FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Pingjiang Li

Semantic Reasoning on the Edge of Internet of Things

Master's Thesis
International Master's Programme in Computer Science and Engineering
(Ubiquitous Computing)
November 2016

Li P. (2016) Semantic Reasoning on the Edge of Internet of Things. University of Oulu, Degree Programme in Computer Science and Engineering. Master's thesis, 78 p.

ABSTRACT

The Internet of Things (IoT) is a paradigm where physical objects are connected with each other with identifying, sensing, networking and processing capabilities over the Internet. Millions of new devices will be added into IoT network thus generating huge amount of data. How to represent, store, interconnect, search, and organize information generated by IoT devices become a challenge. Semantic technologies could play an important role by encoding meaning into data to enable a computer system to possess knowledge and reasoning. The vast amount of devices and data are also challenges. Edge Computing reduces both network latency and resource consumptions by deploying services and distributing computing tasks from the core network to the edge.

We recognize four challenges from IoT systems. First the centralized server may generate long latency because of physical distances. Second concern is that the resource-constrained IoT devices have limited computing ability in processing heavy tasks. Third, the data generated by heterogeneous devices can hardly be understood and utilized by other devices or systems. Our research focuses on these challenges and provide a solution based on Edge computing and semantic technologies.

We utilize Edge computing and semantic reasoning into IoT. Edge computing distributes tasks to the reasoning devices, which we call the Edge nodes. They are close to the terminal devices and provide services. The newly added resources could balance the workload of the systems and improve the computing capability. We annotate meaning into the data with Resource Description Framework thus providing an approach for heterogeneous machines to understand and utilize the data. We use semantic reasoning as a general purpose intelligent processing method.

The thesis work focuses on studying semantic reasoning performance in IoT system with Edge computing paradigm. We develop an Edge based IoT system with semantic technologies. The system deploys semantic reasoning services on Edge nodes. Based on IoT system, we design five experiments to evaluate the performance of the integrated IoT system. We demonstrate how could the Edge computing paradigm facilitate IoT in terms of data transforming, semantic reasoning and service experience. We analyze how to improve the performance by properly distributing the task for Cloud and Edge nodes. The thesis work result shows that the Edge computing could improve the performance of the semantic reasoning in IoT.

Keywords: Edge Computing, Cloud Computing, Internet of Things, Semantic, Reasoning, Performance

TABLE OF CONTENTS

FOREWORD

I am thankful to my supervisors Professor Jukka Riekki and Dr. Xiang Su for the endless support as well as the countless times spent on supervision. I also thank Ubiquitous Computing Group and University of Oulu who gave me a precious study opportunity and wonderful life experience in Finland.

I also thank my family and friends who gave me meticulous care and warm accompany during these two years living aboard.

Oulu, Finland November 7, 2016

Pingjiang Li

ABBREVIATIONS

API	Application programming interface
ASP	Application service provider
ATN	Augmented transition network
BGP	Basic Graph Pattern
CISCO	Cisco Systems, Inc.
CPU	Central Processing Unit
DARPA	Defense Advanced Research Projects Agency
DL	Description Logic
DNS	Domain Name System
DTLS	Datagram Transport Layer Security
EN	Entity Notation
GUI	Graphical User Interface
HART	Highway Addressable Remote Transducer Protocol
HTTP	Hypertext Transfer Protocol
IM	Instant Messaging
IMAP	INTERNET MESSAGE ACCESS PROTOCOL
IoT	Internet of Things
IP	Internet protocol
LAN	Local Area Network
MEC	Mobile Edge Computing
MQTT	MQ Telemetry Transport
OWL	Web Ontology Language
OGSA	Open Grid Service Architecture
OGSA-DAI	Open Grid Service Architecture Data Access and Integration
OGSI	Open Grid Services Interface
PaaS	Platform as a Service
PACs	Programmable Automation Controllers
POP	Post Office Protocol
QoS	Quality of Service
RACs	Radio Applications Cloud Servers
RAM	Random-access Memory
RDBMS	Relational Database Management Systems
RDQL	RDF Data Query Language
RFID	Radio Frequency IDentification
RQL	RDF Query Language
SASL	Simple Authentication and Security Layer
SPARQL	SPARQL Protocol and RDF Query Language
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
URL	Uniform Resource Locator
WSRF	Web Services Resource Framework
XSLT	Extensible Stylesheet Language Transformations

1. INTRODUCTION

1.1. Background

Internet of Things (IoT) is the internetworking of physical objects that are provided with unique identifiers and the ability to transfer data over a network without requiring human-to-human interaction. The worldwide IoT market spending will grow from $591.7 billion in 2014 to $1.3 trillion in 2019 with a compound annual growth rate of 17%. The installed base of IoT units will grow from 9.7 billion in 2014 to more than 25.6 billion in 2019, finally reaching 30 billion in 2020 [1]. IoT can help organizations utilize their business infrastructures and assets in innovative ways to offer new services and deliver additional revenue. Individuals and societies could also gain benefits from IoT technologies.

Semantic technologies include a fairly diverse family of technologies aiming to help derive meaning from information. Semantic technologies encode meaning into data to enable computer systems to possess knowledge and reasoning. IoT connects the devices and the semantic technologies enable linking knowledge together. The semantic technologies have been developed for more than 40 years. There are many technologies and systems, for example, Formal logic [2], Knowledge Representation System [3] in AI, Semantic Network and ATN [4], DARPA and European Commission Program in information integration, Tractable Logic [5], Relational Algebra and Schema in database system [6].

Applying semantic technologies to IoT promotes interoperability among IoT objects and facilitates data access, data integration, resource discovery, semantic reasoning and knowledge extraction [7]. The communication and data exchange between IoT devices are essential activities in IoT. Semantic technologies provide a method for different machines to understand the data thus improving the interoperability and data integration. Semantic annotation of IoT objects and services is beneficial to the search and discovery for them.

Edge computing is "the enabling technologies allowing computation to be performed at the edge of the network, on downstream data on behalf of Cloud services and upstream data on behalf of IoT services [8]". The Edge computing could also be recognized as Mobile Edge Computing (MEC). The Edge computing is a general concept and the MEC is a more specific concept from architecture's prospectives and specify the Edge node as Mobile devices. The definition of MEC, which is "An open cloud platform that uses some end-user clients and located at the mobile edge to carry out a substantial amount of storage (rather than stored primarily in cloud data centers) and computation (including edge analytics, rather than relying on cloud data centers) in real time, communication (rather than routed over backbone networks), and control, policy and management (rather than controlled primarily by network gateways such as those in the LTE core) [9]." MEC aims at reducing both network latency and resources consumptions by shifting computing and storage capacities from the Internet cloud to the mobile edge [10]. Mobile Edge Computing enables innovative service scenarios that can ensure enhanced users experiences and optimized network operations [11]. The MEC will provide convergence of computer and connectivity, flexibility for application developers, new revenue stream for service providers and benefits of the network equipment vendors [9].

1.2. Challenges

Although IoT creates new possibilities and provides a great vision for the future technologies and industry, IoT still presents multiple challenges.

Research [12] shows that there are current five key challenges in IoT. First, the current architecture of the data centre cannot fully support the incredible increment of the data in personal and enterprise area. Juniper Research [13] estimates that the sales volume of smart devices in the world will reach 15 million units by 2013. The sales of these smart devices will amount to 70 million units by 2017. Second, the new data management systems need to be developed. Third, Data Mining is an essential technology for data process in big amount. IoT requires processing steaming data produced by various equipments. Early data mining tools are not entirely supporting structured data. Fourth, IoT data carries sensitive personal information such as health situation, location, movement, and purchasing preference. Protecting privacy is a major concern for both enterprise and the users. Fifth, the personal specific data could also cause security problem. The evolution of IoT technologies is in a hyper accelerated innovation cycle which means the scale of an IoT system could be extremely huge. A small error may not bring down a system but in the hyper-connected world [14], it could cause disorder throughput of the network.

Beside these, we also recognize some issues in the current IoT systems:

- The centralized server or Cloud server may be located far away from users and the physical distance generate a long latency.
- The IoT devices such as the sensors have usually limited computing and storage resources and they are unable to execute complex task or perform heavy computing activities.
- The heterogeneous devices will generate data in their own format which can hardly be understood or utilized by other devices or systems.

1.3. Research Objectives and Contribution

The increasing amount of data and devices in IoT requires new technologies. The data should be represented in a way that heterogeneous devices in the network could understand and utilize easily. Data processing requires intelligent method in order to provide more accurate and comprehensive results. New architectures are required for handle the vast amount of devices and provide high performance computing and high quality services. Hence, the semantic technologies and Edge computing paradigm has draw attention from researchers in IoT. We establish the three technologies together and study the influence of the integration. The thesis intents on investigating the benefits and issues in utilizing Semantic technologies, Cloud computing and Edge computing paradigm in IoT systems. It aims to discover the advantages of Edge computing paradigm and semantic technologies in IoT, the related problems caused by integration and technical barriers thus helping developing high performance future IoT systems.

More specifically, we focus on the following research questions:

1. How is the reliability of IoT systems integrated with Edge computing paradigm and semantic technologies? How is the performance and scalability of the new systems?

2. What benefits and shortcomings the Edge computing and semantic technologies have when introduce them in IoT systems?

Our contributions are discussed as follows. First , we develop an IoT system with semantic reasoner and Edge computing paradigm to study the benefits of integrating Edge computing paradigm into this IoT system. The system could do rule-based reasoning and help us study the performance of each process. Second, we carry out experiments to research the performance of the data transferring and semantic processing base on our system. Our experiments analyze the performance of semantic reasoning on the Edge of IoT. The results give constructive suggestions to the future IoT systems design and implementation.

1.4. Thesis Structure

The rest of the Thesis consists of the following structure: Chapter 2 describes the background knowledge about Knowledge Representation, semantics and IoT with the state-of-the-art technologies, framework, standards in this area. Chapter 3 presents an Edge semantic reasoning system for taxi traffic scenario data processing. Chapter 4 describes the experiments based on Taxi scenario and analyze the results. Chapter 5 discuss the findings and future works and chapter 6 summaries the research results draw a conclusion about this study.

2. LITERATURE REVIEW

This research focuses on three key technologies: IoT, the semantic technologies and the Edge computing paradigm. In this chapter, we will introduce the related concepts and technologies for these three fields. In semantic technologies, we will introduce the Resource Definition Framework and Semantic tools. For Edge computing paradigm, we will also introduce the Cloud computing and Fog computing.

2.1. Internet of Things

IoT means "a world-wide network of interconnected objects uniquely addressable, based on standard communication protocols" [15]. IoT systems will connect physical world objects via wireless and wired ways. Moreover, it will connect both the inanimate and living things, for example, the Cow Tracking Project monitors cows for illness and track behavior in the herd [16]. Researchers predicate that in the next decade, IoT will get a distinct development and at that time, the seamless fabric of classic networks and objects will provide ubiquitous service for us [17].

Research about IoT starts from early 2000's [18]. The new vision, which is presented in Figure 1, equips objects in daily life with identifiers, wireless connectivity and other digital modules, then the objects can communicate and coordinate with each other in order to form a network of physical things and be managed by machines automatically without the help from human beings. Figure 1 shows the main concepts, technologies and standards from three prospectives. "Things" oriented perspective focuses on physical objects. "Internet" oriented perspective focuses on networking between objects. "Semantic" oriented perspective focuses on cooperative processing.

Figure 2 shows the evolution from pure IoT to IoT with semantic technologies. There are two steps[20] to make IoT as a semantic IoT powered by Web technologies. The first step has been to offer interconnectivity to everything. The second step is to connect things to the Web. The current IoT platforms, systems and products are focusing on connection and integration among heterogeneous IoT resources. For these purpose, new standards, lightweight communication protocols such as CoAP [21] are designed and utilized. After coping with the connection issue, the market pays more attention to data and services. How to describe knowledge in a common way to make heterogeneous systems and applications understand and utilize universally becomes a new challenge.

Research [7] shows that IoT systems includes the vast amount of raw data produced by IoT devices and structured machine-readable information from the raw data for interoperability. What the human and service required is not the raw data but high-level abstractions and perceptions. The semantic technologies could transform the raw data into universally actionable and understandable knowledge.

IoT contains multiple technologies which cover the domain of software and hardware. These technologies enable the objects to have three major capabilities: identifiable ability, communication ability and interactive ability. Bandyopadhyay divided the technologies into 13 categories [19] including:

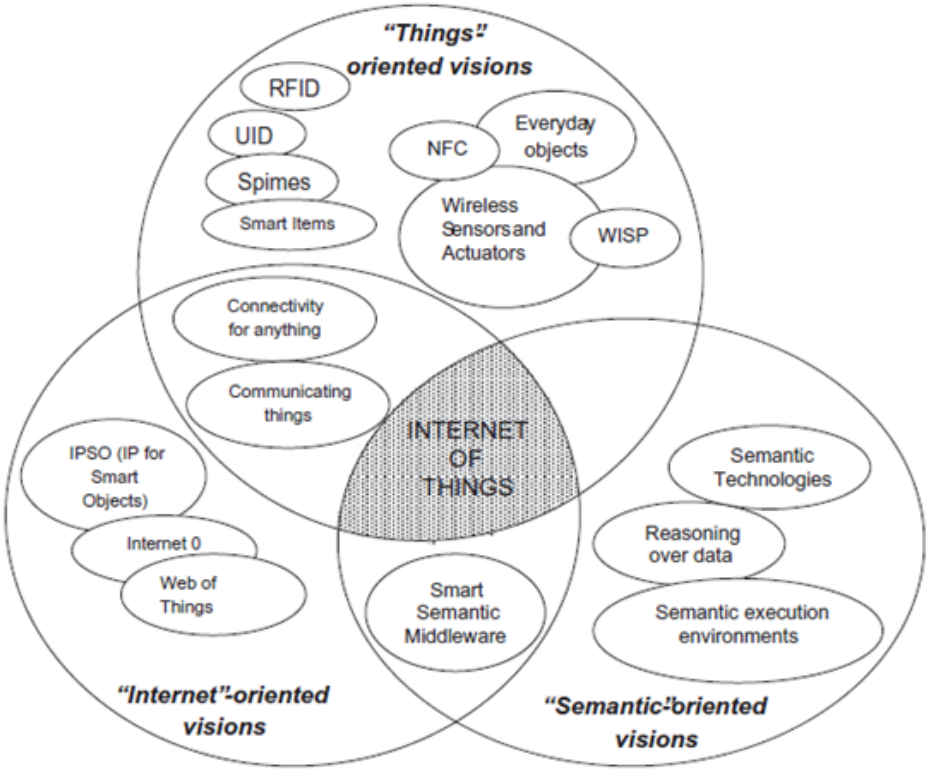

Figure 1. Vision of Internet of Things. [19]

- Identification: The function of identification is to map a unique identifier or UID to make an object an identifiable and retrievable entity without ambiguity.
- IoT Middleware Technology: This technology covers the middleware architectures for IoT following a particular approach such as the service oriented architecture approach.
- Communication Technology: Communication between IoT devices and systems.
- Networking technology: IoT deployment requires developments of suitable network technologies for implementing the vision of IoT to seek objects in the physical world and to bring them into the Internet.
- Network Discovery Mechanisms: In IoT paradigm, the networks will dynamically change and continuously evolve.
- Software and Algorithms: Software and algorithms used for data processing.
- Hardware: Research on nano-electronics devices is focusing on miniaturization, low cost and increased functionality in design of wireless identifiable systems.

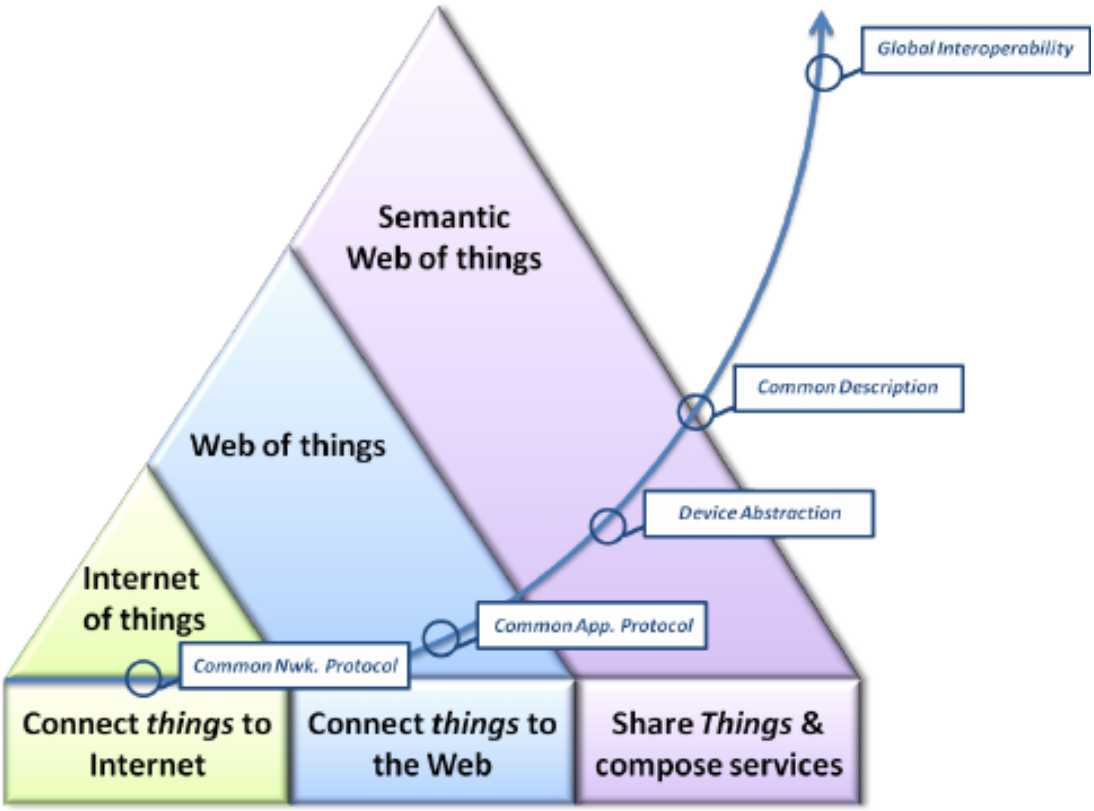

Figure 2. From the Internet of Things to the semantic Web of Things. [20]

- Data and Signal Processing Technology: Developing a standardized way to express the data contract, trust, process, work flow, message, and other data semantics for the nodes.
- Discovery and Search Engine Technologies: IoT requires the development of lookup or referral services to link objects to information and services and to support secure access to information and services.
- Power and Energy Storage Technologies: The autonomous things operating in IoT applications and performing either sensing or monitoring of the events need power and energy to perform the required job.
- Security and Privacy Technologies: Privacy of the humans and confidentiality of the business processes.
- Standardization: Standards should be designed to support a wide range of applications and address common requirements from a wide range of industry sectors.

Based on the above mentioned classification and our research background, we summarize the technologies into a simplified categories which contains 5 domains in Table 1. The first domain is Identification technology which is the same as Bandyopadhyay's Identification. The second one is Communication Technology and it includes "Networking Technology" and "Network Discovery Mechanisms". This technology

mainly focuses how to build a communication network and how could the items communicate with each other inside the network. The third one is “Hardware”, and it includes both “Hardware” and “Power and Energy Storage Technologies”. The fourth is Data and Signal Processing Technology. The fifth is Organization and Management, it includes “Discovery and Search Engine Technologies", “Security and privacy technologies” and “Relationship Network Management Technologies”. The rest items on Bandyopadhyay’s category such as “Software and Algorithms” and “Standardization” are included in other categories we summarized above.

In the following sub-chapter, we will introduce Identification Technology and Communication Technology. We do not focus on special hardware, so we skip “Hardware” technology. We use semantic technologies for data processing and it will be introduced in semantic technologies chapter. We use semantic technologies and Edge computing paradigm for organization and management, so we will introduce related technologies in semantic technologies chapter and computing technology chapter.

Table 1. Key Technologies in IoT

	Key Technologies
1	Identification
2	Communication Technology
3	Hardware
4	Data and Signal Processing Technology
5	Organization and Management

2.1.1. Identification Technology

Identification technology in IoT domain aims to solve the problem that how to name objects so that other objects can find them without confusion. Identifiers could be a random unstructured bit string. Otherwise it could be a structured name which consists of different meaningful name spaces, for example, the Uniform Resource Locator (URL) [22].

Uniform Resource Locator (URL) [22] is used for locating a web resource in Web-based systems. It consists of DNS name of the server and the path of the document. URL is a subset of URI because locators could also be used as identifier. Internationalized Resource Identifier (IRI) [23] is a complement to the Uniform Resource Identifier (URI), and it is a sequence of characters from the Universal Character Set (Unicode/ISO 10646) while URI supports only ASCII encoding. A mapping from IRIs to URIs is defined, which means that IRIs can be used instead of URIs, where appropriate, to identify resources. A Universal Unique Identifier (UUID) [24] is a 128-bit number used to uniquely identify objects or entities. UUID has different version. For example, the first version UUID combines a MAC address and a timestamp to produce sufficient uniqueness.

Another problem is how to locate the particular object according to the unique name in a network. Solutions includes broadcasting, multi-casting, forwarding pointers, home-based tracking approaches, distributed hash tables, hierarchical approaches, Domain Name System (DNS), etc.

2.1.2. Communication Technologies

One biggest challenge in IoT is that large number of nodes which will exist in IoT network and it is important to keep stable communication. As the device quantity increased, the available IPv4 address has almost been consumed, IPv6 has been popularized these years. But for IoT environment, IoT devices need to address the challenge to efficiently use the low power and limited bandwidth. There are various types of local area connections such as RFID, NFC, Wi-Fi, Bluetooth, and Zigbee. There are also many wide area connectivity such as GSM, GPRS, 3G, and LTE. An RFID system has readers and tags that communicate with each other by radio. RFID provides inexpensive communication with tags and readers for tracking devices, human beings and animals. The RFID tags are made of tiny chip, which is usually called the integrated circuit (IC), with antenna connected to it. The chips have memory component which stores information about the object such as the product's electronic product code (EPC) [25].

Sensor networks will also play a crucial role in IoT. Wireless Sensor Network (WSN) [26] is composed of small cheap sensors with limited processing and computing resources. Sensor could perceive the ambient situation and gain information from the environment, for example, the temperature. There are mainly five types of WSNs: terrestrial WSN, underground WSN, underwater WSN, multi-media WSN, and mobile WSN.

The communication protocol consists of five standard protocol layers for packet switching: application layer, transport layer, network layer, data-link layer, and physical layer. IoT has special requirement to address large scale of devices with limited computing resources. Internet Engineering Task Force (IETF) developed a number of standards and protocols for IoT. Figure 3 shows the landscape of IoT protocols. The landscape shows part of the widely used protocols according to layers. We will introduce some of the protocols based on the layer classification in Figure 3 and other important ones which are not showed in the figure.

For data-link layer and physical layer, IEEE 802.15.4 is for Near Field Communication (NFC) for resource limited devices, developed by IEEE 802.15 personal area network (PAN) working group. WirelessHART [28] and ZigBee [29] protocols are based on IEEE 802.15.4. 6LoWPAN protocol which is an acronym of IPv6 over Low power Wireless Personal Area Networks. IPv6 addresses are expressed by means of 128 bits in order to define 10^{38} addresses for low-power wireless communication nodes.

ZigBee [29] is an IEEE 802.15.4-based specification for a suite of high-level communication protocols with low-power digital radios. The low power consumption also restricts the communication distance.

WirelessHART [28] is a WSN protocol based on the Highway Addressable Remote Transducer Protocol (HART) with a time synchronized, self-organizing, and self-healing mesh architecture.

Home Automation (RFC5826), Industrial Control (RFC5673), Urban Environment (RFC5548) and Building Automation (RFC 5867) standards [30] are developed for Network layer by Lossy and Low-power Networks (RoLL) working group. Routing protocol for LLN (RPL) support point-to-point point-to-multipoint and multipoint-to-point communication.

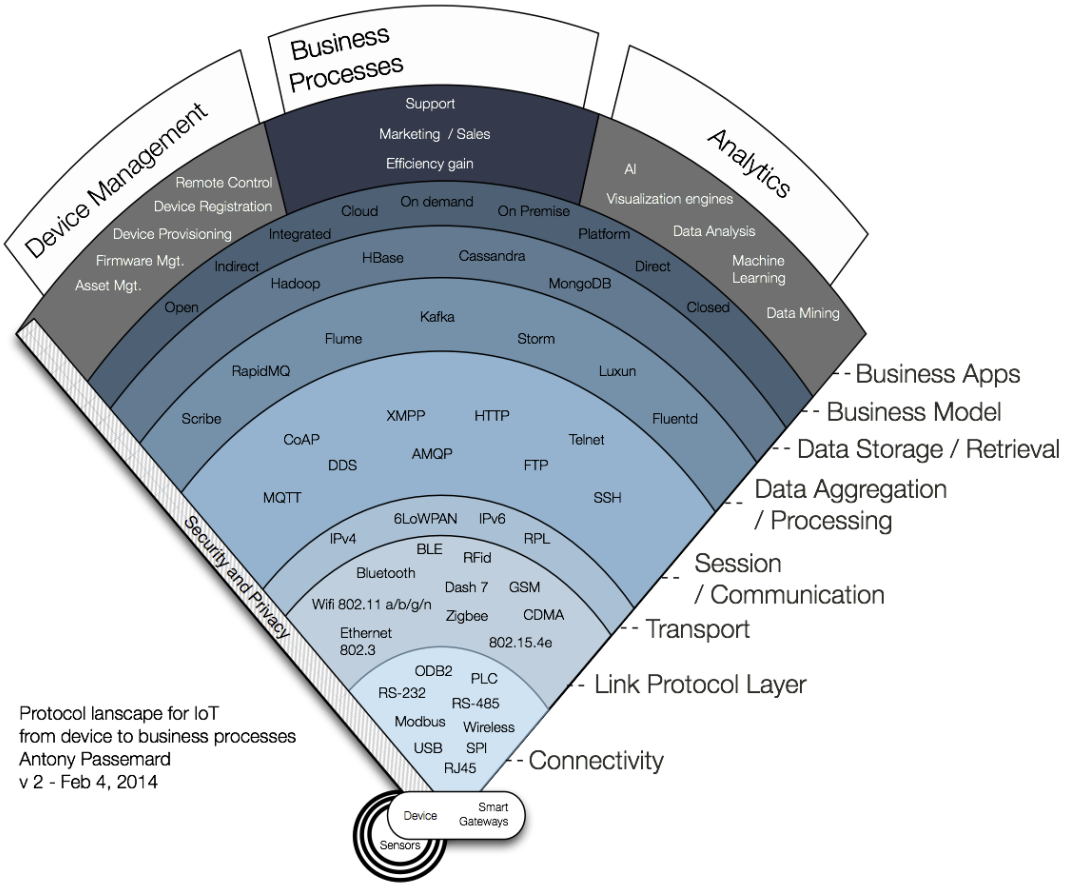

Figure 3. Protocol Landscape for IoT. [27]

In application layer, there are many protocols, for example CoAP [21] and MQTT, in next part, we will discuss the functionality, features , pros and cons of the most widely used protocols.

The constrained Application Protocol (CoAP) [21] is a web transfer protocol with REST [31] style. It is designed for easy stateless mapping with HTTP. It provides machine-to-machine interaction. CoAP employs UDP protocols as TCP protocol is more complex for the resource-constrained devices under IoT environment. CoAP enable to process millions of devices. CoAP has similar features with HTTP, but HTTP employs TCP which increases the computation complexity. CoAP provides URI, REST method and IP multicast. Comparing with TCP, UDP is relatively unstable. To overcome this shortcoming, CoAP implements a retransmission mechanism and resource discovery mechanism. CoAP employs a two layer structure, the lower layer is for asynchronously processing UDP messages and the upper layer is for REST request and response. For security consideration, CoAP utilizes Datagram Transport Layer Security (DTLS) [32] protocol for integrity, authentication and confidentiality [33].

Embedded binary HTTP (EBHTTP) [34] is a binary-formatted, space-efficient, stateless encoding of the standard HTTP/1.1 protocol. EBHTTP is primarily designed

the transportation of small scale data between resource-constrained nodes, which is similar with CoAP.

Message Queue Telemetry Transport (MQTT) [35] is a publish/subscribe messaging transport protocol. It employs a topic-based publish-subscribe architecture. The clients could perform as publishers, subscribers or both. Publishers publish message to a specific channel with a unique topic name. All the subscribers who subscribe this topic can receive the message. MQTT utilize TCP protocol to provide stable communication. MQTT has three level of Quality of Service (QoS). The lowest level will only deliver the message for one time, the middle level delivers the message at least once for confirmation, and the highest level employs a four-way handshake mechanism.

Comparing with MQTT, CoAP is more lightweight as it utilizes UDP. CoAP use URI to locate resources while MQTT uses topic for message. Research shows MQTT will generate longer delay when data packet lost [36].

MQTT-SN is a pub/sub protocol, which optimized for implementation on low-cost, battery-operated devices with limited processing and storage resources. MQTT-SN will delivery short message by dividing the original message into 3 sub-messages. For compressing the message length, it also uses "Pre-defined" topic IDs and "short" topic names [35].

The Advanced Message Queuing Protocol (AMQP) is an open standard and message-oriented binary protocol for message-oriented middleware. Comparing with the text-base protocol and binary protocol, package the data into binary will save more space. AMQP is a wire-level protocol, which means the format of the data that is sent across the network as a stream of octets. AMQP has an exchange which routes the message to the particular queue based on rules and criteria. Then the message will be stored in a message queue and wait for consumers [37].

The Extensible Messaging and Presence Protocol (XMPP) [38] is a message-oriented protocol for streaming XML elements with a real time behavior. XMPP was called "Jabber" and provides instant messaging (IM) to connect people to other people via text messages. XMPP is an open standard available for everyone. Secure authentication (SASL) and encryption (TLS) have been built into the core of XMPP specifications. The XMPP does not support QoS and text-based communication made XML data heavy.

Lightweight M2M provides an underlying layer agnostic protocol. OMA Lightweight M2M [39] is a protocol from the Open Mobile Alliance for M2M or IoT device management. The OMA Lightweight M2M enabler includes device management and service enablement for LWM2M Devices.

2.2. Semantic Technologies

Semantic technology is a concept in computer science that aims to bring semantics, which includes the meaning and context behind words and sentences. A number of approaches to implement the concept have been developed, ranging from artificial intelligence to formal, machine-readable descriptions of content. The Web is a key focal point for semantic technologies, though it may benefit business and academic fields as well.

The World Wide Web is one of the focal points for semantic technologies, and the Semantic Web attempts to address some of the semantic shortcomings of the current web, by introducing a much more versatile and dynamic markup to the documents that comprise the network. We will introduce this concept first.

The semantic technologies mainly address meanings for data. Making the data semantic is to represent the data as a knowledge that all the other devices and systems could understand it. We will introduce this in Knowledge Representation and Logic section. After understanding the data, the system could generate new knowledge based on the existing ones. This is introduced in Reason section. Then we will introduce technologies about encoding the knowledge in semantic way in Resource Description Framework, Web Ontology Language, RDF serialization format sections, technologies about semantic processing in Semantic Reasoner and Editor section, and RDF Database section.

2.2.1. Semantic Web

Berners Lee gave a formal definition [40] for Semantic Web: "The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation." Semantic Web technologies contains a family of technology standards from the World Wide Web Consortium (W3C) that are designed to describe and relate data on the Web and inside enterprises. For example, Resource Description Framework is a flexible data model, RDFS and OWL are schema and ontology languages for describing concepts and relationships, SPARQL [41] is a query language, RIF [42] is a rules exchange language.

The Concept of Semantic Web has been utilized in real application. Here are some examples of the successful running applications: The British Broadcasting Corporation (BBC) begin to use semantic web technologies in 2010 for the World Cup [43]. Every player on every team in every group had their own web page, and you could navigate from one piece of content to the next with ease. Later BBC utilize the semantic web technologies in BBC Sport, BBC Education, BBC Music, News projects and more. BBC developed Ontolgies [44] for Business News, content management systems, curricula, food, journalism, politics, provenance, sports, storyline and wildlife. There are also many Semantic Web applications [45]. The Open Graph Protocol is Facebook's version of the Semantic Web since 2010. When users press the LIKE button, Facebook will get not just a link, but a specific object of the specific type. The Biogen Idec utilizes Semantic Web technologies for Supply Chain Management [45], Chevron has been experimenting with Semantic Web technologies in Data Integration in Oil and Gas [45].

2.2.2. Knowledge Representation and Logic

Knowledge Representation [46] was developed in the 1970's. It concerned with how to represent the knowledge symbolically so that the computer systems could understand and utilize them for reasoning. It is divided roughly into two categories: logic-based representations, which evolved out of the intuition that predicate calculus could be used

unambiguously to capture facts about the world; and non-logic-based representations, which are developed by building on more cognitive notions.

First-order Logic [47] is a collection of formal systems for mathematics, philosophy, linguistics and computer sciences. First-order Logic uses quantified variables over objects. The First-order Logic is developed from propositional logic. Comparing with propositional logic, First-order Logic adds assertion, predicate and quantifier. Proposition could be true or false while assertion is always thought to be true. First-order logic is the standard for the formalization of mathematics into axioms and is studies on the foundations of mathematics.

The Description Logic (DL) [48] is a subset of the first-order-predication logic and a family of logic-based knowledge representation languages that can be used to represent the terminological knowledge of an application domain in a structured way. The key characteristic features of Description Logics reside in the constructs for establishing relationships between concepts. DL could use a network-base representation structure, where nodes represent concepts and links represent relationships.

Concept descriptions can be used to build statements in a DL knowledge base, which typically comes in two parts: a terminological and an assertional one. In the terminological part, there is the TBox, which describes the relevant notions of an application domain by stating properties of concepts and roles, and relationships between them—it corresponds to the schema in a database setting. The assertional part of the knowledge base, called ABox, is used to describe a concrete situation by stating properties of individuals—it corresponds to the data in a database setting [49].

2.2.3. Reasoning

Reasoning is a process of thought that yields a conclusion from percepts, thoughts, or assertions. The basic inference on concept expressions in Description Logics is subsumption, typically written as $C \subseteq D$. Determining subsumption is the problem of checking whether the concept denoted by D (the subsumer) is considered more general than the one denoted by C (the subsumee). In other words, subsumption checks whether the first concept always denotes a subset of the set denoted by the second one.

The definite clause logic is monotonic in the sense that anything that could be concluded before a clause is added can still be concluded after it is added; adding knowledge does not reduce the set of propositions that can be derived. A logic is non-monotonic if some conclusions can be invalidated by adding more knowledge. The logic of definite clauses with negation as failure is non-monotonic. Non-monotonic reasoning is useful for representing defaults. A default is a rule that can be used unless it overridden by an exception [50].

Intermittently, non-monotonic reasoning is related to inductive reasoning and monotonic to deductive reasoning. Non-monotonic reasoning is more close to human learning process, where old knowledge can be negated after new knowledge has been discovered [51].

There are two ways to study and build the intelligence systems: from the top down and from the bottom up [52]. The top-down study focuses on intelligent behaviour such as thought and reason. To simulate the same behaviour on computers, people need to figure out what an agent needs to know in order to trigger that behavior and what com-

putational mechanisms could allow them gain new knowledge based on the existing ones. Semantic reasoning is one of the intelligent behaviours focusing on reasoning based on linguistic meaning of the data. It is a top down method. For doing semantic reasoning, there are also two important concepts: knowledge and inference. Semantic reasoning is a process to inferring new knowledge base on the existing knowledge. The same knowledge could have several different ways to be represented.

Inference on the Semantic Web can be characterized by discovering new relationships. On the Semantic Web, data is modeled as a set of named relationships between resources. "Inference" means that automatic procedures can generate new relationships based on the data and based on some additional information in the form of a vocabulary, e.g., a set of rules [40].

To make the reasoning process simple enough for people without professional background to use, one solution is to set conditions [53]. Embedded rules are one means for implementing setting condition. Embedded rules have the form of "if s, then if a, then c" where "s" is a setting condition, "a" is an antecedent, and "c" is a consequent. A rule-based system may be viewed as consisting of three basic components: a set of rules (rule base), a data base (fact base) and an interpreter for the rules. The rule-based reasoning is to store the rule and fact as knowledge inside the system to interpret information in a useful way.

2.2.4. Resource Description Framework

Resource Description Framework (RDF) [54] has been a widespread data format for the Semantic Web. RDF provides an approach for knowledge representation of the resources for computers. RDF is a framework for modeling data with general-purpose language to represent information in the Web.

RDF is a graph-based data model. The core structure of RDF is a set of triples and each triple consists of a subject, a predicate and an object. The subject is an internationalized resource identifier (IRI) [23] or a blank node. The predicate is an IRI and the object is an IRI, a literal or a blank node. Anything in real world is a resource.

The IRI [23] was defined in 2005 as a new Internet standard to extend upon the existing uniform resource identifier (URI) scheme, which supports Universal Character Set (Unicode/ISO 10646). For example, the IRI could be:

```
http://www.w3.org/1999/02/22-rdf-syntax-ns\#XMLLiteral
```

Resources could have IRIs as its referent or literal as literal value. The predicate itself is an IRI and denotes a property. An RDF vocabulary is a collection of IRIs intended for use in RDF graphs. RDF graphs define RDF vocabularies. An RDF vocabulary is a collection of IRIs. RDF vocabularies include built-in classes and properties, which are used to describe the resources and relationships. Properties are for describing relationships between the subject and the object of the statement and are used as a predicate in a statement. An RDF data set is a collection of RDF graphs. In an RDF data set, all statements are assumed to be true only in the named graph they belong to. RDF semantics enable making complex deductions and infers from the RDF graphs.

The RDF statement is an expression following a specific grammar that names a specific resource, a specific property (attribute), and gives the value of that property for

that resource. A RDF graph contains a set of RDF triples. Each triple contains a subject, a predicate and an object expressed by RDF statement. A RDF graph is also called a RDF data model. The RDF graph is a directed labeled graph. The statement itself is also a resource and one statement is the simplest form of RDF graph. The representation of the statement “The name of the user LiPingjiang in GitHub is Li Pingjiang” in RDF statement is “< https://github.com/LiPingjiang><foaf:name>“Li Pingjiang” “. The Subject is “< https://github.com/LiPingjiang>” and the Predicate “<foaf:name>” is IRIs and the Object"Li Pingjiang” is a literal value. The abbreviations “foaf” refers to name space of friend-of-friend vocabulary.

The data manipulated in the Semantic Web is usually described using triples or statements. A triple is a unit data element with the form: “<s,p,o>”. “s” represents the subject, “p” represents the predicate and the “o” represents the object. So a triple could be interpreted as a statement: “object o stands in relationship p with subject s”. A set of triples could be described as a graph in RDF graph model called “triple store” or “RDF database”. The graph RDF applies a field of logic called description logic. RDF Semantics specification [55] and RDF Schema (RDFS) define a base semantic on top of RDF and allow developers to define standardized vocabularies. The power of RDF relies on the flexibility in representing arbitrary structure without a priori schemas.

The data manipulated in the Semantic Web is usually described using triples or statements. The RDF statement is an expression following a specific grammar that names a specific resource, a specific property (attribute), and gives the value of that property for that resource. A triple is a unit data element with the form: “<s,p,o>”. "s” represents the subject, “p” represents the predicate and the “o” represents the object. So a triple could be interpreted as a statement: “object o stands in relationship p with subject s”. A set of triples could be described as a graph in RDF graph model called “triple store” or “RDF database”. A RDF graph is also called a RDF data model. The RDF graph is a directed labeled graph. The statement itself is also a resource and one statement is the simplest form of RDF graph. The representation of the statement “The name of the user LiPingjiang in GitHub is Li Pingjiang” in RDF statement is “< https://github.com/LiPingjiang><foaf:name>"Li Pingjiang” “. The Subject is “< https://github.com/LiPingjiang>” and the Predicate “<foaf:name>” is IRIs and the Object"Li Pingjiang” is a literal value. The abbreviations “foaf” refers to name space of friend-of-friend vocabulary. RDF Semantics specification [55] and RDF Schema (RDFS) define a base semantic on top of RDF and allow developers to define standardized vocabularies. The power of RDF relies on the flexibility in representing arbitrary structure without a priori schemas.

Table 2. Triple Example

Subject	Predicate	Object
<https://github.com/LiPingjiang>	<foaf:name >	Li Pingjiang

RDF Schema (RDFS) [55] provides a data-modelling vocabulary for RDF data extended from the basic RDF vocabulary. It provides mechanisms for describing groups of related resources and the relationships between these resources. For example, one relationship in RDFS is “subclass” and if one class is subclass of another class, it must be inherented all the properties the other class has. To present RDF graph data in computer science, there are various formats for publishing and exchanging the RDF data.

RDF/XML, Notation 3 (N3) [56], Turtle, and N-Triples are the most popular formats. Most of these formats are based on triple structure, which are subject, predicate and object.

RDFS enables to express the relationships between objects by standardizing in a flexible, triple-based format and then providing a vocabulary which can be used to describe objects.

2.2.5. Web Ontology Language

The word "ontology" is a philosophy term at the beginning. According to Thomas Gruber [57], an ontology is "an explicit specification of a conceptualization". He states that, the term 'ontology' signifies a systematic account of existence in philosophy. Then the concept of "ontology" is transformed from philosophy to Information Systems [58]. Nicola Guarino [59] explains, an IS ontology is "an engineering artifact, constituted by a specific vocabulary used to describe a certain reality, plus a set of explicit assumptions regarding the intended meaning of the vocabulary words".

Ontology is based on Open World Assumption (OWA) or Closed World Assumption (CWA). In OWA, the assumption is that "everything is not known and anything that is not stated at the moment can be true or false". In contrast, CWA is based on the assumption that "anything that is not stated is false and everything is known" [50]. When we want to represent knowledge with Ontology and discover new information, we need to apply Open World Assumption as OWA applies when a system has incomplete information.

Web Ontology Language (OWL) [60] is an ontology language descended from Description Logics. It is produced by the W3C Web Ontology Working Group in 2004. OWL is recommended to be a major formalism for Semantic Web. The official exchange syntax for OWL is RDF/XML [61].

An ontology language is a language which describes information about different kinds of objects in the domain of discourse. An ontology is an explicit specification of a conceptualization. The term is borrowed from philosophy, where an Ontology is a systematic account of Existence [57]. Different kinds of Ontologies has been developed for describing various concept in difference scales.

The latest version of OWL is OWL 2. Comparing with OWL 1, OWL 2 have more useful structure for modeling [62]: Qualified Cardinality Restrictions, relational expressivity, datatype expressivity and keys [62]. Beside the this, OWL 2 also have advantages on syntax, meta modeling, imports, versioning, annotation, species validation, etc.

2.2.6. RDF Serialization Format

RDF data can be saved in a storage with various serialization formats. JSON-LD [61], TriG, N-Triples [61], N-Quads [63], Turtle [61], RDF/XML [61] and RDFa are recommended standard formats by W3C Recommendations. Beside these, there are other formats such as Notation 3 (N3) [56], Embedded RDF (eRDF), RXR [64], TriX and Entity Notation (EN). The eRDF and RDFa are good choices for augmenting HTML

documents with small amounts of additional metadata. Turtle [61] or RDF/XML [61] can easily be transformed to RDF model. RXR, TriX, and N-TRIPLES are easy to parse to triples but they have poor human-readability.

RDF/XML [61] is the most widely used RDF format and the only format specified by the RDF Recommendations. It is very human-readable format with concise syntax. But RDF/XML is not easy to implement as XML structure thus making it hard for parser. Another shortcoming is that some RDF graphs cannot be serialised to RDF/XML, and other reasonable constructs cannot use the shorthand forms for lists and collections. RDF/XML encodes the RDF graph in XML format. RDF/XML uses XML QNames as defined in name spaces in XML (XML-NAMES) to represent IRIs. QNames stands for "qualified names" and defines a valid identifier for elements and attributes. QNames requires both the name of the name space and local part. In addition, QNames can either have a short prefix or be declared with the default name space declaration.

EN [65] is a lightweight data representation designed for ambient intelligent systems. EN produces compact packets for efficient processing and communication. EN packets could be transform into RDF or ontology models in an unambiguous fashion.

N-Triples is extremely simple to implement serializers and parsers and efficient for streaming. As N-Triples does not use abbreviation, it looks very verbosely and is lack of human-readability.

The Manchester OWL syntax [61] is a compact syntax for OWL 2 ontologies. The original Manchester syntax, developed by the CO-ODE project at The University of Manchester, is created for OWL 1. The Manchester syntax for OWL 2 ontologies is defined using a standard Backus–Naur Form notation. It is designed to produce a concise syntax, which did not use DL symbols, and is quick and easy for human to read and write. Comparing with DL syntax, the Manchester syntax replace mathematical symbols such as

$$\exists, \forall, \neg$$

with keywords "some", "only", and "not".

Turtle [56] is the Terse RDF Triple Language. It is an extension of N-Triples used in RDF model recommended by W3C. Turtle is compatible with N3 and could be used inside the SPARQL query language. It describes RDF graph using triples consist of a subject, a predicate and a object. Turtle has compact expression about object list and collections comparing with RDF/XML. Prefix is separated from the triples data. Turtle is intended to be compatible with, and a subset of N3 [56]. N3 is a machine-readable syntax defined by the context-free grammar.

The OWL functional-style syntax [66] is a concrete syntax. A functional-style syntax ontology document is a sequence of Unicode characters. The OWL functional-style Syntax is the product of the W3C OWL Working Group. It is used for defining OWL 2 in the W3C Specs.

TriX [67] is an XML syntax, that adopts a flat triples style syntax like N-Triples, but it also allows XSLT to be used to provide syntactic extensions that render down to the un-extended TriX format. TriX supports an extensible syntax via XSLT. The format is not readable for human and inefficient for transforming.

RXR [64] is a triple-based XML serialization of RDF. It removes syntactic extensibility and named-graphs and adds a concise syntax for collections. It is a simpler version of TriX, so it has similar pros and cons with it.

RDFa [68] is an extension of XHTML that allows RDF triples to be embedded in an HTML document.

Embedded RDF (eRDF) is a syntax for writing HTML in such a way that the information in the HTML document can be extracted (with an eRDF parser or XSLT style sheet) into RDF.

Header Dictionary Triples (HDT) [69] is a compact data structure and binary serialization format for RDF that keeps big datasets compressed to save space while maintaining search and browse operations without prior decompression.

2.2.7. RDF Database

RDF is a logical data model consisting of triples. The traditional approaches for indexing, querying and storing data is not adapted to the dynamic graph-structured character of Semantic Web. There is a realistic requirement of efficient tools and technologies for storing and querying knowledge using the ontologies and the related resources. For example, the relational database management systems (RDBMS) can hardly handle the non-static static Semantic Web data.

For querying RDF data, there are several alternative query languages: RDF Query Language (RQL) , RDF Data Query Language (RDQL) and finally the W3C Recommendation SPARQL Protocol and RDF Query Language (SPARQL) [41]. SPARQL is a semantic query language for databases, able to retrieve and manipulate data stored in RDF format. SPARQL queries RDF graphs. SPARQL gives the users access to create, combine, and consume structured data by accessing a web of Linked Data while SQL does this by accessing tables in relational databases. SPARQL is based on graph pattern matching. There are many graph patterns such as Basic Graph Patterns, Group Graph Pattern and Optional Graph patterns. Basic Graph Pattern (BGP) [70] are designed for efficient indexing RDF data. It is essentially conjunctive queries. A basic graph pattern is a conjunction of series of triple. For example, a BGP consists of m triples, triple "i" is represented by (s_i, p_i, o_i). The BGP is $(s_1, p_1, o_1) \wedge ,,, \wedge (s_m, p_m, o_m)$.

Figure 4 shows the classification of IoT storage technologies. There are two types of RDF data management systems: the native storage and the non-native storage. The native storage means the system is designed based on RDF structure, for example, Virtuoso, Mulgara, AllegroGraph, Garlik JXT. The non-native storage means the system is set up with a third party general purpose database, which is not originally design for RDF data, for example, relational databases MySQL. The native storage will have better performance [71]. As the non-native storage cannot directly process the RDF data, there are DBMS based approaches to cope with this issue. There are three main physical organization techniques [72]: triple table, property table and the vertical partitioning approaches. Triple table stores triple in three column in Relational Database, one column for subjects, one for predicates and one for objects. As all the triples are stored in one table, query operation may be slow. The triple table approach has been used for Oracle, 3store, Redland, RDFStore and rdfDB. Property table stores one subject and several fixed properties in each named table to avoid self-joins on subject column. But it will generating many NULL values. This approach has been used by tools such as Sesame, Jena2, RDFSuite and 4store. Vertical partitioning [73] rewrites

triples table into n two column tables where n is the number of unique properties in the data. In each of these tables, the first column contains the subjects that define that property and the second column contains the object values for those subjects. The swStore first use this approach.

There are five criteria to evaluate the quality of RDF database [72]. First criterion is what data storage mechanism the storage system used. Second criterion is what level of inference it supports. There are two strategies for inference, the system could do the inference when the data comes, so the results are already stored before querying. Another approach is to execute the inference on demand. Third criterion is about data update. Fourth, the storage should support efficient updating and modification. Fifth, distribution network issues should be considered.

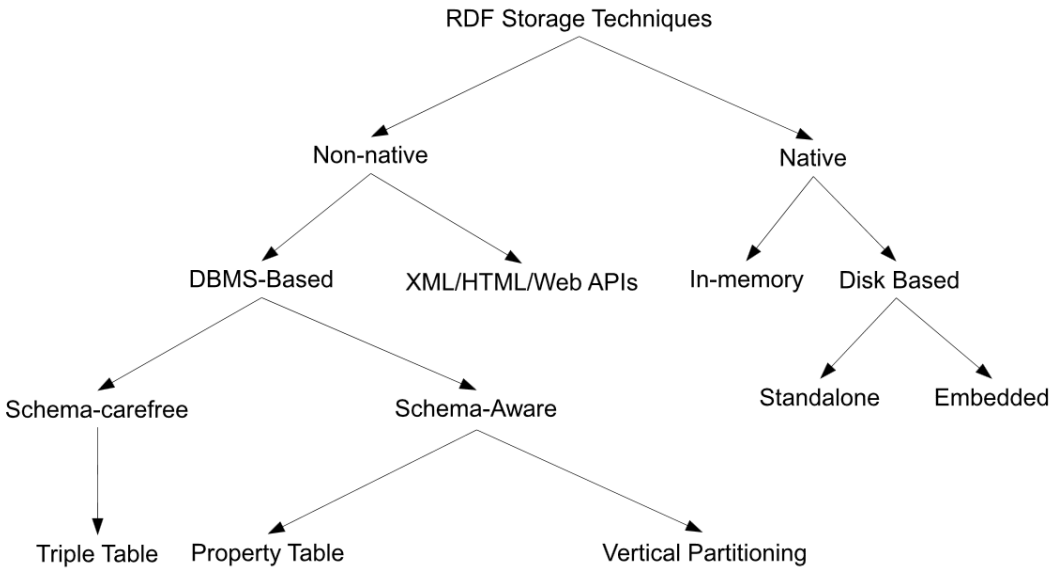

Figure 4. Classification of RDF Data Storage Approaches. [72]

Researchers have compared the RDF storage approaches based on the previous criteria [72].The paper was written in 2012, so some of the software have improved. We update the data according to W3C “LargeTripleStores” page [74]. The result shows in Table 3.

Table 3. Storage and Database

Store	Storage Scheme	Storage Support			Query Language	US	IS		Scalability (Million)	DS
		DBMS	File	In RAM			C	R		
Sesame	PT	✓	✓	✓	RQL	✓	✓		70	
Virtuoso	MI	✓			SPARQL	✓		✓	14500	✓
RDFSuite	PT	✓			RQL	✓	✓	✓	-	
3store	TT	✓			RDQL /SPARQL		✓	✓	100	
rdfDB	TT	✓		✓	SquishQL -like	✓	✓	✓	-	
RDFStore	TT	✓	✓	✓	RDQL /SPARQL		✓	✓	-	
Redland	TT	✓			RDQL /SPARQL	✓			-	
Allegro Graph	-		✓		SPARQL				1000000	✓
sw-Store	VP	✓			SPARQL				55	
roStore	HPP	✓			SPARQL			✓	1,3	
4store	PT	✓			SPARQL	✓	✓		15000	✓
YARS	MI		✓	✓	N3 extensions	✓			7000	
Kowari	MI	✓			iTQL /RDQL				160	
Hexastore	MI	✓		✓	SPARQL				-	
RDFJoin	MI	✓			SPARQL				44	
RDFKB	MI	✓			-	✓	✓		44	
BitMat	MI			✓	SPARQL -like				47	
TripleT	MI			✓	-				6	
RDF-3X	MI	✓			SPARQL	✓			51	
iStore	MI		✓		-				6	
Parliament	MI		✓		-	✓	✓	✓	4,5	
Brahms	MI			✓	-				6	
RDFCube	MI				-				0,1	✓

2.2.8. Semantic Reasoners and Editors

A semantic reasoner is a program which could infer logical consequences from a set of explicitly asserted facts or axioms. It could also be called as “reasoning engine” or “rules engine”. The inference is based on rules which are specified by ontology. First-order logic is often used to perform reasoning with forward chaining or backward chaining. Table 4 shows the semantic reasoners or framework which support reasoner. Cys and KAON2 are close source software. Gandalf, Cwm, Drools, Flora-2, Jena,

Prova, dot15926 Editor, Apache Marmotta, CB, CEL, ELK reasoner, ELLY, FaCT++, HermiT, leancor, Pellet and RDFFox are open sources software. All of them are free software.

Jena Framework [75] is a Java framework for building Semantic Web applications. It provides an extensive Java libraries that handle RDF, RDFS, RDFa, OWL and SPARQL in line with published W3C recommendations. Jena includes a rule-based inference engine to perform reasoning based on OWL and RDFS ontologies, and a variety of storage strategies for storing RDF triples in memory or on disk. In Jena Framework, there are basically three types of reasoner: the RDFS reasoner, the rule-based OWL reasoner, the transitive reasoner. The rule-based reasoners have three rule engines: the forward reasoning, the backward reasoning and the hybrid reasoning. There are two internal rule engines: the forward chaining RETE engine and the tabled datalog engine.

Many reasoners are designed for OWL. For example, HermiT [76], Owlgres [77] and Pellet [78].

Semantic editors are able to edit the RDF file and Ontology. There are many different types of editors [79], for example, Protégé is a free, open-source ontology editor and framework for building intelligent systems; WebProtégé is an online version of Protégé; Fluent Editor, is a tool for editing, manipulating and querying complex ontologies written in OWL, RDF or SWRL; Vitro is a general-purpose web-based ontology and instance editor with customizable public browsing; gFact is a Graph-based Faceted Exploration of RDF Data; PoolParty is a world-class semantic technologies suite; RDFaCE is a RDFa Content Editor based on TinyMCE; TopBraid Composer Standard Edition (TBC-SE) is a powerful semantic web modeling tool for building and testing configurations of ontologies and RDF graphs; OWLGrEd is a free UML style graphical editor for OWL ontologies; VocBench is a web-based, multilingual, editing and workflow tool that manages thesauri, authority lists and glossaries using SKOS-XL.

2.3. Computing, Edge and Fog Technology

2.3.1. Cloud Computing

The market of Cloud computing grew significantly during these years. It becomes popular both in industry and academy. There is no consensus definition about “Cloud Computing” [80] yet. The first definition of the term “Cloud Computing” was given by Prof. Chellappa in 1997 [81]. He thought Cloud computing as a “computing paradigm where the boundaries of computing will be determined by economic rationale rather than technical limits alone.” Ian Foster [82] defined from technical point of view: “A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet."

Cloud has three essential features [83]: first, it provides infinite computing resources available on demand; second, the elimination of an up-front commitment; last but not least, the ability to pay for use of computing resources on a short-term basis. Consider

Table 4. Semantic Reasoners

	Free	Open Source	Charactor
Cyc	✓	✗	A forward and backward chaining inference engine.
KAON2	✓	✗	Managing OWL-DL, SWRL, and F-Logic ontologies.
Gandalf	✓	✓	Decision rules engine on PHP.
Cwm	✓	✓	A forward-chaining reasoner.
Drools	✓	✓	A forward-chaining inference-based rules engine.
Flora-2	✓	✓	An object-oriented, rule-based knowledge-representation and reasoning system.
Jena	✓	✓	An open-source semantic-web framework.
Prova	✓	✓	A semantic-web rule engine.
dot15926 Editor	✓	✓	Ontology management framework.
Apache Marmotta	✓	✓	A rule-based reasoner.
CB	✓	✓	A resolution based reasoner for OWL EL. Superseded in 2011 by ELK.
CEL	✓	✓	CEL is an open-source polynomial-time Classifier for the OWL 2 EL profile.
ELK Reasoner	✓	✓	ELK is an ontology reasoner that aims to support the OWL 2 EL profile.
ELLY	✓	✓	ELK is a Java-based, open source reasoning for OWL EL.
FaCT++	✓	✓	FaCT++ is a DL reasoner. It supports OWL DL and (partially) OWL 2 DL.
HermiT	✓	✓	HermiT is reasoner for ontologies written using the OWL.
leancor	✓	✓	A fork of leanCoR that aims description logics reasoning.
Pellet	✓	✓	Pellet is an open-source Java based OWL 2 reasoner.
RDFox	✓	✓	RDFox is a highly scalable in-memory RDF triple store that supports shared memory parallel datalog reasoning.

about one task which needs one server to run 1000 hours, by using Cloud technology, users could finish the task in one hour on 1000 servers with the same price.

The Cloud computing has several advantages to business [84]. Figure 5 shows the benefits from Cloud Computing from a more comprehensive view. The benefits cover from the business aspects to computing aspects. For example, there is no up-front investment as the pay-as-you-go pricing model allow customer to start developing after purchasing. Cloud reduces operating cost by scale benefits. The system is highly and easily scalable. The Cloud servers and services are easily accessed as the provider will develop customized interface for users. It will also reduce business risks and maintenance expenses. Traditional business model is a one-time payment for unlimited use. But this will cause over-provisioning or under-provisioning problem. For example, one company bought a high performance server for web site host. The hardware could fully meet the requirement and still have extra capability for unexpected volume and traffic. At beginning, the CPU is always idle and storage is partly used. Several years later, new software requires better CPU and data need larger storage, the server need update because of the inadequate performance.

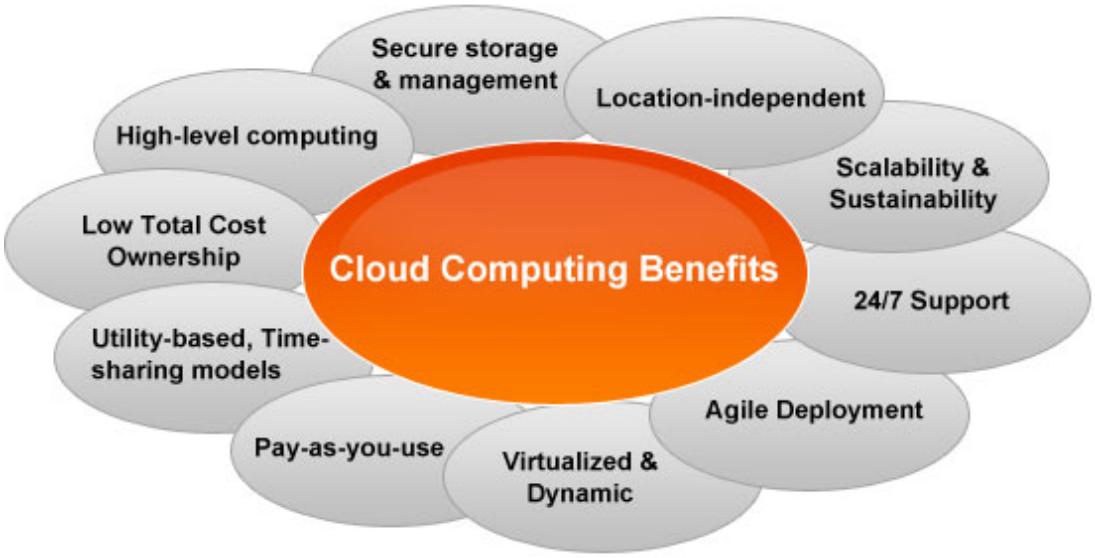

Figure 5. Benefits of Cloud Computing. [85]

The current challenges [84] are about automated service provisioning, virtual machine migration, server consolidation, energy management, data security, storage technologies, data management, software frameworks and traffic management&analysis.

Platform as a service (PaaS) is a category of cloud computing services that provides a platform allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app. There are many PaaS providers: Windows Azure, Google App Engine, VMware Cloud Foundry, Salesforce, Heroku, Amazon Elastic Beanstalk, Engine Yard Cloud, Engine Yard Orchestra, CumuLogic, etc. Table 5 compares 3 most popular Cloud platforms based on 13 features. Amazon EC2 provides the largest temporary storage and more Operation System choices. Microsoft Azure VM

provides largest memory among these three platforms. Google CE does not provide outstanding feature but the ecosystem of Google could benefit the developers.

Table 5. Azure VM vs Amazon EC2 vs Google CE: Cloud Computing Comparison [86]

	Amazon EC2	Google CE	Microsoft Azure VM
Numberof instance templates available	39	18	40
GPU acceleration	Yes	No	No
Custom instance creation feature	No	Yes	No
CPU Limits	1 – 40	1 Shared – 32 dedicated CPU	1 – 32 CPU
Memory Limits	0,5 – 244 GB	0,6 — 208 GB	0,75 — 448 GB
Temporary Storage Limits	Up to 48 TB (Multiple Disks)	3 TB	2 TB
Network feature ssupported	CDN, Direct connection, DNS, Load Balancing, Virtual private cloud network, VPN Gateway		
Number of OS supported	11	9	9
Number of Databases supported	5+	3	3
Autoscaling	Yes, clone building	Yes, clone building	Yes, presettable group
Size change	Available	Available	Available
Cloudberry Support	Yes	In progress	Yes
Free trial	Yes, time-limited on one instance	Yes, time and resource limited on any instance	Yes, time and resource limited on any instance

2.3.2. Edge Computing and Fog Computing

The de facto computing architectures can hardly meet the requirements of IoT. CISCO [87] noticed that the traditional computing architectures or even the Cloud computing cannot meet IoT timeliness requirement because data analysis needs to be very fast as IoT devices generate data constantly. For more detail, IoT computing architectures needs to meet five requirements showing in Table 6. First, IoT computing needs to minimize processing latency. Second, it needs to minimize the bandwidth consumption. Offshore oil rigs generate 500 GB of data weekly. Commercial jets generate 10 TB for every 30 minutes of flight. But network condition on these devices are relatively limited. The third requirement is about security, as IoT data is generated from the environment or the person, it carries private information about the users. These sensitive data needs to be protected properly. The reliability is also in need. Last but

not least, IoT devices usually work in an harsh environment. For example, speed sensor and surveillance camera on the railway. So geographical requirement is also taken into consideration.

Table 6. CISCO: IoT Requirements [87]

1	Minimize latency
2	Conserve network bandwidth
3	Address security concerns
4	Operate reliably
5	Collect and secure data across a wide geographic area

Mobile computing is design to meet the new requirements of IoT. There are mainly two kinds of mobile computing solution: the Edge computing and Fog computing.

Mobile Edge Computing (MEC) [10], as a form of Edge computing, has recently been proposed to address the increasing demand of network resources. MEC reduces the network workload by shifting computational efforts from the core network to the mobile edge. Traditional base stations just forward traffic deployed, but do not actively process the request of the network resources form the users. MEC deploys new elements which have computing and storage capabilities. The edge means a device deployed at the edge of the network. It could be a base station as it is the last communication infrastructure and it will forward the network to the users. After the end users send requests of a service, the traffic first comes to the base station and then is forwarded to the Internet infrastructure providers through the router. The Internet infrastructure providers will forward the requests to application service providers (ASPs), which host applications within data centers and content delivery networks. MEC equips a Mobile Edge Computing Server near the base station. The Internet infrastructure providers and the application service providers will deploy rule sets, filters and other services on MEC server, so the MEC server is capable of participating both in user traffic and control traffic. Special users requests will be processed at the edge of the network and do not need to be forwarded to centric server through core network infrastructure. Figure 6 shows the relationship between Edge servers and IoT infrastructures.

Although Cloud computing provides an efficient way for data processing as resources in the cloud are closely centralized and the server can gain access to enormous bandwidth through the core network. It cannot cope with the congestion and bandwidth limitation in user network. MEC could address this problem. Another trend is the incredible increment of the raw data produced from the users. Not all the raw data is useful, redundant raw data wastes bandwidth and increases the latency. MEC servers will consume most of the raw data at the edge instead of forwarding them to remote servers. As the MEC servers are close to users, the response time will be reduced if the data is processed from MEC server.

Nokia Siemens Networks introduced a real-world MEC platform: Radio Applications Cloud Servers (RACS) [89] in 2014. The server could run applications directly within a mobile base station. The server reduces the delivery latency by providing media-rich services from the base station.

The Fog computing pushes the processing capability down to the data resources. So that the data could be instantly processed after produced. The fog computing utilizes different strategy [87] according to the data type and processing type. Transient

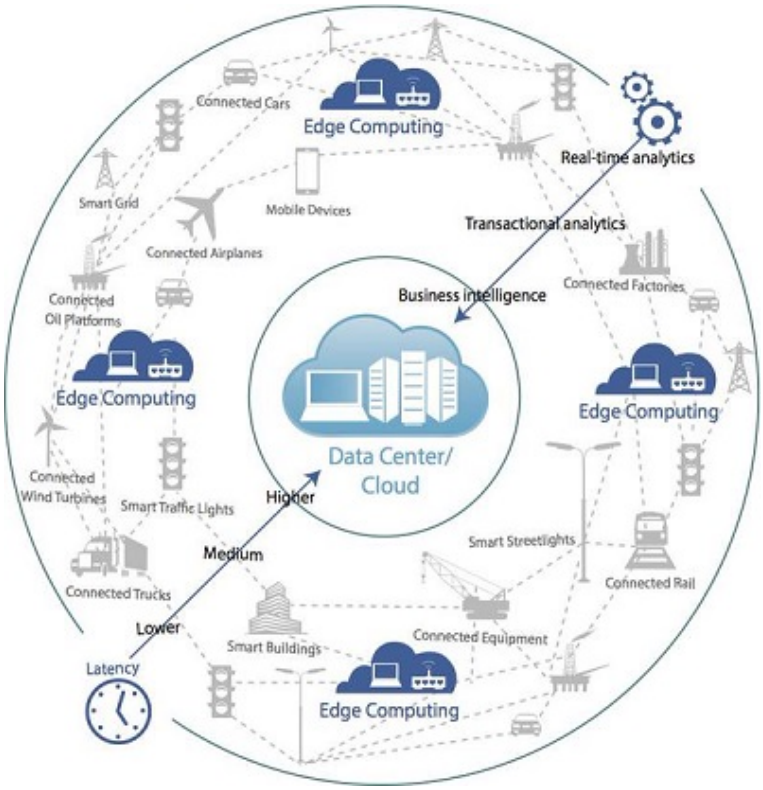

Figure 6. Illustration of Edge Computing paradigm. [88]

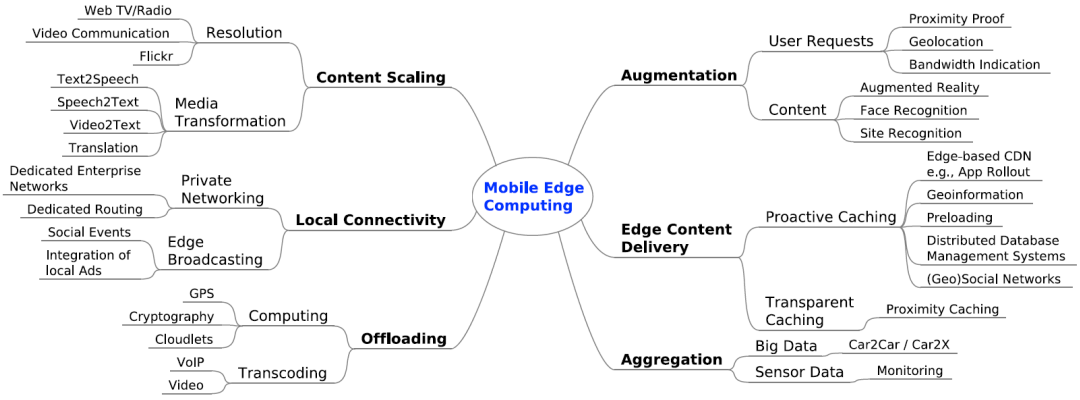

Figure 7. Mobile Edge Computing Applications and Use Cases. [10]

stored data, which need instant response, can be processed on Fog nodes closest to IoT devices, for example, M2M communication Haptics [90]. The data, which needs to be stored for short period (hours or days) and does not need to be responded directly, could be processed in Fog aggregation nodes. Data need long time store without instant response could be forwarded to Cloud server.

CISCO Fog Computing Solutions [91] provide connectivity for wide range of IoT devices, data security, time-sensitive based data processing priority strategy and automatic provision. It contains five main components: Network connectivity, Physical

and cyber security, Fog application development and hosting, Data analytics and Management&automation component. It handles an unprecedented volume, variety, and velocity of data by direct the data to right place for analysis based on time-sensitive via Fog data services.

The major difference between the Edge and Fog architectures is exactly where that intelligence and computing power is placed. Fog computing pushes intelligence down to the local area network level of network architecture, processing data on a fog node or IoT gateway. Edge computing paradigm pushes the intelligence, processing power and communication capabilities of an Edge gateway or appliance directly into MEC server devices such as programmable automation controllers (PACs).

3. SYSTEM DESIGN AND IMPLEMENTATION

This chapter describes the Edge based IoT Semantic reasoning system. It contains different layers of Cloud server, Edge and IoT devices. The ontologies and rules enable the systems to do general purpose rule-based reasoning for different tasks. For research purpose, we will separate reasoning processing into measurable independent steps. By doing this, we could study performance of Semantic IoT systems. In the following sections, we will introduce the system requirements, the scenario, the system architectures design and implementation.

3.1. System Requirements

The system aims to fully support semantic reasoning under simulated IoT environments. There are six requirements for our system:

- Scalability: A stunning amount of smart devices will be connected in the near future. CISCO researchers predicted that in 2020, 50 billions connected devices will be exited in the world and each person will own more than 6 devices [92]. The first challenge from the next evolution of the Internet is the scalability problem. To meet this requirement, our system should be able to process big amount of dynamically generated RDF data from a considerable number of devices.

- Computing Ability: In real IoT environment, the data is always generated by the end devices such as sensors aperiodically. The quantity and speed of data generating are dynamic and unpredictable. In other words, the workload of the whole system is changing all the time. So the system should have enough computing ability to cope with the instantaneous peak of heavy workload.

- Heterogeneous Processing Ability: As the heterogeneous IoT devices may deploy with different Operation System and utilize different semantic annotation methods, the system should be able to provide a cross-platform communication mechanism for cooperation between different modules and process the RDF data which has different formats.

- Semantic Processing Ability: The system should at least support popular RDF format such as RDF/XML, JSON-LD, N3, EN format, etc. In real situations, the reasoning devices such as mobile phones have relatively low memory and slow processing speed, it is impossible to store all the rules and ontologies inside these mobile devices. The ontologies and rules fetching mechanism should be utilized to enable these devices to preform various reasoning tasks.

- Measurement Accuracy: We measure the latency and the data processing time for comparing the performance, so an accurate time measurement method for independent processes is required. Keeping the running server in a stable situation with an independent monopolized system also guarantees the stable measurement.

3.2. Scenario

The continuous reduction of the smart product price makes it possible to be equipped everywhere. For efficient management, taxi companies and bus companies install the smart devices which include sensors and communication components to track their routes and deal with the customer's' queries. The taxi cabs around the City of Oulu have been installed the smart system which include GPS and related software. The system can be used to analyze the annotated locations and driving information from the taxi cabs. The taxis will upload the information to either Cloud Reasoning Server or Edge nodes to do semantic reasoning.

We collect the raw data from real taxi trajectories. The raw data is in XML format and stored in SQLite database. When the GPS sensors of the taxi generate a new value, we package the data into an XML file and call it an individual record or individual data. One individual data consists of 9 properties: an ID of this observation record; a date when the data is generated; an ID of the area where the taxi located; a coordination includes longitude and latitude; a velocity of the taxi; a direction degree; the manual information and a sender ID. To satisfy the requirement of scalability, we extent the raw data into 200 taxis and each taxi has 800 individual data. The raw data is converted to four RDF format: RDF/XML, JSON-LD, N3 and EN.

In Appendix A, we list the same RDF data in four formats: the XML raw data, the RDF/XML data, the JSON-LD data and the EN short package format data.

No.	Attributes	Raw Data	RDF/XML&JSON-LD	Entity Notation
1	ID	✓	✓	✓
2	Time	✓	✓	✓
3	Area	✓	✓	✓
4	Latitude	✓	✓	✓
5	Longitude	✓	✓	✓
6	Velocity	✓	✓	✓
7	Direction	✓	✓	✓
8	ManualInfo	✓	✗	✗
9	Sender	✓	✓	✓
10	TimeStamp	✗	✓	✓
11	Acceleration	✗	✓	✓
12	Distance	✗	✓	✓

Table 7. The Comparison of Different RDF data

The original data set has 9 attributes. For privacy issue, the original raw data uses inaccurate date time. The RDF data generates more accurate timestamps and calculates the acceleration and distance information for simulation and reasoning purpose. The ManualInfo is not necessary for the reasoning and removed from RDF data. So there are 11 attributes in RDF/XML, JSON-LD, N3 and EN data. The difference between RDF formats display in Table 7.

For the EN short packet [93] formatted data, the short formatted data need extra information to be converted to RDF data. The RDF data could be separated as two parts: the A-BOX for individual data and the T-BOX for general knowledge, namely the ontology. The EN short package is mainly designed for individual RDF data. The main

purpose behind the design is to compress the data size by replacing constant information with templates and prefixes. The order in the sequence determine what property it belongs to. For example, as the example in Appendix 1.4 shows, the individual data has 12 attributes. The first one is the UUID of the short package, the rest in this case are properties values of the individual. The corresponding position shows in the Table 8.

Position In EN	Attribute	URI	Datatype
1	ID	obs:hasID	Int
2	Area	obs:hasArea	Int
3	Latitude	obs:hasLatitude	Double
4	Longitude	obs:hasLongitude	Double
5	Velocity	obs:hasVelocity	Double
6	Direction	obs:hasDirection	Int
7	Sender	obs:hasSender	Int
8	Distance	obs:hasDistance	Double
9	Acceleration	obs:hasAcceleration	Double
10	Date	obs:hasDate	Int
11	Time	obs:hasDataTime	Date

Table 8. Properties in EN Short Packet Data

The ontology used in these system is based on Maarala's experiment [94]. There are 12 facts which could be deducted form the rules. 29 rules are used for inference. Table 9 shows the rules.

Table 9. Rules for Oulu Taxi Scenario

Facts	Description
JamSpeed1	If Observation has velocity between 1.0 and 25.0, then it is JamSpeed1.
JamSpeed2	If JamSpeed1 continusly to be JamSpeed1, it becomes JamSpeed2.
JamSpeed3	If JamSpeed2 continusly to be JamSpeed2, it becomes JamSpeed3.
JamSpeed4	If JamSpeed3 continusly to be JamSpeed3, it becomes JamSpeed4.
LowSpeed	If Observation has velocity less than 30.0, then it is with lowSpeed.
Jam	If LowSpeed has durition for longer than 90s, and the average speed of two Observations is less than 40, then the last Obersavation is Jam.
Jam	If LowSpeed has durition for longer than 90s, and the average speed of three Observations is less than 60, then the last Obersavation is Jam.
Jam	If LowSpeed has durition for longer than 90s, and the average speed of four Observations is less than 80, then the last Obersavation is Jam.
Stop1	If Observation has velocity less than 3.0, then it is with Stop1.
Stop2	If Stop1 contiues, it will become Stop2.
Stop3	If Stop2 contiues, it will become Stop3.
Stop4	If Stop3 contiues, it will become Stop4.
LongStop	If Stop has durtion longer than 60s from Stop1 to Stop2, then it is LongStop.
LongStop	If Stop has durtion longer than 60s from Stop1 to Stop4, then it is LongStop.
LongStop	If Stop has durtion longer than 60s from Stop1 to Stop2 to Stop3, then it is LongStop.
VeryLongStop	If LongStop has durtion longer than 180s then it is VeryLongStop.
HighAvgSpeed1	If Observation has velocity greater than 75, then it is HighAvgSpeed1.
HighAvgSpeed2	If HighAvgSpeed1 continusly to be HighAvgSpeed1, it becomes HighAvgSpeed2.
HighAvgSpeed3	If HighAvgSpeed2 continusly to be HighAvgSpeed2, it becomes HighAvgSpeed3.
HighAvgSpeed4	If HighAvgSpeed3 continusly to be HighAvgSpeed3, it becomes HighAvgSpeed4.
HighAvgSpeed	If Observations experience HighAvgSpeed1, HighAvgSpeed2, HighAvgSpeed3 and HighAvgSpeed4 for longer than 120s, it becomes HighAvgSpeed.
RightTurn	If LowSpeed Obeservation moves and the direction turn right (by plusing degree), It becomes RightTurn.
RightTurn	If LowSpeed Obeservation moves and the direction turn right (by minusing degree), It becomes RightTurn.
LeftTurn	If LowSpeed Obeservation moves and the direction turn left (by plusing degree), It becomes LeftTurn.
LeftTurn	If LowSpeed Obeservation moves and the direction turn left (by minusing degree), It becomes LeftTurn.
U Turn	If LowSpeed Obeservation moves back (by plusing degree), It becomes U turn.
U Turn	If LowSpeed Obeservation moves back (by minusing degree), It becomes U turn.
High Deacceleration	If Observation has Acceleration less than -1.5, then it becomes HighDeacceleration.
High Acceleration	If Observation has Acceleration more than 1.4, then it becomes HighAcceleration.

3.3. System Architecture

The system is designed for heterogeneous IoT devices to communicate, transfer data and execute semantic reasoning tasks. IoT devices at lower level collect sensor data and encode the raw data into RDF format. Then IoT devices deliver the RDF data to capable devices, which could execute semantic reasoning.

Inspired by multi-layer architecture for semantic reasoning [95], we provide two solutions for performing semantic reasoning in IoT. the One solution is to deploy semantic reasoner on Cloud server. Cloud computing provides cost-efficient stable high-performance computing and storage services [96]. But considering about the real situation, the lower level IoT devices are physically far away from the Cloud Server, which need to spend more time for communication. So we consider using near-by devices for computing. Smart devices have relatively high computing capability, which could be used as Edge nodes between IoT low level devices and Cloud Server.

Based on the requirements, two architectures are designed for the comparison of semantic reasoning performance. The first architecture directly connects the terminal IoT devices and powerful semantic reasoning resources on the Cloud server. We called the first architecture "Cloud Reasoning Architecture". This is a most typical architecture in IoT environments. Considering the heterogeneous types of devices in real world, some of them have less capability of semantic reasoning, and some has relatively powerful processing ability. Hence it is reasonable to share some workload to the terminal devices for faster processing. The second architecture is designed for this purpose. Relatively powerful computing resources inside or near IoT terminal devices network area are chosen to be "Edge Nodes", which could be used as reasoners beside the Cloud and located close to the terminal devices. The rest of the powerless IoT terminal devices are called "IoT Node". We called this architecture "Edge Reasoning Architecture".

The Cloud Reasoning Architecture, which is presented in Figure 8, separates the system into two parts: a central reasoning computer (Cloud server) and several IoT nodes. IoT nodes could be all kinds of devices include sensor, single-chip, mobile phone, etc. IoT nodes could generate raw data and encode them into RDF format. IoT node could convert the raw data into four RDF formats: the RDF/XML, JSON-LD, N3, and EN. Then IoT node send the RDF data to Cloud server through TCP/IP protocol. In the Cloud server, there are two main components: a semantic reasoner and a RDF database. The semantic reasoner could receive the RDF data and perform rule-based reasoning. Rules and Ontologies should also be stored in the Cloud server. After reasoning, the results, which include the individual RDF data with new tags, will be stored in the RDF database. MQTT, as a lightweight messaging protocol for small sensors and mobile devices in IoT environment, is used to provide publish-subscribe communication.

The Edge Reasoning Architecture, showed in Figure 9, adds Edge layer between IoT nodes and Cloud server. The Edge nodes are sets of devices physically near IoT nodes and they have reasoning capability. The purposes of the Edge nodes are to provide faster reasoning and to reduce the workload of the central reasoning computer. In terms of the network situation and bandwidth limitation of the terminal devices, latency will affect the overall performance. Considering the limitation of the Edge node, the Edge nodes will only take part of the reasoning tasks. The special rules and

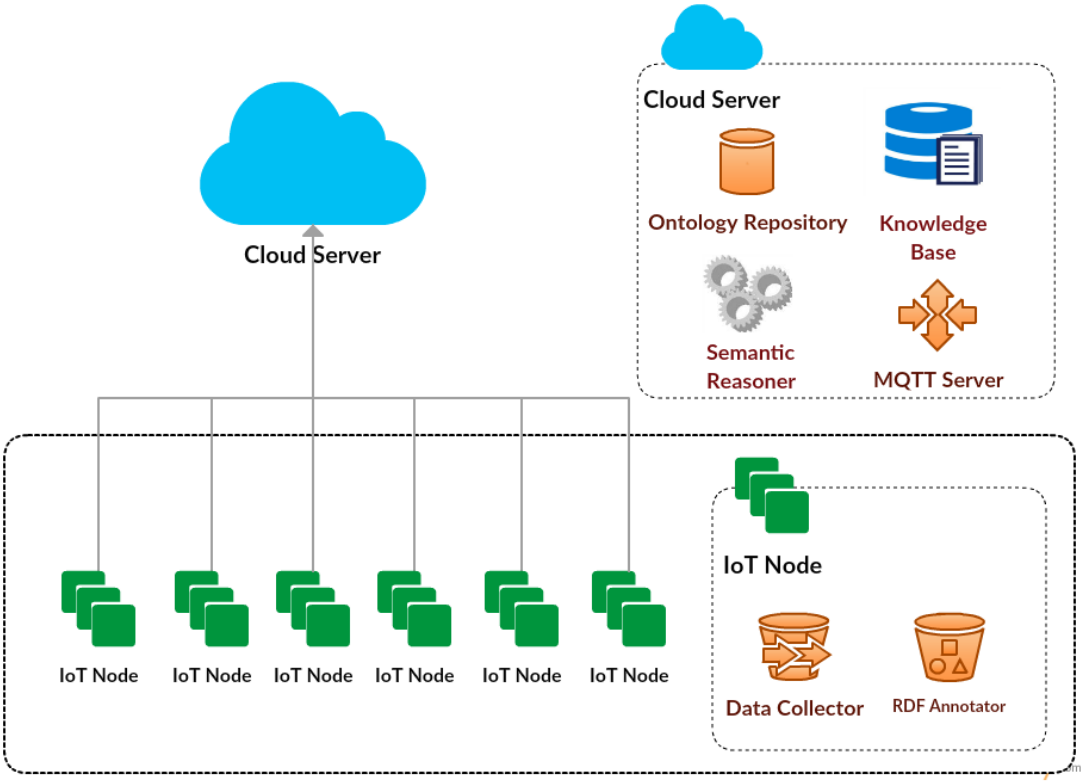

Figure 8. Cloud Reasoning Architecture.

ontologies will be designed for each case and the data files are stored in the Edge nodes' storage. The reasoners will be implemented in the Edge nodes as well. The reasoners in Edge nodes and Cloud server provide the same function including processing different RDF format, etc. The Edge nodes communicate with the Cloud server uses TCP/IP protocol. The Edge nodes and IoT nodes use socket for communication. All the four RDF formats should be supported in Edge nodes.

An upgraded version of the second architecture aims to make the Edge reasoning more flexible. We called it "Ontology Enabled Edge Reasoning Architecture", which uses EN Schema to transform Ontology. The architecture is showed in Figure 10. The transformed ontologies could be part of the original ontologies as the fact that some of the ontologies are quite heavy while the rule-based reasoning only use part of it. There is one component called Semantic Server which is responsible for providing ontology and rules to the reasoning devices. The Semantic Server is a flexible component which could be deployed on Cloud, Edge or other servers in real implementation. For better communication performance, the RDF data is transformed into EN short package format [65].

The software components on taxi cabs receive GPS signal from hardware and annotate them into RDF format. Due to the limitation of the computing resources on the taxi, IoT nodes will deliver the data to the more powerful devices. It has two choices, first, it could deliver data to a specialized server which could do semantic reasoning on it; second, it could seek for Edge nodes, for example, the smart phones or pads of the drivers and passengers. The IoT nodes could send all the RDF data to the server,

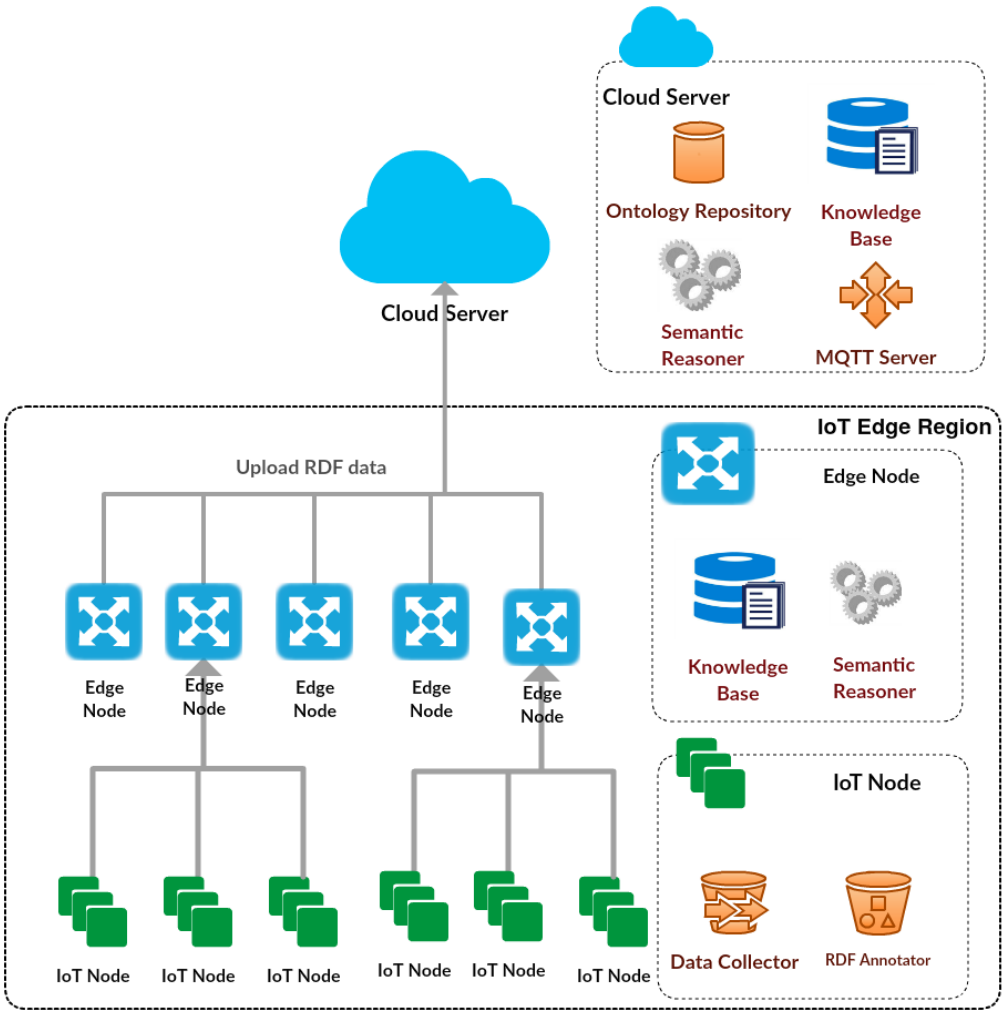

Figure 9. Edge Reasoning Architecture.

but they would be limited by the unstable bandwidth from the fast moving cars and the server may receive much useless data if we consider the situation that the car are in traffic jam and stay for hours, the GPS information will be the same while they will still send similar data to server. Another solution is to use nearby computers to do pre-processing or share some reasoning task of the Cloud Server. Android mobile phones could be used as an Edge node in the middle of car computer. We called the car computer the IoT node. The Edge nodes will share part of the reasoning tasks and then send the results to Cloud Server if necessary.

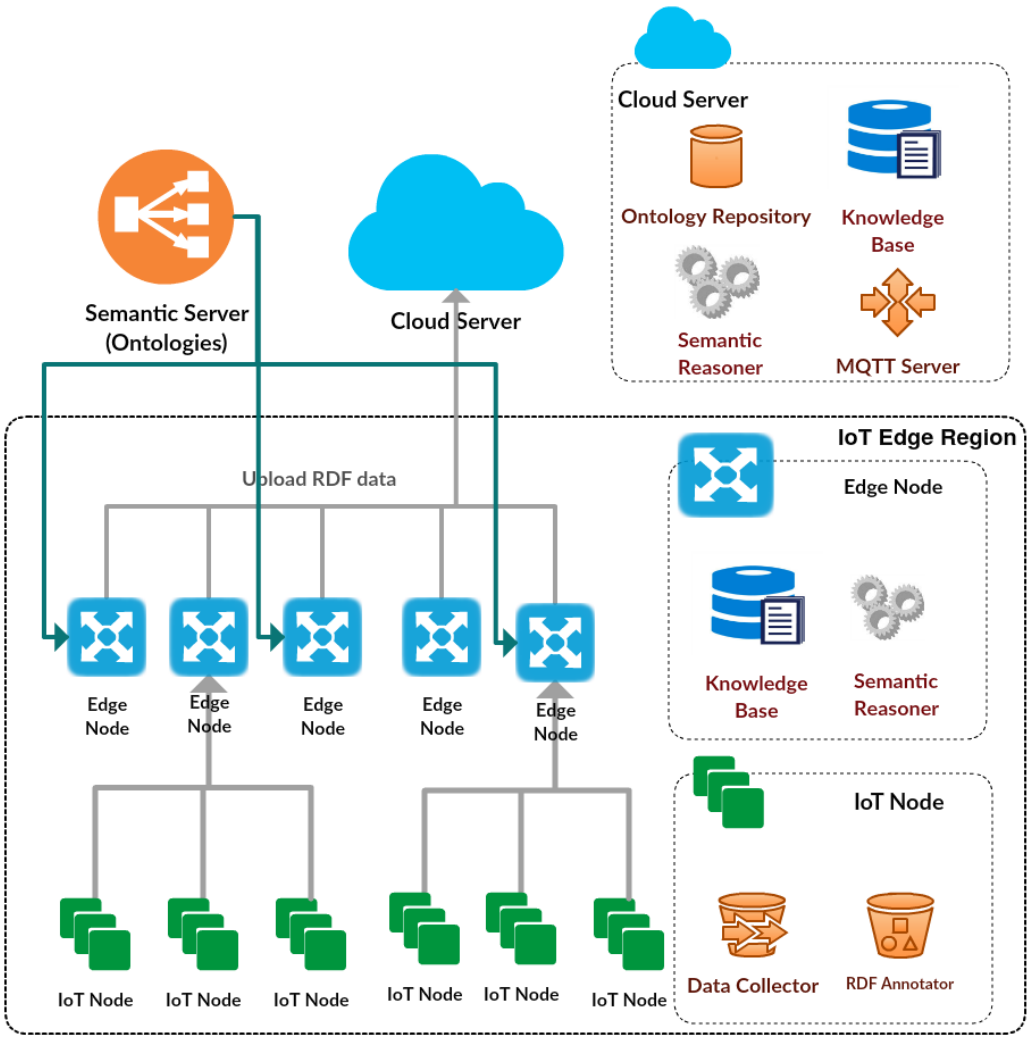

Figure 10. Ontology Enabled Edge Reasoning Architecture.

3.4. Implementation

3.4.1. Hardware

IoT nodes are simulated by a desktop computer. The module is HP Desktop PC Elite Desk, which has Intel Core i5 4590 (3.30 GHz) CPU and 8GB memory. The Edge nodes used LG Nexus 5X phones with Android OS version 6.0.1. The Nexus 5X has 6 CPU cores in total, which consists of four Quad-core 1.44 GHz Cortex-A53 processors and two dual-core 1.82 GHz Cortex-A57 processors, running on Qualcomm MSM8992 Snapdragon 808 chip-set. It has 2GB RAM and 32GB storage. The Cloud Server uses Amazon M4 Deca Extra Large Cloud. It has 160 GB memory with 124.5 EC2 Compute Units. One EC2 Compute Unit provides the equivalent CPU capability to 1.0 1.2 GHz 2007 Xeon processor. This 64-bit system has max bandwidth of 4000 Mbps. The server is located in Frankfurt, Germany.

3.4.2. Software

IoT Node

IoT nodes simulate the software components in the end-point IoT devices. These devices could collect data from the environments but have limited computing capability. IoT nodes have three main functions: the data collection, the data encoding and communication. The data encoding means to encode the raw data into RDF format.

For simulation purpose, IoT nodes have 8 communication modes: sending RDF/XML data to Cloud; sending JSON-LD data to Cloud; sending N3 data to Cloud; sending EN data to Cloud;sending RDF/XML data to Edge; sending JSON-LD data to Edge; sending N3 data to Edge; sending EN data to Edge. IoT nodes maintain an IP address lists locally. For both destinations of the Cloud server and Edge nodes, they could encode the data into four formats. They use MQTT [97] protocol to communicate with Cloud server and use Socket for Edges. All IoT nodes could be executed simultaneously in one experimental computer using threads. As the sensors generate data in really a high frequency, IoT nodes will catch a certain amount of data (50 individual RDF data for one package) and then send them together. We measure the time from generating the RDF data to receiving response from the Cloud or Edge nodes as a conformation of the sending message. So IoT nodes mainly works with 4 steps: reading the data, packaging the data, sending the data and waiting for response.

Edge Node

We deploy the Edge component on Android devices which include Android phones and Android pads. IoT Edge nodes have three functions: receiving data from IoT node, semantic reasoning and sending data to Cloud server. It implements Socket and MQTT client for communication and Jena framework for semantic reasoning. It has two working modes: in the first mode, the Edge nodes store the Ontology locally; in the second mode, Edge nodes use HTTP request to fetch Ontology from remote Ontology Repository Server, which could transfer part of the whole Ontology. The work flow of Edge nodes are showed in Figure11. First, the Edge nodes are waiting for RDF data coming from IoT nodes. Then they could seek for local repository for Ontology or request from Semantic server via HTTP. Then they execute reasoning and send the data to Cloud server simultaneously.

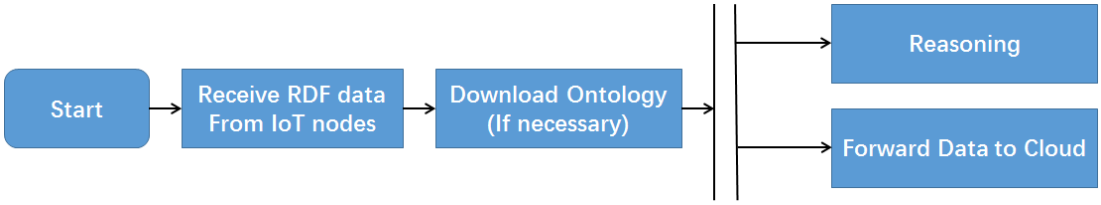

Figure 11. Work Flow of The Edge Node.

Figure 12 shows the software structure of the Edge node. Socket client is used for receiving RDF data and MQTT client is used for forwarding RDF data to the Cloud server. All the RDF data will be stored into Jena Model. A rule-based reasoner is

build with Jena Generic Rule Reasoner. More specifically, we used Hybrid rule engine in Jena Framework. The reasoner imports the Ontology and Rules from either local storage or remote Semantic server.

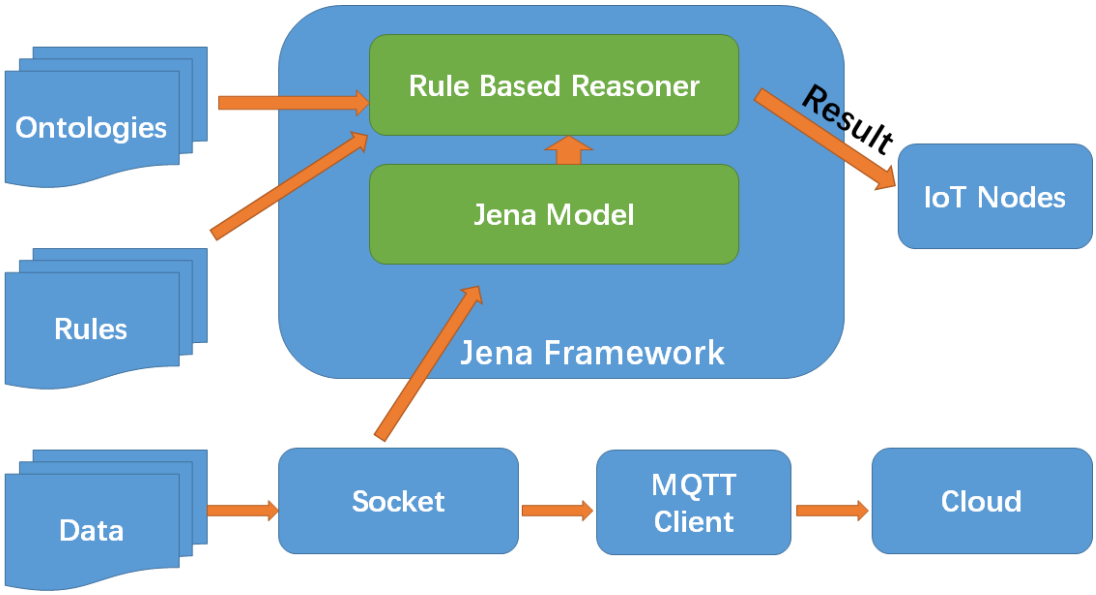

Figure 12. Edge Node Structure.

Cloud Server

The Cloud server implements MQTT [97] server for communication and Jena Framework for semantic reasoning. It also has web services form providing Ontology for Edge nodes. For experimental purpose, the Ontology Repository management function is deployed in Cloud server while it could be deployed on other platform for performance and flexibility.

3.4.3. EN Schema Parser

The Entity Notation Schema (EN Schema) [93] parser is design and implemented for convert common OWL 2 ontology into EN Schema format. Most popular OWL 2 formats are: Functional Syntax [98], Turtle [56], Manchester Syntax [61], RDF/XML [99], JSON-LD [56], N-TRIPLE [56] and OWL/XML [56]. These syntax of ontologies could be converted between each other. We choose Turtle as the OWL 2 syntax for EN Schema Parser. RDF/XML and OWL/XML are based on XML format, the XML syntax makes it clumsy for Ontology representation. JSON-LD also has this shortcoming as it is based on JSON format.

Turtle is based on triples and provides levels of compatibility with the N-Triples. A Turtle document is a textual representation of an RDF graph. It is a Unicode [100] character string encoded in UTF-8 [101]. Unicode characters only in the range U+0000 to U+10FFFF inclusive are allowed. To parse the Turtle to EN Schema, first we need to analyze the Turtle grammar elements.

There are three main parts in Turtle: the comments, the name space information and the triples. White space is widely used to separate two tokens, namely any kind of element. Turtle use “#” to comment the rest of the whole line. For the name space, you can define the base name space and the name space for special prefix. The triple statement consists of 3 elements: subject, predicate, object. Each element is separated by whitespace and the whole statement is terminated by “.’. To identify a subject, predicate or object, Turtle uses IRIs. IRIs is a generalization of URIs that permits a wider range of Unicode characters. Every absolute URI and URL is an IRI, but not every IRI is an URI. Relative or absolute IRIs or prefixed names could be used as IRIs. Relative and absolute IRIs are enclosed in “<” and “>” and may contain numeric escape sequences. Relative IRIs are resolved relative to the current base IRI. The token “a” in the predicate position of a Turtle triple represents the IRI http://www.w3.org/1999/02/22-rdf-syntax-ns#type . The name with prefix is an alternative way to representation resources and objects. This name consists of a prefix label and a local part separating by character “:”. We could first define a prefix “@prefix example: <http://example.org/#>.” and use “example:phones” to represent the same IRIs. Unlabeled blank node could be used for representing fresh RDF blank node, used in both subject and object position represented by “[]”. In side the Square brackets, there could be a predicate lists. The examples are presented in Appendix 1.5.1 Example 1 and 2. The predicate list contains predicates and corresponding objects. Different predicates are separated by “;”. See Appendix 1.5.1 Example 3. Object could be an object list, each object is separated by “,”. The examples are presented in Appendix 1.5.1 Example 3 and 4. Literals, namely the plain texts, may be used as an object and may have a language tag, a datatype IRI, or neither. the language tag is preceded by a “@” (U+0040). For example “@en” indicates the text is in English.The datatype IRIs is fellow the literal and separated by “^^” and it could be an absolute IRI [22], relative IRI [22] or prefixed name. The literals need to be quoted by one of the delimiters: “’” and ““”, which mean a single line literal; “’’’” and ““”””, which mean multi-line literal.

Table 10. Elements for The Parser

No.	Element	Syntax
1	Name Space Defination	@prefix
2	Name Space Defination	@base
3	Separator	White Space ” ”
4	IRIs	“<” and “>”
5	Prefixed Name	Prefix + “:” + Loacal Part
6	End of Triple	“.”
7	Separator of Predicate	“;”
8	Separator of Object List	“,”
9	Collection	“(” and “)”
10	Fresh RDF Node	“[” and “]”
11	Literal Quote	“’” or ““” or “’’’” or ““”””
12	Datatype separator	“^^”
13	Language Tag Separator	“@”
14	Comment	“#”

There are four kinds of situations for the start of the Turtle file. First, useless white space and tab will be the first character. Second, “” will be the first letter. It means the rest part is either “prifix” or “base”. Third case is that the “#” is the first letter and the rest of the whole lines is comment. The last case contains all the rest situation, which means the annotation part of the Turtle file has ended and triples description begins.

For the last case, it could be decided into sub-cases. First there may be an IRIs starting by “<” and ending by “>”. Second, it could be a name with or without prefix. Third it could be an blank RDF node represented by “[]”.

Collection in Turtle will be converted into EN Collection, which is a special entity with items inside.

By doing this, RDF elements such as Subject, Predicate and Object could be recognized and Separated out of the string to build the entity. Anonymous class or individual will become named entity, in the current parser, it will be named as “Anonymous_” plus with special ID. For example, “Anonymous_102”. The prefix of the anonymous class is used for reconverting EN Schema to Turtle and emit the anonymous class and individual definition. Another case is the anonymous class which should be existed when converting to Turtle. We named this object like “SystemAnony_12”.

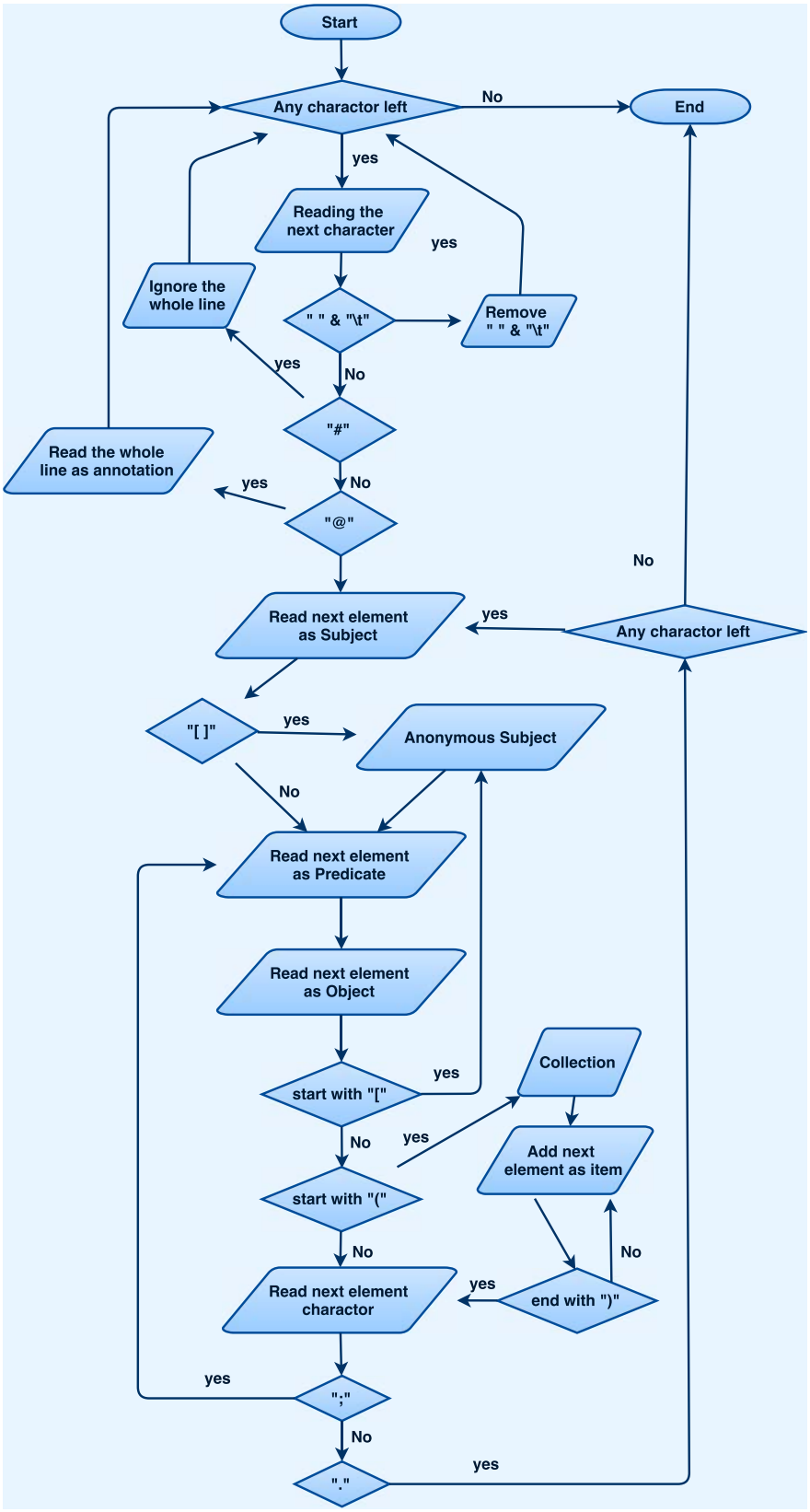

Figure 13. Parser Working Flow.

4. EXPERIMENT & ANALYSIS

This chapter describes the experiments setup, result and analysis. Five experiments are designed to study the performance and related issue of the Edge based IoT systems. The experiments show the influence of the RDF syntax, computing architectures and the task distribution to the performance of the system.

4.1. Setup

The experiments consist of three parts: IoT nodes, the Edge nodes and the cloud server. There are 20-100 IoT nodes simulated by a desktop computer as low-level sensors. There are totally 10 Edge nodes and each of them is deployed on an Android mobile device. The mobile devices have at least 1.5GHz CPU and 1GB memory. The cloud server is deployed on Amazon EC2 Cloud platform. The model of the Amazon cloud server is m4.10xlarge, which contains 40 vCPU, 160GB memory and 4000 Dedicated EBS Bandwidth, located in Germany.

The data is collected from GPS devices of driving taxi cabs around downtown of Oulu. The original data set is from 65,000 separate taxi trajectories including 5,543,348 observations producing 72,063,524 RDF triples. We extent the data to 200 cars and each car with 800 individual observation data. GPS data has highly dynamic spatial and temporal characteristics, thus it is appropriate for testing reasoning and data delivery on IoT environment. Each individual observation data contains 11 elements: the longitude, the latitude, the timestamps, the date, the direction, the velocity, the acceleration, the area, the sender name, the ID and the distance. The data are transformed into RDF/XML, JSON-LD, N3 and EN from XML. The rule-based reasoning is supported by Jena Framework. There are 29 rules in total stored in rule file with Jena rule format and there is an Observation ontology file.

4.2. Experiment

We design five experiments. The first experiment, introduced in Section 4.2.1, analyzes the size of the ontologies with different syntax. The second experiment, introduced in Section 4.2.2, analyzes the processing time based on different RDF syntax. The third experiment, introduced in Section 4.2.3, tests the reasoning by Cloud computing. In the fourth experiment, we test reasoning by Edge computing system. We will introduce it in Section 4.2.4. In the last experiment, we distribute different tasks on Edge nodes and test the performance difference. This will be introduced in Section 4.2.5.

We present the results in eight sections. First section presents the comparison of different ontologies size with different formats. Semantic technologies annotate raw data with structure information. Different RDF formats will result in different data sizes. The data size is an essential factor which affects the bandwidth usage and data transferring of the Edge based IoT system. The second section presents the comparison of ontology Model loading processing time among different formatted data. We choose Jena as the inference engine. In a real semantic IoT system, the server will receive RDF data with a specific format and directly process with this format. Then we will present

four groups of scalability comparison for four selected RDF formats. After combining semantic technologies with IoT, whether the semantic process will affect scalability should be taken into consideration. In the next section, we presents a comparison between two distributed computing architectures: one architecture with Cloud Computing and the other one with Edge computing paradigm. This comparison measures two architectures within the complete data delivery and reasoning processes. The next sub-section presents the performance comparison of Edge computing paradigm and figure out what factors will affect the overall performance and whether the Edge structure will improve the overall performance comparing with Cloud computing. In the last sub-section, we aim to research what kind of tasks distributing strategies between the Cloud and Edge nodes could lead the best overall performance.

4.2.1. Ontology Size Comparison

The first experiment analyzes the different ontology formats by measuring the ontology size. As we added EN Schema, which is a new OWL 2 representation, we also design experiments to compare the Complete EN Schema packet with other knowledge representation method with a set of eight ontologies. The tested ontologies are well known and widely utilized in pervasive computing, IoT, and other domains. Each ontology will be converted into 8 OWL syntax include: Functional Syntax, EN Schema, Turtle, Manchester syntax, RDF/XML, JSON-LD, N-Triple and OWL/XML.

These ontologies include COBRA-ONT [102], IoT-Lite ontology [103], Socially Interconnected Online Communities (SIOC) ontology [104], Semantic Sensor Network (SSN) [105] and the Organization Ontology [106]. COBRA-ONT includes a collection of ontologies for describing vocabularies in an intelligent meeting room use case. COBRA-ONT ontologies (Version 0.6) contains Ebiquity-geo, Ebiquity-meetings and Ebiquity-actions ontologies. IoT-Lite ontology is a lightweight ontology to represent IoT resources, entities and services. It is also a meta ontology that can be extended in different domains. The SSN ontology describes the sensors, observations and related concepts in pervasive environment. SIOC ontology is designed for describing online communities such as forums, blogs, mailing list and wikis. The Organization Ontology is designed to enable publication of information on organizations and organizational structure including governmental organizations, etc. It is designed as a generic, reusable core ontology that can be extended or specialized in particular situations.

In Figure 14, the result shows the ontology data size of eight ontologies with eight formats. The X-axis represents the format and the Y-axis represents the data size in bytes. Functional Syntax, EN Schema, Turtle and Manchester syntax shows better performance of the data size than average syntax of the other four. JSON-LD, N-Triple and OWL/XML appear to be the highest format. The RDF/XML is always with the middle size. OWL/XML is the biggest ontology in four out of eight ontologies.

In IoT-Lite Ontology, The JSON-LD formatted data is 5.2 times than the Manchester syntax formatted ones. In Ebiquity-meeting Ontology, the ratio is 2.3 which is the smallest among all the sets. The average ratio between the biggest format and the smallest one is 3.3 times.

Figure 14. Length Comparison of Eight Widely Used Ontologies.

4.2.2. Ontology Model Loading Performance Comparison

We evaluate the loading performance of different ontology based on Jena Framework. We also use the previous eight ontologies with eight formats. In Jena Framework, a RDF graph is called a model and is represented by the "Model" interface. Jena has "Object" class to represent graphs, resources, properties and literals. The interfaces

representing resources, properties and literals are called Resource, Property and Literal respectively. Jena loads ontologies in the instance of "OntModel" class, which is an extension to the "InfModel" interface. This class does not by itself compute the deductive extension of the graph under the semantic rules of the language. Instead, Jena wraps an underlying model with this ontology interface, that presents a convenience syntax for accessing the language elements. Depending on the inference capability of the underlying model, OntModel will appear to contain triples at the same level.We compare the performance of different RDF syntax by measuring OntModel instance building time.

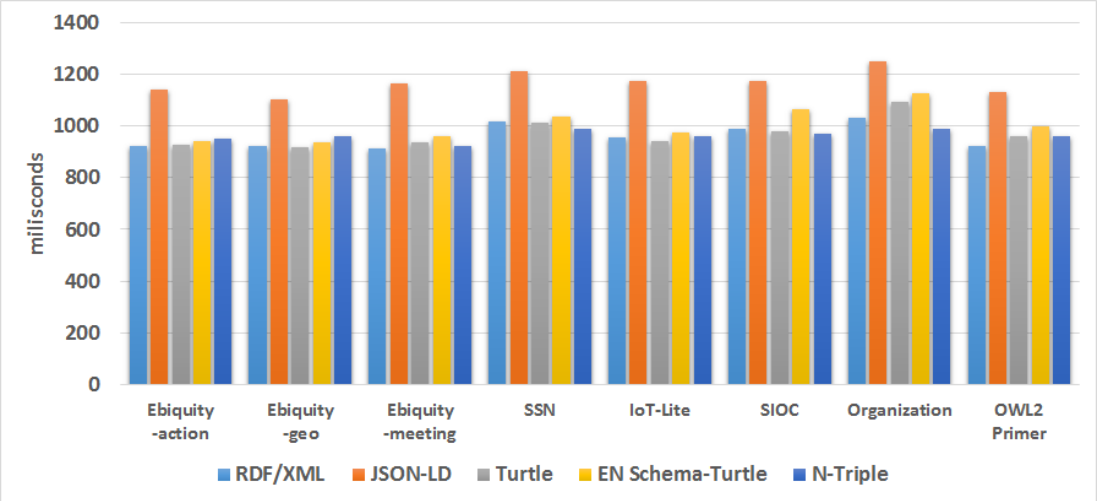

Figure 15. Ontology Model Loading Time on Cloud.

Figure 15 presents the Jena OntModel loading time among different ontologies with five formats. The yellow bar "EN Schema-Turtle" means this is originally formatted with EN Schema but as the Jena Framework cannot directly support EN Schema, we convert the EN Schema to Turtle. The converting time is around 20 milliseconds. We could find out that all different syntax makes minimum efforts to the ontology loading time except the JSON-LD: JSON-LD use longer time than the rest four formats and the rest four formats almost consume the same amount of time. RDF/XML, Turtle, EN Schema and N-Triple appear to be the shortest format. In Ebiquity-meeting Ontology, the JSON-LD format loading time is 27% longer than the shortest RDF/XML one. JSON-LD loading time is 23% longer than the smallest one for all the test cases in average.

4.2.3. Scalability Comparison

This set of experiments tests the scalability of semantic reasoning in the Edge based IoT system. In this experiment, there are IoT nodes and the Cloud server. IoT nodes send data to the Cloud server. As IoT node will generate real time ambient data, we ignore the data generating time. We only measure the total data transforming time. The transforming time starts from building the MQTT client to set up the connection and end with receiving the response. The response is not necessary for the system

but useful for experimental measurement. It is used to confirm that the information is delivered successfully. At the Cloud server part, we measure the time of semantic reasoning and RDF database storage. The measurement starts form building Jena Model, which is used for RDF and ontology, and ends with finishing the storage the inferred facts in RDF format. The Cloud server will reason the RDF data with all the 29 rules. New facts generated after reasoning will be stored into RDF database, the facts are: “RightTurn", “LeftTurn", “UTurn", “Jam", “HighAvgSpeed", “LongStop", “HighAcceleration", “HighDeacceleration", “VeryLongStop” and “LowSpeed”.

Table 11. Semantic Reasoning Experiments Setup for Edge Nodes

Group	No.	Edge node Number	nodes per Edge node	RDF per node	Total RDF data
A	1	2	10	400	8000
	2	4	10	400	16000
	3	6	10	400	24000
	4	8	10	400	32000
	5	10	10	400	40000
B	6	4	10	800	32000
	7	6	10	533	32000
	8	8	10	400	32000
	9	10	10	320	32000
C	10	6	10	533	32000
	11	6	15	355	32000
	12	6	20	266	32000
	13	6	25	213	32000
D	14	6	10	200	12000
	15	6	10	400	24000
	16	6	10	600	36000
	17	6	10	800	48000

We design 17 semantic reasoning test cases on Cloud Reasoning Architecture and Edge Reasoning Architecture. The experiment for Cloud is in Figure 12 and the experiment for Egde nodes is in Figure 11. The test cases have three variables: “IoT node Number” is the total number of the IoT nodes ; “Edge node Number” is the number of the Edge nodes; “nodes per Edge node” means how many IoT nodes connect to an Edge node; “RDF per nodes” means how many RDF observation statements are sent from one IoT node. For example, Task No.1 on Edge Reasoning Architecture means there are two Edge nodes and 20 IoT nodes. 10 IoT nodes connect to the first Edge node and the rest 10 nodes connect to the second Edge node. Each IoT node sends 400 observation RDF statements hence there are 8000 observation RDF statements in total. The corresponding test case for Cloud Reasoning still has 20 IoT nodes. The 20 IoT nodes will send 8000 RDF statements directly to Cloud server. In Edge Reasoning Architecture experiment, Group A changes the “Edge node Number” and keeps other parameters constant. The “Edge node Number” increases from 2 to 10. Group B keeps the total RDF statements with a constant value and changes the “Edge node Number” and “RDF per nodes”. Group C keeps the total RDF statements and the “Edge node Number” in constant and changes the rest variables. Group D keeps both “Edge node

Table 12. Semantic Reasoning Experiments Setup for Cloud

Group	No.	IoT nodes Number	RDF per node	Total RDF data
A	1	20	400	8000
	2	40	400	16000
	3	60	400	24000
	4	80	400	32000
	5	100	400	40000
B	6	40	800	32000
	7	60	533	32000
	8	80	400	32000
	9	100	320	32000
C	10	60	533	32000
	11	90	355	32000
	12	120	266	32000
	13	150	213	32000
D	14	60	200	12000
	15	60	400	24000
	16	60	600	36000
	17	60	800	48000

number" and "nodes per Edge node". The "RDF per nodes" changes from 200 RDF statements to 800. The minimum RDF data is 8000 RDF statements and the maximum is 48000 in total. In Cloud Reasoning Architecture experiment, as the test cases is corresponding to the Cloud Reasoning Architecture experiment by test case number, there are few redundant cases.

We perform all the test cases on Cloud Reasoning Architecture in this section. For each group, we present 2 figures, one for data transferring time and one for reasoning time. The data transferring time is from IoT nodes to Cloud server. The reasoning time is calculated from the Cloud server. For the same experiment, the results of different format are grouped by RDF data quantity.

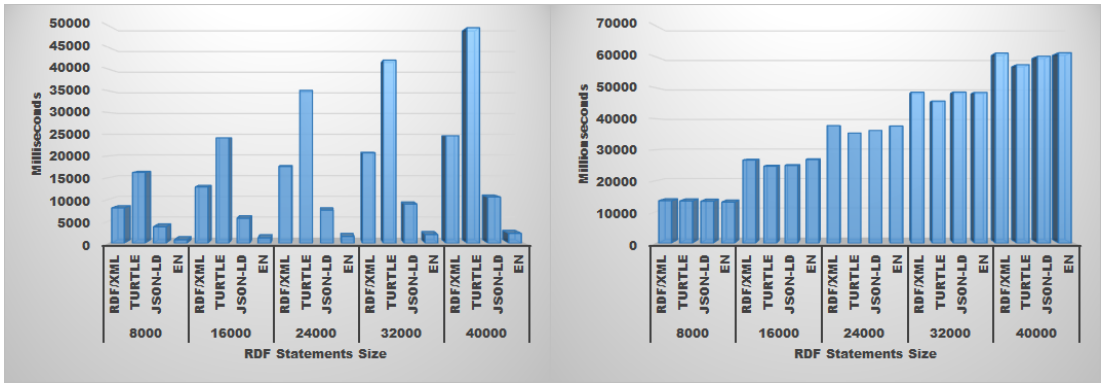

Figure 16. Scalability Results for Group A (Left:Transferring, Right:Reasoning).

In Figure 16, the left figure shows the result of Group A data transferring time comparison and the right one shows the Group A reasoning time comparison. The X-axis

represents the total data size. For example, “8000” means 8000 RDF statements in total. The Y-axis represents the time consumption in millisecond. The data transferring time increases with the rise of the total RDF data size. Comparing with four data formats, data with EN syntax consumes the shortest time, The JSON-LD is the second shortest, the RDF/XML is the third one and the Turtle format is the longest one. In test case “24000 RDF data" (No.3), the transferring time of Turtle data is 22 times longer than EN format. The transferring time of JSON-LD format is 4.7 times longer than EN in average, the transferring time of RDF/XML format is 10.7 times longer than EN in average and the transferring time of Turtle format is 21.2 times longer than EN in average. In the reasoning time comparison, the total reasoning time growth almost linearly when the data size increase. For the same amount of data, different formatted data shows equal performance.

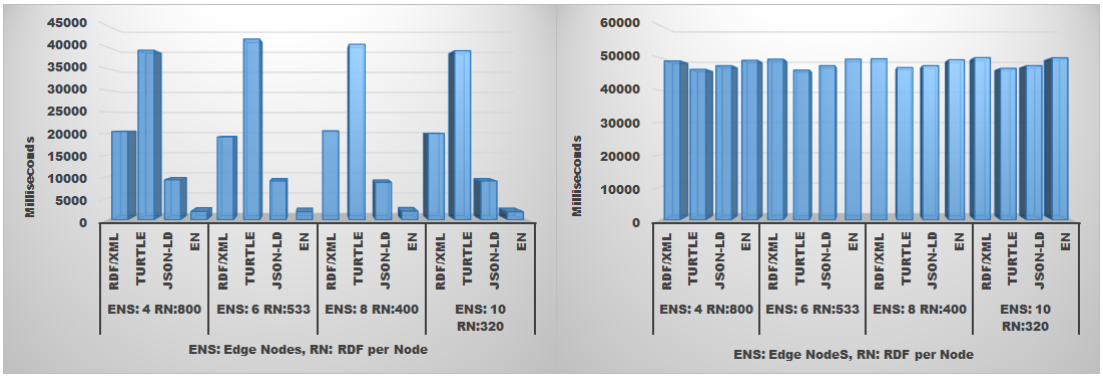

Figure 17. Scalability Results for Group B (Left:Transferring, Right:Reasoning).

In Figure 17, the left figure shows the result of Group B transferring time comparison and the right one shows the Group B reasoning time comparison. The X-axis represents the combination of amount of Edge nodes and “RDF per node”. The abbreviation “ENS” means “Edge Nodes Number”. The abbreviation “RN” means “RDF Statements per IoT Node”. The experiment binds 10 IoT nodes to each Edge node.

$$Total\ IoT\ nodes\ Number = Edge\ Server\ Number * IoT\ node\ per\ Edge\ Server$$

For example, “ENS:4 RN:800” means there are four Edge nodes in total and 800 RDF statements will be sent from each IoT node. So for this case, there are 40 IoT nodes in total. The Y-axis represents the time consumption in milliseconds. The transferring time increases of four data formats differs with each other in the test case and for the different cases, the time consumptions for the same format are in the same level. EN formatted data consumes shortest time, The JSON-LD is the second shortest, the RDF/XML is the third one and the Turtle format is the longest one. In test case “ENS:6 RN:533" (No.7), the transferring time of Turtle data is 22.4 times longer than EN format. The transferring time of JSON-LD format is 4.6 times longer than average EN time , the transferring time of RDF/XML format is 10.4 times longer than average EN time and the transferring time of Turtle format is 20.9 times longer than average EN time. In the reasoning time comparison, the total reasoning time of different cases and different formats is similar.

In Figure 18, the left figure shows the result of Group C data transferring time comparison and the right one shows the Group C reasoning time comparison. In corre-

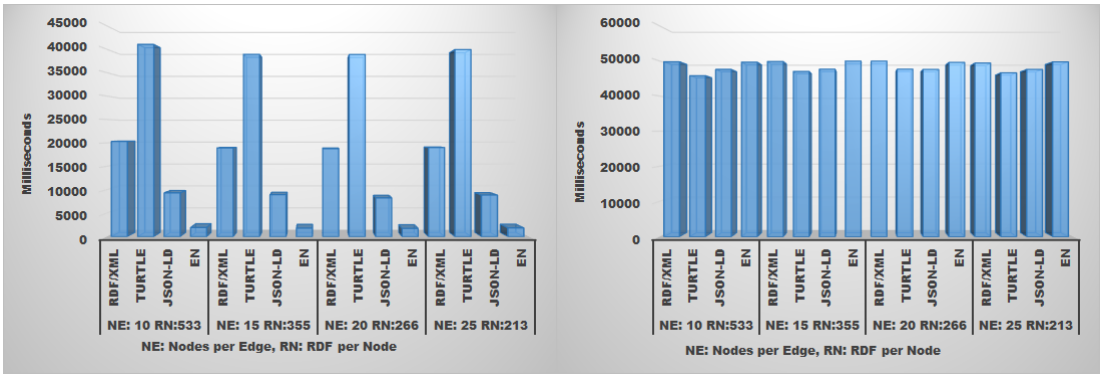

Figure 18. Scalability Results for Group C (Left:Transferring, Right:Reasoning).

sponding Edge Reasoning Architecture, the test cases use six Edge nodes with the total 32000 RDF observation statements. The X-axis represents the combination of "nodes per Edge node" and "RDF per node". The total RDF individual data size to 32000. For example, "NE:10 RN:533" means one Edge node will be responsible for 10 IoT nodes and each IoT node will send 533 individual RDF data. For this case, there are 60 IoT nodes in total. The Y-axis represents the time consumption in milliseconds. The transferring time increases of four data formats differ with each other in the test cases and for the different cases, the time consumptions for the same format are in the same level. EN formatted data consumes the shortest time, The JSON-LD is the second shortest, the RDF/XML is the third one and the Turtle format is the longest one. In experiment of "NE:20 RN:266" (No.12), the data transferring time of Turtle data is 21.9 times longer than EN format. The transferring time of JSON-LD format is 4.7 times longer than EN in average, the transferring time of RDF/XML format is 10.3 times longer than EN in average and the transferring time of Turtle format is 21.2 times longer than EN in average. In the reasoning time comparison, The total reasoning times of the different cases and different format keep in the same level.

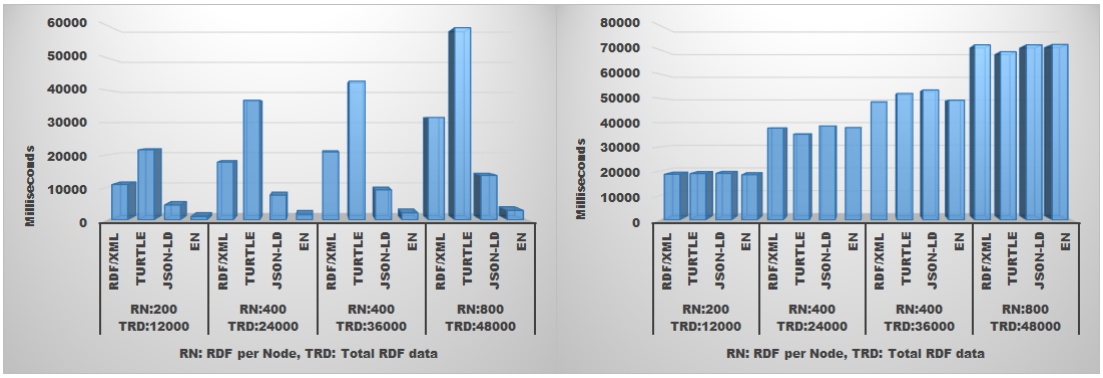

Figure 19. Scalability Results for Group D (Left:Transferring, Right:Reasoning).

In Figure 19, the left figure shows the result of Group D data transferring time comparison and the left one shows the Group D reasoning time comparison. The test cases use six Edge nodes and connect 10 IoT nodes to each Edge node. The X-axis represents the amount of "nodes per Edge node" and the corresponding total RDF data size. For example, "RN:200 TRD:12000" means each IoT node will send 200 RDF

individual data and there are 12000 individual data in total. The Y-axis represents the time consumption in milliseconds. The transferring time increases with the rise of the total RDF data size. Comparing with four data formats, EN formatted data consumes shortest time, The JSON-LD is the second shortest, the RDF/XML is the third one and the Turtle format is the longest one. In experiment of “RN:400 TRD:24000" (No.15), the transferring time of Turtle data is 21.3 times longer than EN format. The transferring time of JSON-LD format is 4.4 times longer than EN in average, the transferring time of RDF/XML format is 10.3 times longer than EN in average and the transferring time of Turtle format is 20.5 times longer than EN in average. In the reasoning time comparison, The total reasoning time increase when the data size increase. For the same amount of data, different formatted data shows equal performance.

4.2.4. Comparison with Cloud Reasoning and Edge Reasoning

In this section, we will compare the performance of Cloud Reasoning Architecture and Cloud Reasoning Architecture. Similar with Cloud Reasoning Architecture, we measure the data transforming time from IoT nodes and reasoning time from Cloud server. Moreover, we also measure the reasoning time form the Edge nodes and the data transforming time from Edge nodes to Cloud server.

The Edge component will reason all the 17 tasks with 2 rules: High Acceleration and High De-acceleration. The rest rules will be reasoned on the Cloud server. We have four RDF syntax. The RDF/XML, JSON-LD and Turtle syntax are used in Edge architecture in Figure 9 while EN data is used in Ontology Enabled Edge Architecture showed in Figure 10. The Cloud reasoning is based on architecture in Figure 8.

As the Edge nodes consume “High Acceleration” and “High De-acceleration” tasks, the Cloud server will not use “obs:hasAcceleration” property, hence we could only send the rest ten properties to Cloud server instead.

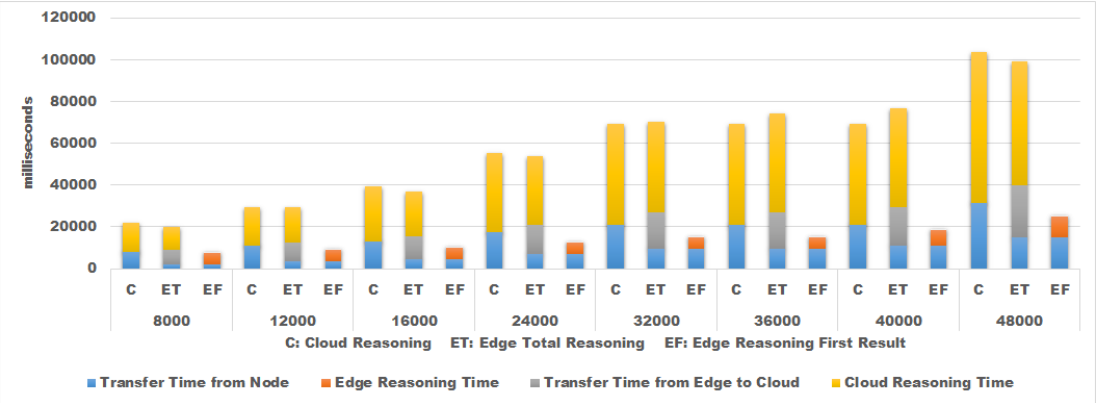

Figure 20. Reasoning Performance Comparison between Two Architecture with RDF/XML.

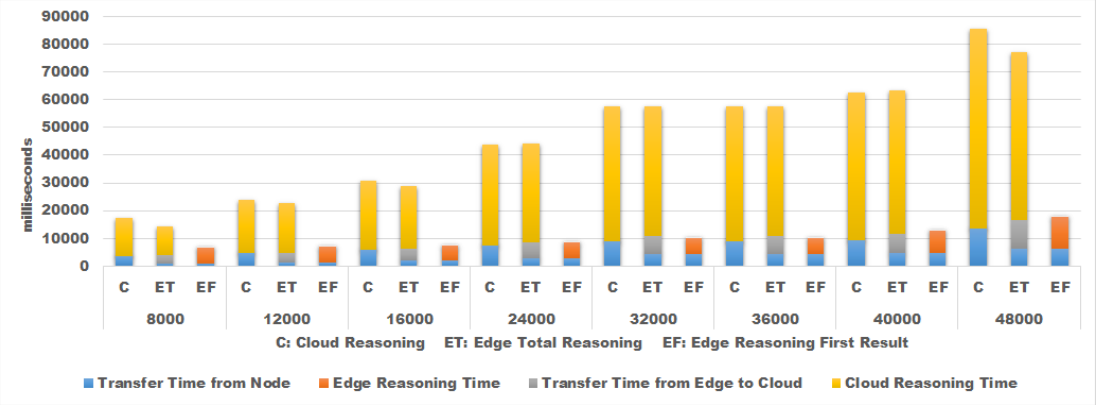

Figure 21. Reasoning Performance Comparison between Two Architecture with JSON-LD.

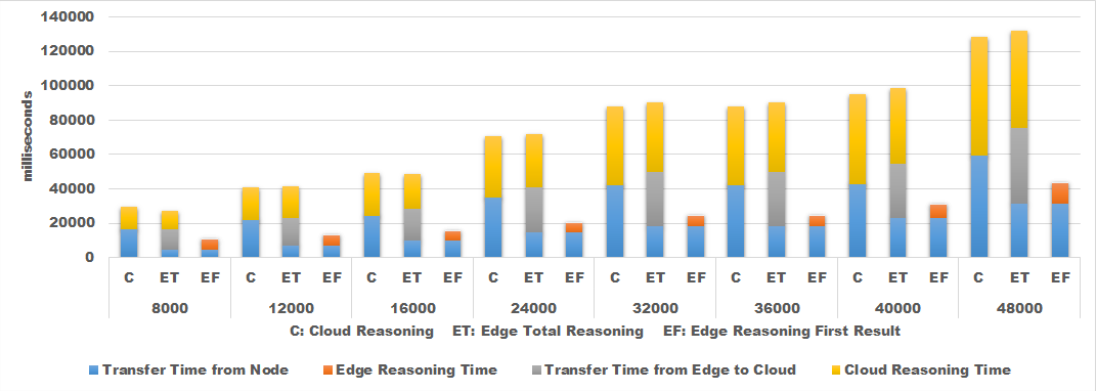

Figure 22. Reasoning Performance Comparison between Two Architecture with Turtle.

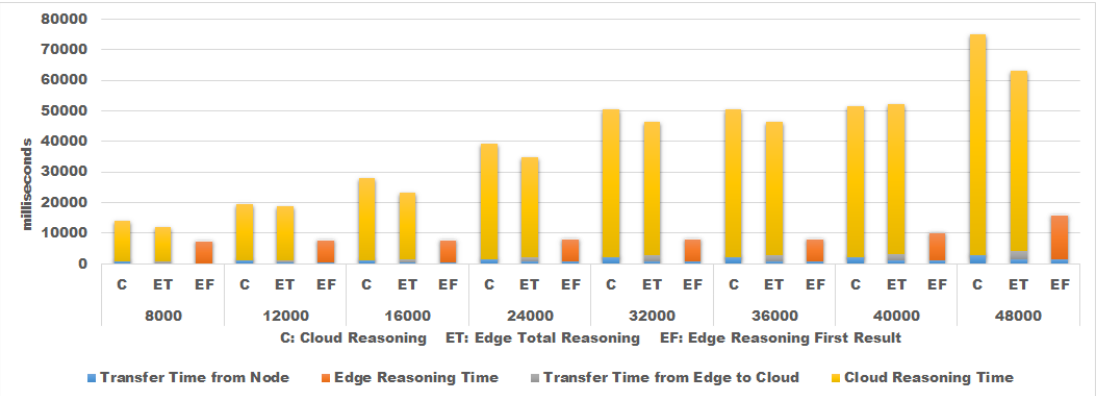

Figure 23. Reasoning Performance Comparison between Two Architecture with EN.

Figure 20, Figure 21, Figure 22 and Figure 23 present the performance comparison between Cloud Reasoning Architecture and Edge Reasoning Architecture. There are three columns in each test case. As the Edge nodes will take part of the tasks, both Cloud server and Edge nodes will get the reasoning results. In our experiments, the

results from Edge nodes come earlier than the Cloud. We call the result from the Edge nodes the “first result”. The first result processing time includes the time IoT node transferring data to Edge node and the reasoning time of Edge node. The left column represents the total processing time of the Cloud Computing architecture; the middle column represents the total processing time of the Edge Computing architecture; the right column represents the result processing time of the Edge Computing architecture. We choose 8 out of 17 experiments based on the total RDF data size. The Y-axis of the figure represents the total RDF data size. For example, the first test case “8000” means there are 8000 RDF statements to process. The total time of Cloud and Edge are with the same level. The first result will came much faster than the Cloud server’s result. In EN experiments, the third test case with 16000 RDF statements shows the Cloud reasoning time is 10 times longer than the first result come from Edge node. The average ratio between Cloud reasoning time and first result time from Edge Computing for RDF/XML is 5.1 times, for Turtle is 4 times, for JSON-LD is 6.6 times and for EN is 8.9 times. For all the cases, the reasoning on Cloud consumes most of the whole time. After utilizing Edge nodes, the average Cloud reasoning time reduces 12.4% for RDF/XML format, 12.3% for Turtle format, 6.2% for JSON-LD format and 12.1% for EN format. The total transferring time on Edge of Turtle is 1.2 times of reasoning time in average and the time of EN is 6% of the average reasoning time.

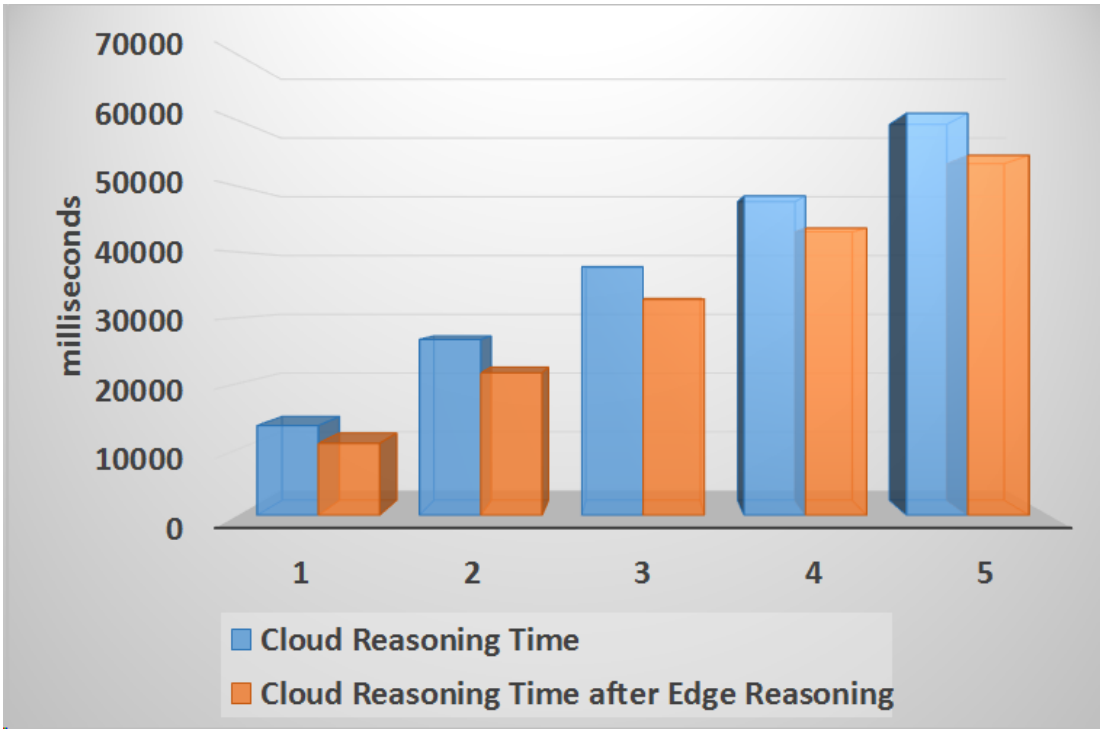

Figure 24. Cloud Side Computing Reasoning Time Comparison.

Figure 24 shows the comparison between reasoning the whole data and reasoning the part of the data on Cloud side. The Edge node will do part of the reasoning using “High Acceleration” property in RDF data. Then the Edge nodes remove the property and send the rest of the data to Cloud, the Cloud server does the rest of the reasoning. We chose all the five experiments from Group A in Table 11. In the five experiment, the Cloud server reduces 14% processing time after Edge nodes reasoning.

4.2.5. Edge Reasoning Performance Comparison with Different Rule Set

Last experiment aims to test the performance improvement of Edge Computing. We deploy different rules on Edge nodes and measure the overall processing time. The more rules processed on Edge nodes, the less rules processed on Cloud server. We choose RDF/XML as the RDF data format. In the experiment, there are 8 Edge nodes and each Edge node will process 10 IoT nodes. Each IoT node will send 400 individual RDF data to either Edge node or Cloud server. So for Cloud architecture, 80 IoT nodes will send 32000 RDF data in total. This is the No.4 experiment in Table12.

We design four groups of task distributions: Group A implements all the reasoning tasks on the Cloud server, so there is only the "Cloud" column; Group B implements rules which related with "High Average Speed" (Rule 17,18,19,20,21 in Figure 9) on the Edge nodes and the rest on Cloud; Group C implements rules related with High Acceleration and High De-acceleration (Rule 28,29 in Figure 9) on Edge nodes and the rest on Cloud; Group D implements rules related with both "High Average Speed", "High Acceleration" and "High De-acceleration" (Rule 17,18,19,20,21,28,29 in Figure 9) on the Edges node and the rest on the Cloud.

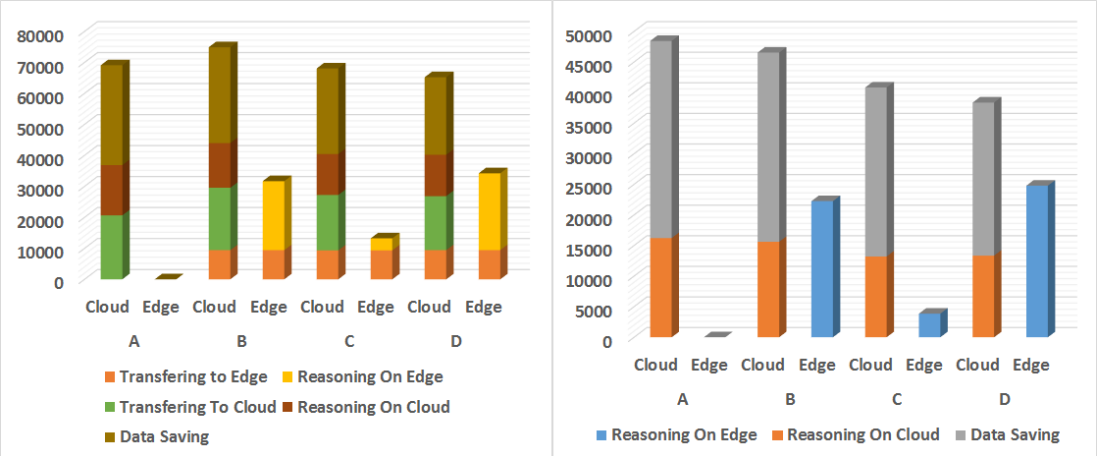

Figure 25. Performance Comparison with Different Rule Set.

This experiment shows the performance comparison between different task distribution on Edge nodes. Figure 25 presents the result.

In each group, the overall time consumption is:

$$TIME_{Overall} = \max\left(TIME_{Cloud}, TIME_{Edge}\right)$$

In Figure 25, the left chart shows the overall processing time include data transferring time and reasoning time and the right chart shows only the reasoning time. The "Cloud" column represents the total processing time from IoT node to Edge nodes and from Edge nodes to Cloud server. The "Edge" column only counts the processing from IoT nodes to Edge nodes including data receiving time and reasoning time. For example, in Group B, IoT nodes first send the data to Edge, and then the Edge nodes simultaneously reason the data and send data to Cloud. So both columns have the same orange part, which is data sending time from IoT nodes to Edge nodes. Then the yellow part in "Edge" column, which represents the reasoning time on Edge. The green part represents the data sending time from Edge

to Cloud. And the red part represents the data reasoning time on Cloud and the brown part represents the data storage time on Cloud. As the Cloud and Edge doing task simultaneously, we could calculate the overall reasoning time with the formula: $TIME_{Overall} = \max(TIME_{CloudColumn}, TIME_{EdgeColumn})$. From the right chart we could see that the more rules implemented on the Edge node, the less reasoning time needed from Cloud server. Comparing with Cloud reasoning in Group A, Group B reduces 4% reasoning time, Group C reduces 15.6% reasoning time and Group D reduces 20.7% reasoning time. From the left chart, Comparing with Cloud reasoning time in Group A, Group B increases 8% reasoning time, Group C reduces 1.4% reasoning time and Group D reduces 5% reasoning time.

4.3. Analysis

From the Ontology Size Comparison section, we could find out the format affects the size of the OWL 2 Ontologies. The largest OWL 2 ontology is more than three times than the smallest one in average. It means selecting a proper format will save storage and bandwidth for storing and transferring the ontology data. But we could also notice that the size of the ontology is not only affected by the format because the same format will perform differently in different ontology. The reason is that different syntax expresses the same syntax structure in different ways. For example, RDF/XML syntax encode Instance definition with shorter text comparing with OWL/XML:

RDF/XML Syntax

```
<Person rdf:about="Mary"/>
```

OWL/XML Syntax

```
<ClassAssertion>
   <Class IRI="Person"/>
   <NamedIndividual IRI="Mary"/>
</ClassAssertion>
```

While OWL/XML performs better in expressing Class Disjointness:

RDF/XML Syntax

```
<owl:AllDisjointClasses>
  <owl:members rdf:parseType="Collection">
    <owl:Class rdf:about="Woman"/>
    <owl:Class rdf:about="Man"/>
  </owl:members>
</owl:AllDisjointClasses>
```

OWL/XML Syntax

```
<DisjointClasses>
    <Class IRI="Woman"/>
    <Class IRI="Man"/>
</DisjointClasses>
```

From the result of Jena Model loading time comparison, we could find that JSON-LD format will consume more time in Model building step. In Jena Framework, all the

different formatted ontology data will be converted to Jena Model. So after the Jena Model is built, the format will not affect.

$$TIME_{Complete\ Reasoning} = TIME_{Jena\ Model\ Building} + TIME_{Jena\ Model\ Inference}$$

All the formats consume the same time in inference process as they have been loaded in Jena Model. The complete reasoning time is only different according to Jena Model Building part. The longer Model loading time it needs, the loner complete time it consumes. From this point of view, the more time consumed in Model building process, the more time needed for reasoning.

From the result of RDF data transferring and reasoning time comparison, we could find that the transferring time is related to data formats. For this scenario, the structure of each individual RDF data is fixed. Even with different amount of data, the encoding ratios between different format keep in the same level. In another words, the length of four formatted data keeps a similar ratio for all the test cases. The transferring time is based on the total data size and network status. We package 50 individual data together and send to Cloud server, we measure the total size of the 50 individual data and find the size has the same sequence order:

$$SIZE_{Turtle} > SIZE_{RDF/XML} > SIZE_{JSON-LD} > SIZE_{EN}$$

The reasoning time result shows that the overall reasoning time is related to the size of RDF data. The reasoning time grows linearly as RDF statements added. RDF formats make no difference to the reasoning performance. The reason is that the main time consumption is Jena reasoning. Jena performs reasoning using Jena Model and RDF data with different formats will be loaded to the same Jena Model for inference.

The Cloud Computing and Edge Computing paradigm comparison shows that adding Edge nodes into IoT environments could accelerates the result processing time and reduces the bandwidth of the core network. The Cloud Reasoning architecture will generate the first result 10 times slower than the Edge architecture. After removing property “obs:hasAcceleration", the average size of 50 RDF/XML individual data is 59 739 bytes, which is 90.7% of the original size.

We also find out the transferring time heavily depends on the network situation. As our IoT Edge nodes are located in Finland and the Cloud Server is in Germany, the long distance and unstable network could affect the latency.

Table 13. panOULU wireless router

	Comply with	Transfer Speeds
Cisco Aironet 1200	802.11b,802.11a	54 Mbps
Linksys WRT54GL	802.11g	54 Mbps
Cisco Aironet 1240	802.11g,802.11a	54Mbps
Cisco Aironet 1140	802.11n	72Mbps
Strix OWS 2400	802.11a,802.11g	54Mbps

The Network equipments could affect the general performance as well. We are using panOULU [107] for Edge nodes reasoning experiments. panOULU has 5 types of Wireless Router, which is displayed in Table 13. Most of their maximum transfer

speed is 54 Mbps, which equals to 6.75 MB/S. For example, one individual RDF/XML format data is 1.4 KB on average, so the Experiment No.1 transfers 10.93 MB RDF data in 2281 milliseconds. which means the average transfer speed is 4.8 MB/S . It is close to the maximum capacity of the network as a 54 Mbps public router cannot fully used in practise.

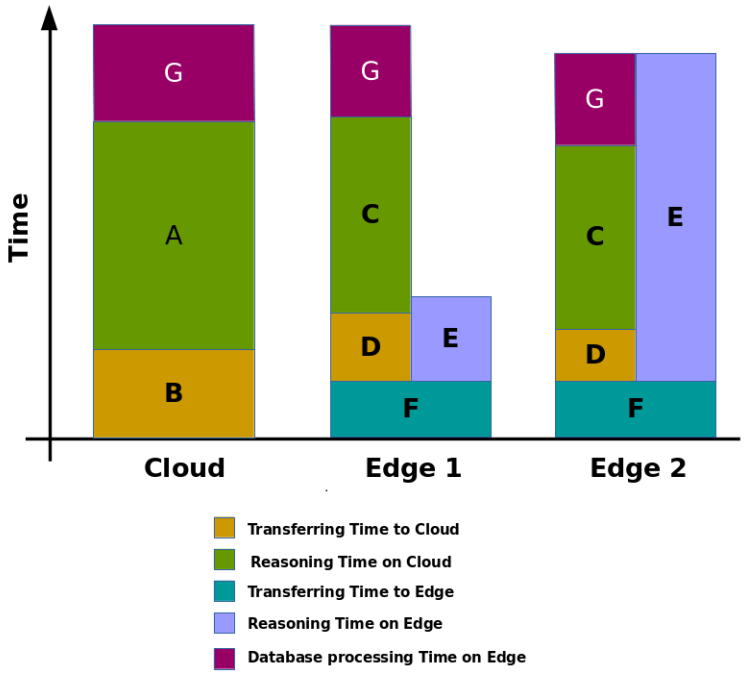

Figure 26. Time Consumption Analysis.

Figure 26 shows the analysis of the time consumption between Cloud Reasoning Architecture and Edge Reasoning Architecture. The total processing time of Cloud Reasoning Architecture consists of data transferring Time from IoT nodes to Cloud (Part A), Cloud reasoning Time (Part B) and Cloud database processing time (Part G). The total processing time of Edge Reasoning Architecture consists of data transferring time from IoT nodes to Edge (Part F), Edge reasoning time (Part E), data transferring time from IoT nodes to Cloud (Part D) and Cloud reasoning time (Part C). As the Edge nodes will transfer the data and do the reasoning in parallel, so the Part C and Part D is parallel with Part E from the same starting point.

We calculate the computation time consumption based on the formulas:

$$TIME_{Cloud} = TIME(A) + TIME(B)$$

$$TIME_{Edge} = TIME(F) + \min\left(TIME(C) + TIME(D), TIME(E)\right)$$

There are two strategies for designing the Edge based IoT system based on different requirements. First, from the prospective of fast response, we could minimize the Part F and Part E on Edge. “Edge 1” in Figure 26 shows the pattern. As the Edge nodes aim for services, it will not do much extra reasoning tasks for the Cloud. Second, from the prospective of the overall computation performance, we could gain the minimum processing time by balancing the reasoning on Edge (Part E) and the processing on

Cloud (Part C, Part D and Part G). In another words, the more reasoning tasks deployed on Edge nodes, the less workload will be undertook on the Cloud. “Edge 2” in Figure 26 shows the diagram of the balance of processing time consumption between Cloud and Edge nodes.

Performance comparison between different rule sets shows the same result in Figure 25. The more workload distributed on the Edge, the less processing time the Cloud server needs. When they reach a balance and the Cloud processing time is similar with Edge processing time, the overall processing time will be optimized.

Then we analyze the performance about distribution of reasoning task. First we need to analyze the reasoning mechanism. We use Jena Framework rule based reasoner. Jena rule based reasoner has three reasoning engines: the forward engine, the backward engine and the hybrid engine. We use the default one: hybrid engine. All the engine works in similar way: matching the facts according to the rules. All the elements in both RDF data and rules will be scanned. The amount of facts and the amount of rules are two factors which affect the whole reasoning processing time. The facts are stored in RDF data. We can also conclude that the more input RDF elements the system receives, the longer reasoning processing time it needs; the more rules,whether or not they will be triggered, the longer reasoning processing time the system consumes. If we do further analyze, the complexity of the rules and how many rules are triggered also affect the performance. For the rules, the more elements and judgements they have, the more processing time the system needs. Some rules will only be triggered when another rules satisfied and new facts added. We list the factors in Table 14. If we could split the rules and RDF statement into several parts without overlapping and distribute tasks properly on Edge nodes, the overall processing time will be reduced. But if there is overlapping, the analysis will become complicated. If we separate the rules and RDF properties into two parts, one part is implemented on the Edge nodes and the second part is on the Cloud server. And we do reasoning in sequential execution manner, for example, the Edge node will reason first and then the Cloud server reason the second part, the overall processing time will be longer as the Edge components are much slower than the Cloud server. In our Edge reasoning architecture, we organize the Edge nodes and Cloud server in a parallel way. In this manner, we could execute both Edge nodes and Cloud server together. For Cloud processing time and Edge nodes reasoning time , the slower one will not affect the overall processing time. If we keep IoT nodes as fixed amount and increase the Edge node amount, each Edge node will reason less RDF data and we could reduce the processing time. The more Edge nodes we have, the less overall processing time the system needs.

Table 14. Factors of the Reasoning Processing Time Comsumption

No.	Factors
1	Amount of RDF Data
2	Amount of Rules
3	Complexity of Rules
4	Rules Trigger Situation
5	Amount of Edge nodes

How much time will be saved is based on the system structure and the speed of the each component. For example, in our example shows in Figure 25, the Edge nodes

could definitely reduce the reasoning processing time but may not reduce the overall time. The reason is that we reduce the reasoning time but we also increase the transferring time. If the reduced reasoning processing time is larger than the increased transferring time, we could save overall time. Comparing with Cloud server, the Edge nodes are resources-limited devices. Moving the tasks to resources-limited devices will increase the processing time but if we execute Edge reasoning in parallel manner, we could save time. If the Edge nodes have higher processing ability, they will improve more performance for the whole system. For the area with poor network connection, for example the moving vehicles, the data transferring time via Internet will be slow thus the Edge nodes will have more spare time to undertake more tasks and improve the performance. Increasing the quantity of Edge nodes will provide more processing resources but the structure of the network becomes complected.

5. DISCUSSION

The growth of IoT requires new technologies for data representation, data storage, data interconnection, search, information organization, high performance computing, etc. This thesis work utilizes semantic technologies and Edge computing paradigm in IoT systems to improve the data management and semantic processing performance. We focus on analyzing the performance of semantic reasoning in IoT systems using Edge computing paradigm. An Edge based IoT system with semantic technologies is build for the research. In the system, we implement semantic reasoners for general purpose rules based reasoning in real IoT environments. Computing and storage resources are both deployed on the Cloud servers and Edge nodes. The Edge computing paradigm have three function based on different deployments. First it could accelerate the services response time. Second, it could improve the semantic reasoning performance. Third, it could improve the core network performance by reducing the bandwidth usage. The Edge nodes, which are located close to IoT nodes, communicate with IoT nodes and Cloud server. By undertaking part of the whole tasks, the Edge nodes reduce the computing burden of the Cloud server and reduce the bandwidth usage in core network by filtering useless data.

Five experiments are carried out to study the research questions. To figure out the reliability of the system integrated with three different technologies, we test the scalability with a large data set. To research the influence of semantic technologies on performance, we design experiments to compare the size and loading time of ontologies with different RDF syntax. For studying the influence of Edge computing paradigm, we compare the semantic reasoning performance on the system with only Cloud server and the system with both Cloud server and Edge nodes. We also distribute different amount of task on Edge server and evaluate the performance influence.

This study shows that different RDF formats could encode the same ontology into different size. The size of the RDF data is an essential factor for data transferring and storage. So choosing a proper format for an ontology could facilitate the performance of semantic processing. However, we also found out the same format may perform differently on different cases. That is because that different format use different expression for the same syntax. We recommend the developers to use Functional Syntax, EN, TURTLE and Manchester Syntax as they show better encoding efficiency meanwhile they need to consider the syntax usage in that specific ontology from statistic point of view. The different formats show similar performance when building the Jena Model will the JSON-LD consumes more time.

This study also finds out that the semantic data transferring time is closely related to the total RDF data size while the overall semantic processing time include reasoning and storing for different syntax is in the same level. Changing the amount of IoT nodes and Edge nodes can not significantly affect the overall performance.

The study shows that adding Edge nodes into IoT systems could generate result faster, reduce the bandwidth usage on core network and share the workload of the centre server. The physical proximity between Edge nodes and IoT nodes facilitates the transferring efficiency. The added Edge nodes will definitely reduce the reasoning processing time of the Cloud server and if the developers properly distribute the tasks under a stable network, the Edge server could reduce the overall processing time.

Our current work focuses on analyzing the performance of a Edge based IoT systems. We deploy the system on the Amazon EC2 cloud platform and carry out the experiment with Android devices. The result shows the benefits of Edge computing paradigm for semantic reasoning in IoT. We test and evaluate the performance based on time consumption. But the processing time depends on the hardware and software systems. For example, our system runs on a CentOS [108] Linux system. The CentOS Linux Operation System already runs a lot of services in background, which consume the computing resources. Our experiment does not count how many background services are running during our test. So the result from anther Cloud server with the same hardware and Operation systems may slightly differ. In another words, we could choose more objective measurement standards for computing consumption in the future. For example, how many CPU instructions are executed for the specific software or process. Similarly, the measurement of transferring time of the RDF data is also affected by network situation. It depends on the hardware and the network situation. In real IoT systems, IoT devices may join different network with different routers, for example, the mobile phone of a driver. Crowd network, bad signal and low speed router will affect to the high latency. New methods of measuring the latency are also required.

We measure the storage efficiency of RDF data with different formats based on the data size. We found the same format have different efficiency if the RDF data has different structure. So more comprehensive measurement could be carried out based on the RDF data structure.

Our scalability experiments test large scale data set while we need to consider the fact that the amount of IoT devices will increase incredibly in the future. Everything around us will become IoT smart devices, for example, the refrigerator, the oven, the bed, the table or even the pencil. Very huge scale experiment could be designed in the future for further evaluation.

The comparison between Cloud computing and Edge computing paradigm is based on our Oulu Taxi scenario. The rules and ontologies are pre-defined. More experiments could be carried out to study whether the result will differ if the data, rules and ontology changed. We perform part of the reasoning tasks to Edge nodes and the rest tasks on Cloud server. How to assign the tasks and what criteria would optimize the performance could be future researches. The researches also need to figure out how to divide the RDF data, rules and ontology for optimization purpose.

Executing the reasoning tasks on Edge computing paradigm in parallel manner will reduce the processing time and thus improving the performance. The degree of improvement is based on the relationship between the transferring speed, reasoning speed and storage speed. But if the relationship between these speeds is fixed, the performance improvement could finally reach a maximum at a balance status. Increasing the quantity of Edge nodes will also improve the overall performance. In the future, it is necessary to research on how to split the rules and RDF data in a more optimized manner.

6. CONCLUSION

To study the semantic reasoning on the Edge of Internet of Things, we design and implement an Edge based IoT system with semantic technologies. This system could perform rule-based semantic reasoning for general purpose. With the system, we do five experiments to research the performance of semantic reasoning in real IoT environments. The experiments are designed to figure out how RDF syntax, computing architectures and task distribution in Edge computing could affect the reasoning performance.

Revisit the research questions, we could conclude that the Edge computing paradigm could facilitate IoT with semantic processing. By utilizing Edge nodes, we could reduce the reasoning and data transferring time, response the request in a short period, save the bandwith usage of the core network. The system and experiments also shows that the semantic technology could provide a new way to represent and understand knowledge in IoT.

The first research question is about reliability, scalability and performance of IoT systems integrated with Edge computing paradigm and semantic technologies. Our system successfully address these requirements. After integrating the three technologies, the system shows good performance and scalability.

The second research question studies the benefits and shortcomings of the Edge computing and semantic technologies for IoT systems. For the second question, we could conclude that the Edge computing technology could facilitate the semantic reasoning in IoT systems. The Cloud server is involved in both Cloud Computing and Edge Computing. In another words, the Edge computing is extended to Cloud computing. As the physical proximity from Edge nodes to the users' devices, the Edge nodes could provide faster services thus improving the user experience. The Edge is more flexible than the Cloud and could be deployed anywhere. The Edge nodes could also facilitate the Cloud Computing by reducing the results delivery time and save the bandwidth for the core network. Different options of semantic technologies could also influence the reasoning performance. The RDF format is an essential factor which could affect the transferring and storage efficiency for semantic processing. The total RDF data size will affect the semantic reasoning time. For better performance, we need to choose proper RDF format for RDF data and Ontology. For gaining faster result, we need to reduce the scale of the data and increase the number of Edge nodes.

Our research focuses on the integration of IoT, semantic technologies and Edge computing paradigm. Although all these three technologies are hot topics, the integration research has rarely been done before. Our work demonstrates that the Edge computing paradigm technology could facilitate IoT with semantic technologies. We also recognize how the semantic technologies and computing architectures could influence IoT. The results shed the light on the design and implementation of the future IoT systems. The semantic technologies have been utilized in Web for years and our research shows the potential of semantic technologies for IoT. Our work for semantic reasoning on the Edge of IoT promotes the utilization of semantic technologies and Edge computing paradigm in IoT.

7. REFERENCES

[1] MacGillivray C. (2016) Worldwide internet of things forecast update: 2015 – 2019. International Data Corporation (IDC), Tech. Rep .

[2] Jeffrey R.C. & Burgess J.P. (1981) Formal logic: Its scope and limits, vol. 1967. Cambridge Univ Press.

[3] Guarino N. (1995) Formal ontology, conceptual analysis and knowledge representation. International journal of human-computer studies 43, pp. 625–640.

[4] Bundy A. & Wallen L. (1984) Augmented transition network. In: Catalogue of Artificial Intelligence Tools, Springer, pp. 7–7.

[5] Domingos P. & Webb W.A. (2012) A tractable first-order probabilistic logic. In: Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, AAAI'12, AAAI Press, pp. 1902–1909. URL: http://dl.acm.org/citation.cfm?id=2900929.2900997.

[6] Smith J.M. & Chang P.Y.T. (1975) Optimizing the performance of a relational algebra database interface. Communications of the ACM 18, pp. 568–579.

[7] Barnaghi P., Wang W., Henson C. & Taylor K. (2012) Semantics for the internet of things: early progress and back to the future. International Journal on Semantic Web and Information Systems (IJSWIS) 8, pp. 1–21.

[8] Shi W., Cao J., Zhang Q., Li Y. & Xu L. (2016) Edge computing: Vision and challenges. IEEE Internet of Things Journal 3, pp. 637–646. URL: http://dx.doi.org/10.1109/jiot.2016.2579198.

[9] Ahmed E. & Rehmani M.H. (2016) Mobile edge computing: Opportunities, solutions, and challenges. Future Generation Computer Systems .

[10] Beck M.T., Werner M., Feld S. & Schimper S. (2014) Mobile edge computing: A taxonomy. In: Proc. of the Sixth International Conference on Advances in Future Internet, Citeseer, pp. 48–54.

[11] Hu Y.C., Patel M., Sabella D., Sprecher N. & Young V. (2015) Mobile edge computing—a key technology towards 5g. ETSI White Paper 11.

[12] Lee I. & Lee K. (2015) The internet of things (iot): Applications, investments, and challenges for enterprises. Business Horizons 58, pp. 431–440.

[13] Consulting J..P. (2013), Overview of the russian and world smart devices market, 2011 – 2017. URL: http://www.json.ru/en/poleznye_materialy/free_market_watches/analytics/obzor_rossijskogo_i_mirovogo_rynka_smart-devajsov_2011_-_2017/.

[14] Settembre M. (2012) Towards a hyper-connected world. In: Telecommunications Network Strategy and Planning Symposium (NETWORKS), 2012 XVth International, IEEE, pp. 1–5.

[15] Bassi A. & Horn G. (2008) Internet of things in 2020: A roadmap for the future. European Commission: Information Society and Media .

[16] Kumar M., Vetripriya M., Brigetta A., Akila A. & Keerthana D. (2016) Analysis on internet of things and its application. International Journal of Scientific Research in Science .

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

[18] Weber R.H. (2010) Internet of things–new security and privacy challenges. Computer Law & Security Review 26, pp. 23–30.

[19] Bandyopadhyay D. & Sen J. (2011) Internet of things: Applications and challenges in technology and standardization. Wireless Personal Communications 58, pp. 49–69.

[20] Jara A.J., Olivieri A.C., Bocchi Y., Jung M., Kastner W. & Skarmeta A.F. (2014) Semantic web of things: an analysis of the application semantics for the iot moving towards the iot convergence. International Journal of Web and Grid Services 10, pp. 244–272.

[21] Shelby Z., Hartke K. & Bormann C. (2014) The constrained application protocol (coap). Tech. rep., Universitaet Bremen TZI.

[22] Masinter L., Berners-Lee T. & Fielding R.T. (2005) Uniform resource identifier (URI): Generic syntax.

[23] Dürst M. & Suignard M. (2004) Internationalized resource identifiers (IRIs).

[24] Leach P.J., Mealling M. & Salz R. (2005) A universally unique identifier (uuid) urn namespace.

[25] Welbourne E., Battle L., Cole G., Gould K., Rector K., Raymer S., Balazinska M. & Borriello G. (2009) Building the internet of things using rfid: the rfid ecosystem experience. IEEE Internet Computing 13, pp. 48–55.

[26] Yick J., Mukherjee B. & Ghosal D. (2008) Wireless sensor network survey. Computer networks 52, pp. 2292–2330.

[27] Antony (2014), Iot protocol stack. URL: `https://entrepreneurshiptalk.wordpress.com/2014/01/29/the-internet-of-thing-protocol-stack-from-sensors-to-business-value/`.

[28] Song J., Han S., Mok A., Chen D., Lucas M., Nixon M. & Pratt W. (2008) Wirelesshart: Applying wireless technology in real-time industrial process control. In: Real-Time and Embedded Technology and Applications Symposium, 2008. RTAS'08. IEEE, IEEE, pp. 377–386.

[29] Kinney P. et al. (2003) Zigbee technology: Wireless control that simply works. In: Communications design conference, vol. 2, vol. 2, pp. 1–7.

[30] Sheng Z., Yang S., Yu Y., Vasilakos A.V., McCann J.A. & Leung K.K. (2013) A survey on the ietf protocol suite for the internet of things: Standards, challenges, and opportunities. IEEE Wireless Communications 20, pp. 91–98.

[31] Fielding R. (2000) Representational state transfer. Architectural Styles and the Design of Netowork-based Software Architecture , pp. 76–85.

[32] Rescorla E. & Modadugu N. (2012) Datagram transport layer security version 1.2. Tech. rep., Internet Engineering Task Force (IETF).

[33] Chen X. (2014), Constrained application protocol for internet of things. URL: http://www1.cse.wustl.edu/~jain/cse574-14/ftp/coap/.

[34] Tolle G. (2010) Embedded binary http (ebhttp). ID: draft-tolle-core-ebhttp-00 .

[35] Stanford-Clark A. & Truong H.L. (2008) Mqtt for sensor networks (mqtt-s) protocol specification. International Business Machines Corporation version 1.

[36] Thangavel D., Ma X., Valera A., Tan H.X. & Tan C.K.Y. (2014) Performance evaluation of mqtt and coap via a common middleware. In: Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on, IEEE, pp. 1–6.

[37] Vinoski S. (2006) Advanced message queuing protocol. IEEE Internet Computing 10, p. 87.

[38] Saint-Andre P. (2004) Extensible messaging and presence protocol (xmpp): Instant messaging and presence. Tech. rep., Internet Engineering Task Force (IETF).

[39] Tian L. (2012) Lightweight m2m (oma lwm2m). OMA device management working group (OMA DM WG), Open Mobile Alliance (OMA) .

[40] Berners-Lee T., Hendler J., Lassila O. et al. (2001) The semantic web. Scientific american 284, pp. 28–37.

[41] Prud'Hommeaux E., Seaborne A. et al. (2008) Sparql query language for rdf. W3C recommendation 15.

[42] Kifer M. (2008) Rule interchange format: The framework. In: International Conference on Web Reasoning and Rule Systems, Springer, pp. 1–11.

[43] Guess A. (2012), Interview: Semantic tech at the bbc. URL: http://www.dataversity.net/interview-semantic-tech-at-the-bbc/.

[44] Corporation T.B.B. (2016), Bbc ontology. URL: http://www.bbc.co.uk/ontologies.

[45] Semantics C. (2016), Example semantic web applications. URL: https://www.cambridgesemantics.com/semantic-university/example-semantic-web-applications#data-integration-in-oil-and-gas.

[46] Sowa J.F. (1999) Knowledge representation: logical, philosophical, and computational foundations. Book in preparation. To be published by PWS Publishing Company, Boston, Massachusetts .

[47] Smullyan R.M. (1995) First-order logic. Courier Corporation.

[48] Baader F. (2003) The description logic handbook: Theory, implementation and applications. Cambridge university press.

[49] Van Harmelen F., Lifschitz V. & Porter B. (2008) Handbook of knowledge representation, vol. 1. Elsevier.

[50] Poole D.L. & Mackworth A.K. (2010) Artificial Intelligence: foundations of computational agents. Cambridge University Press.

[51] Shi Z. (2011) Advanced artificial intelligence, vol. 1. World Scientific.

[52] Brooks R.A. (1991) Intelligence without representation. Artificial intelligence 47, pp. 139–159.

[53] Frye D., Zelazo P.D. & Palfai T. (1995) Theory of mind and rule-based reasoning. Cognitive Development 10, pp. 483–527.

[54] Lassila O. & Swick R.R. (1999) Resource description framework (RDF) model and syntax specification.

[55] Brickley D., Guha R.V. & McBride B. (2004) Rdf vocabulary description language 1.0: Rdf schema. w3c recommendation (2004). URL http://www. w3. org/tr/2004/rec-rdf-schema-20040210 .

[56] Hitzler P., Krötzsch M., Parsia B., Patel-Schneider P.F. & Rudolph S. (2009) Owl 2 web ontology language primer. W3C recommendation 27, p. 123.

[57] Gruber T.R. (1995) Toward principles for the design of ontologies used for knowledge sharing? International journal of human-computer studies 43, pp. 907–928.

[58] Zúñiga G.L. (2001) Ontology: its transformation from philosophy to information systems. In: Proceedings of the international conference on Formal Ontology in Information Systems-Volume 2001, ACM, pp. 187–197.

[59] Guarino N. (1998), Formal ontology and information systems, formal ontology in information systems. fois'98, 6-8 june 1998., trento.

[60] Bechhofer S. (2009) Owl: Web ontology language. In: Encyclopedia of Database Systems, Springer, pp. 2008–2009.

[61] Horridge M. & Patel-Schneider P.F. (2009) Owl 2 web ontology language manchester syntax. W3C Working Group Note .

[62] Grau B.C., Horrocks I., Motik B., Parsia B., Patel-Schneider P. & Sattler U. (2008) Owl 2: The next step for owl. Web Semantics: Science, Services and Agents on the World Wide Web 6, pp. 309–322.

[63] Cyganiak R., Harth A. & Hogan A. (2008), N-quads: Extending n-triples with context.

[64] OASIS (2009), Rxr wiki. URL: https://wiki.oasis-open.org/office/RXR.

[65] Su X., Riekki J. & Haverinen J. (2012) Entity notation: enabling knowledge representations for resource-constrained sensors. Personal and Ubiquitous Computing 16, pp. 819–834.

[66] W3C (2012), Owl 2 web ontology language structural specification and functional-style syntax (second edition). URL: https://www.w3.org/TR/owl2-syntax/.

[67] Carroll J.J. & Stickler P. (2004) Rdf triples in xml. In: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters, ACM, pp. 412–413.

[68] Adida B. & Birbeck M. (2008) Rdfa primer: Bridging the human and data webs. Retrieved June 20, p. 2008.

[69] RDFHDT (2013), Hdt - your binary format for rdf. URL: http://www.rdfhdt.org/.

[70] Stocker M., Seaborne A., Bernstein A., Kiefer C. & Reynolds D. (2008) Sparql basic graph pattern optimization using selectivity estimation. In: Proceedings of the 17th international conference on World Wide Web, ACM, pp. 595–604.

[71] Sankar S., Sayed A. & Bani-Younis J.A. (2014) A schematic analysis on selective-rdf database stores. International Journal of Computer Applications 86.

[72] Faye D.C., Curé O. & Blin G. (2012) A survey of rdf storage approaches. Arima Journal 15, pp. 11–35.

[73] Abadi D.J., Marcus A., Madden S.R. & Hollenbach K. (2007) Scalable semantic web data management using vertical partitioning. In: Proceedings of the 33rd international conference on Very large data bases, VLDB Endowment, pp. 411–422.

[74] W3C (2016), Large triple stores. URL: https://www.w3.org/wiki/LargeTripleStores#Jena_TDB_.281.7B.29.

[75] McBride B. (2002) Jena: A semantic web toolkit. IEEE Internet computing 6, p. 55.

[76] Shearer R., Motik B. & Horrocks I. (2008) Hermit: A highly-efficient owl reasoner. In: OWLED, vol. 432, vol. 432, p. 91.

[77] Stocker M. & Smith M. (2008) Owlgres: A scalable owl reasoner. In: OWLED, vol. 432, vol. 432.

[78] Sirin E., Parsia B., Grau B.C., Kalyanpur A. & Katz Y. (2007) Pellet: A practical owl-dl reasoner. Web Semantics: science, services and agents on the World Wide Web 5, pp. 51–53.

[79] SemanticWeb (2012), List of semantic tools. URL: http://semanticweb.org/wiki/Tools.html.

[80] Geelan J. et al. (2009) Twenty-one experts define cloud computing. Cloud Computing Journal 4, pp. 1–5.

[81] Chellappa R. (1997) Cloud computing: emerging paradigm for computing. INFORMS 1997 .

[82] Foster I., Zhao Y., Raicu I. & Lu S. (2008) Cloud computing and grid computing 360-degree compared. In: 2008 Grid Computing Environments Workshop, Ieee, pp. 1–10.

[83] Armbrust M., Fox A., Griffith R., Joseph A.D., Katz R., Konwinski A., Lee G., Patterson D., Rabkin A., Stoica I. et al. (2010) A view of cloud computing. Communications of the ACM 53, pp. 50–58.

[84] Zhang Q., Cheng L. & Boutaba R. (2010) Cloud computing: state-of-the-art and research challenges. Journal of internet services and applications 1, pp. 7–18.

[85] Benefits of cloud computing. http://www.it-support-singapore.com/it-services/cloud-virtualization-microsoft-hyper-v-vmware-vsphere-citrix-xenserver/benefits-of-cloud-computing/. Accessed: 2012-09.

[86] Lab C. (2016), Azure vm vs amazon ec2 vs google ce: Cloud computing comparison. URL: http://www.cloudberrylab.com/blog/azure-vm-vs-amazon-ec2-vs-google-ce-cloud-computing-comparison/.

[87] CISCO (2015), Fog computing and the internet of things: Extend the cloud to where the things are. URL: https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf.

[88] CISCO (2014), Edge computing helps ensure that the right processing takes place at the right time and place. URL: http://ubiquity.acm.org/article.cfm?id=2822875.

[89] NOKIA (2014), Radio applications cloud servers. URL: https://www-03.ibm.com/press/us/en/pressrelease/40490.wss.

[90] Minsky M.D.R. (1995) Computational haptics: the sandpaper system for synthesizing texture for a force-feedback display. Ph.D. thesis, Massachusetts Institute of Technology.

[91] CISCO (2015), Cisco fog computing solutions: Unleash the power of the internet of things. URL: http://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-solutions.pdf.

[92] Evans D. (2011) The internet of things. How the Next Evolution of the Internet is Changing Everything, Whitepaper, Cisco Internet Business Solutions Group (IBSG) 1, pp. 1–12.

[93] Xiang S. (2016) LIGHTWEIGHT DATA AND KNOWLEDGE EXCHANGE FOR PERVASIVE ENVIRONMENTS. Doctoral dissertation, University of Oulu, University of Oulu, Faculty of Information Technology and Electrical Engineering, Computer Science and Engineering.

[94] Maarala A.I., Su X. & Riekki J. (2014) Semantic data provisioning and reasoning for the internet of things. In: Internet of Things (IOT), 2014 International Conference on the, IEEE, pp. 67–72.

[95] Su X., Fucci D. & Riekki J. (2010) A framework to enable two-layer inference for ambient intelligence. In: Ambient Intelligence and Future Trends-International Symposium on Ambient Intelligence (ISAmI 2010), Springer, pp. 29–36.

[96] Kondo D., Javadi B., Malecot P., Cappello F. & Anderson D.P. (2009) Cost-benefit analysis of cloud computing versus desktop grids. In: Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on, IEEE, pp. 1–12.

[97] Stanford-Clark A. & Nipper A. (2014) MQ Telemetry Transport.

[98] Motik B., Patel-Schneider P.F., Parsia B., Bock C., Fokoue A., Haase P., Hoekstra R., Horrocks I., Ruttenberg A., Sattler U. et al. (2009) Owl 2 web ontology language: Structural specification and functional-style syntax. W3C recommendation 27, p. 159.

[99] McGuinness D.L., Van Harmelen F. et al. (2004) Owl web ontology language overview. W3C recommendation 10, p. 2004.

[100] Consortium U. et al. (1997) The Unicode Standard, Version 2.0. Addison-Wesley Longman Publishing Co., Inc.

[101] Yergeau F. (1996) UTF-8, a transformation format of Unicode and ISO 10646.

[102] Chen H., Finin T. & Joshi A. (2003) An ontology for context-aware pervasive computing environments. The knowledge engineering review 18, pp. 197–207.

[103] Edo M.B. (2015), Iot-lite ontology. URL: https://www.w3.org/Submission/iot-lite/.

[104] Bojars U. & Breslin J. (2010) Semantically-interlinked online communities ontology. NUI Galway Std .

[105] Compton M., Barnaghi P., Bermudez L., GarcíA-Castro R., Corcho O., Cox S., Graybeal J., Hauswirth M., Henson C., Herzog A. 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, pp. 25–32.

[106] Reynolds D. (2014) An organization ontology. URL http://www. w3. org/TR/vocab-org .

[107] Ojala T., Orajärvi J., Puhakka K., Heikkinen I. & Heikka J. (2011) panoulu: Triple helix driven municipal wireless network providing open and free internet access. In: Proceedings of the 5th International Conference on Communities and Technologies, ACM, pp. 118–127.

[108] Negus C. & Boronczyk T. (2009) CentOS Bible. Wiley Publishing.

Appendices

A. DATA SAMPLE

1.1. XML Raw Data

```
<Observation ID="4">
   <Time>2013-04-05T13:00:17+00:00</Time>
   <Area>92413</Area>
   <Coordinates Y="6500.877" X="2546.591"/>
   <Velocity>33</Velocity>
   <Direction>212</Direction>
   <ManualInfo>0</ManualInfo>
   <Sender>51709293<Sender/>
</Observation>
```

1.2. RDF/XML Data

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#
      "
   xmlns:obs="http://localhost/SensorSchema/ontology#" >
  <rdf:Description rdf:about="http://localhost/
     SensorSchema/ontology#Observation_51709293_4_ca18b2fc
     -edd6-455a-bfb1-d1fa47592617">
   <obs:hasDateTime>2013-04-05T13:00:17</obs:hasDateTime>
   <obs:hasID rdf:datatype="http://www.w3.org/2001/
      XMLSchema#int">4</obs:hasID>
   <obs:hasDirection rdf:datatype="http://www.w3.org
      /2001/XMLSchema#int">212</obs:hasDirection>
   <obs:hasAcceleration rdf:datatype="http://www.w3.org
      /2001/XMLSchema#double">-0.10051043494427193</obs:
      hasAcceleration>
   <obs:hasArea rdf:datatype="http://www.w3.org/2001/
      XMLSchema#int">92413</obs:hasArea>
   <rdf:type rdf:resource="http://localhost/SensorSchema/
      ontology#Observation"/>
   <obs:hasVelocity rdf:datatype="http://www.w3.org/2001/
      XMLSchema#double">33.0</obs:hasVelocity>
   <obs:hasDate rdf:datatype="http://www.w3.org/2001/
      XMLSchema#long">1365156017606</obs:hasDate>
   <obs:hasDistance rdf:datatype="http://www.w3.org/2001/
      XMLSchema#double">52.20267510687888</obs:
      hasDistance>
   <obs:hasSender rdf:datatype="http://www.w3.org/2001/
      XMLSchema#int">51709293</obs:hasSender>
   <obs:hasLongitude rdf:datatype="http://www.w3.org
      /2001/XMLSchema#double">25.46591</obs:hasLongitude>
```

```
    <obs:hasLatitude rdf:datatype="http://www.w3.org/2001/
       XMLSchema#double">65.00877166666668</obs:
       hasLatitude>
  </rdf:Description>
</rdf:RDF>
```

1.3. JSON-LD data

```
{
  "@graph" : [ {
    "@id" : "obs:Observation_51709293_4_ca18b2fc-edd6-455a
       -bfb1-d1fa47592617",
    "obs:hasAcceleration" : -0.10051043494427193,
    "obs:hasArea" : {
      "@type" : "http://www.w3.org/2001/XMLSchema#int",
      "@value" : "92413"
    },
    "obs:hasDate" : {
      "@type" : "http://www.w3.org/2001/XMLSchema#long",
      "@value" : "1365156017606"
    },
    "hasDate:Time" : "2013-04-05T13:00:17",
    "obs:hasDirection" : {
      "@type" : "http://www.w3.org/2001/XMLSchema#int",
      "@value" : "212"
    },
    "obs:hasDistance" : 73.72751275943494,
    "obs:hasID" : {
      "@type" : "http://www.w3.org/2001/XMLSchema#int",
      "@value" : "4"
    },
    "obs:hasLatitude" : 65.02544166666667,
    "obs:hasLongitude" : 25.50801166666667,
    "obs:hasSender" : {
      "@type" : "http://www.w3.org/2001/XMLSchema#int",
      "@value" : "51709293"
    },
    "obs:hasVelocity" : 33.0,
    "http://www.w3.org/1999/02/22-rdf-syntax-ns#type" : {
      "@id" : "obs:Observation"
    }
  }],
  "@context" : {
    "hasDateTime" : {
      "@id" : "http://localhost/SensorSchema/ontology#
         hasDateTime",
      "@type" : "@id"
```

```
},
"hasArea" : {
  "@id" : "http://localhost/SensorSchema/ontology#
     hasArea",
  "@type" : "@id"
},
"hasDate" : {
  "@id" : "http://localhost/SensorSchema/ontology#
     hasDate",
  "@type" : "@id"
},
"hasLatitude" : {
  "@id" : "http://localhost/SensorSchema/ontology#
     hasLatitude",
  "@type" : "@id"
},
"hasDistance" : {
  "@id" : "http://localhost/SensorSchema/ontology#
     hasDistance",
  "@type" : "@id"
},
"hasVelocity" : {
  "@id" : "http://localhost/SensorSchema/ontology#
     hasVelocity",
  "@type" : "@id"
},
"hasLongitude" : {
  "@id" : "http://localhost/SensorSchema/ontology#
     hasLongitude",
  "@type" : "@id"
},
"hasSender" : {
  "@id" : "http://localhost/SensorSchema/ontology#
     hasSender",
  "@type" : "@id"
},
"hasDirection" : {
  "@id" : "http://localhost/SensorSchema/ontology#
     hasDirection",
  "@type" : "@id"
},
"hasAcceleration" : {
  "@id" : "http://localhost/SensorSchema/ontology#
     hasAcceleration",
  "@type" : "@id"
},
"hasID" : {
```

```
    "@id" : "http://localhost/SensorSchema/ontology#
       hasID",
    "@type" : "@id"
   },
   "obs" : "http://localhost/SensorSchema/ontology#"
  }
}
```

1.4. Entity Notation Data

```
[urn:uuid:7bcf39 4 92413 65.02544166666667
   25.50801166666667 33.0 212 51709293 52.20267510687888
   -0.10051043494427193 1365156017606 2013-04-05T13
   :00:17+00:00]
```

1.5. Examples

1.5.1. Example 1

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

# Someone knows someone else, who has the name "Bob".
[] foaf:knows [ foaf:name "Bob" ]
```

1.5.2. Example 2

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

[ foaf:name "Alice" ] foaf:knows [
   foaf:name "Bob" ;
   foaf:knows [
      foaf:name "Eve" ] ;
   foaf:mbox <bob@example.com> ] .
```

1.5.3. Example 3

```
<http://example.org/#spiderman> <http://www.perceive.net/
   schemas/relationship/enemyOf> <http://example.org/#
   green-goblin> ;
                        <http://xmlns.com/foaf/0.1/name>
                           "Spiderman" .
```

1.5.4. Example 4

```
<http://example.org/#The Greatest> <http://xmlns.com/foaf
   /0.1/name> "Muhammad Ali", "Cassius Marcellus Clay Jr
   ."@en .
```

1.5.5. Example 5

```
<http://example.org/#The_Greatest> <http://xmlns.com/foaf
   /0.1/name> "Muhammad Ali", "Cassius Marcellus Clay Jr
   ."@en .

which is equal to:

<http://example.org/#The_Greatest> <http://xmlns.com/foaf
   /0.1/name> "Muhammad Ali" .
<http://example.org/#The_Greatest> <http://xmlns.com/foaf
   /0.1/name> "Cassius Marcellus Clay Jr."@en .

but not equal to :

<http://example.org/#The_Greatest> <http://xmlns.com/foaf
   /0.1/name> ("Muhammad Ali" "Cassius Marcellus Clay Jr
   ."@en) .

As the name is not a collection but one of the items in
   the collection.
```

1.5.6. Example 6

```
:weddingCouple <http://xmlns.com/foaf/0.1/name> (:Lily :
   Sam) .

which can not be separated, so it is not equal to:
:weddingCouple <http://xmlns.com/foaf/0.1/name> :Lily .
:weddingCouple <http://xmlns.com/foaf/0.1/name> :Sam .

As Lily is a person not a couple. So does Sam.
```