Ewya: An Interoperable Fog Computing Infrastructure with RDF Stream Processing

Eugene Siow, Thanassis Tiropanis, and Wendy Hall

Electronics & Computer Science, University of Southampton
{eugene.siow,t.tiropanis,wh}@soton.ac.uk

Abstract. Fog computing is an emerging technology for the Internet of Things (IoT) that aims to support processing on resource-constrained distributed nodes in between the sensors and actuators on the ground and compute clusters in the cloud. Fog Computing benefits from low latency, location awareness, mobility, wide-spread deployment and geographical distribution at the edge of the network. However, there is a need to investigate, optimise for and measure the performance, scalability and interoperability of resource-constrained Fog nodes running real-time applications and queries on streaming IoT data before we can realise these benefits. With Eywa, a novel Fog Computing infrastructure, we 1) formally define and implement a means of distribution and control of query workload with an inverse publish-subscribe and push mechanism, 2) show how data can be integrated and made interoperable through organising data as Linked Data in the Resource Description Format (RDF), 3) test if we can improve RDF Stream Processing query performance and scalability over state-of-the-art engines with our approach to query translation and distribution for a published IoT benchmark on resource-constrained nodes and 4) position Fog Computing within the Internet of the Future.

Keywords: Fog Computing, Stream Processing, Interoperability, Internet of Things, Query Translation, Linked Data, Workload Management

1 Introduction

In the science-fiction motion picture Avatar[1], Eywa is the name of a biological internet on the planet Pandora, made up of trees which are distributed over the surface of the planet, that store and process information and memories. The flora and fauna of the planet form an ubiquitous sensor network feeding Eywa.

The Internet of Things (IoT) is growing to become a similar ubiquitous network of sensors and actuators for our planet. Fog computing is an emerging technology which seeks to bridge a gap for the IoT, like the fictional Eywa on Pandora, in between the ground where sensors and actuators are deployed and collect/act on data, and the cloud [12], where larger amounts of resources for processing and storing data can be provisioned dynamically.

[1] http://www.avatarmovie.com/

Some challenges for existing and past proposals of distributed system architectures within the IoT lie in 1) how the heterogeneity of device, platform and data (which is often overlooked at the architecture level) is managed, 2) how multiple streams of data can be processed in a performant, scalable way in real-time and 3) how and what resources can and should be provisioned for these real-time applications. Furthermore, the architecture should also provide a means to manage the *data plane* of the network, e.g. the streams, and the *control plane* of the network, e.g. distribution of application processing on streams.

Bonomi *et al.* [7], in their paper on Fog Computing for the IoT, introduce the defining characteristics and requirements of Fog Computing systems as 1) *interoperability* to support heterogeneous devices and data, 2) *low latency/high performance* for streaming data and real-time applications and 3) distributed infrastructures that have the ability to *scale* horizontally due to the potential wide-spread distribution and large number of resource-constrained Fog nodes.

As Fog Computing extends to the edge of the network, it benefits from low and predictable latency due to data locality, location awareness, mobility, wide-spread deployment and geographical distribution [6]. Furthermore, Fog Computing infrastructures deployed on a network of resource-constrained devices can enhance and support big data processing and analysing data in the cloud by taking advantage of data locality (therefore low latency), mobility and distribution, to provide performant, interoperable and scalable services for data integration and supporting streaming, real-time applications before cloud processing. An example Fog Computing application given by Bonomi *et al.* [6] is a Smart Traffic Light system that has local subsystem latency requirements in the order of <10ms while requiring deep analytics over long-periods in the cloud.

The contribution of this paper is to propose a new Fog Computing architecture, Eywa, and provide the components for stream processing with it, focused on streams and data interoperability and to evaluate it based on the key metrics of interoperability, performance and scalability (measures of a Fog Computing system as defined by Bonomi *et al.* [7]) by:

1. Formally defining and implementing a means of distributing query workload in the *control plane* with an inverse publish-subscribe and push mechanism.
2. Showing how streaming data from heterogeneous devices can be integrated and made interoperable through organising data as Linked Data in the Resource Description Format (RDF) and taking advantage of the shape of time-series IoT data to optimise performance in the *data plane*.
3. Evaluating our approach and framework, by the metrics of latency and scalability, against state-of-the-art RDF Stream Processing engines using a published Smart City benchmark, CityBench [1]. We record improved performance and scalability for real-time applications with streaming queries on inexpensive, mobile, resource-constrained Fog nodes.

The rest of the paper is organised as follows: we formally define a framework for Fog Computing that allows the distribution of stream query workload for real-time applications in Section 2 and go on in Section 3 to introduce Linked Data and the RDF format and how it can be used to integrate and make interoperable

input and output from heterogeneous IoT devices. Next, Section 4 explains RDF stream processing (RSP), how it can be optimised for time-series IoT streams and how its implemented in Eywa. We then evaluate Eywa in Section 5 with results and discussion in Section 6 based on the metrics of performance and scalability. Finally, we argue for the position of Fog Computing within the Internet of the Future as an IoT solution in Section 7 and discuss related work in Section 8.

2 Eywa: An Infrastructure for Fog Computing

The purpose and contribution of this section is to introduce and formally define Eywa, 1) an infrastructural Fog Computing framework where stream processing can be performed on resource-constrained lightweight computer nodes like Raspberry Pis' (RPis'), 2) where the processing workload can be distributed amongst nodes within a inverse-pub-sub *control plane* and 3) each node can maintain its own independence and control over access, resources, security and privacy.

An Eywa network, that forms the basis for the Eywa Fog Computing infrastructure, is explained in Definition 1. The utility of this network for processing streams is to facilitate 1) *stream query delivery* to relevant nodes, 2) *distributed processing* and 3) *results delivery* to requesting nodes. To this end, Definition 2 defines the 3 types of nodes in the network, Section 2.1 explains query delivery, Section 2.2 distributed processing and Section 2.3 results delivery.

Definition 1 (Ewya Network, ε). *An Eywa network, ε, consists of a set of nodes, N and connections, C, such that $\varepsilon = (N, C)$. Each node, $n \in N$, can be a source(s), client(τ) or broker(b) node, such that $n = \{s, \tau, b\}$. Each connection, $c \in C$ exists uniquely between two nodes such that $C \subseteq N \times N$.*

Definition 2 (Source, s, Client, τ and Broker, b nodes). *Given the set of source nodes, S, client nodes, T, and broker nodes, B, where $N = S \cup T \cup B$. A source node, $s \in S$, is a node that produces a set of time-series streams, Γ. A client node, $\tau \in T$ is a node with a set of queries Q expecting a set of corresponding results, R. A broker node, $b \in B$, establishes a connection, c, for new source and client nodes to enter the network.*

As *source* and *client* nodes form up in a Eywa network, *broker* nodes 1) provide a point of entry for new nodes into the network, 2) not store or process but forward data, 3) consume minimal resources and 4) employ redundancy (multiple separate instances) within the network so as not be become single points of failure. Hence, facilitating both the formation and data flow in Eywa.

2.1 Stream Query Delivery by Inverse-Publish-Subscribe

Once an Eywa network has been formed, stream processing can take place as clients (τ) issue stream queries (Q). Traditionally, source nodes (s) publish data while client nodes subscribe to data. However, in Eywa, it is desirable for clients

to collaborate with the sources to share the workload, hence, we propose an inverse-publish-subscribe mechanism for query delivery.

Each *source* node subscribes to a topic for each of its streams. *Client* nodes then publish queries to the relevant topics. Uniform Resource Identifiers (URIs), proven to work for the web, are used to provide a means of uniquely identifying and exchanging topic names. Definition 3 formally describes the mechanism.

Definition 3 (Inverse-publish-subscribe). *Given the set of topics, M, a source node, s, subscribes to a topic, $\mu \in M$ for each stream within Γ to form $\bigcup_{\mu \in M_\Gamma} sub(\mu)$, where M_Γ is the set of all topics of s and $sub(\mu)$ is a function that produces a subscription to μ. For each query, $q \in Q$, from a client node, τ, a distribution function, α, builds a set of query-topic pairs $\alpha(q) = (Q_\mu, M_q)$ where Q_μ is the set of all sub-queries in q, each referencing a particular topic μ and M_q is the set of all topics referenced in q. Each sub-query, $q_\mu \in Q_\mu$ is published to it's particular topic $\mu_q \in M_q$ by the publish function of query x to topic y, $pub(x, y)$, so all publications from q is represented by $\bigcup pub(q_\mu, \mu_q)$.*

2.2 Distributed Processing

Source nodes receive the queries, perform part of the processing and deliver the results to *clients* that process the results. This forms the axis of client-source collaboration and distributing processing workload as defined in Definition 4. *Source* nodes control their own resources and response to queries. Quality of service is not in the scope of this work but can be configured for best effort, service-level agreements, trustless networks or consensus protocols.

Definition 4 (Distributed Source Node Processing). *A source node, s, receives a query, q, for a topic, μ, and converts it into a work function, ω with a conversion function, $\lambda(q) \to \omega$. The work function, ω is applied to the corresponding stream for topic μ, γ_μ, where $\gamma_\mu \in \Gamma$, so that $\omega(\gamma_\mu) = \gamma_r$ and the resulting stream, γ_r, is pushed to the requesting client, τ.*

2.3 Push Results Delivery and Sequence Diagram

Client nodes receive results streams via a direct push from *source* nodes. Operations involving multiple streams, like aggregations are performed on the clients and results of the queries are published to topics as output to applications. Definition 5 details the process of results delivery and query output.

Definition 5 (Push Results Delivery). *A client node, τ, receives a set of result streams, Γ_r, by push delivery. For each query, $q \in Q$, in the set of queries for that client, a work function, ω_τ is produced by $\lambda_\tau(q) \to \omega_\tau$ and executed on $\bigcup \gamma_r$, where γ_r are all the result streams corresponding to the query, q. The result, γ_q, from $\omega_\tau(\bigcup \gamma_r) = \gamma_q$ is published to a client results topic, μ_τ.*

Fig. 1 shows the full sequence diagram of the query processing process beginning from the *source* nodes subscribing to topic URIs, receiving queries when published by *client* nodes, distributed processing and result delivery. Table 1 shows a glossary of symbols used and their corresponding definitions.

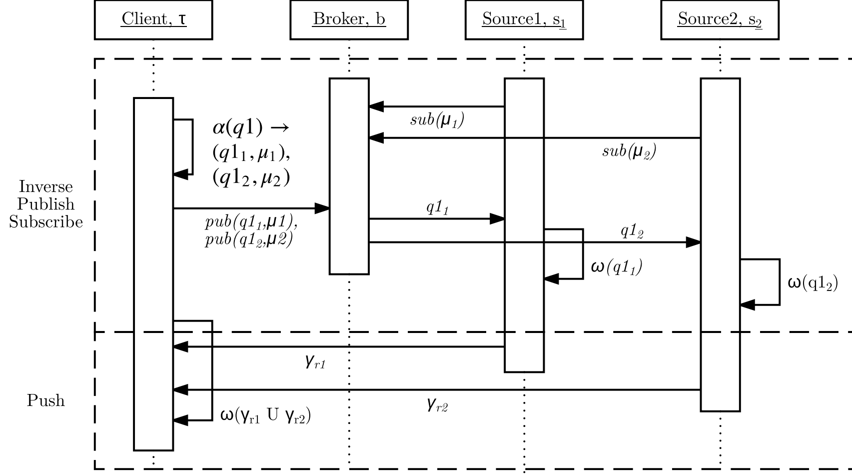

Fig. 1. Sequence Diagram of Stream Query Processing in Eywa

Table 1. Glossary of special symbols and their Definitions

	Description	Definition(s)
τ	Client node	2
μ	Topic (Uniformed Resource Identifier)	3
γ, Γ	Stream, Set of streams	3, 4, 5
$\alpha()$	Distribution function, build query-topic pairs	4
$\lambda()$	Conversion function, converts query to work function	4, 5
$\omega()$	Work function, applied to streams	4, 5

3 Linked Data for Interoperability

The Internet of Things (IoT) is currently beset by product silos and to unlock its potential, an open ecosystem based upon open standards for identification, discovery and interoperation of services is required[2]. As previously noted, interoperability is an important metric for Eywa. To provide for interoperability, we consider Linked Data, a means of publishing data on the Web so that distributed structured data can be interconnected, exchanged and retrieved through semantic queries [5]. A common way of representing Linked Data is with the Resource Description Format (RDF) where the use and referencing of common identifiers and ontologies helps to integrate data. Furthermore, sensor ontologies like the Semantic Sensor Ontology (SSN) [9] have been developed for Linked Data which encourages providing machine-interpretable descriptions within RDF to describe what data represents, where it originates from, how it can be related to its surroundings, who is providing it, and what its attributes are e.g. a unit

[2] https://www.w3.org/2014/12/wot-ig-charter.html

of measure for each sensor reading, its sensor platform and location. Barnaghi *et al.* [3] further support the view that Linked Data is a means of connecting and integrating cyber, social or physical world data in the IoT.

RDF documents are composed of tuples of triples formed from a subject, predicate and object. For example, in the statement 'sensor1 has weatherObservation1', the subject is *sensor1*, the predicate is *has* and the object is *weatherObservation1*. 'weatherObservation1 hasValue 30knots' is another triple and the set of these triples forms an RDF graph in which we know sensor1 had an observation with a value of 30knots. This is formally expressed in Definition 6.

Definition 6 (RDF triple, t and RDF graph, G). *t is a tuple $(s, p, o) \in (IB) \times I \times (IBL)$ where s, p and o are subject, predicate and object respectively. I, B and L are disjoint infinite sets of Internationalised Resource Identifiers, blank nodes and literals respectively. G is a set of triples such that $G = \{x : x \in t\}$.*

The use of Internationalised Resource Identifiers (IRIs), URIs that support unicode characters, like 'http://purl.oclc.org/NET/sao/hasValue', enables resources to be uniquely identified and referenced. Blank node's are anonymous resources without an IRI and literals are values like strings and dates. SPARQL is a language used for querying RDF.

Hence, Linked Data provides IoT interoperability as 1) RDF graphs allow structured data to be interconnected with IRIs providing common identifiers that can be referenced across sources, 2) SPARQL provides a means for semantically querying RDF graphs and 3) there exist many ontologies like the SSN Ontology [9] that provide a reference and model for organising sensor data.

4 RDF Stream Processing (RSP) for the IoT

Stankovic [18] explains that the IoT will increasingly be composed of "a very large number of real-time sensor data stream" and that a "given stream of data will be used in many different ways". RDF stream processing (RSP) is the area of work which enables Linked Data to be produced, transmitted and continuously queried as RDF streams[3]. RSP enables us to take advantage of the interoperability of RDF on streams while preserving the power of semantic queries. However, as Buil-Aranda *et al.* [8] have shown with Linked Data on the web, performance is an issue especially when dealing with a series of non-trivial queries.

We have previously surveyed about 20,000 unique IoT schema from public IoT data streams and discovered that a large majority of the sampled devices had *flat* schemata and *wide* schemata (99.5% and 76.3% respectively) [17]. Flat schemata have no nested layers (table-like rather than tree-like) and wide schemata have more than one property besides the timestamp.

We then looked at the ontologies for integrating time-series sensor data in RDF and observed that linked sensor data is produced from these ontologies as 1) *device metadata* like the location and specifications of sensors, 2) *observation*

[3] https://www.w3.org/community/rsp/

metadata like the units of measure and types of observation and 3) *observation data* like timestamps and actual readings. A large portion of the triples produced consisted observation metadata which was repetitive and made up of IRIs with additional 128-bit universally unique identifier strings. Our results showed across various scenarios that this data was often redundant and not used eventually.

It follows that Eywa's RSP for time-series streams can be optimised to:

1. Store *observation data* succinctly in flat and wide rows instead of graphs.
2. Abstract the small set of *device metadata* to store as RDF mappings.
3. Compress *observation metadata* as bindings and only materialise if needed.
4. Distribute part of a query to be applied on streams on a Eywa source node which can reduce the bandwidth required and share the workload.

When an RSP query is registered, it is translated with reference to the metadata expressed in mappings, part of the query is distributed to the relevant source streams and the rest of the query is applied on the subset of stream data received on the client side as a continuous query. Any additional metadata is materialised and added to the results. We walkthrough an example query from a smart city scenario published in CityBench [1], a published benchmarking suite for streaming applications on smart city data gathered from IoT sensors deployed within the city of Aarhus in Denmark from February 2014 to November 2015.

Table 2 shows a weather observation from a CityBench stream. This is a stream of actual flat and wide time-series observation data and consists of humidity, temperature and wind speed readings connected to a timestamp. A source node with this stream subscribes to the URI 'http://...#AarhusWeatherData0'.

Table 2. CityBench: Observation from AarhusWeatherData0 Stream

timestamp	hum	tempm	wspdm
2014-08-01T00:00:00	56.0	18.0	7.4

Listing 1.1 is the corresponding RDF mapping that stores the sensor and observation metadata of the weather data stream. It also contains bindings to fields from the underlying stream data e.g. 'AarhusWeatherData0.tempm'.

Listing 1.1. CityBench AarhusWeatherData0 RDF Mapping (abbreviated)

```
@prefix ssn:<http://purl.oclc.org/NET/ssnx/ssn#>
@prefix sao:<http://purl.oclc.org/NET/sao/>
@prefix ct:<http://.../citytraffic#>
@prefix ns:<http://../SampleEventService#>
@prefix iot:<http://iot.soton.ac.uk/s2s/s2sml#>
_:obs1 a ssn:Observation;
  ssn:observedProperty ns:Property-1;
  sao:hasValue "AarhusWeatherData0.tempm"^^iot:literalMap;
  ssn:observedBy ns:AarhusWeatherData0.
ns:Property-1 a ct:Temperature.
_:obs2 a ssn:Observation;
  ssn:observedProperty ns:Property-2;
  sao:hasValue "AarhusWeatherData0.hum"^^iot:literalMap;
  ssn:observedBy ns:AarhusWeatherData0.
ns:Property-2 a ct:Humidity.
_:obs3 a ssn:Observation;
  ssn:observedProperty ns:Property-3;
  sao:hasValue "AarhusWeatherData0.wspdm"^^iot:literalMap;
  ssn:observedBy ns:AarhusWeatherData0.
ns:Property-3 a ct:WindSpeed.
```

This mapping references various ontologies like the Semantic Sensor Network (SSN) Ontology[4] and City Traffic Ontology, providing a common way of describing sensor data, increasing interoperability. A formal definition of the mapping language is covered in our previous paper [16] and as a specification[5].

Similarly, a mapping is available for the stream of traffic data at various locations in the city. The traffic stream consists of fields like *avgSpeed* and *congestionLevel* connected to a *timestamp*.

Listing 1.2 shows Query 2 of CityBench expressed in the W3C recommended RSP-QL syntax for RSP engines[6], that at the time of writing no other engines support yet. When registered, it processes both traffic and weather streams for observations of traffic congestion level and weather (e.g. temperature), from a particular stretch of road, for the last 3 seconds. The semantic expressivity of the RDF graph query provides for interoperability, however, the underlying values *v1* to *v4* are actually from just 2 fields from each flattened stream.

Listing 1.2. CityBench Query 2: Finding the traffic congestion level and weather conditions of my planned journey (abbreviated)

```
SELECT ?v1 ?v2 ?v3 ?v4
FROM NAMED WINDOW :traffic ON <http://...#AarhusTrafficData158505> [RANGE PT3S]
FROM NAMED WINDOW :weather ON <http://...#AarhusWeatherData0> [RANGE PT3S]
WHERE {
  WINDOW :weather {
    ?obId1 a ssn:Observation;
     ssn:observedProperty ?p1;
     sao:hasValue ?v1;
     ssn:observedBy ns:AarhusWeatherData0.
    ?p1 a ct:Temperature. ... }
  WINDOW :traffic {
    ?obId4 a ssn:Observation;
     ssn:observedProperty ?p4;
     sao:hasValue ?v4;
     ssn:observedBy ns:AarhusTrafficData158505.
    ?p4 a ct:CongestionLevel. } }
```

This query is then translated using the RDF mappings as explained in Section 4.1, distributed by Eywa's inverse-pub-sub, processed and returned to the client for processing as in 4.2 and are streamed to downstream applications. The entire architecture of Eywa's RSP engine is summarised in 4.3.

4.1 Query Translation

Firstly, the RSP-QL query is broken down into algebra as in Diagram 1.1.

The algebra tree is traversed from leaf to root. At the $Window_{:weather}$ node, the basic graph pattern (BGP), $BGP_{weather}$, which compromises:

```
?obId1 a ssn:Observation;
  ssn:observedProperty ?p1;
  sao:hasValue ?v1;
  ssn:observedBy ns:AarhusWeatherData0.
?p1 a ct:Temperature. (...humidity and wind speed parts)
```

is matched against the *AarhusWeatherData0* RDF mapping from Listing 1.1. This matching can be done by any in-memory SPARQL engine and in our im-

[4] http://www.w3.org/TR/vocab-ssn/
[5] https://github.com/eugenesiow/sparql2sql/wiki/S2SML
[6] https://www.w3.org/community/rsp/

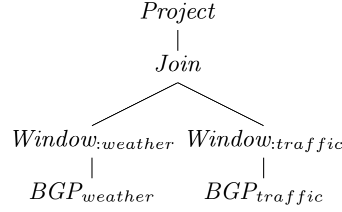

Diagram 1.1. Algebra of CityBench Query 2 from Listing 1.2

plementation we provide interfaces for popular open source engines[7]. The results of this step includes a result set bindings variable *?v1* from $BGP_{weather}$ to the field *AarhusWeatherData0.tempm* in the stream and so forth. Similarly, from $Window_{:traffic}$ we retrieve the binding of variable *?v4* to *AarhusTrafficData158505.congestionLevel* from the result set.

At the *Join* node, since no variables overlap from the *weather* and *traffic* windows, the result sets are passed upwards to the *Project* node.

At the *Project* node, variables $?v1$ to $?v4$ are projected. Any variable renaming/aliases will be taken care of in the projection as well. The simplified stream query expressed in Event Processing Language (EPL)[8] is shown in Listing 1.3:

Listing 1.3. CityBench Query 2 simplified and translated to EPL

```
SELECT AarhusWeatherData0.tempm AS v1 ,
  AarhusWeatherData0.hum AS v2 ,
  AarhusWeatherData0.wspdm AS v3 ,
  AarhusTrafficData158505.congestionLevel AS v4
FROM AarhusWeatherData0.win:time(3 sec) ,
  AarhusTrafficData158505.win:time(3 sec)
```

Query 5 which discovers the traffic congestion level on the road where a given cultural event is happening[9] has algebra (in Diagram 1.2):

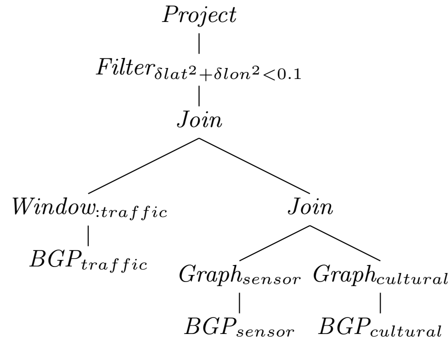

Diagram 1.2. Algebra of CityBench Query 5

$Window_{:traffic}$ produces a similar result set as in Query 2. However, $Graph_{sensor}$ and $Graph_{cultural}$ reference a static/graph source instead of a stream, hence the

[7] https://github.com/eugenesiow/sparql2sql/wiki/SWIBRE
[8] http://www.espertech.com/products/esper.php
[9] https://github.com/eugenesiow/Benchmark/wiki/Q05

simplified translated syntax produced is SQL instead of EPL, though the method of executing against an RDF mapping is the same e.g. $Graph_{cultural}$ produces the following SQL when translated against the *AarhusCulturalEvents* mapping:

```
SELECT title, lat, lon FROM AarhusCulturalEvents
```

So at the first *Join* in the tree from the leaf, the SQL is integrated into the FROM clause of the translated EPL statements produced:

```
FROM sql:AarhusCulturalEvents [ 'SELECT title, lat, lon
    FROM AarhusCulturalEvents' ] AS AarhusCulturalEsvents ,
sql:SensorRepository [ 'SELECT lat, lon FROM SensorRepository' ] AS SensorRepository
```

At the next *Join* above, as there is a variable *?p2* present in both inbound nodes, a join is performed between the RDF metadata from $Window_{:traffic}$ and the previous *Join*, so the SQL within becomes (note the extra WHERE clause):

```
... [ 'SELECT lat, lon FROM SensorRepository
    WHERE propId=\'Property-b9f9...\'' ] AS SensorRepository,
AarhusTrafficData158505.win:time(3 sec)
```

Finally, a filter on the addition of the square delta of latitude and longitude constrains the traffic sensor and cultural event locations to within 0.1 units. All queries and corresponding translations can be found on our engine's wiki[10].

4.2 Query Distribution

Query distribution is the process whereby part of a query workload, ω_τ, is distributed from the client (where the query is registered and the results are expected), to the source node (where the data is produced or stored).

For example, for Query 2 in CityBench (Listing 1.2), at the *Project* operator at the top of the algebra tree, the engine tracks that *hum*, *tempm* and *wspdm* are the fields required from $Window_{:weather}$ while only *congestionLevel* is required from $Window_{:traffic}$. Hence, the projection of these fields is the work function, ω_τ, pushed to each source node producing the traffic and weather streams. The projection tree for the streams in Query 2 are shown below in Diagram 1.3.

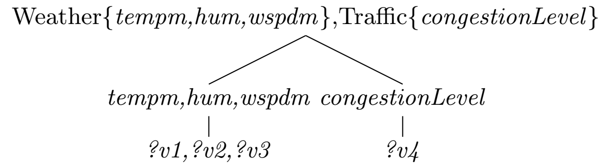

Diagram 1.3. Projection tree for Query 2 streams

Following RSP Query Translation (Section 4.1), the resulting simplified, translated EPL query is registered on the client. The root of the projection tree is distributed through Eywa's inverse-pub-sub mechanism to the subscribing source nodes of the IRIs of $Window_{:weather}$ and $Window_{:traffic}$. Hence, each source node pushes only an effective subset of each stream, γ_q, with only the projected fields to the client which further processes the EPL stream.

[10] https://github.com/eugenesiow/sparql2sql/wiki

For static sources, the work function, ω_τ, also comprises SQL queries distributed to the source nodes. In Query 5 of CityBench, an SQL query:

```
SELECT lat , lon FROM SensorRepository WHERE propId=\'Property-b9f9...\'
```

is executed on the static *SensorRepository* source node, returning results to the client using a Java Database Connectivity (JDBC) connection.

4.3 Eywa's RSP Architecture

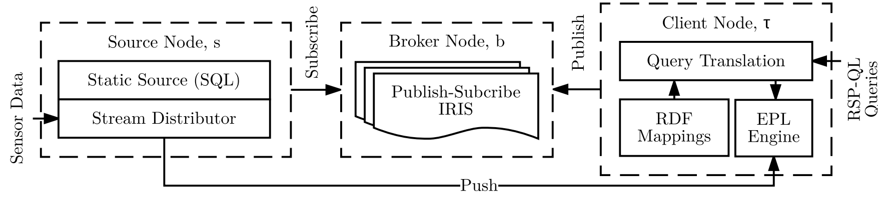

Fig. 2. Architecture of Eywa's RSP Fog Computing framework

Fig. 2 shows the architecture of Eywa's RSP implementation consisting *client, source* and *broker* nodes. The *brokers* use lightweight ZeroMQ sockets and binary transmission protocol with minimal overhead to support publish-subscribe IRI topics. *Client* nodes receive queries in RSP-QL and projected streams via push from *source* nodes. They process complex events in queries using an EPL engine that registers translated queries with RDF mappings. *Source* nodes produce streams of time-series IoT data or store static data, e.g. sensor location data, cultural event data. A stream distributor component pushes projected streams, γ_q, to client nodes as required. Static queries utilise JDBC connections.

5 Experimentation Evaluating Eywa

The experiment uses the Smart City benchmark for Linked Data, CityBench [1], that features real-time datasets (e.g. vehicle traffic, parking, weather, pollution, etc.) from the city of Aarhus and streaming queries based on smart city application requirements (e.g. parking space finder, admin console). Linked Data goes towards fulfilling the interoperability metric and hence we evaluate Eywa's RSP framework by latency (performance) and scalability metrics.

The streaming and static source nodes, client nodes and broker nodes used resource-constrained lightweight computers, in the form of Raspberry Pi 2 Model B+s', suitable for inexpensive, mobile and broad Eywa Fog Computing networks. Each had 1GB RAM, a 900MHz quad-core ARM Cortex-A7 CPU and Class 10 SD Cards. 256mb (the recommended and default 1/4th of system memory) was

assigned to the Java Virtual Machine on Raspbian 4.1. Ethernet connections were used between the nodes for reliable transport.

We compared our approach, Eywa RSP, against state-of-the-art native RSP streaming engines CQELS [11] and C-SPARQL [2]. For each query, we varied the amount of concurrent queries and the number of data streams.

For each query and experimental configuration, we tested each engine for the latency and memory consumption. Tests were run for 15 minutes and averaged over 3 runs. The goal was to measure the performance (latency of queries) and scalability (the memory consumption while varying the number of data streams and concurrent queries). Additionally, to measure the benefits in scalability of our Fog Computing infrastructure, Eywa-RSP was tested and compared with client, source and broker on a single node against when they were across multiple nodes (3 nodes) with workload distribution.

6 Results & Discussion

6.1 Latency Evaluation

The latency of a query refers to the average time consumed by the engine between when a stream observation arrives and when the results from the query output are generated. Nodes are synchronised using the Network Time Protocol (NTP).

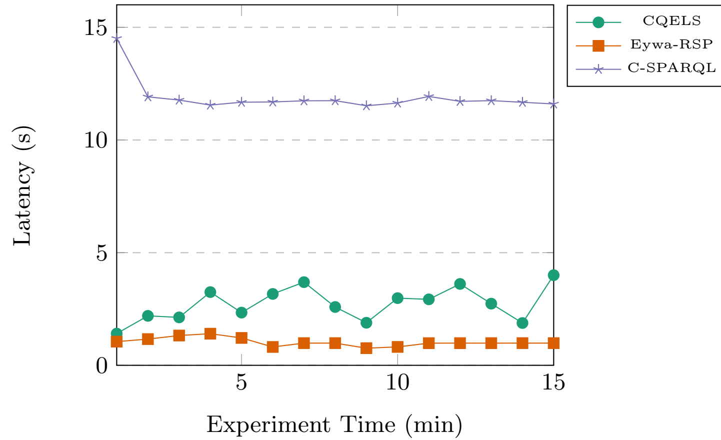

Fig. 3. Latency of CityBench Q1 across time

Fig. 3 shows the latency over time of each of the engines for Query 1[11] which measures the traffic congestion level on two roads. As we can see, Eywa-RSP has the lowest latency and hence best query performance. All other queries

[11] https://github.com/eugenesiow/Benchmark/wiki/Q01

show similar results, with Eywa having the lowest latency over time and are summarised by averaging over the 15 minute interval as shown in Fig. 4.

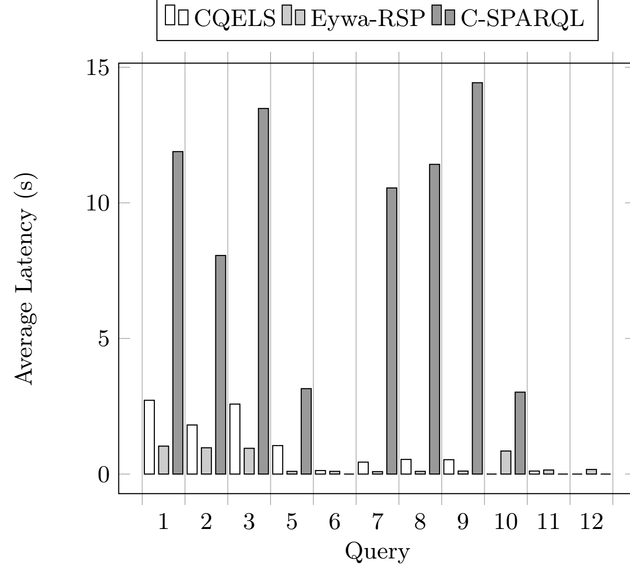

Fig. 4. Average latency of CityBench Queries on RSP Engines

Bars with zero values are queries that cannot be run on that engine (Eywa:4, CQELS:4,10,12, C-SPARQL:4,6,11,12)

The reason that Eywa has a lower latency than CQELS and C-SPARQL is because it abstracts metadata triples to mappings, which are processed in the query translation process as opposed to in the stream itself.

Diagram 1.4 shows the algebra of the translated query 1 from CityBench, registered on the client node. II is the operator for retrieving the projected columns (e.g. congestionLevel) from each event in the stream window.

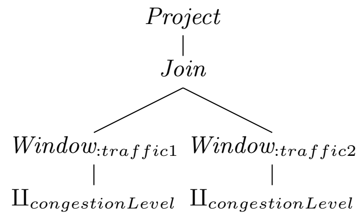

Diagram 1.4. Algebra of Query 1 for Eywa-RSP

The algebra for CQELS and C-SPARQL on the other hand, as shown in Diagram 1.5, requires the retrieval of, τ_{BGP}, for each event in the stream. The

extra triples are shown in Listing 1.4, representing the observation metadata (lines 1 to 3) and relation to sensor metadata (line 4).

Listing 1.4. Additional metadata triples

```
?obId1 a ?ob.
?obId1 ssn:observedProperty ?p1.
?obId1 sao:hasValue ?v1.
?obId1 ssn:observedBy <...#AarhusTrafficData182955>.
```

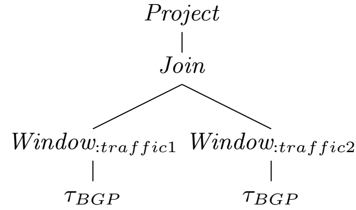

Diagram 1.5. Algebra of Query 1 for CQELS and C-SPARQL

It is more expensive in terms of overall latency when retrieving the extra triples and processing the query without workload distribution at the source. Hence, Eywa-RSP proves faster than the other tested engines.

6.2 Scalability Evaluation

Scalability evaluation looks at the amount of memory resources used by each RSP engine and the results when increasing the number of concurrent queries and increasing the number of data streams. The lower the memory resources used, the more scalable the system is, especially on resource-constrained nodes.

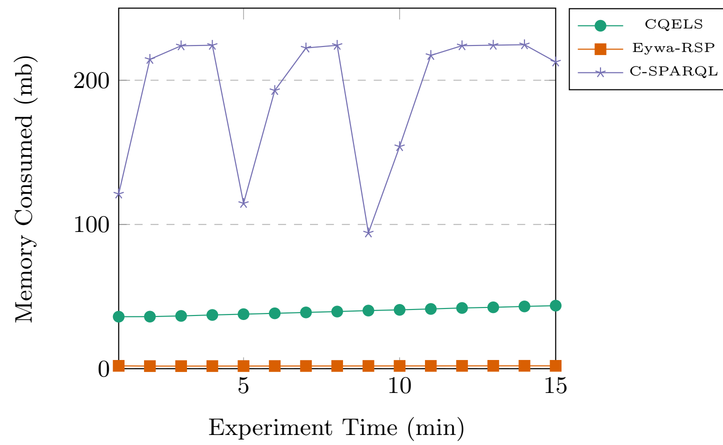

Fig. 5. Memory Consumed by CityBench Query 2 across time on client

Fig. 5 shows the memory consumption of each engine across time. Eywa-RSP streams a more concise format of just time-series data and only the necessary projected fields from the source node, as compared to the verbose set of triples, in Listing 1.5, for other engines. This is consistent for other queries in CityBench with the more complex Q5, a filter operation on 2 graphs and a stream, taking the most memory and Q11, checking for observations from weather sensors on a single stream, the least on all engines.

Listing 1.5. Triples from a row/event of the Traffic Stream

```
:Property1 a ct:CongestionLevel.
:Observation1 a ssn:Observation;
  ssn:observedProperty :Property1;
  sao:hasValue "congestionLevel";
  ssn:observedBy ns:AarhusTrafficData158505;
  time:inXsdDateTime "timestamp".
:Observation2 a ssn:Observation;
  ssn:observedProperty :Property2;
  sao:hasValue "avgSpeed";
  ssn:observedBy ns:AarhusTrafficData158505;
  time:inXsdDateTime "timestamp".
```

As expected, when increasing the number of concurrent queries on each engine, Eywa-RSP once again achieved the lowest memory consumption. The memory consumed was also the only one consistent and did not increase over time like C-SPARQL and CQELS. This can be seen from Fig. 6 which shows the results of Query 5 at two different configurations of a single query and 20 concurrent queries. There were consistent results from these 2 configurations for other CityBench queries. Q2 and Q8 took up significantly more memory for C-SPARQL, Q1 and Q3 took up significantly more memory for CQELS while other queries took slightly more memory (about 10MB). Eywa-RSP had a slight, stable 2-5MB increase in memory consumption on increasing concurrent queries.

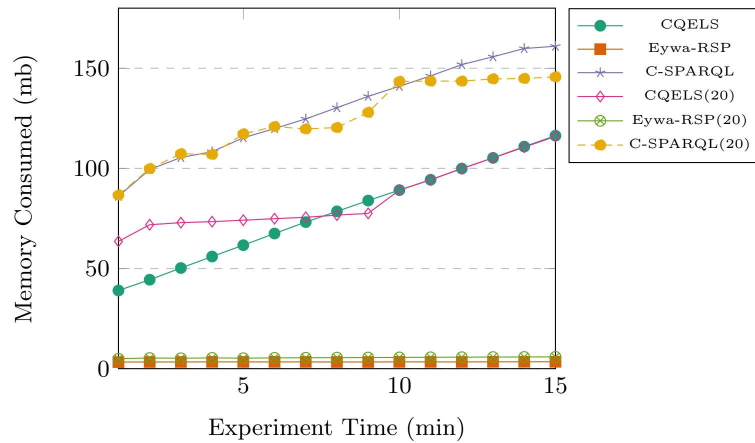

Fig. 6. Memory Consumed when increasing # Concurrent Queries on CityBench Q5

Fig. 7 shows the memory consumption when increasing the number of pollution data streams from 2 to 5 in CityBench's query 10 which looks for the most polluted area in the city in real-time[12]. Eywa-RSP once again has the lowest memory consumption and C-SPARQL actually runs out of memory on the resource-constrained client in the 5 stream configuration just before 15 minutes.

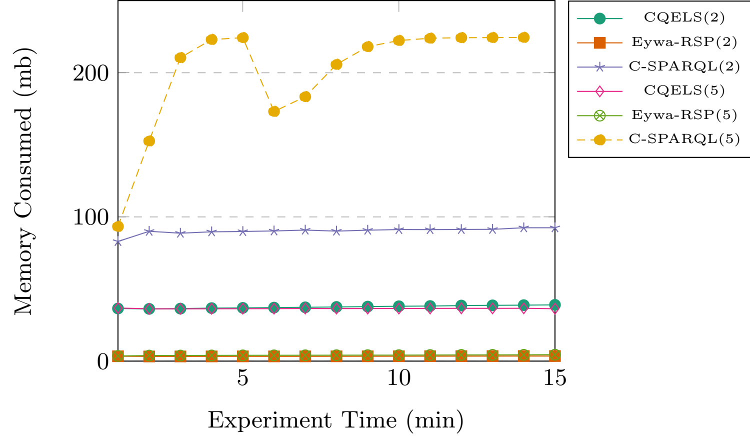

Fig. 7. Memory Consumed by CityBench Q10 with varying data stream configurations

Finally, we want to observe how much the Eywa fog computing framework, through the projection operator on the source node, actually improves the memory consumption and scalability across an increasing amount of data streams. Fig. 8 shows the amount of memory consumed by the distributed fog execution of the query and the single node execution of the query.

As the projection operator is distributed to the source nodes and only the applicable part of the stream is sent to the client, there is a significant difference in memory consumed when the number of incoming data streams are increased. At 2 and 5 streams for query 10, the memory consumed is similar for fog and single-node (non-fog) setups. When there were 8 streams on query 10, however, the fog computing approach consumed less memory. This is due to the flow of data being significantly large enough that the projection operation passed down to the source node of each stream also has a significant effect on the overall memory consumed. Hence, as the amount of streams increase, the Eywa's Fog Infrastructure provides greater scalability. Evaluation experiments forked from the CityBench benchmark harness are available on Github[13].

[12] https://github.com/eugenesiow/Benchmark/wiki/Q10
[13] https://github.com/eugenesiow/Benchmark

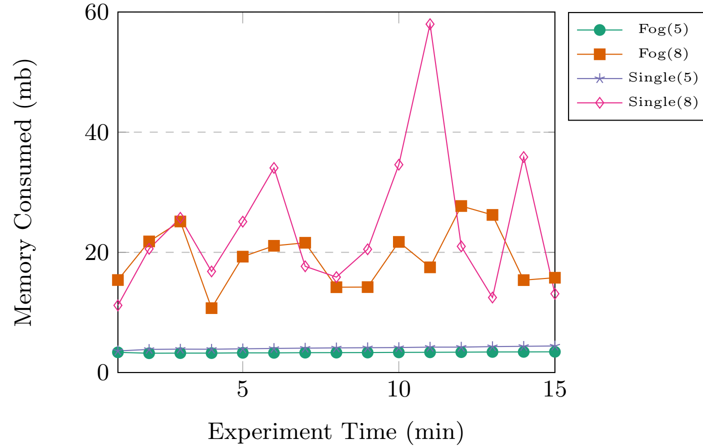

Fig. 8. Fog vs Single-Node: Memory Consumed while increasing streams on Q10

7 Fog Computing within the Internet of the Future

The Internet has been a great motor of socio-economic activity in the past decade and the Internet of the Future has the potential to do even more. Like the fictional Eywa in Avatar, Eywa as a Fog Computing infrastructure has the potential to make the IoT and the planet more intelligent, more connected and at the same time more engaged and social while valuing privacy and security.

Learning from the web, that grew as a free, open and standards-based information space on top of the internet [4], the IoT needs to endeavour to support interoperability. Linked Data not only encourages interoperation through the use of URIs and common ontologies, it also represents concepts as machine-readable graphs [5] which goes towards solving challenges like discovery and data integration in the IoT. Fog Computing that serves as a layer between sensors and actuators and the cloud and possesses comparatively more compute and storage than sensors, serves as the entry level of interoperability and integration.

The current business model for many web services is that free content is exchanged for our personal data. However, as Berners-Lee states, 'as our data is held in proprietary silos, out of sight to us, we lose out on the benefits we could realise if we had direct control over this data and chose when and with whom to share it[14]'. Fog Computing on lightweight, distributed computers means that data that is collected from sensors and devices are stored and processed locally. As access controls evolve, specific privacy policies and access control with additional trust and fault tolerance mechanisms can be created [15].

Furthermore, as data is stored and processed by some applications locally, we minimise the need to shuttle data to and from the cloud, there is more quality

[14] https://www.theguardian.com/technology/2017/mar/11/tim-berners-lee-web-inventor-save-internet

of service guarantee for mission-critical IoT apps and there is less dependency on supporting high bandwidth global connections. In disaster management IoT scenarios, where last-mile connectivity is lost, having data locality and offline access is especially valuable. There are also performance benefits over storing encrypted data in traditional clouds as a means to maintain privacy because it is easier to perform processing (no need for crypto-processors or to apply special encryption functions) over data. When necessary and access controls permit, data can still be sent to the cloud for big data processing.

Finally, as the Internet and IoT advance, studying the social-technical aspects of the intersection between intelligent, cooperative autonomous machines in the IoT and human users is gaining importance. Fog Computing, as a distributed, interoperable application layer between the network, autonomous IoT end-devices, applications and human end-users can serve to 1) build and manage this social network, 2) facilitate information flow/sharing, 3) host applications and 4) serve as an observation platform for the study of these emerging networks.

Hence, we argue for the importance of research in Fog Computing technologies as part of the future, next-generation internet.

8 Related Work

Bonomi *et al.* [6] introduce Fog Computing as a platform for the Internet of Things, its defining principles, its utility for real-time streaming applications and the concept of *distributed orchestration*. These principles provide guidelines for the formal design and implementation of our framework which registers queries on streams and pushes results to real-time applications while introducing a novel inverse publish-subscribe model for workload management and orchestration.

In Wireless Sensor Networks literature, identifies base station nodes as "powerful devices with PC-like capabilities" which support the idea of Eywa Fog Computing nodes and are already widely-deployed. The innovation of a Fog Computing framework should come from managing the workload on and optimising performance of these nodes.

There has been previous work on RSP with the C-SPARQL [2] engine that supports continuous pull-based queries over RDF data streams by using Esper[15], a complex event processing engine, to form windows in which SPARQL queries can be executed on an in-memory RDF model. Another engine is CQELS [11], which is a purely native RSP, supporting both push and pull queries. Due to the 'white-box' approach, there is full control over query optimisation and execution. We compare against both of these engines.

Efficient SPARQL-to-SQL translation has been investigated by Rodriguez-Muro *et al.* [14] and Priyatna *et al.* [13]. The state-of-the-art reduces redundant self-joins and applies query containment and semantic query optimisation to translations. However, neither of the engines are designed to work on IoT streams or for Fog Computing scenarios yet.

[15] http://www.espertech.com/products/esper.php

There are a few benchmarks on streaming Linked Data including: SRBench [19], which in our previous work we have evaluated on [17], CityBench [1], which we compare against and LSBench [10] comprising social network stream data.

9 Conclusion and Future Work

The Internet of Things has huge research potential and Fog Computing, we argue, that is implementing a layer between the 'ground' and the 'cloud', can benefit from efficient, interoperable RDF stream processing (RSP) to support real-time applications on lightweight computers and networks. This does not replace, but seeks to complement large-scale processing and analytics in the cloud by providing collection, integration and continuous querying capabilities across distributed nodes. We go on to formally define and implement a Fog Computing infrastructure, Eywa, that utilises inverse-publish-subscribe and push as a distribution mechanism together with a query translation approach to RSP optimised for IoT time series data. In evaluation benchmarks, Eywa showed better performance and scalability as compared to state-of-the-art RSP approaches while preserving the interoperability of Linked Data.

Furthermore, with specialised distribution mechanisms for Fog Computing, we can maintain control, security and privacy on a per node level.

There is also potential to explore query distribution that pushes down a range of operators to the source nodes, Quality of Service guarantees and Service Level Agreements for source nodes and consensus algorithms for streams and sensors.

References

1. Ali, M.I., Gao, F., Mileo, A.: CityBench: A configurable benchmark to evaluate RSP engines using smart city datasets. Lecture Notes in Computer Science (2015)
2. Barbieri, D.F., Braga, D., Ceri, S., Valle, E.D., Grossniklaus, M.: Querying RDF streams with C-SPARQL. ACM SIGMOD Record 39(1), 20 (2010)
3. Barnaghi, P., Wang, W., Henson, C., Taylor, K.: Semantics for the Internet of Things: early progress and back to the future. International Journal on Semantic Web and Information Systems 8(1), 1–21 (2012)
4. Berners-Lee, T., Fischetti, M., Foreword By-Dertouzos, M.L.: Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor. HarperInformation (2000)
5. Bizer, C., Heath, T., Berners-Lee, T.: Linked Data - The Story So Far. International Journal on Semantic Web and Information Systems 5, 1–22 (2009)
6. Bonomi, F., Milito, R., Natarajan, P., Zhu, J.: Fog Computing: A Platform for Internet of Things and Analytics, vol. 546 (2014)
7. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog Computing and Its Role in the Internet of Things. Proceedings of the first edition of the MCC workshop on Mobile cloud computing pp. 13–16 (2012)
8. Buil-Aranda, C., Hogan, A.: SPARQL Web-Querying Infrastructure: Ready for Action? In: Proceedings of the International Semantic Web Conference (2013)

9. Compton, M., Barnaghi, P., Bermudez, L., García-Castro, R., Corcho, O., Cox, S., Graybeal, J., Hauswirth, M., Henson, C., Herzog, A., Huang, V., Janowicz, K., Kelsey, W.D., Le Phuoc, D., Lefort, L., Leggieri, M., Neuhaus, H., Nikolov, A., Page, K., Assant, A., Sheth, A.: The SSN ontology of the W3C semantic sensor network incubator group. Journal of Web Semantics 17, 25–32 (2012)
10. Danh, L.P., Minh, D.T., Pham, M.D., Boncz, P., Eiter, T., Michael Fink: Linked Stream Data Processing: Facts And Figures. Proceedings of International Semantic Web Conference 2012 (2012)
11. Le-Phuoc, D., Dao-Tran, M., Xavier Parreira, J., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and linked data. Proceedings of the International Semantic Web Conference (2011)
12. Mell, P., Grance, T.: The NIST Definition of Cloud Computing. Tech. rep. (2011), http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf
13. Priyatna, F., Corcho, O., Sequeda, J.: Formalisation and Experiences of R2RML-based SPARQL to SQL Query Translation using Morph. In: Proceedings of the 23rd International Conference on World Wide Web. pp. 479–489 (2014)
14. Rodriguez-Muro, M., Rezk, M.: Efficient SPARQL-to-SQL with R2RML mappings. Web Semantics: Science, Services and Agents on the WWW 33, 141–169 (2014)
15. Roman, R., Zhou, J., Lopez, J.: On the features and challenges of security and privacy in distributed internet of things. Computer Networks 57(10), 2266–2279 (2013)
16. Siow, E., Tiropanis, T., Hall, W.: Interoperable & Efficient : Linked Data for the Internet of Things. In: Proceedings of the 3rd International Conference on Internet Science (2016)
17. Siow, E., Tiropanis, T., Hall, W.: SPARQL-to-SQL on Internet of Things Databases and Streams. In: Proceedings of 15th International Semantic Web Conference (2016)
18. Stankovic, J.A.: Research Directions for the Internet of Things. IEEE Internet of Things Journal 1(1), 3–9 (2014)
19. Zhang, Y., Duc, P.M., Corcho, O., Calbimonte, J.P.: SRBench: A streaming RDF/SPARQL benchmark. In: Proceedings of the International Semantic Web Conference. Lecture Notes in Computer Science (2012)