect to a gateway. Also, most of these approaches construct an individual model per service type and/or do not support dynamic indexing in which indexing should be refreshed and updated as many IoT resources become available, and others become unavailable due to source state or battery life.

It is worth-noting that indexing approaches for resources (e.g. devices, sensors, services) are more static in terms of the type of services they offer (e.g. temperature, air pollution). However, underlying IoT resources might have dynamic spatial attributes (e.g. location, quality) while publishing their data. On the other hand, data indexing approaches are more dynamic in which data has potentially frequent updates. Moreover, indexing based on higher-level abstractions (e.g. events, patterns) can be static or dynamic. In such cases, indexing approaches are static when they assume a predefined set of patterns to construct indexes based on grouping nodes with the same type of events. The indexing methods can be adaptive when the indexes tend to be constructed based on inferring new contextual information from data published by various resources.

Overall, dynamicity, scalability, and distribution are the common problems of conventional indexing mechanisms for IoT data and resources. IoT resources are deployed in distributed environments over a wide geographical area. To this end, IoT data is generated in highly distributed and dynamic environments. The data/resource indexing should be constructed in a way that allows dynamic and on-line updates with minimal computation overhead despite the number of connected IoT resources.

While indexing allows a fast and efficient access to IoT resources and their published data, discovery methods use indexing to support query and search processes by providing higher-level and actionable information extracted from IoT data and resources that can be provided to higher-level applications and services. Various search and discovery approaches are presented and discussed.

The approaches provide either an exact search to match user queries as in [Rakthanmanon et al. 2013] or an approximate search as in [Cassar et al. 2010; Le-Phuoc et al. 2011] to get the best match for user queries. Overall, data discovery has a trade-off between time, modality, and quality as shown in Fig. 6. Intuitively, the more quality of data is required; the more time is needed to find the suitable resources. End-users (i.e. human or machine users) should be able to execute on-line queries on IoT data. The query is often constituted of a set of attributes such as type (e.g. temperature, humidity), location (e.g. London, Guildford), time (e.g. freshness ≤ 5 seconds) and other attributes. Different types of queries have been discussed in [Barnaghi et al. 2013b]. Response to user queries should be presented in a human-readable and/or machine interpretable format.

There is a variety of criteria to rank IoT resources such as latency, trust, availability, and reliability. However, selecting the criteria is also based on the application domain. IoT domain lacks having applications that allow users to choose the criteria based on their needs. Most of the current ranking approaches are focused on the network level (e.g. latency, energy efficiency), and there is limited research on the ranking and user requirements in the IoT domain. It also worth noting that collecting and monitoring data for ranking can create an additional overhead in IoT networks.

The major differences between indexing, discovery and ranking solutions on the Web and their counterparts in IoT applications is evident. The Web solutions are often tailored to deal with a collection of documents that are relatively static data and the Web methods exploit the links between the documents. Web search query consists of a word or a set of words. Answering the query is to locate documents containing the query word(s), and query results are ranked based on their relevance to the requested query and user preferences. Unlike the Web, IoT data is generated in dynamic and high-velocity distributed environments. IoT data has intrinsic characteristics as discussed earlier. In particular, IoT data is more dynamic, multi-modal and spatio-temporal. Indexing IoT data and resources can not be separated from data search and discovery. IoT requires designing efficient solutions for distributed indexing and discovery to enable selection, access and use of the suitable resources at the right time to answer user queries. The approaches should be tailored to the needs of highly dynamic and distributed IoT environments. We still also lack automated annotation for publishing IoT sensor data that can help to query them. The number of devices (Things) that are connected to the Web is increasing rapidly, and also the rate of querying the IoT data. Optimised query processing mechanisms are required to allow continuous queries. It is noteworthy that tackling these challenging tasks can help building distributed frameworks and search engines for the IoT.

As stated earlier, IoT data can be represented as streaming data. Some work has been proposed on the topic of enabling continuous query processing over data streams. In particular, Data Stream Management Systems (DSMSs) adapt the traditional relational model from databases to model relational data streams and enable creating streaming applications [Golab and Özsu 2003]. Several existing solutions exploit the relations between different attributes. Relational streaming systems and engines examples include TelegraphCQ [Chandrasekaran et al. 2003], Aurora [Abadi et al. 2003; Carney et al. 2002], Borealis [Abadi et al. 2005], STanford stREam datA Manager (STREAM) [Arasu et al. 2004], NiagaraCQ [Chen et al. 2000] and Nile [Hammad et al. 2004]. Complex Event Processing (CEP) is a special case of stream processing systems to detect and infer events/patterns from different data sources. Prominent examples of CEP systems include Cayuga [Demers et al. 2007]. TelegraphCQ, Borealis, STREAM and Nile do not handle spatio-temporal feature of data streams [Mokbel et al. 2005]. It is worth mentioning that Aurora, Borealis, and STREAM systems are no longer active [Gorawski et al. 2014]. We have provided more details about each one of these systems in the Appendix F. Interested readers can refer to [Babcock et al. 2002; Cugola and Margara 2012; Gorawski et al. 2014] for more discussion about data stream processing systems and

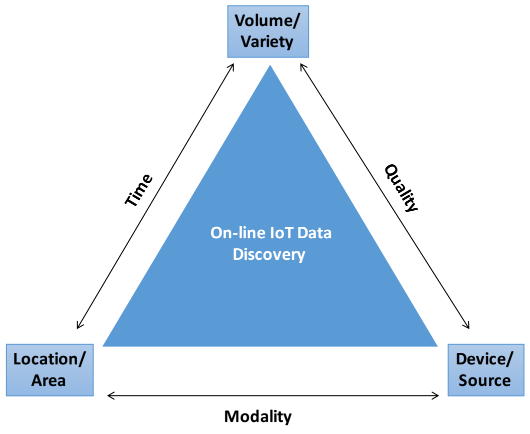

Fig. 6. Trade-off triangle of on-line IoT data discovery

streaming query languages and to [Golab and Özsu 2003; Stonebraker et al. 2005] for a summary of the major requirements for data streaming management and processing systems.

Overall, the streaming management systems and engines are designed to deal with data streams with high-volume to enable creating stream processing and monitoring applications. The systems/engines are mainly to provide continuous queries over streams in which users request queries and receive updated results of the queries continuously without having to request the same queries. Some stream processing systems/engines are commercialised such as Aurora/Borealis and TelegraphCQ or have not been comprehensively evaluated to meet the requirements of IoT data (heterogeneity, spatio-temporality characteristics) [Golab and Özsu 2003]. In particular, the existing solutions and systems are mainly tailored to retrieve values or attributes of data streams without having a distributed, scalable and efficient indexing, ranking and/or analysis for the data streams to gain insights and extract information.

Some other commercialised streaming systems have been developed for processing data streams, however, there is no much attention in the literature to them such as TIBCO StreamBase[22], IBM InfoSphere Streams [Biem et al. 2010], Microsoft StreamInsight [Ali 2010] and others. Some other open source streaming systems have also been developed such as Apache Storm[23], Apache Samza[24], Esper[25] and others. We have provided more details about each one of these systems in the Appendix F. Interested reader can refer to the work of [Cugola and Margara 2012; Zámečníková and Kreslíková 2015] for a detailed discussion about different streaming systems. Overall, most of these approaches have an indexing structure that is based on one-dimensional feature and do not offer answers to the queries that might need aggregation or join for large portions of published data [Dehne et al. 2013]. Although some of these systems support data gathered from real-world with a high rate, the solutions are not suitable for working with the fluctuation of data rates in real-time [Kumbhare et al. 2013]. Most of those systems have a deprivation of spatial features of the resources in which the spatial features are considered as symbolic data [Dao et al. 2014]. However, the spatial feature of IoT data is essential characteristic. Therefore, such systems are not suitable for answering range and

[22]http://www.tibco.com/products/event-processing/
[23]http://storm.apache.org/
[24]http://samza.incubator.apache.org
[25]http://www.espertech.com/

approximate spatial queries. The systems also do not support automated query rewriting [Schultz-Møller et al. 2009]. Interested readers can refer to [Fülöp et al. 2010] with regards to other issues and open research questions in streaming systems.

However, there are some initial works that provide a search engine for the IoT. As examples, we summarise the characteristics of existing IoT search engines in Table 7 and their advantages, disadvantages, and example of their queries in Table 8. Other examples of some of the existing industrial IoT applications and platforms are summarised in Table 9. However, we provide more details about each one of these platforms in Appendix G. Other search engines are discussed in [Römer et al. 2010] and other IoT platforms are presented in [Mineraud et al. 2015]. Some other IoT Cloud platforms are discussed in [Al-Fuqaha et al. 2015]. It is worth mentioning that Cloud Computing provides an effective solution for IoT service management such that it makes it easy to implement applications that use the data produced by various resources (e.g. devices and services) [Lee et al. 2010]. The Cloud-based solutions can also offer fast configuration models for IoT sensors and devices [Botta et al. 2016]. Interested reader can refer to the work of [Botta et al. 2016] for a detailed discussion about the integration of Cloud computing into IoT applications from the communication, storage, and computation perspectives. The work also highlights the main technical and business-related issues that remain unsolved for allowing full integration of Cloud computing into IoT.

In Tables 7 and 9, a query type is the data type of a query (e.g. text, numeric). Query terms constitute a query expression (e.g. keywords, location). The query result is either based on exact or approximate search in which the former is to find a response that exactly matches a given query while the latter is to find the best similarity to a requested query. Indexing, data discovery, and ranking have been defined before in Sections 6 and 7. Crawling focuses on how resources can be detected and their features can be integrated into the indexes. Manual discovery in the tables means that the engine/platform does not support (auto-) discovery for resources and IoT resources have to connect to the network manually (e.g. a device should be registered by its owner). On-line means that the search engine/platform can be updated by any changes that might happen in data/resource.

Overall the main problems with the current IoT search engines are distribution, on-line queries and/or scalability as shown in Tables 7 and 8 and none of the existing solutions provides efficient discovery for their sensory data. The data is searchable/queryable; however, there is no deep analysis and mining for the sensory devices in complex queries for IoT applications. Furthermore, industrial IoT applications and platforms are either lack scalability/distribution or do not provide (on-line) discovery and/or details about their architecture that allows integration of more Web resources and services.

Table 7. Characteristics of different IoT search engines

Search Engine	**Craling (resource discovery)**	**Indexing**	**Data Discovery**	**Ranking**	**On-line**	**Query**		
	Manual/Auto-discovery	**Centralised/ Distributed**	**Centralised/ Distributed**			**Type**	**Terms**	**Result**
Dyser [Elahi et al. 2009; Ostermaier et al. 2010]	Manual discovery (no auto-discovery functionality is supported)	Centralised indexing functionality is supported (i.e. central local database) [Elahi et al. 2009]	No support for data discovery	Supported	Resource change notification is supported	Text (Devices state)	Key-words	Exact

Snoogle [Wang et al. 2010]	Manual discovery (no auto-discovery functionality is supported)	Distributed indexing functionality is supported (aggregated based on geographical area)	No support for data discovery	Supported	Resource change notification is supported	Text	Keywords	Approximate (Find sensors that might have queried keywords)
MAX [Yap et al. 2005]	Manual discovery (no auto-discovery functionality is supported)	No indexing functionality is supported	No support for data discovery	Supported	No resource change notification is supported	Text	Keywords	Approximate (find objects in a relative location)
DiscoIoT [Mayer and Guinard 2011]	Manual discovery (no auto-discovery functionality is supported)	No indexing functionality is supported	Centralised data discovery is supported (semantic discovery)	Not supported	Resource change notification is supported	Text	URL	Exact
SenseWeb/ SenseMap [Kansal et al. 2007]	Manual discovery (no auto-discovery functionality is supported)	Centralised indexing functionality is supported (SenseDB)	No support for data discovery	Not supported	Resource change notification is supported	Text	Keywords and location	Exact
Thingful†	Manual discovery (no auto-discovery functionality is supported)	No available information	No support for data discovery	Supported (i.e. ThingRank Algorithm)	No resource change notification is supported	Text	Service type (e.g. temperature) and/or location (e.g. London)	Exact and approximate (near me option)

† http://www.thingful.net/

Table 8. Comparison of different IoT search engines

Search Engine	**Advantage**	**Disadvantage**	**Query Example**
Dyser [Elahi et al. 2009; Ostermaier et al. 2010]	• Search for devices states • Ranking is based on predictive model	• Crawling HTML pages for sensors (not applicable for all sensory devices) • Users have to know the state names for all physical objects to query them (e.g. occupy: empty) • Not scalable (centralised index)	Find bicycle rental stations which have currently available bikes
Snoogle [Wang et al. 2010]	• Ranking objects based on query • Distributed query processing • Aggregating objects indexes	• Index is based on IPs • Not scalable (change in meta-data requires update KeyIPs) • Using Bloom filter requires recreating the filter afresh when new sensors are connected • False positive result (IPs that can not provide an answer)	Search for a textual description of a specific object (e.g. book) with/without a specific location

https://mc.manuscriptcentral.com/csur

Page 39 of 67 Computing Surveys

MAX [Yap et al. 2005]	• It assumes that each device has a passive RFID (no power supply is required)	• It needs configuration based on physical space • It does not support indexing or ranking approaches • Not scalable (overhead of broadcasting for all base-stations and tags)	Search for a textual description of a specific object (e.g. Book, Harry Potter, Rowlling)
DiscoIoT [Mayer and Guinard 2011]	• Semantically description for Web resources at run-time • Describe Web resources in different formats (e.g. RDFa, Microformats and JSON) • It is based on RESTful interface	• Centralised discovery unit • Not scalable to integrate with existing services on the Web [Cirani et al. 2014]	Search for a specific sensor by its URL
SenseWeb/ SenseMap [Kansal et al. 2007]	• Dynamic access to sensors and their readings • Allows aggregation for sensor data • Combine requests for access same data • Caching most recent data	• Search only for static meta-data [Römer et al. 2010] • Central point of access to all applications (via coordinator) • Not scalable (central repository for all meta-data [Römer et al. 2010])	Get all sensor readings from region (Longitude = -123.00;and latitude = 47.00)[26]
Thingful	• It has geographical index • Ranking connected devices • Allow users to verify their ownership for their connected devices • Query could be by location and/or service type	• It does not provide real-time query • It lacks of data-refreshness • There is no available details about technical architecture	What: (e.g. temperature) and/or Where: (e.g. London)

Table 9. Characteristics of different IoT platforms/applications

Platform/ Application	**Crawling/ Resource Discovery**	**Indexing**	**Data Discovery**	**Ranking**	**On-line**	**Query**		
	Manual/Auto-discovery	**Centralised/ Distributed**	**Centralised/ Distributed**			**Type**	**Terms**	**Result**
Wolfram Data Drop⊕	Manual discovery (no auto-discovery functionality is supported)	No available information	Centralised data discovery is supported	Not supported	Resource change notification is supported (every 30 seconds)	Text	Databin	Exact
Ericsson IoT Framework∓	Manual discovery (no auto-discovery functionality is supported)	No available information	Distributed data discovery is supported (i.e. simple aggregation)	No available information	No resource change notification is supported	Text	Service type (e.g. temperature) and/or location (e.g. London)	Exact
Open.Sen.se‡	Manual discovery (no auto-discovery functionality is supported)	No indexing functionality is supported	Centralised data discovery is supported (simple semantic discovery)	Not supported	Resource change notification is supported	Text	URL	Exact

[26]http://research.microsoft.com/en-US/projects/senseweb/default.aspx/

https://mc.manuscriptcentral.com/csur

ThingSpeak†	Manual discovery (no auto-discovery functionality is supported)	No available information	No support for data discovery	Not supported	Resource change notification is supported	Text	Fields and location	Exact and approximate
Xively★	Manual discovery (no auto-discovery functionality is supported)	Distributed indexing functionality is supported	No support for data discovery	Not supported	Resource change notification is supported	Text	Different attributes (location, name, type of data and others	Exact

⊕ http://datadrop.wolframcloud.com/
∓ http://github.com/EricssonResearch/iot-framework-engine
‡ http://open.sen.se/
† http://www.thingspeak.com/
★ http://xively.com/

9 CONCLUSIONS

With the prevalence of mobile devices, low-cost sensors and network-enabled (Internet-connected) devices, the concept referred to as the Internet of Things (IoT) has gained momentum in recent years to push the boundaries between physical and digital worlds. The massive amounts of heterogeneous and multi-modal real-world data play a key role in developing situation-aware applications that are capable of inferring knowledge from real-world data. However, the conventional indexing and discovery methods and ranking solutions on the Web are not suitable for multi-model and dynamic IoT data which is usually numerical observation and measurement data. The IoT data can be an individual observation, or it can be streaming data represented as time-series. IoT data can also include quality and other descriptive meta-data. IoT data resources are distributed and often provide ad-hoc and dynamic data. However, the Web search engines often work efficiently with textual data, and they usually use archived data on the Web servers.

In the IoT, data can be published in the network on gateways, or it could be stored in repositories. The indexing and discovery methods, however, can also be based on resources that can provide the data. The dynamic and multi-model nature of IoT data/resources, links that need to be processed to find and access the data and different levels of abstractions to represent the data make IoT data different from most of the existing Web data for which the current Web search engines are optimised to index and search.

In this paper, we discuss the process chain of IoT data starting from publication and resource discovery up to making the resources and their data searchable and discoverable. We show that machines and human users can interact with IoT applications in different scenarios; searching for a resource, an abstraction or the observation and measurement data. We discuss a framework for on-line IoT data indexing and discovery.

While there are many attempts to leverage the ubiquity of the IoT, it requires further investigation for efficient, scalable and distributed indexing, ranking and discovery solutions. Developing efficient and scalable indexing and discovery solutions for the IoT will play a similar role that the Web search engines played in making the Web data more accessible and widely available for different users. A searchable IoT will change the way the applications are used and developed in various domains.

Some of the future research directions in this area include the deployment of dynamic and adaptable indexing and discovery mechanisms for distributed and pervasive networks and providing machine-to-machine search and discovery support in automated application scenarios and in dynamic and distributed IoT networks. The future of IoT discovery and search systems will also depend on creating large-scale ecosystems of IoT systems that can work and collaborate with each other to share and exchange data and services. While scalability analysis linked to quality and granularity of the data and access policies are key components of designing future IoT systems, security, provisioning, reliability, and trust will also be crucial components of any design in future IoT data/service access and discovery systems. To conclude, this paper summarises the current state-of-the-art, provides an analysis of the advantages and disadvantages of the existing solutions and proposes some ideas for further research in this domain.

APPENDICES

A COMMON CONSTRAINED OPERATING SYSTEMS FOR IOT

Various operating systems are developed for sensor nodes or "motes". The purpose of an operating system is to hide the complexity of low-level hardware specifications by providing a set of Application Program Interfaces (APIs) to deal with abstracted functionalities of hardware. For example, TinyOS [Levis and Culler 2002; Levis et al. 2005] is one of the most widely used Operating Systems (OS) on sensor nodes. Maté [Levis and Culler 2002] is a virtual machine (VM) of TinyOS. TinyOS is an event-driven concurrency operating system model for WSN. It includes a set of software components that have been built in the nesC environment [Gay et al. 2003] which is a dialect of C language for sensor networks. Powerful applications can be constructed on the top of TinyOS. However, only some devices use it such as Tinynode[27] and Mica2[28]. The main advantage of TinyOS is that it represents complex programs with minimum code size to adhere the challenges of resource constraints in WSN; power, communications and memory consumptions. However, developing applications might be difficult on top of TinyOS; a complex task should be split into a set of small processes (TinyOS only supports non-preemptive and non-blocking processes).

Contiki is another event-driven and a light-weight operating system for sensor nodes that allows dynamically load/unload of processes at run-time and supports multi-threading. The kernel is event-driven, but each process can be performed in a pre-emptive multi-threading manner at the application level [Phani et al. 2007]. The main difference between TinyOS and Contiki is that the programs are expressed in a state machine fashion in the former, whereas they are written in a sequential manner in the latter. Linking between applications is another difference, wherein components are linked statically in TinyOS [Gay et al. 2003] and this prevents sending out the updated codes between components once the linking between them is established. However, the components are linked dynamically in Contiki [Dunkels et al. 2004].

Sensor Operating System (SOS) is another dynamic event-driven operating system [Han et al. 2005]. It allows dynamic linking similar to Contiki. Unlike TinyOS in which updating modules requires to re-compile and load the full system image on the node [Han et al. 2005; Phani et al. 2007], SOS offers dynamically-loaded modules which make it easy to update only necessary modules. Therefore, SOS is more efficient than TinyOS in dynamic environments, wherein updating code might be frequent. MANTIS Operating System (MOS) [Abrach et al. 2003] is a multi-threaded module-based operating system. MOS is similar to TinyOS considering that updating codes require rewriting the full system image to the mote. MOS has a power-efficient scheduler to send the micro-controller thread to sleep in idle time until a task is available to be executed [Phani et al. 2007]. MOS also supports cross-platform through a set of APIs and allows remote management, dynamic programming and debugging for network sensors [Bhatti et al. 2005]. Table 10 summarises different constrained operating systems for the IoT. In Table 10, component-based OS provides a way to select a set of components (services) at compile-time (no support for the dynamic selection of the components). Module-based OS has a set of modules (similar to individual functions) that

[27]http://www.tinynode.com/?q=catalog/10
[28]http://tinyos.stanford.edu/tinyos-wiki/index.php/MICA2

collaborate with each other. Multi-threading refers to the ability of OS to execute multiple tasks concurrently. Event-driven OS can execute processes as a reaction to an event.

Table 10. Taxonomy of common constrained operating systems for IoT

	Programming Model			Architecture Design	
OS	Event-driven	Multi-threading	Langauge	Component-based	Module-based
TingOS	√	×	nesC	√	×
Contiki	√	√	C	×	√
SOS	√	×	C	×	√
MANTIS	×	√	C	×	√

B DIMENSIONALITY REDUCTION

The prime goal of dimensionality reduction is to downsize data with size N to data with size m, where $m < N$ or $m << N$. Singular Value Decomposition (SVD) [Golub and Reinsch 1970], Multi-Dimensional Scaling (MDS) [Kruskal and Wish 1978], Principal Component Analysis (PCA) [Moore 1981] and Latent Discriminants Allocation (LDA) [Blei et al. 2003] are standard linear techniques for dimensionality reduction. PCA and SVD typically transform a high-dimensional data into lower dimensional data with some loss of information by linear projection. PCA finds top k projections (i.e. principal components) that maximise variance and minimises least square error. The transformation is performed by selecting the largest eigenvalues of eigenvectors in covariance matrix with minimal least square error [Wall et al. 2003]. SVD is not adequate when data size increases (error is increased while data starts to scale) [Korn et al. 1997]. MDS finds embedded features that preserve distances between data points. If the distance is Euclidean, it is expected that both MDS and PCA give similar results [Tenenbaum et al. 2000]. [Barnaghi et al. 2013b] use SVD to reduce dimensions of annotated sensor data in an off-line mode. This has a significant effect on decreasing the required time for distributing data into various repositories (i.e. clusters). However, SVD is computationally intensive and is not tailored to large-scale datasets that can not fit in memory (disk-resident) [Ding et al. 2008]. [Guha et al. 2003] present an incremental streaming SVD approach by getting a linear correlation (similarity) between data streams. The experiments have been conducted on a small dataset, and it is expected to have a high complexity to re-compute SVD periodically for new arrival data streams. However, these linear techniques can not capture non-linear structure of data streams [Tenenbaum et al. 2000].

LDA is a generative probabilistic model that automatically discovers and extracts a set of latent topics that best describe a large collection of documents (i.e. corpus), wherein topics are distributed over words independent of the word order. LDA reduces the high-dimensional data vector into a lower-dimensional representation in a latent space. The model is based on two main assumptions; the order of documents in corpus does not affect the model. The model is also based on "bag of words", in which order of words within a document is ignored. LDA uses variational methods to estimate the posterior probability to tackle the inference problem (learning topic distribution in documents). On-line LDA is introduced by [Hoffman et al. 2010] to provide an on-line variational step using a stochastic optimisation approach to approximate posterior probability to analyse large set of streaming documents.

Other non-linear techniques for dimensionality reduction include: Local Linear Embedding (LLE) [Roweis and Saul 2000] and ISOMAP [Tenenbaum et al. 2000]. LLE maps high-dimensional data into lower-dimensional data by preserving local configurations in the nearest neighbours. LLE is sensitive to outliers, and neighbourhood selection does not guarantee the quality of the resultant less granular data. On the other hand, ISOMAP is a global geometric framework for non-linear dimensionality reduction. It is similar to LLE, but it preserves neighbourhood based on geodesic

https://mc.manuscriptcentral.com/csur

shortest distances and it is efficient (induces less computation overhead) and converges to global optimality.

To cope with high computations and complexities of dimensionality reduction techniques, [Mitliagkas et al. 2013] propose a PCA method with limited memory settings. [Law et al. 2004] propose incremental ISOMAP, but the model is not entirely on-line. While much work has focused on high computations and efficiency for dimensionality reduction, the techniques stated above mainly are not suitable for time-series on-line data (i.e. IoT data). Either the reduction process is not entirely performed on-line (i.e. applied in an off-line), and/or a high computational complexity is imposed when myriad of on-line data streams arrive which hinders their scalability.

Many techniques have been successfully used to reduce the dimensionality of time-series data. For example, Discrete Fourier Transform (DFT) [Cooley and Tukey 1965], Discrete Wavelet Transform (DWT) [Chan and Fu 1999], Piecewise Aggregate Approximation (PAA) [Keogh et al. 2001a; Keogh and Pazzani 2000; Yi and Faloutsos 2000], Adaptive Piecewise Constant Approximation (APCA) [Keogh et al. 2001b] and Symbolic Aggregate approXimation (SAX) [Lin et al. 2003]. DFT is the first proposed technique to reduce time-series data dimension. It transforms any complex signal from a time domain to a frequency domain by extracting f features (i.e. the first/best f DFT coefficient) [Faloutsos et al. 1994]. The major drawback of DFT is that the signal is discretised into a frequency domain. Thus, it has a deprivation of a temporal aspect. It lacks capturing a specific on-line event and at a particular time which both situations are often essential in IoT applications. Also, DFT cannot deal with streams that have a different length. DWT transforms the signal into a set of basic functions (i.e. wavelets) which are based on a recursive function [Wu et al. 2000]. DWT is more efficient than DFT in preserving time and frequency dimensions. It allows capturing event location at a certain time. However, it works efficiently if the input length is an integral power of 2 [Keogh et al. 2001a].

PAA is another approach that divides time-series data into k equal segments, wherein an average value of each segment is calculated and stored. It is worth mentioning that if the time-series data is split into k equal chunks of power 2 (i.e. 2^n where n = 1, 2, 3, 4, etc.), DWT and PAA will have the same representation of the signal [Cai and Ng 2004]. Moreover, if $k = N$, where k is as defined before and N is the number of points in the time-series sequence, the original and transformed signals are identical [Keogh and Pazzani 2000] and if $k = 1$ it means that the transformed signal is the average of the original one [Keogh et al. 2001a]. PAA is simple, easy to understand and implement and faster to compute. Moreover, it supports weighted Euclidean distance [Keogh et al. 2001a], whereas DFT and DWT cannot support it. APCA is another approach for reducing dimensionality. It outperforms standard PAA regarding query response time; it approximates the original signal better than PAA. Unlike PAA that splits the sequence into equal parts (segments with equal length), APCA allows arbitrary length for each segment. This allows creating more segments in regions and less in other regions based on the activity level. Furthermore, APCA surpasses other timer-series dimensionality reduction techniques because it induces less minimal error [Keogh et al. 2001b]. [Keogh et al. 2001b] show that APCA outperforms the techniques discussed above in CPU and Input/output (I/O) costs. However, similar to PAA, APCA does not preserve the shape of time-series data within segments (i.e. two segments can have same mean value. However, they have different shape) [Bettaiah and Ranganath 2014].

The techniques mentioned above are limited to represent real-valued time-series data. Different representations of time-series data are summarised in [Lin et al. 2003]. However, representing the data by strings might be beneficial; symbolic representation allows applying text retrieval and mining techniques [Lin et al. 2007b]. SAX is the first symbolic approach that allows dimensionality reduction (i.e. reducing data dimensionality by selecting smaller data representation). Time-series streaming data is transformed into a discrete symbolic representation (i.e. words) in a linear time, wherein a word, for instance, “abacab” is a vector of symbols $V = \{a, b, a, c, a, b\}$ [Lin et al. 2007b; Pham et al. 2010]. Symbolic representations are obtained by first normalising data ($Z \sim \mathcal{N}(0, 1)$) and dividing the distribution into equiprobable regions (i.e. by specifying breakpoints). PAA is then applied as an intermediate step to capitalise its advantages in reducing data dimensions and having lower measure distance bound on symbolic data that is less than the actual distance in raw data [Lin et al. 2003; Shieh and Keogh 2008]. PAA coefficients (i.e. segments) are subsequently symbolised (i.e. are discretised into symbols). In this case, the symbolic representation requires less space for representing the data; symbols need fewer bits than numbers comparing to DFT and DWT. Overall, SAX depends on intrinsic benefits of PAA and it does not require to access all of the time-series data in advance before creating its symbolic representation. The latter makes SAX a powerful symbolic representation approach comparing to other existing approaches for time-series [Lin et al. 2003]. However, SAX representations depend on the word length which controls PAA segments and cardinality (i.e. alphabet size). The latter is sometimes called “resolution” [Castro and Azevedo 2010]. Resolution controls the granularity for each segment [Lin et al. 2003]. It also depends on the sliding window size that captures the dynamicity of time-series within a specific sliding window.

It is worth noting that determining the values of the number of PAA segments and alphabet size parameters rely heavily on data. In addition, the major drawback of SAX is that time-series data in SAX is assumed to have a Gaussian distribution. [Pham et al. 2010] extend SAX to adaptive SAX (aSAX) by adaptively improving breakpoints over time using $k-$means clustering to address the issue of Gaussian distribution assumption in classic SAX. It is claimed that aSAX outperforms classical SAX, especially when SAX’s assumption about data distribution (i.e. uniformly distribution) does not hold. aSAX initially utilises Gaussian distribution to boost classical SAX symbols as an initialisation step. Then, an adaptive improvement of breakpoints vector is performed. This improvement is obtained by specifying k cardinality (i.e. alphabet size) and clustering the breakpoints (i.e. segments) based on this pre-defined parameter k using k-means clustering algorithm. The clustering step is considered as a pre-processing phase on PAA coefficients before they are discretised into symbols (SAX symbols). However, aSAX is only performed on a dataset with 100K time-series (experiments need to be conducted on a larger dataset to have robust conclusions and a fair comparison with SAX).

[Sun et al. 2012] argue that aSAX is not adaptive in the frequency and time domains due to having fixed time windows. Therefore, a variance-wise dynamic segmentation method is proposed. [Sun et al. 2012] change the sliding window size dynamically. The basic idea is that the window size varies based on a threshold value that is proportional to the standard deviation of the dataset. However, the experiments are conducted on an artificially modified ECG and the method is not entirely adaptive; it considers the changes only in the frequency domain. Similarly, [Ganz et al. 2013] propose SensorSAX which is an enhancement of SAX to adaptively modify the window size value based on the spread of values in a distribution (standard deviation) to reduce transmission overhead. The experiments are conducted on a real dataset (UK Channel Coastal Observatory resources); the abstracted data has 13% less size than the raw data. [Lkhagva et al. 2006] underline the importance of detecting extreme points from PAA segments, especially in financial domain. Therefore, Extended SAX (ESAX) is proposed wherein each PAA segment is fully represented not only by mean but also by minimum and maximum points without losing the simplicity of symbolic representation. The preliminary results in [Lkhagva et al. 2006] show high accuracy in capturing useful patterns compared to the classic SAX in financial applications. However, experiments were not conducted on the same representation size for both SAX and ESAX; ESAX representation is three times longer than SAX representation.

Another drawback of SAX that might affect some applications is that two different time-series data could be symbolised into the same representation due to its normalisation step. Therefore, [Esmael et al. 2012] extend the plain SAX approach by adding symbols (U, D, S) for up, down and straight, receptively to define the direction of time-series data. Overall representing data using SAX shows better result than using the raw data due to dimensionality reduction approach [Lin et al. 2003]. However, SAX strongly assumes that raw data has inherently Gaussian distribution and applies z-normalisation, in which magnitude is lost. However, [Keogh and Kasetty 2003] argue that normalisation is an essential step for measuring the similarity between time-series data, [Lin et al. 2003] also shows that time-series data tends to be highly Gaussian distributed and [Rakthanmanon et al. 2013] normalise time-series data to cluster them correctly. SAX patterns have been used in various existing works in the IoT domain. For example, [Barnaghi et al. 2012a] apply SAX to reduce sensor data dimensionality and find patterns in semantically annotated sensor data. Then, Parsimonious Covering Theory (PCT) [Reggia and Peng 1987] is used to derive abstractions from SAX patterns to analyse sensor data. [Zoumboulakis and Roussos 2007] detect patterns to describe complex events with reasonable accuracy by reducing dimensionality through converting time-series sensor data into SAX representations.

In IoT, the Gaussian assumption does not hold all the time, and it is mainly application dependent. IoT data is spatio-temporal, wherein data changes over time and sometimes is produced at a different pace and with diverse granularities. Various types of dimensionality reduction techniques are discussed. However, some are performed in batch/off-line; others have a high complexity and/or have strong assumptions about the data. Selecting dimensionality reduction technique is highly dependent on the application, processing capabilities, accuracy and representation requirements.

C GATEWAY MODEL AND RESOURCE DISCOVERY

The primary goal of resource discovery service is to crawl, find and allow IoT resources (e.g. sensor nodes, devices and services) that can publish their data and services to be discovered automatically or by manual registration. The resources should make the network spontaneously notified of the changes in their data and services. Resource discovery is also known as network discovery [Edwards 2006] that could either be active or passive [Guinard et al. 2010]. Active (i.e. automatic) is when devices initiate a communication channel to share their information and services on a network. When devices are registered in a network manually (e.g. by their URI), it is called passive.

IoT lacks a unified standardisation to allow communication and integration between heterogeneous sensory devices, WSN and other mobile communication networks [Zhu et al. 2010]. Integration of ubiquitous things (sensory devices and actuation services) into the Web can be direct where the objects must be IP-enabled or indirect in which a proxy (i.e. smart gateway) is used to provide uniform access to the Web [Zeng et al. 2011].

Gateway or middleware plays a vital role in providing a flexible bridge between network-enabled devices in a local mesh network and the global network (Internet). Gateway/middleware hides the complexity and heterogeneity of underlying networks by providing common interfaces and protocol compatibility that allow network-enabled devices to share and publish their data and services seamlessly on a global network.

Middleware is defined as a distributed service (abstract layer) that lies in-between operating system and applications that could be distributed to different network components (as shown in Fig. 7). It facilitates application development and deployment on different platforms. There is often a trade-off between the generality of middleware and domain-specific applications that can be built on top of them [Yu et al. 2004]. However, a separation between the generic functionality of middleware and unique features of each application is required.

In IoT, middleware is to provide a standardised and homogeneous interface to communicate with heterogeneous IoT resources with different specifications. Many middlewares are either complex or domain specific [Kamilaris et al. 2011]. Others are inflexible; they focus on centralised approach, in which collected sensor data is stored in a centralised database. Moreover, conventional middlewares are typically complex; they include communication, energy and memory overhead [Yu et al. 2004]. However, IoT requires light-weight solutions by adhering the constraints of sensor network technologies. WSN middleware design requirements and challenges are discussed in [Römer et al. 2002] and in more detail in [Yu et al. 2004]. In the latter, a multi-layer distributed cluster-based middleware architecture for collaborative data processing between nodes is proposed. Sensor nodes are grouped into clusters to form a virtual machine (cluster and resource management layer) to interact with the application layer. Each cluster has many sensors and is controlled by an elected cluster head. The head deals with the upper layer (resource management), in which Quality of Service (QoS) requirements should meet the application domain. The cost model of QoS is to minimise communication and energy overhead. However, more investigation and extensive experiments are required to test the proposed architecture. There are several types of middleware that have different approaches to develop applications for resource-constrained WSN including: query-based, service-based, agent-based, QoS, semantic-based and context-aware.

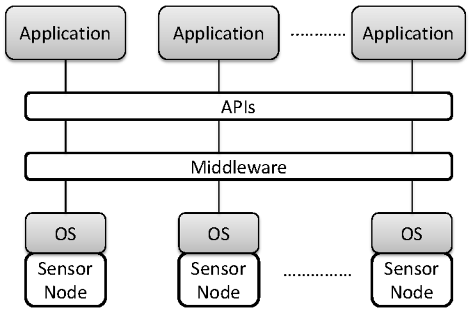

Fig. 7. The function of middleware

Sensor data can be gathered and stored in a database. Distributed in-network processing for sensor databases and declarative query approaches are introduced in Cougar[29] [Yao and Gehrke 2002]. The architecture of Cougar middleware is divided into leader nodes, non-leader nodes and a gateway. Every node has a query proxy to facilitate the interaction with routing and application layers. Each gateway has a query optimiser to generate distributed and efficient in-networking query processing schemes in the network. The processing scheme receives different declarative queries and then generates distributed query processing plan through a query optimiser. Data is subsequently collected from non-leader sensor nodes, and in-network aggregation is performed by a leader node based on the queries. Leader node delivers the aggregated data to the gateway. The main advantage of Cougar is its in-network computations to decrease energy and communication consumptions in WSNs.

TinyDB[30] [Madden et al. 2005] is another middleware that is based on extracting sensor data from TinyOS sensor devices using a declarative approach. It supports (power-efficient) in-network processing mechanisms to reduce power, latency and transmission. Unlike Cougar, in which data is aggregated incrementally to answer user queries, in TinyDB data is collected from motes based on

[29] http://www.cs.cornell.edu/bigreddata/cougar/index.php
[30] http://telegraph.cs.berkeley.edu/tinydb/

user query. The data is then filtered and aggregated. TinyDB relies on a network routing spanning tree structure to answer user queries by specifying a start-point (root) as the query's location. When a new node is connected, an updating mechanism is propagated through the network (up to the root of the tree) to update tree structure. TinyDB allows users to write SQL-like queries to query data and also specify data intervals (i.e. data re-freshness rate) as a parameter while querying the data (i.e. sample period). These make TinyDB different from other conventional middleware solution.

SenseWrap [Evensen and Meling 2009] is a service-oriented middleware. It represents different physical sensors as virtual ones, allows users to discover and interact with them effortlessly (i.e. zero-configuration). The middleware allows extracting the common functionalities of sensors from application development step and allows wrapper implementation for different sensor models to interact with them. It supports publish/subscribe communication style. It also allows a user to query data through declarative queries in a SQL-like form. The main advantage of SenseWrap is self-configuration, in which sensors are registered automatically to the network.

Agilla[31] [Fok et al. 2005b] is a mobile agent-based middleware for WSN that is based on Maté which is a virtual machine for TinyOS. Each Agilla agent is a virtual machine with a set of instructions and has a local tuple space that allows decoupling between sending and receiving agents. Each node has four agents; this allows each node to support different applications. Maté supports only one application per node. Upon agent execution, its instructions are executed allowing interaction between different agents and the environment. When an agent completes its task, it is terminated. However, its tuple space could be retrieved by other agents. Coordination between agents is performed through Linda-like tuple spaces [Fok et al. 2005b]. The main advantage of Agilla is flexibility (new agents can be injected into the network) and adaptability to any changes in environment or user needs; it has a context-aware discovery mechanism (e.g. a fire tracking application in [Fok et al. 2005a]). Another advantage is its high mobility; when agents are moving/cloning from a location to another, they carry their codes and states so that they can resume executing at the new location. However, in SensorWare [Boulis et al. 2003], only the node state is transferred while cloning or moving and the nodes have to start execution from the beginning in their new location. On the other hand, Agilla does not consider that there might be nodes with less power and communication capabilities during the running of complex applications/tasks. Due to its low level of abstraction, it is hard to maintain applications based on Agilla [Kwon et al. 2006].

BiSNET (Biologically-inspired architecture for Sensor NETworks) [Boonma and Suzuki 2007] is an agent-based middleware that is inspired by bee colony and is built on top of TinyOS. A BiSNET agent senses surrounding environment and emits different types of pheromone based on local conditions (sensor measurement types such as temperature) in the environment. Each sensor node has many BiSNET agents. Only data with high pheromone concentration (significant changes) is communicated to a base station. BiSNET agents could replicate data similar to Agilla. In the former, the replication/migration occurs based on pheromone concentration. BiSNET adaptively adjusts the sleep period parameter value (idle time) for sensor nodes based on sensor data reading in the environment without human intervention. It also hides the complexity of lower-level computing, networking, communication and agent behaviours by providing a set of services to access sensor measurement and observation data.

Middleware Linking Applications and Networks (MiLAN) [Murphy and Heinzelman 2002] is a solution that is designed specifically for Quality of Service (QoS) regarding reliability and lifetime of applications for personal health monitoring (heart monitoring), surveillance and security applications. MiLAN is based on a graph-based approach to allow users specify their application

[31] http://mobilab.wustl.edu/projects/agilla/

and performance requirements in advance, and it tries to meet these requirements. However, there is always a trade-off between performance and network cost on one side and system complexity and its lifetime on the other side. The main advantage of MiLAN is that it extends the network protocols to provide service discovery by providing an API as an abstraction to communicate the commands over a particular network protocol (e.g. Bluetooth, IEEE 802.11b) through network plug-ins. However, MiLAN is still not fully adaptive to dynamic environments due to the interoperability issues between various protocols and services. Other IoT platforms and middlewares are discussed in [Perera et al. 2013] such as Linked Sensor Middleware (LSM) [Le-Phuoc et al. 2011], Global Sensor Network (GSN) [Aberer et al. 2007], Microsoft SensorMap [Nath et al. 2007] and COSM[32]. However, these approaches have limited search functionalities [Perera et al. 2013].

[Ganz et al. 2011] present a semantic context-aware middleware solution that is typically based on collecting context information (e.g. battery status, signal strength) about each node in the network. Gateways subsequently decide which nodes to communicate with based on the gathered context information. Upon connection between gateways and nodes, a semantic pre-defined template based on W3C SSN ontology is provided for interactions with higher layers. It is thus different from Shaman approach [Schramm et al. 2004], in which nodes establish an arbitrary connection with gateways. In the work of [Ganz et al. 2011], the middleware is flexible; it cooperates with various platforms by a plug-and-play approach. However, sending updated context information has overhead when the number of sensors grows. A resource mobility scheme is proposed in [Ganz et al. 2012] to tackle the mobility and availability limitations (i.e. the network is not updated about the availability of IoT services/resources) of the middleware in their early mentioned work [Ganz et al. 2011]. The scheme allows a service to be available for users all the time by overcoming two main issues; handover delays and coverage loss. To address these problems, a caching approach is developed to solve the former and a tunnelling mechanism is to overcome the latter. While a node is moving from one gateway to another, the cache mechanism uses the last cached data to make the service up-and-running. Tunnelling is beneficial at the gateway level in which one gateway can forward to another if their mutual node is disconnected from one gateway and connected to another. This scheme is not suitable for some IoT applications wherein exact on-line data is important such as smart health and disaster monitoring. In addition, scalability is still an issue when the number of connected sensors to a gateway increases, and in cases that there is a high data congestion.

[De et al. 2011] present a semantic service modelling framework that extends Ontology Web Language for Services (OWL-S) ontology to the IoT services. In this model, a "Device-Entity-Resource-Service" relationship is proposed to describe IoT resource capabilities; a device is attached to an entity that constitutes "Thing" in the IoT. A service allows accessing a resource that is associated with this entity. It also allows embedding entities into the digital world (i.e. Internet) to provide interoperability between IoT data and existing data on the Web, as well as, between services. In addition, the model allows association of domain knowledge, measurement units and location specification as meta-data information.

Another approach is a zero-configuration for deployment sensor nodes automatically in WSN which is proposed in [Schor et al. 2009]. The approach is IP-based, and it depends on 6LoWPAN to allow integration with IPv6. A RESTful Web service API is deployed on each node to avoid updating gateways while new devices are being connected. Nodes publish their services or request information from other nodes by Multicast Domain Name Service (mDNS) protocol. API response is a JSON-based data. The main drawback of this mechanism is that it assumes all nodes are IP-enabled. However, gateways are required to enable connecting non-IP-enabled devices to WSN. [Schramm

[32]http://www.cosm.com

et al. 2004] present a Java-based service gateway to integrate heterogeneous sensor-actuator modules (SAMs) into a network. It uses network proxies to unburden devices to their memories, power and communication constraints, allows automatic sensor integration and translates user requests into SAM commands. However, this approach is off-line because users submit their requests to be translated into SAM commands via proxies and the requests are queued into a request queue.

[Guinard et al. 2010] propose a framework for discovery, query and selection for the IoT. The system is a dynamic Service Oriented Architecture (SOA) that supports RESTful and Web services (WS-*) such as Simple Object Access Protocol (SOAP) and Web services Description Language (WSDL) to allow seamless cooperation and integration between heterogeneous devices and services and the Web. However, such standardisation Web services are not light-weight and lack simplicity [Guinard et al. 2011]. In addition, the centralised integration architecture is inefficient regarding the scalability.

Another type of middleware is semantic wireless sensor middleware described in [da Rocha et al. 2009]; it is a rule- and knowledge-based middleware. It facilitates analysis and controlling of complex tasks in monitoring applications. [Bimschas et al. 2010] provide a middleware with smart gateways, in which applications use middleware's APIs and tasks are executed on the gateways. It also has an intelligent caching and discovery mechanisms; the current value is predicted based on learning from its previous values using a Bayesian model. [Elahi et al. 2009] sort sensors based on their probability estimation to find a sensor with high probability to match user queries. Table 11 summarises the middleware approaches (the taxonomy of middleware approaches is also shown in Fig. 8).

Table 11. Middleware classification

Middleware Classification	Example
Virtual-machine	Maté
Query-based	Cougar, TinyDB
Service-oriented	Shaman, SenseWrap
Agent-based	Agilla, BiSNET
Quality of Service (QoS)	MiLAN

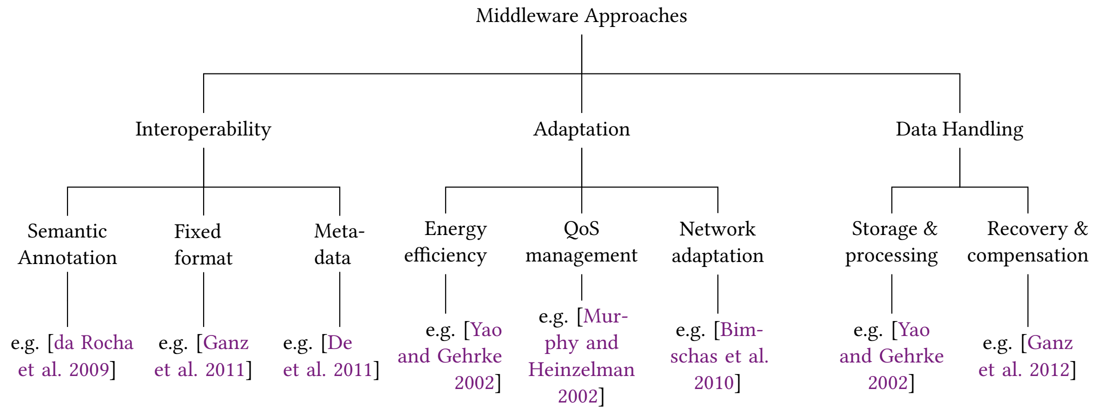

Fig. 8. Taxonomy of middleware approaches

D SYMBOLIC DATA INDEXING APPROACH

There is some existing work to use symbolic data representations for indexing. For example, [Pham et al. 2010] propose an indexable adaptive SAX (iaSAX) which is an indexed aSAX approach. Similar to iSAX, iaSAX allows representing time-series data with different granularities (multi-resolution property). However, it has no assumption about data distribution, and the breakpoints are adaptively determined similarly to aSAX. It is argued that iaSAX outperforms iSAX. However, the experiments are conducted on a time-series dataset with only 100K of length 256 which is not enough to have robust conclusions.

The main hindrance for creating indexes for massive datasets is the time and complexity to build index structure [Camerra et al. 2010]. For instance, iSAX requires more than 6 days to index 100 million (10^8) time-series data [Camerra et al. 2010]. However, [Camerra et al. 2014] argue that it requires two days to build the same data size and 20 days to build 500 million (5×10^8) time-series data. iSAX requires a long time to build indexes because two main reasons: *a*) Inefficient splitting policy *b*) No bulk loading scheme. To reduce the time complexity of building indexes, [Camerra et al. 2010] propose iSAX 2.0, which is similar to iSAX, but instead of building the whole index structure instantly, it provides building indexes for sub-trees gradually, wherein one sub-tree is built at a time until all sub-trees are constructed. A bulk loading algorithm is used to reduce the I/O access by buffering time-series data in the available memory and fetch them to the disk once the memory limit reaches. It also provides a node splitting policy to efficiently distribute time-series data equally between the two child nodes in the index structure [Camerra et al. 2014]. The main drawback of iSAX 2.0 is the scalability; while the size of time-series data grows, the number of node splits increases rapidly. In addition, the node splitting mechanism requires access/write for all raw time-series data and its iSAX representation from/to disk that is also a hindrance to scalability. Also, [Zoumpatianos et al. 2014] argue that it takes more than one day to index 1 billion (10^9) time-series data using iSAX 2.0 on the most advanced server machines (Intel Xeon machine with 64GB of RAM and 4x 2TB, SATA, 7.2K RPM Hard Drives in RAID0).

[Camerra et al. 2014] extends bulk loading algorithm of iSAX 2.0 by proposing iSAX 2.0 clustered and iSAX2+ index to reduce the time to construct index structure. The former is based on clustering raw time-series data using an approximation for its iSAX representation to reduce access time to disk, whenever a node splits. All time-series data is subsequently fetched to disk according to their leaf nodes. Some time-series data is not going to be split during the construction process. However, they are still accessed to create approximation. By avoiding accessing these data, the access time is reduced by writing the data directly to the disk. Therefore, iSAX2+ is to effectively avoid reading and fetching the data that is related to nodes that should not be split later. The avoidance is performed by "splitting once" approach. The approximation of already existing raw time-series data is written to the disk and a pointer to its raw data is created, while newly time-series data and its approximations are written directly to the disk in the right leaf node. Intuitively, since each node has at most two child nodes, it is guaranteed that only one split is performed per node. Once index structure is built, all time-series pointers are solved by removing pointers and writing data to the disk. Overall, the SAX extensions are not suitable for on-line indexing where building and updating indexes should be done in an on-line, efficient manner and with minimal overhead.

E HIGH DIVERSITY REAL-WORLD DATA

IoT data is dynamic, spatio-temporal and often high dimensional and diverse. These characteristics impose challenges in using conventional machine learning algorithms for analysing IoT data. For example, Fig. 9 shows an estimation of the distribution of weather data streams[33]. We use Kernel

[33]http://mesonet.agron.iastate.edu/ASOS/

Page 51 of 67 Computing Surveys

Density Estimation (KDE) to estimate the distribution of the weather data points (3, 000 data points) to get the number of peaks (can be obtained by zero-crossing). The number of peaks represents the dispersion of the data which can be used as the number of clusters (i.e. k) which is a required parameter setting for most of the clustering algorithms. However, the number of peaks is almost 1500 (half of the number of data points) due to the high diversity of the data. Therefore, conventional parametric clustering/machine learning algorithms are not often suitable for real-world data with high diversity.

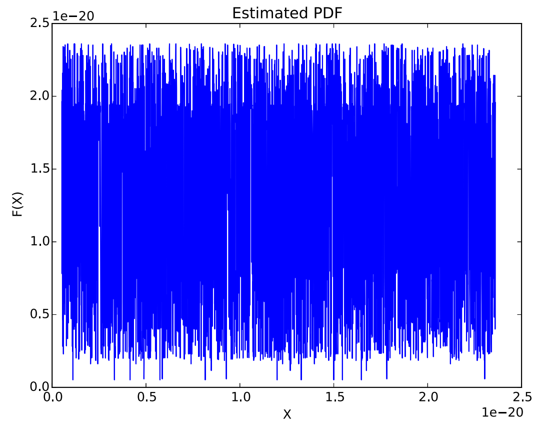

Fig. 9. Diversity of real-world data stream

F STREAM PROCESSING SYSTEMS AND ENGINES

There are several existing work on developing stream database systems and stream processing engines. For example, **NiagaraCQ**[34] is an XML-based engine that allows querying distributed XML documents (resources) using XML Query Language (XML-QL). NiagaraCQ supports two different types of execution for continuous queries: change-based and time-based queries. Change-based queries are executed if relevant data to the requested query becomes available, while time-based queries are executed at certain time intervals. NiagaraCQ addresses scalability by grouping continuous queries that have similar signatures (inputs) to share computations or execution plans. A new query is also combined into an existing group queries that match the query signature. NiagaraCQ is only suitable for data that is described as XML documents, and it assumes that requests for continuous queries have some similarities or common structure which is not always the case. Moreover, NiagaraCQ does not support approximate queries; continuous queries are executed on XML data of the static resources [Motwani et al. 2003].

TelegraphCQ[35] is an adaptive data-flow stream processing system. It offers dynamic partitioning of data streams for parallel processing. TelegraphCQ uses an adaptive query mechanism which is based on Eddies [Avnur and Hellerstein 2000]. An Eddy is a data-flow operator that passes tuples through a query plan. Each tuple in Eddy is associated with a vector of "Ready bits" and "Done bits" to mark the operators that can process the tuple and the operators that have already processed the tuple, respectively. It is worth noting that each operator in Eddy runs in an independent thread [Avnur and Hellerstein 2000] and might create a circular data-flow in which operator has one or two inputs from Eddy and returns an output tuple. Eddies are combined dynamically to

[34]http://www.cs.wisc.edu/niagara
[35]http://telegraph.cs.berkeley.edu

https://mc.manuscriptcentral.com/csur

provide an adaptive query processing mechanism. Eddies provide flexibility in re-ordering operators (e.g. join, group, aggregate) of query execution in a query plan at runtime. The reader is referred to [Avnur and Hellerstein 2000] for more details about Eddy. TelegraphCQ relies on Eddies to enable continuous query optimisation. However, Eddy deals with data and can not identify objects that generate the data. Moreover, TelegraphCQ does not support temporal relationships between queries/events which are crucial in the IoT. Interested readers can refer to a complete discussion about TelegraphCQ in [Krishnamurthy et al. 2003]. It is worth mentioning that TelegraphCQ is commercialised as Truviso[36] by CISCO.

Aurora[37] is a stream processing engine that is mainly developed to enable continuous streams of data for monitoring applications [Carney et al. 2002]. Similar to Eddies, Aurora architecture divides the processing of a query into multiple threads [Abadi et al. 2003]. Aurora provides users with a graphical interface for constructing a data flow of a query plan. The data flow is composed of a set of boxes to represent operators and arrows (arcs) to connect the boxes that reflect the flow of the data. Each stream s is represented as a tuple $< d, t >$, where d is data fields of the stream s and t is the time-stamp. In Aurora, a tuple can not be updated once it is placed in a stream which impedes run-time updating of a certain attribute of a stream [Abadi et al. 2005]. Aurora is also a centralised system which limits its scalability and reliability in large-scale IoT deployments [Abadi et al. 2005; Gorawski et al. 2014]. A distributed version of Aurora ($Aurora^*$) is proposed in [Cherniack et al. 2003], where each node in a distributed network sends query results either to users or other nodes for aggregation and further processing.

Borealis[38] is a successor stream processing of Aurora to provide an efficient distributed stream processing engine. Unlike Aurora, Borealis supports revision tuples to update tuples to recover and correct the query results. Borealis model supports insertion, deletion, and replacement (update) of tuples. Borealis receives the queries and processes them simultaneously by distributing the query processing into several nodes. Each node has an instance of Aurora query processing engine. Unlike TelegraphCQ, which does not support temporal feature of a tuple, data model in Borealis (tuples) has a time-stamp in the header of the tuple. However, Borealis does not support spatial queries. Queries in Borealis are text-based. In particular, queries are written in XML format. Aurora and Borealis have been integrated in the commercial StreamBase[39] tool.

STREAM[40] is a stream database management system to answer continuous queries over continuous streams. The queries are formulated using Continuous Query Language (CQL) [Arasu et al. 2006] which is an extension of the standard SQL. In particular, CQL implements an abstract semantics data type to express data streams and relations. A requested query is added to a query plan. The query plan consists of three main components (operators, queues, and synopses). Unlike Borealis, which does not convert stream tuple into a relation, STREAM transforms a stream into relation using stream-to-relation operator. STREAM has three different types of operators which are stream-to-relation, relation-to-relation (i.e. it produces a relation given a set of relations) and relation-to-stream (i.e. it produces a stream with a change that made by a relation). STREAM also supports negative tuple (i.e. delete tuple) [Mokbel et al. 2005]. Queues are used to buffer tuples (e.g. output of an operator is buffered and processed later as an input to another operator(s)). Synopses are used to summarise information about tuples to enable answering approximate queries. STREAM is a centralised system that assumes all data streams are in a single system. However, data is generated from distributed resources in real-world applications.

[36] http://www.cisco.com/c/en/us/about/corporate-strategy-office/acquisitions/truviso.html
[37] http://cs.brown.edu/research/aurora/
[38] http://cs.brown.edu/research/borealis/public/
[39] http://www.tibco.com/products/event-processing/
[40] http://www-db.stanford.edu/stream

Page 53 of 67 Computing Surveys

Nile[41] is a query processing engine for data streams. Nile supports continuous queries based on sliding windows. Nile extends SQL operators to support sliding window queries. In Nile, a source of a data stream (e.g. sensor device) is modelled using a stream data type "StreamType". [Ali et al. 2005] propose Phenomenon Detection and Tracking framework (Nile-PDT) as an extension of Nile to enable developing monitoring applications. Nile-PDT can query a group of sensors with similar behaviour (e.g. value, summary) over a period (time interval). Nile-PDT relies on Sensor Network join (SN-join) operator to get the similarity between a pair of data streams produced by different sensors. Nile-PDT also uses other operators to aggregate sensor values to find an event (e.g. number of sensors in a phenomenon). Nile-PDT provides a client application that allows users to submit their queries. Nile-PDT has an incremental processing feature to track the appearance and disappearance of resources. Pervasive Location-Aware Computing Environments (PLACE) server is a location-aware stream server which extends Nile to enable continuous spatio-temporal queries for moving objects [Mokbel et al. 2005]. Similar to NiagaraCQ, PLACE has a shared execution feature to merge related queries together. Like Borealis and STREAM, PLACE handles negative tuples (i.e. delete tuples). PLACE also supports queries with a temporal window to get objects at a certain time-stamp and spatial window to get objects in spatial locations regardless of their time-stamps. Moreover, PLACE has an incremental evaluation feature to update answered queries continuously. However, it has two types of updates which are positive and negative updates. A positive update indicates that an object is added to the query while the negative update indicates that an object is removed from the query. PLACE takes regular snapshots of objects and queries while executing the queries over moving objects [Lin et al. 2007a; Mokbel et al. 2005]. PLACE does not support efficient updating and processing for snapshots to provide efficient query processing [Lin et al. 2007a].

Cayuga[42] is a complex event processing system. Cayuga uses a publish/subscribe mechanism, in which users can subscribe to their events of interest. Users request their queries in the Cayuga Event Language (CEL) format. Unlike other stream database systems that support query processing for sliding windows (such as TelegraphCQ), Cayuga does not support sliding windows [Herbst et al. 2015]. Although, Cayuga supports detection of sequential tuples for event streams [Demers et al. 2007], it can not detect successive events within a specific time interval [Li et al. 2011]. In Cayuga, events are represented as a sequence of relational tuples. Cayuga relies on non-finite automata to allow arbitrary relations between input streams (which are represented as tuples) to match patterns (events) with requested queries. Each automata state represents relational tuple, and the transitions between states are based on predicates. Cayuga stores incoming tuples until a requested event is detected, and subscribed users to that event are then notified. However, Cayuga model does not support the temporal and spatial features of data streams.

TIBCO StreamBase is a commercial CEP platform that provides processing and analysis for data streams. The platform offers rapid building of different applications and the analysis of historical and real-time data. The platform also provides LiveView Datamart[43] to consume massive data streams published by various resources (e.g. IoT resources) in real-time [Wähner 2014]. End-users can subscribe to different events or aggregate data from different resources. Users are continuously notified whenever changes occur to their subscribed events. StreamBase also has a component that is called Spotfire for extracting events and patterns from the historical data. Some promising IoT applications can be developed using the StreamBase framework such as real-time monitoring for traffic, weather and driver behaviours.

[41] http://www.cs.purdue.edu/Nile/
[42] http://www.cs.cornell.edu/bigreddata/cayuga/
[43] http://www.tibco.com/products/event-processing/complex-event-processing/streambase-liveview/

https://mc.manuscriptcentral.com/csur

Apache Storm is an open source distributed real-time processing system for data streams. The data is encapsulated into tuples (i.e. a collection of (key, value) pairs). Although, Storm provides low-level control for grouping streams in topology architecture. To this end, it requires higher tools and streaming operators to access and analyse the data for naive users. Storm is based on master-workers architectures. To avoid a single point of failure, Storm supports fault-tolerance (e.g. auto-restart for stopped workers or re-run stopped worker processes). Storm has been integrated within Twitter's platform for developing Twitter analytics services such as anti-spam and content mining and discovery [Toshniwal et al. 2014]. Overall, Storm is suitable for applications that require real-time analysis, fault-tolerance and high response rate. However, Storm does not support stateful operators (e.g. aggregation and join) which are essential for analysing data in some real-time applications such as recommendation and decision support systems. Storm Trident[44] has been proposed to support stateful operators. However, Storm does not support multi-dimensional data which is crucial for IoT applications [Dehne et al. 2013] in which the data is processed in the form of (key,value) pairs [Bahmani et al. 2012].

IBM InfoSphere Streams [Biem et al. 2010] is a commercial stream processing platform for continuous processing and analysis large-scale data streams. InfoSphere supports different operators (e.g. aggregate, join). InfoSphere receives data from different resources in the form of tuples. It can fuse various kinds of data and offers complex analysis (e.g. correlation, filtering, summarisation) over potentially continuous data streams. InfoSphere has some potential services to work in highly distributed environments such as management services including scheduling, parallelism and synchronisation [Nabi et al. 2014]. Other features and services are discussed in details in [Ballard et al. 2010]. In benchmark study in [Nabi et al. 2014] to compare between InfoSphere and Storm for email classification for on-line spam detection, InfoSphere significantly outperforms Storm. InfoSphere is effective in using CPU power. There is no much literature available about InfoSphere.

Microsoft StreamInsight [Ali 2010] is a commercial stream processing platform that has been merged with Microsoft SQL Server to allow processing of data streams. StreamInsight allows processing data streams from different resources for extracting meaningful patterns and events. StreamInsight does not support spatial streaming applications. To this end, [Kazemitabar et al. 2010] propose GeoInsight which combines both of Microsoft SQL Server Spatial libraries and StreamInsight. To support different search approaches, [Miller et al. 2011] extend StreamInsight by developing a set of search operators (range and K-nearest neighbour search). StreamInsight allows data queries using .NET Language Integrated Query (LINQ), however, it does not support dynamic queries (queries at run-time). In other words, StreamInsight does not support dynamic event processing to handle the characteristics of continuous data streams [Gorawski et al. 2014].

Esper is an open source CEP engine that was developed by EsperTech. Esper supports both centralised and distributed deployments [Cugola and Margara 2012]. Esper uses a SQL-like language called Event Processing Language (EPL) with some additional operators (e.g. time windows). The data streams are stored and processed in forms of tuples. Esper supports both of a push-based (i.e. users subscribe to specific data resources or events and they are continuously notified of any changes) and pull-based (i.e. users retrieves requested query results frequently at fixed intervals) delivery for results of requested queries. However, Esper does not support some operators such as join (i.e. merge two or more data streams from different types) and union (i.e. merge two or more data streams from the same type) operators [Calbimonte et al. 2012] which is crucial for many real-time applications.

[44]http://storm.apache.org/releases/current/Trident-API-Overview.html

G IOT/WOT SEARCH ENGINES, PLATFORMS AND APPLICATIONS

[Ostermaier et al. 2010] introduce **Dyser** which is a real-time search engine for real-world entities. It assumes that each sensor has an HTML page that contains dynamic meta-data and can be crawled. This approach is typically based on a sensor ranking approach that is presented in [Elahi et al. 2009] in which probabilistic models are used to predict which sensor/device to be contacted at the time of the query. In Dyser, indexing is based on crawling the HTML pages and indexing them in a centralised database. The ranking approach is based on an adjustment process in which rankings are predicted and improved based on previous rankings. To answer user queries while sensor observations are being changed, a predicted model is used to predict which sensor devices might have a response to a user query. It is worth mentioning that users must know the state names (e.g. occupy: empty) for all objects (e.g. room) to query them. Dyser is scalable in terms of using predictive models to find responses to queries instead of communicating with all objects; however, the indexing is centralised [Ostermaier et al. 2010].

Snoogle search engine [Wang et al. 2010] provides indexing and ranking for real-world objects. The indexing approach is based on building inverted indexes for all connected objects' IPs and managing all IPs at a Key Index Point (KeyIP). Indexes are periodically aggregated based on their locations. The main drawback of Snoogle is that indexes are based on IPs that might change and every change in sensor's meta-data require updating KeyIP, which hinders its scalability [Römer et al. 2010]. However, it provides efficient compression approach by using Bloom filter to represent the existing keys in sensor nodes. Then, a ranking approach is used by getting top k results that match user queries based on a probabilistic model to estimate which sensor has an answer to the requested query. The query processing approach is distributed. However, approximate query results might include IPs that do not have the queried data (due to using Bloom filter) [Römer et al. 2010].

Similar to Snoogle, [Yap et al. 2005] present a human-centric search called **MAX** to search for physical objects. However, MAX does not support any indexing or ranking approaches, but instead it aggregates physical objects with relative locations into base stations. MAX assumes that the devices have passive RFID tags. MAX base station receives queries, and it then broadcasts them to all sub-stations and tags to find a physical object that can provide a response. This query mechanism is the main hider for MAX's scalability. The sub-stations communicate with the physical objects based on the Received Signal Strength Indicator (RSSI), and they select the objects that have the maximum RSSI responses to answer the requested queries.

[Mayer and Guinard 2011] propose **DiscoWoT** which is a semantic discovery service for Web-enabled smart things. It uses a RESTful solution to allow integration of the things into to the Web. DiscoWoT provides different discovery strategies that can be updated at the run-time. It also allows users to describe Web sources semantically in the run-time. DiscoWoT supports source description in different formats such as JSON and Microformats. [Kansal et al. 2007] propose an infrastructure **(SenseWeb)** for shared sensing resources between multiple applications. Sensor data is collected based on the query itself. This enables accessing sensor readings dynamically. However, all applications can be accessed through one entry (coordinator) and the indexing is built in a central database (SenseDB). SenseDB minimises the overhead by combining requests that need access to the same data and caching the most frequently accessed data.

Thingful is a search engine for the IoT that connects Web-enabled objects. It includes geographical indexing and rank approaches. The ranking scheme is based on ThingRank algorithm. It allows the verification of the ownership of the registered sensory devices. However, its main drawback is the data freshness; search results could be 3 or 4 months old. Sensor data is not re-published with their changes/updates. It is possible to query (search) devices based on their locations and/or type of services; query term has What (e.g. weather, transport) and/or Where (e.g. London). It also has “Near Me” option in which users should allow Thingful to know their locations to provide them with all nearby sensory devices with their services. The indexing approach also seems to be centralised.

Wolfram Data Drop allows users to register Internet-connected things such as sensors, devices, Twitter, email, Arduino[45], Raspberry Pi[46] and others. It is based on using data discovery, visualising, analysis and modelling from WolframAlpha computational knowledge engine[47]. Each registered object has a unique “databin”. Data Drop obtains the data from registered objects every 30 seconds[48]. The main issue in Data Drop is that user should know the databin for the devices they want to query. Data Drop combines and aggregates data from different databins using Wolfram Data Framework (WDF) to summarise the combined data and convert it into a meaningful form[49]. However, there is no available information about its architecture and technical details. The **Ericsson IoT Framework** is available through a REST API and as an open source platform[50]. It collects data with time-stamps from sensors (e.g. humidity, temperature, air pollution) that have IP connectivity. Data is saved in a local central database that affects the framework’s scalability. However, it supports publication/subscription approach that is suitable for resource-constrained devices. It also supports simple mathematical aggregation (e.g. MAX, MIN) from sensor data and provides a prediction for sensor values.

ThingSpeak is an open source platform[51]. It enables storing and retrieving numeric and alphanumeric data. It also supports different data formats; XML, JSON and CSV and allows up to 8 data fields to describe a connected device. ThingSpeak allows users to search in their channel feed (exact search) and search for public channels in a specific location or within a distance. **Open.Sen.se** is another IoT platform. It has a tool called “Funnel” which processes data from several data sources and aggregates them. The platform does not have an indexing approach. The platform provides a RESTful API, and the responses are presented in JSON format[52]. Published data in the platform is always private; users need authentication (i.e. private keys) to access it. **Xively** is also a cloud-based Platform that allows real-time communication and storing data in a distributed framework [Mazhelis and Tyrvainen 2014]. Xively provides a RESTful API for retrieving data from registered sources. It supports different formats such as CSV, JSON and XML. Users can query data based on requested attributes (location, name, type of data and others) [Mineraud et al. 2015] that are represented as keyword/tag meta-data.

ACKNOWLEDGMENTS

This work is supported by the European Commission’s Seventh Framework Programme (EU FP7) for the CityPulse projectnder Grant No.: 609035 (http://www.ict-citypulse.eu/) and the EU Horizon 2020 (EU H2020) for the FIESTA-IoT projectnder Grant No.: 643943 (http://fiesta-iot.eu/).

REFERENCES

D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, et al. 2005. The Design of the Borealis Stream Processing Engine.. In *CIDR*, Vol. 5. 277–289.

[45]http://www.arduino.cc/
[46]http://www.raspberrypi.org/
[47]http://www.wolframalpha.com/
[48]http://blog.wolfram.com/2015/03/04/the-wolfram-data-drop-is-live/
[49]http://www.wolfram.com/data-framework/
[50]http://github.com/EricssonResearch/iot-framework-engine
[51]http://github.com/iobridge/ThingSpeak
[52]http://open.sen.se/dev/

D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. 2003. Aurora: a new model and architecture for data stream management. *The VLDB JournalâĂŤThe International Journal on Very Large Data Bases* 12, 2 (2003), 120–139.

K. Aberer, M. Hauswirth, and A. Salehi. 2007. Infrastructure for data processing in large-scale interconnected sensor networks. In *2007 International Conference on Mobile Data Management (MDM)*. IEEE, 198–205.

G. D. Abowd and E. D. Mynatt. 2000. Charting past, present, and future research in ubiquitous computing. *ACM Transactions on Computer-Human Interaction (TOCHI)* 7, 1 (2000), 29–58.

H. Abrach, S. Bhatti, J. Carlson, H. Dai, J. Rose, A. Sheth, B. Shucker, J. Deng, and R. Han. 2003. MANTIS: System support for multimodal networks of in-situ sensors. In *Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications*. ACM, 50–59.

M. Abu-Elkheir, M. Hayajneh, and N. A. Ali. 2013. Data management for the internet of things: Design primitives and solution. *Sensors* 13, 11 (2013), 15582–15612.

C. C. Aggarwal, N. Ashish, and A. Sheth. 2013. The internet of things: A survey from the data-centric perspective. In *Managing and Mining Sensor Data*. Springer, 383–428.

R. Agrawal, C. Faloutsos, and A. Swami. 1993. Efficient similarity search in sequence databases. In *Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms*. Springer, 69–84.

A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. 2015. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials* 17, 4 (2015), 2347–2376.

M. Ali. 2010. An introduction to microsoft sql server streaminsight. In *Proceedings of the 1st International Conference and Exhibition on Computing for Geospatial Research & Application*. ACM, 66.

M. H. Ali, W. G. Aref, R. Bose, A. K. Elmagarmid, A. Helal, I. Kamel, and M. F. Mokbel. 2005. NILE-PDT: A phenomenon detection and tracking framework for data stream management systems. In *Proceedings of the 31st international conference on Very large data bases*. VLDB Endowment, 1295–1298.

AMQP Working Group and others. 2012. About AMQP. *A General-Purpose Middleware Standard* 10 (2012). available at: http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf (last accessed: 22-12-2015).

A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom. 2004. Stream: The stanford data stream management system. *Book chapter* (2004).

A. Arasu, S. Babu, and J. Widom. 2006. The CQL continuous query language: semantic foundations and query execution. *The VLDB JournalâĂŤThe International Journal on Very Large Data Bases* 15, 2 (2006), 121–142.

E. Asmare and J. A. McCann. 2014. Lightweight Sensing Uncertainty MetricâĂŤIncorporating Accuracy and Trust. *IEEE Sensors Journal* 14, 12 (2014), 4264–4272.

L. Atzori, A. Iera, and G. Morabito. 2010. The internet of things: A survey. *Computer networks* 54, 15 (2010), 2787–2805.

R. Avnur and J. M. Hellerstein. 2000. Eddies: Continuously adaptive query processing. In *ACM SIGMoD Record*, Vol. 29. ACM, 261–272.

B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. 2002. Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 1–16.

B. Bahmani, A. Goel, and R. Shinde. 2012. Efficient distributed locality sensitive hashing. In *Proceedings of the 21st ACM international conference on Information and knowledge management*. ACM, 2174–2178.

R. Baldoni, S. Bonomi, G. Lodi, M. Platania, and L. Querzoni. 2011. Data dissemination supporting complex event pattern detection. *International Journal of Next Generation Computing* 24 (2011).

C. Ballard, D. M. Farrell, M. Lee, P. D. Stone, S. Thibault, S. Tucker, et al. 2010. *IBM InfoSphere Streams Harnessing Data in Motion*. IBM Redbooks.

P. Barnaghi, F. Ganz, C. Henson, and A. Sheth. 2012a. Computing perception from sensor data. In *IEEE Sensors, 2012*. IEEE, 1–4.

P. Barnaghi, A. Sheth, and C. Henson. 2013a. From Data to Actionable Knowledge: Big Data Challenges in the Web of Things [Guest Editors' Introduction]. *IEEE Intelligent Systems* 28, 6 (Nov 2013), 6–11.

P. Barnaghi, W. Wang, L. Dong, and C. Wang. 2013b. A Linked-Data Model for Semantic Sensor Streams. In *2013 IEEE International Conference on Green Computing and Communications (GreenCom) and IEEE Internet of Things(iThings) and IEEE Cyber, Physical and Social Computing(CPSCom)*. IEEE, 468–475.

P. Barnaghi, W. Wang, C. Henson, and K. Taylor. 2012b. Semantics for the Internet of Things: early progress and back to the future. *International Journal on Semantic Web and Information Systems (IJSWIS)* 8, 1 (2012), 1–21.

A. Bartoli, M. Dohler, J. Hernández-Serrano, A. Kountouris, and D. Barthel. 2011. Low-power low-rate goes long-range: the case for secure and cooperative machine-to-machine communications. In *NETWORKING 2011 Workshops*. Springer, 219–230.

D. Beckett and A. Barstow. 2001. N-Triples-W3C RDF Core WG Internal Working Draft. (2001). available at: http://www.w3.org/2001/sw/RDFCore/ntriples (last accessed: 22-12-2015).

D. Beckett, T. Berners-Lee, E. PrudâĂŹhommeaux, and G. Carothers. 2011. TurtleâĂŤTease RDF Triple Language, W3C Candidate Recommendation. (2011). available at: http://www.w3.org/TeamSubmission/turtle/ (last accessed: 22-12-2015).

D. Beckett and B. McBride. 2004. RDF/XML syntax specification (revised). *W3C Recommendation* 10 (2004). available at: http://www.w3.org/TR/REC-rdf-syntax/ (last accessed: 22-12-2015).

N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. 1990. The R*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, Vol. 19. ACM.

D. J. Berndt and J. Clifford. 1994. Using Dynamic Time Warping to Find Patterns in Time Series. In *Workshop on Knowledge Knowledge Discovery in Databases (KDD)*, Vol. 10. Seattle, WA, AAAI Press, 359–370.

T. Berners-Lee and D. Connolly. 2011. Notation3 (N3): A readable RDF syntax. *W3C Team Submission* (2011). available at: http://www.w3.org/TeamSubmission/n3/ (last accessed: 22-12-2015).

V. Bettaiah and H. S. Ranganath. 2014. An effective subsequence-to-subsequence time series matching approach. In *Science and Information Conference (SAI), 2014*. 112–122.

A. R. Bharambe, M. Agrawal, and S. Seshan. 2004. Mercury: supporting scalable multi-attribute range queries. In *ACM SIGCOMM Computer Communication Review*, Vol. 34. ACM, 353–366.

A. Bhattacharya, A. Meka, and A. K. Singh. 2007. MIST: Distributed indexing and querying in sensor networks using statistical models. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB)*. VLDB Endowment, 854–865.

S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han. 2005. MANTIS OS: An embedded multithreaded operating system for wireless micro sensor platforms. *Mobile Networks and Applications* 10, 4 (2005), 563–579.

A. Biem, E. Bouillet, H. Feng, A. Ranganathan, A. Riabov, O. Verscheure, H. Koutsopoulos, and C. Moran. 2010. IBM infosphere streams for scalable, real-time, intelligent transportation services. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 1093–1104.

D. Bimschas, H. Hellbrück, R. Mietz, D. Pfisterer, K. Römer, and T. Teubler. 2010. Middleware for smart gateways connecting sensornets to the internet. In *Proceedings of the 5th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks*. ACM, 8–14.

C. Bisdikian, L. M. Kaplan, and M. B. Srivastava. 2013. On the quality and value of information in sensor networks. *ACM Transactions on Sensor Networks (TOSN)* 9, 4 (2013), 48.

C. Bisdikian, L. M. Kaplan, M. B. Srivastava, D. J. Thornley, D. Verma, and R. I. Young. 2009. Building principles for a quality of information specification for sensor information. In *Information Fusion, 2009. FUSION'09. 12th International Conference on*. IEEE, 1370–1377.

C. Bizer, T. Heath, and T. Berners-Lee. 2009. Linked data-the story so far. *International Journal on Semantic Web and Information Systems (IJSWIS)* 5, 3 (2009), 1–22.

D. M. Blei, A. Y. Ng, and M. I. Jordan. 2003. Latent dirichlet allocation. *the Journal of Machine Learning Research* 3 (2003), 993–1022.

F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. 2012. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 13–16.

P. Boonma and J. Suzuki. 2007. BiSNET: A biologically-inspired middleware architecture for self-managing wireless sensor networks. *Computer Networks* 51, 16 (2007), 4599–4616.

C. Bormann, A. P. Castellani, and Z. Shelby. 2012. Coap: An application protocol for billions of tiny internet nodes. *IEEE Internet Computing* 2 (2012), 62–67.

A. Botta, W. De Donato, V. Persico, and A. Pescapé. 2016. Integration of cloud computing and internet of things: a survey. *Future Generation Computer Systems* 56 (2016), 684–700.

M. Botts, G. Percivall, C. Reed, and J. Davidson. 2008. OGC® sensor web enablement: Overview and high level architecture. In *GeoSensor Networks*. Springer, 175–190.

M. Botts and A. Robin. 2007. OpenGIS sensor model language (SensorML) implementation specification. *OpenGIS Implementation Specification OGC* 7, 000 (2007). available at: http://portal.opengeospatial.org/files/?artifact_id=21273 (last accessed: 22-12-2015).

A. Boulis, C.-C. Han, and M. B. Srivastava. 2003. Design and implementation of a framework for efficient and programmable sensor networks. In *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services*. ACM, 187–200.

S. Brin and L. Page. 2012. The anatomy of a large-scale hypertextual web search engine. *Computer Networks* 56, 18 (2012), 3825–3833.

Y. Cai and R. Ng. 2004. Indexing spatio-temporal trajectories with Chebyshev polynomials. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*. ACM, 599–610.

J.-P. Calbimonte, H. Y. Jeung, O. Corcho, and K. Aberer. 2012. Enabling query technologies for the semantic sensor web. *International Journal on Semantic Web and Information Systems* 8, EPFL-ARTICLE-183971 (2012), 43–63.

A. Camerra, T. Palpanas, J. Shieh, and E. Keogh. 2010. iSAX 2.0: Indexing and mining one billion time series. In *2010 IEEE 10th International Conference on Data Mining (ICDM)*. IEEE, 58–67.

A. Camerra, J. Shieh, T. Palpanas, T. Rakthanmanon, and E. Keogh. 2014. Beyond one billion time series: indexing and mining very large time series collections with iSAX2+. *Knowledge and Information Systems* 39, 1 (2014), 123–151.

D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik. 2002. Monitoring streams: a new class of data management applications. In *Proceedings of the 28th international conference on Very Large Data Bases*. VLDB Endowment, 215–226.

G. Cassar, P. Barnaghi, and K. Moessner. 2010. Probabilistic methods for service clustering. In *Proceedings of 4th International Workshop on Semantic Web Service Matchmaking and Resource Retrieval*. ISWC.

G. Cassar, P. Barnaghi, and K. Moessner. 2014. Probabilistic matchmaking methods for automated service discovery. *IEEE Transactions on Services Computing* 7, 4 (2014), 654–666.

N. Castro and P. J. Azevedo. 2010. Multiresolution Motif Discovery in Time Series. In *SDM*. SIAM, 665–676.

K.-P. Chan and A.-C. Fu. 1999. Efficient time series matching by wavelets. In *Proceedings of the 15th International Conference on Data Engineering, 1999*. IEEE, 126–133.

S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah. 2003. TelegraphCQ: continuous dataflow processing. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM, 668–668.

G. Chen and D. Kotz. 2002. Context aggregation and dissemination in ubiquitous computing systems. In *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*. IEEE Computer Society, 105–114.

J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. 2000. NiagaraCQ: A scalable continuous query system for internet databases. In *ACM SIGMOD Record*, Vol. 29. ACM, 379–390.

M. Chen, S. Mao, and Y. Liu. 2014. Big data: A survey. *Mobile Networks and Applications* 19, 2 (2014), 171–209.

M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. B. Zdonik. 2003. Scalable Distributed Stream Processing.. In *CIDR*, Vol. 3. 257–268.

S. Cirani, L. Davoli, G. Ferrari, R. Léone, P. Medagliani, M. Picone, and L. Veltri. 2014. A scalable and self-configuring architecture for service discovery in the internet of things. *IEEE Internet of Things Journal* 1, 5 (2014), 508–521.

D. Comer. 1979. Ubiquitous B-tree. *ACM Computing Surveys (CSUR)* 11, 2 (1979), 121–137.

M. Compton, P. Barnaghi, L. Bermudez, R. García-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, et al. 2012. The SSN ontology of the W3C semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web* 17 (2012), 25–32.

J. W. Cooley and J. W. Tukey. 1965. An algorithm for the machine calculation of complex Fourier series. *Math. Comp.* 19, 90 (1965), 297–301.

J. Cooper and A. James. 2009. Challenges for database management in the internet of things. *IETE Technical Review* 26, 5 (2009), 320–329.

A. Corsaro and D. C. Schmidt. 2012. *The Data Distribution Service-The Communication Middleware Fabric for Scalable and Extensible Systems-of-Systems*. INTECH Open Access Publisher. available at: http://www.intechopen.com/books/system-of-systems/the-data-distribution-service-the-fabric-for-building-scalable-and-extensible-system-of-systems (last accessed: 22-12-2015).

G. Cugola and A. Margara. 2012. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys (CSUR)* 44, 3 (2012), 15.

A. R. da Rocha, F. C. Delicato, J. N. de Souza, D. G. Gomes, and L. Pirmez. 2009. A semantic middleware for autonomic wireless sensor networks. In *Proceedings of the 2009 Workshop on Middleware for Ubiquitous and Pervasive Systems*. ACM, 19–25.

M.-S. Dao, S. Pongpaichet, L. Jalali, K. Kim, R. Jain, and K. Zettsu. 2014. A real-time complex event discovery platform for cyber-physical-social systems. In *Proceedings of International Conference on Multimedia Retrieval*. ACM, 201.

L. David, R. Vasconcelos, L. Alves, R. André, and M. Endler. 2013. A DDS-based middleware for scalable tracking, communication and collaboration of mobile nodes. *Journal of Internet Services and Applications* 4, 1 (2013), 1–15.

I. Davis, T. Steiner, and A. Hors. 2013. RDF 1.1 JSON Alternate Serialization (RDF/JSON). *W3C Working Group Note* (2013). available at: http://www.w3.org/TR/2013/NOTE-rdf-json-20131107 (last accessed: 22-12-2015).

S. De, P. Barnaghi, M. Bauer, and S. Meissner. 2011. Service modelling for the Internet of Things. In *2011 Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 949–955.

F. Dehne, Q. Kong, A. Rau-Chaplin, H. Zaboli, and R. Zhou. 2013. A distributed tree data structure for real-time OLAP on cloud architectures. In *Big Data, 2013 IEEE International Conference on*. IEEE, 499–505.

Y. Demchenko, P. Grosso, C. De Laat, and P. Membrey. 2013. Addressing big data issues in scientific data infrastructure. In *Collaboration Technologies and Systems (CTS), 2013 International Conference on*. IEEE, 48–55.

A. J. Demers, J. Gehrke, B. Panda, M. Riedewald, V. Sharma, W. M. White, et al. 2007. Cayuga: A General Purpose Event Monitoring System.. In *CIDR*, Vol. 7. 412–422.

M. Demirbas and X. Lu. 2007. Distributed quad-tree for spatial querying in wireless sensor networks. In *IEEE International Conference on Communications (ICC)*. IEEE, 3325–3332.

P. Desai, A. Sheth, and P. Anantharam. 2015. Semantic Gateway as a Service architecture for IoT Interoperability. In *IEEE International Conference on Mobile Services (MS), 2015*. 313–319.

O. Diallo, J. J. Rodrigues, and M. Sene. 2012. Real-time data management on wireless sensor networks: a survey. *Journal of Network and Computer Applications* 35, 3 (2012), 1013–1021.

H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. 2008. Querying and mining of time series data: experimental comparison of representations and distance measures. *Proceedings of the Very Large Data Bases (VLDB) Endowment* 1, 2 (2008), 1542–1552.

A. Dunkels, B. Gronvall, and T. Voigt. 2004. Contiki-a lightweight and flexible operating system for tiny networked sensors. In *29th Annual IEEE International Conference on Local Computer Networks, 2004*. IEEE, 455–462.

M. Durvy, J. Abeillé, P. Wetterwald, C. O'Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, N. Tsiftes, N. Finne, et al. 2008. Making sensor networks IPv6 ready. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*. ACM, 421–422.

W. K. Edwards. 2006. Discovery systems in ubiquitous computing. *IEEE Pervasive Computing* 5, 2 (2006), 70–77.

B. M. Elahi, K. Romer, B. Ostermaier, M. Fahrmair, and W. Kellerer. 2009. Sensor ranking: A primitive for efficient content-based sensor search. In *International Conference on Information Processing in Sensor Networks (IPSN), 2009*. IEEE Computer Society, 217–228.

L. Ericsson. 2011. More than 50 billion connected devices. *White Paper* (2011). available at: http://www.akos-rs.si/files/Telekomunikacije/Digitalna_agenda/Internetni_protokol_Ipv6/More-than-50-billion-connected-devices.pdf (last accessed: 22-12-2015).

B. Esmael, A. Arnaout, R. K. Fruhwirth, and G. Thonhauser. 2012. Multivariate time series classification by combining trend-based and value-based approximations. In *Computational Science and Its Applications–ICCSA 2012*. Springer, 392–403.

C. Esposito. 2011. Data distribution service (DDS) limitations for data dissemination wrt large-scale complex critical infrastructures (LCCI). *MobiLab, Università degli Studi di Napoli Federico II, Napoli, Italy, Tech. Rep* (2011). available at: http://www.mobilab.unina.it/Reports/DataDiss.pdf (last accessed: 22-12-2015).

S. Evdokimov, B. Fabian, S. Kunz, and N. Schoenemann. 2010. Comparison of discovery service architectures for the internet of things. In *2010 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC)*. IEEE, 237–244.

P. Evensen and H. Meling. 2009. SenseWrap: A service oriented middleware with sensor virtualization and self-configuration. In *2009 5th International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*. IEEE, 261–266.

C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. 1994. Fast Subsequence Matching in Time-series Databases. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data (SIGMOD '94)*, Vol. 23. ACM, 419–429.

E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi. 2007. In-network aggregation techniques for wireless sensor networks: a survey. *IEEE Wireless Communications* 14, 2 (2007), 70–87.

Y. Fathy, P. Barnaghi, S. Enshaeifar, and R. Tafazolli. 2016. A Distributed In-network Indexing Mechanism for the Internet of Things. In *Internet of Things (WF-IoT), 2016 IEEE 3nd World Forum on*. IEEE.

D. Figo, P. C. Diniz, D. R. Ferreira, and J. M. Cardoso. 2010. Preprocessing techniques for context recognition from accelerometer data. *Personal and Ubiquitous Computing* 14, 7 (2010), 645–662.

C.-L. Fok, G.-C. Roman, and C. Lu. 2005a. Mobile agent middleware for sensor networks: An application case study. In *Fourth International Symposium on Information Processing in Sensor Networks (IPSN), 2005*. IEEE, 382–387.

C.-L. Fok, G.-C. Roman, and C. Lu. 2005b. Rapid development and flexible deployment of adaptive wireless sensor network applications. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS), 2005*. IEEE, 653–662.

L. J. Fülöp, G. Tóth, R. Rácz, J. Pánczél, T. Gergely, A. Beszédes, and L. Farkas. 2010. Survey on complex event processing and predictive analytics. In *Proceedings of the Fifth Balkan Conference in Informatics*. Citeseer, 26–31.

A. Gani, A. Siddiqa, S. Shamshirband, and F. Hanum. 2016. A survey on indexing techniques for big data: taxonomy and performance evaluation. *Knowledge and Information Systems* 46, 2 (2016), 241–284.

F. Ganz, P. Barnaghi, and F. Carrez. 2013. Information abstraction for heterogeneous real world internet data. *Sensors Journal, IEEE* 13, 10 (2013), 3793–3805.

F. Ganz, P. Barnaghi, F. Carrez, and K. Moessner. 2011. Context-aware management for sensor networks. In *Proceedings of the 5th International Conference on Communication System Software and Middleware*. ACM, 6.

F. Ganz, R. Li, P. Barnaghi, and H. Harai. 2012. A resource mobility scheme for service-continuity in the Internet of Things. In *2012 IEEE International Conference on Green Computing and Communications (GreenCom)*. IEEE, 261–264.

F. Ganz, D. Puschmann, P. Barnaghi, and F. Carrez. 2015. A Practical Evaluation of Information Processing and Abstraction Techniques for the Internet of Things. *IEEE Internet of Things Journal* (2015).

D. Gay, P. Levis, R. Von Behren, M. Welsh, E. Brewer, and D. Culler. 2003. The nesC language: A holistic approach to networked embedded systems. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, Vol. 38. ACM, 1–11.

L. Golab and M. T. Özsu. 2003. Issues in data stream management. *ACM Sigmod Record* 32, 2 (2003), 5–14.

G. H. Golub and C. Reinsch. 1970. Singular value decomposition and least squares solutions. *Numer. Math.* 14, 5 (1970), 403–420.

M. Gorawski, A. Gorawska, and K. Pasterak. 2014. A survey of data stream processing tools. In *Information Sciences and Systems 2014*. Springer, 295–303.

J. Grass and S. Zilberstein. 1996. Anytime algorithm development tools. *ACM SIGART Bulletin* 7, 2 (1996), 20–27.

B. Greenstein, S. Ratnasamy, S. Shenker, R. Govindan, and D. Estrin. 2003. DIFS: A distributed index for features in sensor networks. *Ad Hoc Networks* 1, 2 (2003), 333–349.

S. Guha, D. Gunopulos, and N. Koudas. 2003. Correlating synchronous and asynchronous data streams. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 529–534.

D. Guinard. 2011. *A web of things application architecture-Integrating the real-world into the web.* Ph.D. Dissertation. Citeseer.

D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio. 2010. Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services. *IEEE Transactions on Services Computing* 3, 3 (2010), 223–235.

D. Guinard, V. Trifa, F. Mattern, and E. Wilde. 2011. From the internet of things to the web of things: Resource-oriented architecture and best practices. In *Architecting the Internet of Things*. Springer, 97–129.

M. A. Hammad, M. F. Mokbel, M. H. Ali, W. G. Aref, A. C. Catlin, A. K. Elmagarmid, M. Eltabakh, M. G. Elfeky, T. M. Ghanem, R. Gwadera, et al. 2004. Nile: A query processing engine for data streams. In *Data Engineering, 2004. Proceedings. 20th International Conference on*. IEEE, 851.

C.-C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava. 2005. A dynamic operating system for sensor nodes. In *Proceedings of the 3rd International Conference on Mobile systems, Applications, and Services (MobiSys)*. ACM, 163–176.

L. Harada. 2002. Pattern matching over multi-attribute data streams. In *String Processing and Information Retrieval*. Springer, 187–193.

A. Harth and S. Decker. 2005. Optimized index structures for querying rdf from the web. In *Third Latin American Web Congress (LA-WEB'2005)*. IEEE, 10–pp.

W. He and L. Da Xu. 2014. Integration of distributed enterprise applications: a survey. *IEEE Transactions on Industrial Informatics* 10, 1 (2014), 35–42.

S. Herbst, N. Pollner, J. Tenschert, F. Lauterwald, G. Endler, and K. Meyer-Wegener. 2015. An algebra for pattern matching, time-aware aggregates and partitions on relational data streams. In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*. ACM, 140–149.

M. Hoffman, F. R. Bach, and D. M. Blei. 2010. Online learning for latent dirichlet allocation. In *Advances in Neural Information Processing Systems 23*. Curran Associates, Inc., 856–864.

S. Hoseinitabatabaei, P. Barnaghi, R. Tafazolli, and C. Wang. 2014. Method and Apparatus for Scalable Data Discovery in IoT Systems. (United States Patent: CNV12174, May 2014).

S. Huang, Y. Chen, X. Chen, K. Liu, X. Xu, C. Wang, K. Brown, and I. Halilovic. 2014. The next generation operational data historian for IoT based on informix. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. ACM, 169–176.

J. W. Hui and D. E. Culler. 2008. Extending IP to low-power, wireless personal area networks. *Internet Computing, IEEE* 12, 4 (2008), 37–45.

U. Hunkeler, H. L. Truong, and A. Stanford-Clark. 2008. MQTT-SâĂŤA publish/subscribe protocol for Wireless Sensor Networks. In *3rd International Conference on Communication System Software and Middleware (COMSWARE), 2008*. IEEE, 791–798.

I. Ishaq, D. Carels, G. K. Teklemariam, J. Hoebeke, F. V. d. Abeele, E. D. Poorter, I. Moerman, and P. Demeester. 2013. IETF standardization in the field of the internet of things (IoT): a survey. *Journal of Sensor and Actuator Networks* 2, 2 (2013), 235–287.

C. Jennings, J. Arkko, and Z. Shelby. 2015. Media types for Sensor Markup Language (SENML). (2015). available at: http://datatracker.ietf.org/doc/draft-jennings-core-senml/ (last accessed: 22-12-2015).

F. T. Johnsen, T. H. Bloebaum, M. Avlesen, S. Spjelkavik, and B. Vik. 2013. Evaluation of transport protocols for web services. In *Military Communications and Information Systems Conference (MCC), 2013*. IEEE, 1–6.

P. G. Jones and P. K. Thornton. 2000. MarkSim: software to generate daily weather data for Latin America and Africa. *Agronomy Journal* 92, 3 (2000), 445–453.

A. Kamilaris, A. Pitsillides, and V. Trifa. 2011. The smart home meets the web of things. *International Journal of Ad Hoc and Ubiquitous Computing* 7, 3 (2011), 145–154.

A. Kansal, S. Nath, J. Liu, and F. Zhao. 2007. Senseweb: An infrastructure for shared sensing. *IEEE MultiMedia* 4 (2007), 8–13.

V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate. 2015. A Survey on Application Layer Protocols for the Internet of Things. *Transaction on IoT and Cloud Computing (TICC)* 3, 1 (2015), 11–17.

S. J. Kazemitabar, U. Demiryurek, M. Ali, A. Akdogan, and C. Shahabi. 2010. Geospatial stream query processing using Microsoft SQL Server StreamInsight. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 1537–1540.

E. Keogh. 1997. Fast similarity search in the presence of longitudinal scaling in time series databases. In *Proceedings Ninth IEEE International Conference on Tools with Artificial Intelligence, 1997.* IEEE, 578–584.

E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. 2001a. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and Information Systems* 3, 3 (2001), 263–286.

E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. 2001b. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM SIGMOD Record* 30, 2 (2001), 151–162.

E. Keogh and S. Kasetty. 2003. On the need for time series data mining benchmarks: a survey and empirical demonstration. *Data Mining and Knowledge Discovery* 7, 4 (2003), 349–371.

E. Keogh and J. Lin. 2005. Clustering of time-series subsequences is meaningless: implications for previous and future research. *Knowledge and Information Systems* 8, 2 (2005), 154–177.

E. Keogh and C. A. Ratanamahatana. 2005. Exact indexing of dynamic time warping. *Knowledge and Information Systems* 7, 3 (2005), 358–386.

E. J. Keogh and M. J. Pazzani. 2000. Scaling up dynamic time warping for datamining applications. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, 285–289.

R. Khan, S. U. Khan, R. Zaheer, and S. Khan. 2012. Future internet: the internet of things architecture, possible applications and key challenges. In *Frontiers of Information Technology (FIT), 2012 10th International Conference on.* IEEE, 257–260.

M. Kirsche and R. Klauck. 2012. Unify to bridge gaps: Bringing XMPP into the internet of things. In *2012 IEEE International Conference on Pervasive Computing and Communications (PERCOM) Workshops.* IEEE, 455–458.

A. Klein and W. Lehner. 2009. Representing data quality in sensor data streaming environments. *Journal of Data and Information Quality (JDIQ)* 1, 2 (2009), 10.

J. M. Kleinberg. 1999. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)* 46, 5 (1999), 604–632.

G. Klyne and J. J. Carroll. 2006. Resource description framework (RDF): Concepts and abstract syntax. (2006). available at: http://www.w3.org/TR/rdf-concepts/ (last accessed: 22-12-2015).

J. Ko, J. Eriksson, N. Tsiftes, S. Dawson-Haggerty, M. Durvy, A. Terzis, A. Dunkels, and D. Culler. 2011. Beyond interoperability: Pushing the performance of sensornet IP stacks. In *In Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys).* ACM, 1–11.

F. Korn, H. V. Jagadish, and C. Faloutsos. 1997. Efficiently supporting ad hoc queries in large datasets of time sequences. *ACM SIGMOD Record* 26, 2 (1997), 289–300.

K. Kotis and A. Katasonov. 2012. Semantic interoperability on the web of things: The semantic smart gateway framework. In *2012 Sixth International Conference on Complex, Intelligent and Software Intensive Systems (CISIS).* IEEE, 630–635.

P. Kranen and T. Seidl. 2009. Using Index Structures for Anytime Stream Mining. In *Very Large Data Bases (VLDB) PhD Workshop.* VLDB Endowment.

B. Krishnamachari, D. Estrin, and S. Wicker. 2002. The impact of data aggregation in wireless sensor networks. In *Proceedings 22nd International Conference on Distributed Computing Systems Workshops, 2002.* IEEE, 575–578.

S. Krishnamurthy, S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Madden, F. Reiss, and M. A. Shah. 2003. TelegraphCQ: An architectural status report. *IEEE Data Eng. Bull.* 26, 1 (2003), 11–18.

J. B. Kruskal and M. Wish. 1978. *Multidimensional scaling.* Vol. 11. SAGE.

A. Kumbhare, Y. Simmhan, and V. K. Prasanna. 2013. Exploiting application dynamism and cloud elasticity for continuous dataflows. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis.* ACM, 57.

Y. Kwon, S. Sundresh, K. Mechitov, and G. Agha. 2006. ActorNet: An actor platform for wireless sensor networks. In *Proceedings of the fifth International Joint Conference on Autonomous Agents and Multiagent Systems.* ACM, 1297–1300.

M. H. Law, N. Zhang, and A. K. Jain. 2004. Nonlinear Manifold Learning for Data Stream. In *Proceedings of SIAM International Conference for Data Mining.* SIAM, 33–44.

J. K. Lawder and P. J. King. 2000. Using space-filling curves for multi-dimensional indexing. In *British National Conference on Databases.* Springer, 20–35.

D. Le-Phuoc, H. N. M. Quoc, J. X. Parreira, and M. Hauswirth. 2011. The linked sensor middleware–connecting the real world and the semantic web. *Proceedings of the Semantic Web Challenge* 152 (2011).

J. Ledlie, C. Ng, and D. A. Holland. 2005. Provenance-aware sensor data storage. In *21st International Conference on Data Engineering Workshops (ICDEW'05).* IEEE, 1189–1189.

K. Lee, D. Murray, D. Hughes, and W. Joosen. 2010. Extending sensor networks into the cloud using amazon web services. In *Networked Embedded Systems for Enterprise Applications (NESEA), 2010 IEEE International Conference on*. IEEE, 1–7.

P. Levis and D. Culler. 2002. Maté: A tiny virtual machine for sensor networks. In *ACM SIGPLAN Notices*, Vol. 37. ACM, 85–95.

P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, et al. 2005. TinyOS: An operating system for sensor networks. In *Ambient Intelligence*. Springer, 115–148.

X. Li, Y. J. Kim, R. Govindan, and W. Hong. 2003. Multi-dimensional range queries in sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys)*. ACM, 63–75.

X. Li, J. Liu, Q. Z. Sheng, S. Zeadally, and W. Zhong. 2011. TMS-RFID: Temporal management of large-scale RFID applications. *Information Systems Frontiers* 13, 4 (2011), 481–500.

S. H. Liang and C.-Y. Huang. 2013. Geocens: A geospatial cyberinfrastructure for the world-wide sensor web. *Sensors* 13, 10 (2013), 13402–13424.

D. Lin, B. Cui, and D. Yang. 2007a. Optimizing moving queries over moving object data streams. In *Advances in Databases: Concepts, Systems and Applications*. Springer, 563–575.

J. Lin, E. Keogh, S. Lonardi, and B. Chiu. 2003. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in Data Mining and Knowledge Discovery*. ACM, 2–11.

J. Lin, E. Keogh, L. Wei, and S. Lonardi. 2007b. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery* 15, 2 (2007), 107–144.

Y. Lin, X. Hu, and X. Wu. 2014. Quality of information-based source assessment and selection. *Neurocomputing* 133 (2014), 95–102.

B. Lkhagva, Y. Suzuki, and K. Kawagoe. 2006. New Time Series Data Representation ESAX for Financial Applications. In *Proceedings of the 22Nd International Conference on Data Engineering Workshops*. IEEE Computer Society, 115.

D. Locke. 2010. Mq telemetry transport (mqtt) v3. 1 protocol specification. *IBM developerWorks Technical Library* (2010). available at: http://www.ibm.com/developerworks/webservices/library/ws-mqtt/index.html (last accessed: 22-12-2015).

W. T. Lunardi, E. de Matos, R. Tiburski, L. A. Amaral, S. Marczak, and F. Hessel. 2015. Context-based search engine for industrial IoT: Discovery, search, selection, and usage of devices. In *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*. IEEE, 1–8.

S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. 2005. TinyDB: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems (TODS)* 30, 1 (2005), 122–173.

O. G. Management. 2009. The Real-Time Publish-Subscribe Wire Protocol DDS Interoperability Wire Protocol Specification (DDSI). (2009). available at: http://www.omg.org/spec/DDSI/2.1/PDF/ (last accessed: 22-12-2015).

P. Manzanares-Lopez, J. P. Muñoz-Gea, J. Malgosa-Sanahuja, and J. C. Sanchez-Aarnoutse. 2011. An efficient distributed discovery service for EPCglobal network in nested package scenarios. *Journal of Network and Computer Applications* 34, 3 (2011), 925–937.

S. Mascetti, D. Freni, C. Bettini, X. S. Wang, and S. Jajodia. 2011. Privacy in geo-social networks: proximity notification with untrusted service providers and curious buddies. *The International Journal on Very Large Data Bases (VLDB)* 20, 4 (2011), 541–566.

S. Mayer and D. Guinard. 2011. An extensible discovery service for smart things. In *Proceedings of the Second International Workshop on Web of Things*. ACM, 7.

O. Mazhelis and P. Tyrvainen. 2014. A framework for evaluating Internet-of-Things platforms: Application provider viewpoint. In *IEEE World Forum on Internet of Things (WF-IoT)*. IEEE, 147–152.

A. Meliou, C. Guestrin, and J. M. Hellerstein. 2009. Approximating sensor network queries using in-network summaries. In *International Conference on Information Processing in Sensor Networks (IPSN), 2009*. IEEE, 229–240.

R. Mietz, S. Groppe, K. Römer, and D. Pfisterer. 2013. Semantic models for scalable search in the internet of things. *Journal of Sensor and Actuator Networks* 2, 2 (2013), 172–195.

J. Miller, M. Raymond, J. Archer, S. Adem, L. Hansel, S. Konda, M. Luti, Y. Zhao, A. Teredesai, and M. Ali. 2011. An extensibility approach for spatio-temporal stream processing using microsoft streaminsight. In *International Symposium on Spatial and Temporal Databases*. Springer, 496–501.

J. Mineraud, O. Mazhelis, X. Su, and S. Tarkoma. 2015. Contemporary Internet of Things platforms. *Computing Research Repository (CoRR)* (2015).

D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac. 2012. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks* 10, 7 (2012), 1497–1516.

I. Mitliagkas, C. Caramanis, and P. Jain. 2013. Memory limited, streaming PCA. In *Advances in Neural Information Processing Systems 26*. Curran Associates, Inc., 2886–2894.

M. M. A. Mohamed, A. Khokhar, et al. 2011. Dynamic indexing system for spatio-temporal queries in wireless sensor networks. In *The12th IEEE International Conference on Mobile Data Management (MDM)*, Vol. 2. IEEE, 35–37.

M. F. Mokbel, W. G. Aref, and I. Kamel. 2002. Performance of multi-dimensional space-filling curves. In *Proceedings of the 10th ACM international symposium on Advances in geographic information systems*. ACM, 149–154.

M. F. Mokbel, X. Xiong, M. A. Hammad, and W. G. Aref. 2005. Continuous query processing of spatio-temporal data streams in place. *GeoInformatica* 9, 4 (2005), 343–365.

B. Moore. 1981. Principal component analysis in linear systems: Controllability, observability, and model reduction. *IEEE Trans. Automat. Control* 26, 1 (1981), 17–32.

R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. 2003. Query processing, resource management, and approximation in a data stream management system. CIDR.

A. Murphy and W. Heinzelman. 2002. Milan: Middleware linking applications and networks. *University of Rochester, Tech. Rep. TR-795* (2002), 1–16.

Z. Nabi, E. Bouillet, A. Bainbridge, and C. Thomas. 2014. Of streams and storms. *IBM White Paper* (2014).

S. Nath, J. Liu, and F. Zhao. 2007. SensorMap for Wide-Area Sensor Webs. *Computer* 40, 7 (2007), 90–93.

W. Niu, J. Lei, E. Tong, G. Li, L. Chang, Z. Shi, and S. Ci. 2014. Context-aware service ranking in wireless sensor networks. *Journal of network and systems management* 22, 1 (2014), 50–74.

J. O'Hara. 2007. Toward a commodity enterprise middleware. *Queue* 5, 4 (2007), 48–55.

O. OMG. 2006. *Data Distribution Service for Real-Time Systems.* Technical Report. Technical Report OMG Available Specification formal/07-01-01, OMG. available at: http://community.rti.com/filedepot_download/1795/16 (last accessed: 22-12-2015).

B. Ostermaier, K. RoÌĹmer, F. Mattern, M. Fahrmair, and W. Kellerer. 2010. A real-time search engine for the web of things. In *Internet of Things (IoT), 2010*. IEEE, 1–8.

F. Paganelli and D. Parlanti. 2012. A DHT-based discovery service for the Internet of Things. *Journal of Computer Networks and Communications* 2012 (2012).

M. R. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. A. Grieco, G. Boggia, and M. Dohler. 2013. Standardized protocol stack for the internet of (important) things. *IEEE Communications Surveys & Tutorials* 15, 3 (2013), 1389–1406.

S. Paradesi, P. Doshi, and S. Swaika. 2009. Integrating behavioral trust in web service compositions. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*. IEEE, 453–460.

C. Perera, A. Zaslavsky, P. Christen, M. Compton, and D. Georgakopoulos. 2013. Context-aware sensor search, selection and ranking model for internet of things middleware. In *2013 IEEE 14th International Conference on Mobile Data Management (MDM)*, Vol. 1. IEEE, 314–322.

C. Perera, A. Zaslavsky, C. H. Liu, M. Compton, P. Christen, and D. Georgakopoulos. 2014. Sensor search techniques for sensing as a service architecture for the internet of things. *IEEE Sensors Journal* 14, 2 (2014), 406–420.

N. D. Pham, Q. L. Le, and T. K. Dang. 2010. Two novel adaptive symbolic representations for similarity search in time series databases. In *2010 12th International Asia-Pacific Web Conference (APWEB)*. IEEE, 181–187.

A. M. R. V. A. Phani, D. J. Kumar, and G. A. Kumar. 2007. Operating systems for wireless sensor networks: A survey technical report. (2007). available at: http://dos.iitm.ac.in/publications/LabPapers/adiWirelessOS.pdf (last accessed: 22-12-2015).

R. Piyare and S. R. Lee. 2013. Towards Internet of Things (IOTs): Integration of wireless sensor network to cloud services for data collection and sharing. *Computing Research Repository (CoRR)* (2013).

E. Polytarchos, S. Eliakis, D. Bochtis, and K. Pramatari. 2011. Evaluating discovery services architectures in the context of the internet of things. In *Unique Radio Innovation for the 21st Century*. Springer, 203–227.

E. Prud'Hommeaux, A. Seaborne, et al. 2008. SPARQL query language for RDF. *W3C Recommendation* 15 (2008). available at: http://www.w3.org/TR/rdf-sparql-query/ (last accessed: 22-12-2015).

Y. Qin, Q. Z. Sheng, N. J. Falkner, S. Dustdar, H. Wang, and A. V. Vasilakos. 2016. When things matter: A survey on data-centric internet of things. *Journal of Network and Computer Applications* 64 (2016), 137–153.

R. Rajagopalan and P. K. Varshney. 2006. Data aggregation techniques in sensor networks: A survey. *IEEE Communications Surveys Tutorials* 8, 4 (2006), 48–63.

T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. 2013. Addressing big data time series: Mining trillions of time series subsequences under dynamic time warping. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 7, 3 (2013), 10.

S. Ramabhadran, S. Ratnasamy, J. M. Hellerstein, and S. Shenker. 2004. Brief announcement: prefix hash tree. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*. ACM, 368–368.

C. A. Ratanamahatana and E. Keogh. 2005. Three myths about dynamic time warping data mining. In *Proceedings of SIAM International Conference on Data Mining (SDMâĂŹ05)*. SIAM, 506–510.

S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. 2002. GHT: a geographic hash table for data-centric storage. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*. ACM, 78–87.

J. A. Reggia and Y. Peng. 1987. Modeling diagnostic reasoning: a summary of parsimonious covering theory. *Computer Methods and Programs in Biomedicine* 25, 2 (1987), 125–134.

K. Römer, O. Kasten, and F. Mattern. 2002. Middleware challenges for wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Review* 6, 4 (2002), 59–61.

K. Römer, B. Ostermaier, F. Mattern, M. Fahrmair, and W. Kellerer. 2010. Real-time search for real-world entities: A survey. *Proc. IEEE* 98, 11 (2010), 1887–1902.

A. Rooshenas, H. R. Rabiee, A. Movaghar, and M. Y. Naderi. 2010. Reducing the data transmission in wireless sensor networks using the principal component analysis. In *2010 Sixth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*. IEEE, 133–138.

S. T. Roweis and L. K. Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *Science* 290, 5500 (2000), 2323–2326.

P. Saint-Andre. 2011. Extensible messaging and presence protocol (XMPP): Core. (2011). available at: http://tools.ietf.org/html/rfc6120 (last accessed: 22-12-2015).

S. Salvador and P. Chan. 2007. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis* 11, 5 (2007), 561–580.

J. Schneider, T. Kamiya, D. Peintner, and R. Kyusakov. 2011. Efficient XML Interchange (EXI) format 1.0. *W3C Proposed Recommendation* 20 (2011). available at: http://www.w3.org/TR/exi/ (last accessed: 22-12-2015).

L. Schor, P. Sommer, and R. Wattenhofer. 2009. Towards a zero-configuration wireless sensor network architecture for smart buildings. In *Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*. ACM, 31–36.

P. Schramm, E. Naroska, P. Resch, J. Platte, H. Linde, G. Stromberg, and T. Sturm. 2004. A service gateway for networked sensor systems. *IEEE Pervasive Computing* 3, 1 (2004), 66–74.

N. P. Schultz-Møller, M. Migliavacca, and P. Pietzuch. 2009. Distributed complex event processing with query rewriting. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*. ACM, 4.

J. Schuol and K. Abbaspour. 2007. Using monthly weather statistics to generate daily data in a SWAT model application to West Africa. *Ecological Modelling* 201, 3 (2007), 301–311.

T. Seidl, I. Assent, P. Kranen, R. Krieger, and J. Herrmann. 2009. Indexing density models for incremental learning and anytime classification on data streams. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*. ACM, 311–322.

M. Sekine and K. Sezaki. 2008. Scalable Sensor Data Management under Frequent Content Changes in Peer-to-Peer Network. In *Applications and the Internet, 2008. SAINT 2008. International Symposium on*. IEEE, 133–136.

Z. Shelby, K. Hartke, and C. Bormann. 2014. The constrained application protocol (CoAP). (2014). available at: http://www.rfc-editor.org/rfc/pdfrfc/rfc7252.txt.pdf (last accessed: 22-12-2015).

Z. Sheng, S. Yang, Y. Yu, A. Vasilakos, J. Mccann, and K. Leung. 2013. A survey on the ietf protocol suite for the internet of things: Standards, challenges, and opportunities. *Wireless Communications, IEEE* 20, 6 (2013), 91–98.

A. Sheth, C. Henson, and S. S. Sahoo. 2008. Semantic sensor web. *IEEE Internet Computing* 12, 4 (2008), 78–83.

J. Shieh and E. Keogh. 2008. iSAX: indexing and mining terabyte sized time series. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 623–631.

S. Sigg, D. Gordon, G. von Zengen, M. Beigl, S. Haseloff, and K. David. 2012. Investigation of context prediction accuracy for different context abstraction levels. *IEEE Transactions on Mobile Computing* 11, 6 (2012), 1047–1059.

A. Stanford-Clark and H. L. Truong. 2013. MQTT For Sensor Networks (MQTT-SN) Protocol Specification. (2013). available at: http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf (last accessed: 22-12-2015).

M. Stonebraker, U. Çetintemel, and S. Zdonik. 2005. The 8 requirements of real-time stream processing. *ACM SIGMOD Record* 34, 4 (2005), 42–47.

X. Su, J. Riekki, J. K. Nurminen, J. Nieminen, and M. Koskimies. 2015. Adding semantics to Internet of Things. *Concurrency and Computation: Practice and Experience* 27, 8 (2015), 1844–1860.

C. Sun, D. Stirling, C. Ritz, and C. Sammut. 2012. Variance-wise segmentation for a temporal-adaptive SAX. In *Proceedings of the Tenth Australasian Data Mining Conference*, Vol. 134. Australian Computer Society, Inc., 71–77.

J. B. Tenenbaum, V. De Silva, and J. C. Langford. 2000. A global geometric framework for nonlinear dimensionality reduction. *Science* 290, 5500 (2000), 2319–2323.

S. Thirumuruganathan, N. Zhang, and G. Das. 2013. Rank discovery from web databases. *Proceedings of the Very Large Data Bases (VLDB) Endowment* 6, 13 (2013), 1582–1593.

A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, et al. 2014. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 147–156.

K. Traub, G. Allgair, H. Barthel, L. Burstein, J. Garrett, B. Hogan, B. Rodrigues, S. Sarma, J. Schmidt, C. Schramek, et al. 2005. The EPCglobal architecture framework. *EPCglobal Ratified specification* (2005). available at: http://www.gs1.org/epcglobal (last accessed: 22-12-2015).

E. Troubleyn, J. Hoebeke, I. Moerman, and P. Demeester. 2014. Broadcast Aggregation to Improve Quality of Service in Wireless Sensor Networks. *International Journal of Distributed Sensor Networks* 2014 (2014).

J. Vasseur and A. Dunkels. 2008. IP for smart objects. *IPSO Alliance, White paper* 1 (2008).

S. Vinoski. 2006. Advanced message queuing protocol. *IEEE Internet Computing* 6 (2006), 87–89.

P. Waher. 2015. *Learning Internet of Things.* Packt Publishing.

K. Wähner. 2014. Real-Time Stream Processing as Game Changer in a Big Data World with Hadoop and Data Warehouse. *InfoQ (September 10, 2014)* (2014).

M. E. Wall, A. Rechtsteiner, and L. M. Rocha. 2003. Singular value decomposition and principal component analysis. In *A Practical Approach to Microarray Data Analysis*. Springer, 91–109.

H. Wang, C. C. Tan, and Q. Li. 2010. Snoogle: A search engine for pervasive environments. *IEEE Transactions on Parallel and Distributed Systems* 21, 8 (2010), 1188–1202.

S. Wang, D. Maier, and B. C. Ooi. 2014. Lightweight indexing of observational data in log-structured storage. *Proceedings of the VLDB Endowment* 7, 7 (2014), 529–540.

S. Wang, D. Maier, and B. C. Ooi. 2016. Fast and adaptive indexing of multi-dimensional observational data. *Proceedings of the VLDB Endowment* 9, 14 (2016), 1683–1694.

W. Wang, S. De, G. Cassar, and K. Moessner. 2015a. An experimental study on geospatial indexing for sensor service discovery. *Expert Systems with Applications* 42, 7 (2015), 3528–3538.

W. Wang, F. Yao, S. De, K. Moessner, and Z. Sun. 2015c. A Ranking Method for Sensor Services based on Estimation of Service Access Cost. *Information Sciences* 319 (2015), 1 – 17.

Y. Wang, J.-S. Lee, and I.-C. Choi. 2015b. Indexing by latent dirichlet allocation and an ensemble model. *Journal of the Association for Information Science and Technology* (2015).

Y. Wu and Y. Li. 2009. Distributed indexing and data dissemination in large scale wireless sensor networks. In *Computer Communications and Networks, 2009. ICCCN 2009. Proceedings of 18th Internatonal Conference on.* IEEE, 1–6.

Y.-L. Wu, D. Agrawal, and A. El Abbadi. 2000. A comparison of DFT and DWT based similarity search in time-series databases. In *Proceedings of the ninth International Conference on Information and Knowledge Management.* ACM, 488–495.

Z. Xu, P. Martin, W. Powley, and F. Zulkernine. 2007. Reputation-enhanced QoS-based web services discovery. In *IEEE International Conference on Web Services (ICWS 2007).* IEEE, 249–256.

H. Yang and S. Fong. 2013. Countering the concept-drift problem in Big Data using iOVFDT. In *2013 IEEE International Congress on Big Data (BigData Congress).* IEEE, 126–132.

Y. Yao and J. Gehrke. 2002. The cougar approach to in-network query processing in sensor networks. *ACM SIGMOD Record* 31, 3 (2002), 9–18.

K.-K. Yap, V. Srinivasan, and M. Motani. 2005. MAX: human-centric search of the physical world. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems.* ACM, 166–179.

S. S. Yau and Y. Yin. 2011. Qos-based service ranking and selection for service-based systems. In *Services Computing (SCC), 2011 IEEE International Conference on.* IEEE, 56–63.

B.-K. Yi and C. Faloutsos. 2000. Fast time sequence indexing for arbitrary Lp norms. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB).* Morgan Kaufmann Publishers Inc., 385–394.

Y. Yu, B. Krishnamachari, and V. K. Prasanna. 2004. Issues in designing middleware for wireless sensor networks. *IEEE Network* 18, 1 (2004), 15–21.

K. K. F. Yuen and W. Wang. 2014. Towards a ranking approach for sensor services using primitive cognitive network process. In *2014 IEEE 4th Annual International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER).* IEEE, 344–348.

E. Zámečníková and J. Kreslíková. 2015. Comparison of platforms for high frequency data processing. In *Scientific Conference on Informatics, 2015 IEEE 13th International.* IEEE, 296–301.

D. Zeng, S. Guo, and Z. Cheng. 2011. The web of things: A survey. *Journal of Communications* 6, 6 (2011), 424–438.

D. Zhang, L. T. Yang, and H. Huang. 2011. Searching in internet of things: Vision and challenges. In *Parallel and Distributed Processing with Applications (ISPA), 2011 IEEE 9th International Symposium on.* IEEE, 201–206.

L. Zhang. 2011. Building Facebook Messenger. (2011), 34. available at: http://www.facebook.com/notes/facebook-engineering/buildingfacebook-messenger/10150259350998920 (last accessed: 22-12-2015).

Y. Zhong, J. Fang, and X. Zhao. 2013. VegaIndexer: A Distributed composite index scheme for big spatio-temporal sensor data on cloud. In *2013 IEEE International Geoscience and Remote Sensing Symposium (IGARSS).* 1713–1716.

Y. Zhou, S. De, W. Wang, and K. Moessner. 2014. Enabling Query of Frequently Updated Data from Mobile Sensing Sources. In *Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on.* IEEE, 946–952.

Y. Zhou, S. De, W. Wang, and K. Moessner. 2016. Search Techniques for the Web of Things: A Taxonomy and Survey. *Sensors* 16, 5 (2016), 600.

Q. Zhu, R. Wang, Q. Chen, Y. Liu, and W. Qin. 2010. Iot gateway: Bridgingwireless sensor networks into internet of things. In *IEEE/IFIP 8th International Conference on Embedded and Ubiquitous Computing (EUC).* IEEE, 347–352.

M. Zoumboulakis and G. Roussos. 2007. Escalation: Complex event detection in wireless sensor networks. In *Proceedings of the 2nd European Conference on Smart Sensing and Context.* Springer, 270–285.

K. Zoumpatianos, S. Idreos, and T. Palpanas. 2014. Indexing for interactive exploration of big data series. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. ACM, 1555–1566.

Received February 2007; revised March 2009; accepted June 2009