Towards an Interoperable Internet of Things Through a Web of virtual things at the Fog layer

Behailu Negash[1],
Tomi Westerlund[1], and Hannu Tenhunen[2]

[1]*Department of Future Technologies, University of Turku, Turku, Finland*
[2]*Department of Electronics, KTH Royal Institute of Technology, Stockholm, Sweden*

Abstract

A wide range of Internet of Things devices, platforms and applications have been implemented in the past decade. The variation in platforms, communication protocols and data formats of these systems creates islands of applications. Many organizations are working towards standardizing the technologies used at different layers of communication in these systems. However, interoperability still remains one of the main challenges towards realizing the grand vision of IoT. Intergration approaches proven in the existing Internet or enterprise applications are not suitable for the IoT, mainly due to the nature of the devices envolved; the majority of the devices are resource constrained. To address this problem of interoperability, our work considers various types of IoT application domains, architecture of the IoT and the works of standards organizations to give a holistic abstract model of IoT. According to this model, there are three computing layers, each with a different level of interoperability needs - technical, syntactic or semantic. This work presents a Web of Virtual Things (WoVT) server that can be deployed at the middle layer of IoT (Fog layer) and Cloud to address the problem of interoperability. It exposes a REST like uniform interface for syntactic integration of devices at the bottom layer of IoT (perception layer). An additional RESTful api is used for integration with other similar WoVT servers at the Fog or the Cloud layer. The server uses a state of the art architecture to enable this integration pattern and provides means towards semantic interoperability. The analysis and evaluation of the implementation, such as performance, resource utilization and security perspectives, are presented. The simulation results demonstrate that an integrated and scalable IoT through the web of virtual things can be realized.

Keywords: Internet of Things, Web of Things, Architecture, Fog computing, Interoperability, Thing Description

1. Introduction

A simplified view of the Internet of Things is presented as a global network of real world things embedded with sensors and actuators extending the existing Internet that is dominated mostly by computers and mobile devices. However, this simplification leads to ambiguity in understanding the characteristics and requirements of IoT systems; hence, there are multiple definitions and representation of the vision of IoT including misuse of terminologies [1]. In contrast to the current Internet, which is also referred as the Internet of people [2], the Internet of Things adds connected devices that read real world scenarios and alter them through remotely controlled actuators over the Internet. The diversity of the possible systems that utilize IoT brings a wide range of variations in the requirements of the devices. These requirements are mostly driven by the environment that IoT systems are expected to reside in. Some of these environmental factors include distance from network connectivity, size and nature of the surrounding environment or the host entity. For instance, comparing a smart home system with smart agriculture, the former requires short range communication and its proximity to main electric power grid gives it more freedom than the latter which requires long range communication and battery powered devices. To address these requirements, various network protocols, device platforms and data models have been introduced. As a result of these and related choices of technologies, IoT is fragmented without proper integration to unify it.

The introduction of the World Wide Web in 1989 as a high level abstraction layer on top of the Internet connectivity has significantly enhanced the adoption and integration of Internet systems [3]. It provides a standard way of accessing and formatting the information exchanged between devices connected on the Internet forming a shared open information space. It provides a centralized, yet distributed, layer for the global pool of information generated by people and machines. It is based on three key factors that promotes interoperability: uniform addressing, common protocol and format negotiation of the exchanged information [3]. To leverage the Web for information exchange among IoT devices based on only existing Web standards prohibits the inclusion of a large proportion of IoT components that use non-IP network protocols. It needs careful design decisions to collect the information from devices that are mostly resource constrained and connected over a low bandwidth, low powered, various wired and wireless protocols. The web of things (WoT) is used to refer to the Web dominated by information generated by IoT devices [4]. The heterogeneity of IoT challenges building the WoT to the scale of the Web and grow even further as expected.

To elaborate on the challenges of integrating the IoT systems and the need for the use of the Web as a unifying platform, a side by side view of different IoT stakeholders and systems are shown in Figure 1. On the left side of the figure, you find stakeholders. The interaction between these stakeholders can take place through different application domains, such as smart cities, smart healthcare or smart buildings. Please note that this is not by any measure an exhaustive list, but can be extended to include missing entities to complete other extended usage scenarios. However, a simple use case is where a device manufacturer produce an IoT product targeted for personal use or for companies (such as wearable devices or production machines). These devices are mostly designed to work with the manufacturers private Cloud and customers are granted access with proper authentication to their private data. These types of applications are wide spread in the past few years and are vertically isolated systems. In this model of applications, inputs from standards organizations in the communication protocol or data format might be applied and depending on the communication network internet service providers get involved. Most of the IoT system domains shown on the right side of Figure 1 work in the above model. The involvement of third party service providers, open-source developers, the general public and government is very limited or inconsiderable. A different approach to the private information space is the open model where governments and companies give access to the data collected by IoT devices. This interaction among the stakeholders and the smart systems presented is highly simplified. A realistic presentation of a single smart system contains a wide range of devices with various degree of resource constraints, different interfacing options and standards.

Building the emerging Web of Things, which is capable of realizing the vision of IoT and surpass it, over such fragmented technology needs novel approaches at different levels. These levels range from the way devices are programmed and connected as well as how data is collected, stored, processed and accessed. The research in this area is active in industry and academia and resulted the several contributions, for example, middleware to abstract protocol variations, programming techniques, data processing methods, and solutions for privacy and security. However, there still remains open challenges of how to enable things at the perception layer to collaborate and share information regardless of their platform, protocol and data model. Moreover, a high level unified abstraction of these devices is crucial to build a scalable and truly integrated open Web of Things.

Compared to the above presented IoT models which have small vertical application silos, the model envisioned in this paper is where systems interact horizontally and vertically across domains. Towards this goal, the main contributions of this work built into the Web of virtual things (WoVT) server are:

- **C1**-Uniform communication interface with physical things
- **C2**-Well organized domain specific script stores
- **C3**-Integrated things data model
- **C4**-A virtual things space

The rest of this paper is organized as follows. Section 2 provides different IoT use cases or domains and the various challenges associated with them. This section ends with a motivation for this work. In Section 3, different data models and protocols are shown including related works in different areas of contributions of this paper. In Section 4, the previous server that is used as starting point for this work is discussed. Section 5 provides the design and implementation of this work and the next section, Section 6 gives the analysis of the results and performance evaluation of the work. Finally, Section 7 gives concluding remarks, discussion of the findings and a view to the future of the work.

2. IoT use cases and motivation

Different applications of the Internet of Things bring unique challenges to integration due to variations in the device architecture, network protocol, used data format. The use cases listed in Figure 1 on the right are a small subset of the possible application scenarios. However, they provide insight to the type of used devices and the working environment that indicate the challenges associated with a major portion of IoT elements. Focusing on the interoperability challenges of IoT and the sources of heterogeneity, this section highlights the main motivations of the work. According to a report by

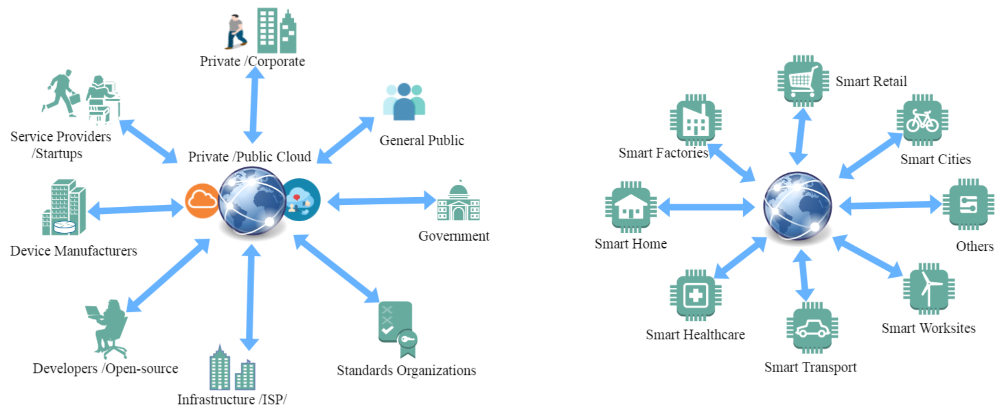

Figure 1: Stakeholders interactions in IoT. On the left side, stakeholders in an interoperable IoT scenario and on the right side are the different use cases or domains. The illustration is simplified as a star assuming IoT as a central system rather than a mesh.

McKinsey [5], 40% of the value of the Internet of Things is yet unavailable due to lack of interoperability. This value is distributed over a number of application domains, where industries, healthcare and cities are the major ones. They present nine use cases of IoT including their corresponding estimated values. Some of these applications are elaborated with real world usage in different settings by Lin *et al.* [6], Kraijack *et al.* [7], Al-Fuqaha *et al.* [8] and Nalbandian S. [9]. Looking through the explanation of these IoT systems, we can also see different proposals for communication, hardware and software computing platforms, data formats and semantics [8].

There are multiple connectivity protocols for things to offload the data they collected or receive commands to trigger a certain event. Some examples of these protocols, the list of which grows as new entrants are added, include Bluetooth (both classic and smart), ZigBee, 6LoWPAN, LoRA, and Sigfox. Each protocol has a unique feature that makes it preferable in specific application domains mentioned above. This makes it obvious that it is not possible to have a single protocol that can satisfy all the requirements for the use cases. Therefore, a means to enable them work together is a winner in this case. Another area of variation is the data model followed by devices to communicate with their peers in the network. For example, the Open Connectivity Foundation (OCF), IP for Smart Objects (IPSO), and W3C are all working to define a standard model for things in IoT. Similarly, a single data model appropriate to describe things in every domain might not be available in the near future. Device manufacturers chose what seems to fit their need in describing their devices. However, an abstract model needs to be used for different application domain to work together regardless of the data model they use internally.

To build a unified IoT system over such a fragmented infrastructure with the current state of the art approaches is practically impossible. It requires pragmatic approaches to the technical challenges and the involvement of all the stakeholders. Moreover, proposed integration methods should be agnostic to the type of application domain it supports to plug a wide range of use cases including others that arise in the future. This ensures that practically all the requirements of the billions of devices that will be connected to the Internet of Things will be satisfied. Similarly, existing and new network protocols and data formats can be included by simple extension without the need to modify existing infrastructure. In this regard, this paper presents an interoperability approach through a server that supports various network protocols over a uniform interface, using different data models and form the building block of the web of things. It can be incrementally extended to support other protocols and models.

3. The IoT landscape - Related works

To identify the gap and carefully position the area where this work fits in the big picture, we elaborate on some of the use cases presented in Section 2, present the challenges and proposals by other works. Moreover, elaborating with examples gives a better understanding of the need for integration and a common description of the things in IoT. To clarity the concepts presented in this section, we discuss the end to end communication from an embedded device all the way to the end user of the data (another device or human user). This communication can be in any of the three ways as presented in RFC 7452, Architectural Considerations in Smart Object Networking [10]. These are: Device to Device, Device to Gateway and Device to Cloud. Even though all of these have specific usage scenario in the Internet of Things, the second one is the most widely used architecture to build the Web of Things. The majority of the devices in IoT are resource constrained to connect directly to the Cloud. These resource constraints lead to different classes of devices and limitations in the network capability [11], [12].

In an effort to bring IP based networking to these resource constrained devices and integrate them directly to the Web of Things, the Constrained

Application Protocol (CoAP) was presented in [13]. Sheng *et al.* [14] present a summary of the Internet Engineering Task Force (IETF) efforts for the Internet of Things. A closer look at the network stack shows that CoAP requires IP as a network layer; this means it is not applicable to non IP protocols such as Bluetooth and ZigBee. As an example to highlight the procedure of building an IoT system, consider a smart home application that needs to know the temperature of the room instruct the air conditioner to adjust the temperature to a certain value, as shown in Figure 2. To fit this example in the above architecture (Device to Gateway), a home gateway is required. Furthermore, let us have a wearable fitness application in addition to the room thermostat and the air conditioner. All these are made by different companies, and therefore each device use different protocol, send data to a different destination and usually each device is accompanied by a mobile application to sync the data or control the device.

To visualize the challenges in integrating the three devices mentioned above, working in two very simplified use cases (healthcare and smart home), we extend the scenario further with hypothetical implementation details. Bluetooth low energy (BLE) [15] is a popular protocol in wearable devices, such as fitness tracking and physiological signal measurement; let the wearable device in this example use BLE. Similarly, ZigBee [16] and 6LoWPAN [17] are two protocols used in our example by the thermostat and the air conditioner respectively (shown in Figure 2). To represent the information sent from these devices in the most expressive way possible, different device descriptions are used. This allows the receiving device to interpret the data without any ambiguity leading to semantic interoperability of the systems built on these devices. Contributions and current approaches, both from the industry and research, in building an integrated system as in our example will be shown next.

3.1. Communication Protocols

To start the integration, we begin by looking at similarities and differences in the communication protocols. All the three protocols use the same physical medium, 2.4 GHz radio [18], [15], to exchange information. However, upper layers of the communication protocol vary to satisfy their unique requirements. For the wearable device to get the room temperature and write a command to the room air conditioner, a developer has to decide the proper GATT (Generic Attribute Profile) attributes, define the right GAP (Generic Access Profile) and security mechanism. Moreover, the BLE module interface requires some type of Serial communication to work with the BLE library. The gateway also requires steps to handle the commands sent from the wearable device. Similar challenges are faced on both ends of the network to integrate the ZigBee and 6LoWPAN based devices; understanding the protocol, using different programming interfaces and translating messages from one format to another.

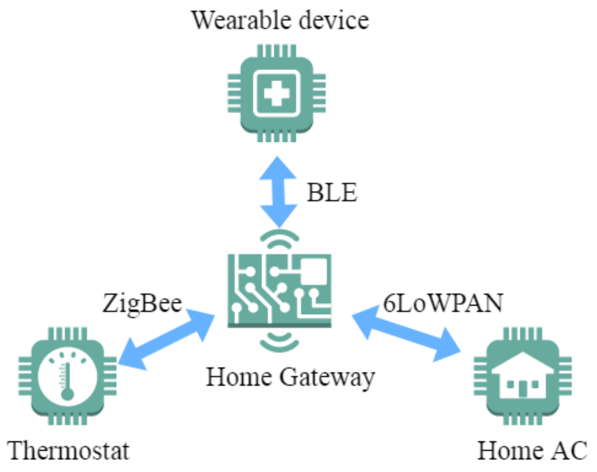

Figure 2: Sample scenario for integration of three IoT devices, from two application domains using different protocols to communicate to a central home gateway.

Each of the protocols use different addressing, packet format and programming approach. A system integrator needs to have these diverse set of knowledge to work with this simple scenario. To simplify this process, a wide range of frameworks have been proposed [19]. Most notable ones are summarized by Razzaque *et al.* [20]. Most of these middleware try to translate between the different protocols at the gateway level. However, this approach gets complicated as more protocols are added and it is not scalable. Some of the network protocols might have various options as application layer protocol. For instance, CoAP can optionally be replaced with MQTT or HTTP over the same network and transport protocols. This choice is also driven by the application logic and architecture followed. Tayur *et al.* [21] review different ways for application layer integration. In another approach to integrate application layer protocols, Lee *et al.* [22] present their work in Software-Defined Network based approach. However, this is only focused on IP based network integration at the application layer. Contribution **C1** of this paper try to address the challenges in this area by providing uniform message header without the need for message translation; instead, virtual things communicate using in-server events inside the gateway.

3.2. Description of Things

The room temperature reading from the thermostat can be formatted in many ways when sent to the gateway. A simple way is to send only the value, for example, as 25.0. However, this is open for interpretation by the receiving end and it could be either in Celsius or Fahrenheit. To clarify this, there are different proposals for describing the things. The IPSO alliance (Internet Protocol for Smart Objects) for example provides different descriptions for smart objects in IoT [23]. For instance, a temperature sensor is described with properties, such as Sensor Value, Min Measured Value, Max Measured Value, Min Range Value, Max Range Value, and Sensor Unit. Each of these properties have corresponding types (for example Float or Int). This specification is formatted in XML and includes schema information for consuming devices. Other related specification to describe things include contributions from Open Connectivity Foundation (OCF) [24] and proposed specifications from the World Wide Web Consortium (W3C) Web of Things interest group [25]. Each of these have different approaches to describing things, organizing properties and formatting. For instance, the OCF description exposes two properties for temperature sensor, temperature as numeric value and enumeration of the unit. Similarly, the W3C proposal gives the value and threshold with their corresponding units as interactions and a separate list of the events associated with the sensor. In addition to differences in the way of organization, naming of the properties, OCF and W3C use JSON compared to IPSO.

In terms of achieving semantic interoperability, Kiljander *et al.* [26] proposed an architecture using semantic information brokers. Virtual representation of physical entities in the architecture is partially related with our work. However, the integration is built using Cloud based resolution infrastructure and virtual representation of the entities is formed by composition of individual physical entities. Strassner *et al.* [26] also proposed an architecture to enable the preservation of meaning of exchanged data between devices. In addition, Mazayev *et al.* [27] implemented the thing description proposal from W3C. Contribution **C3** of this paper deals with this challenge in variations in thing description to enable semantic interoperability among different devices. Our server retains the schema information available in the thing description used by devices during creation of a virtual thing, which helps the consuming applications to resolve the meaning of the values.

3.3. Web of Things

So far the example above has only been focused to enable the exchange of data among the devices in a local area network without any ambiguity. To complete the concept of Internet of Things, these devices have to be connected to the internet, further connected to other systems and become accessible remotely. This interconnection via web technologies forms the Web of Things. However, the mechanism for Web integration is an open research area. For instance, Paganelli *et al.* [28] proposed the use of REST (Representational State Transfer) for exposing the devices to the Web of Things in their implementation of a Smart City system. In another approach, Sakamoto *et al.* [29] developed a dynamic mechanism to expose features of a device to a WoT in the Cloud. A wide set of hypermedia comparison is made by Martins *et al.* [30] among contributions from IETF, W3C, Mozilla and Everything to build the Web of Things. As indicated earlier, most devices connected to IoT are resource constrained. For these devices to be available over WoT and interact with users Abeele *et al.* [31] proposed an HTTP-CoAP proxy. Moreover, Kumar *et al.* [32] developed a solution to enable a web browser to access information from a WoT both in online and offline modes. In comparison, our proposal is not focused on the presentation of the data but on how to unify and make the information accessible. The WoVT Server exposes a RESTful end point for both horizontal integration with other gateways as well as the Cloud layer. The exposed information however is not tied to any of the thing models but a JSON representation of the virtual entity.

In summary, there are many contributions in the area of interoperability at different levels that relate to our work. Some of these efforts have been discussed in relation to the server presented here, focusing on two of the main contributions of the work, **C1 and C3**. Contributions **C2** and **C4** are also elaborated in Section 5 along with the details of the server design and implementation.

4. Domain Specific Script Server

The WoVT server is an extension of a previous conceptual work presented in Negash *et al.* [33]. It was originally started to tackle the challenge of scalability, interoperability and maintainability in IoT. The main contribution of the paper was to enhance the flexibility of re-programming a device. Most updates in device programs are due to a change in functional specification of the system; for example, the basic components or building blocks like reading a sensor or rotating a motor are usually the same but how it should format the value, or the number of rotations change. Hence, a domain specific language is developed to control the configuration of these resources and procedures allowing easy configuration even after deployment of the system.

These configurations written in a domain specific language [34], or the scripts, are stored in WoVTS that dispatches them according to a request by the things. The scripts were organized according to an ontology called IoT-lite [35]. It was extended for the script organization by annotating the device with unique name and path to the script. Requests to this server arrive in different communication protocols over a very simple message format used as a unifying layer. The server parses the request and based on the resource url or device name that is sent from the thing, the path to a corresponding script is queried using *rdflib*. The script is then sent back as a response. In this work, in addition to requests to scripts extended communication with the things is handled forming the WoVT Server. This version removed the need for IoT-lite ontology and relies only on device url to locate the scripts.

4.1. Code on Demand

A holistic view of IoT helps to tackle the problems better. The WoVT Server has a recommended client configuration that fits in the overall architecture. There are two major approaches to programming a device that connects to a network using any protocol. The first and common one is to write the compete procedure of initializing the connected sensors, read the value, and send the values to a defined destination. The second option is to have a service, such as an embedded web server, that receive requests and parse them to respond with proper content, such as the value of the sensor. WoVT Server works well with the first type but the recommended one is different from both types. Both approaches assume that the device has all the resources (such as sensors, radio interface and actuators) to accomplish the task and also the work flow. Our initial work addresses the role of WoVT Server to host domain specific scripts, written in DoS-IL [34], that instruct the device work flow. This makes the smart objects to act in purely client mode, by sending instructions or code to clients on demand, also called mobile code. Mobile code can be implemented using Code on Demand (COD) that is one of the constraints in REST architecture style [36]. It allows the client devices to extend their feature and enhance their maintainability by modifying the scripts hosted on the WoVT Server. Fuggetta *et al.* [37] discuss the advantages of applying mobile code approach to a system. One of these advantages is for remote device control and reconfiguration. Our use of this style for reconfiguration of smart objects is inspired by this model.

5. Web of virtual things

The initial server implementation that inspired the current work is discussed in Section 4. The core principle behind the virtual things server presented in this work is the abstraction of physical entities, which are augmented with sensors and actuators, to have a digital representation or an agent regardless of the data model it follows or the communication protocol used. This will facilitate the integration of multiple protocols and standards forming a unified endpoint for client access through a standard RESTful interface. To achieve this, the architecture of the system plays a critical role. The following sections discuss the system architecture of a single node, the implementation and potential of a distributed network of virtual thing servers.

5.1. System Architecture

The architecture of Web of Virtual Things Server is discuss from different view points [38]; deployment view showing the hierarchical organization of different computing layers in IoT and how WoVTS functions, a component view showing the internal modular organization of the server and finally the dynamic behaviors or interactions at run time. The deployment view of WoVTS can be easily understood using a typical three tiered computing architecture of IoT, which makes use of an intermediate Fog computing layer [39]. At the bottom of the hierarchy is the perception layer where the physical smart objects live. These things can be part of any of the smart systems shown in Figure 3. Devices at this layer use different communication protocols to connect and offload the data they collect or request for information. They are also very diverse in the available resources such as computing, storage, battery life and network bandwidth. The Fog layer serves as gateway for these devices to the external world providing access to the Internet by connecting to a Cloud server or a local network. These gateways can reside near any of the smart systems listed, for instance at home, hospitals, on the

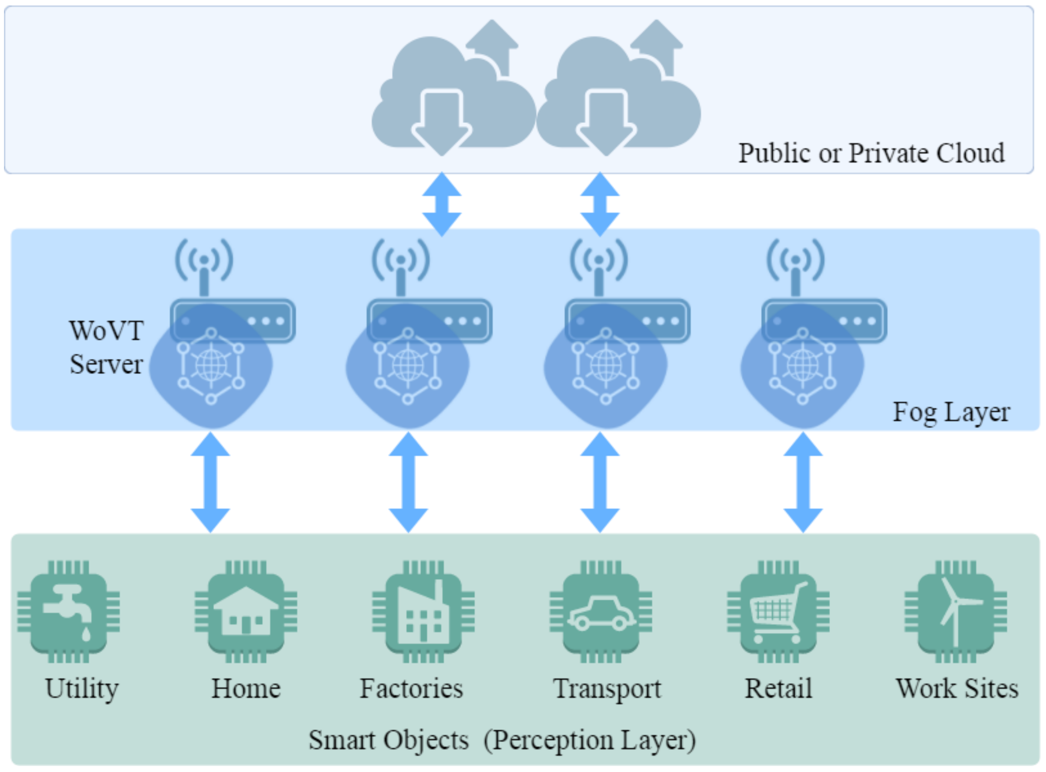

Figure 3: Deployment view of the WoVT Server. Devices are shown in three tiers; perception, Fog and Cloud. The WoVT Server is shown at the Fog layer communicating to upper and lower layers vertically. Horizontal communication between WoVT servers exists but it is omitted for simplification of the Figure.

streets or in factories. They are accessible from their own local networks, by other gateways and Cloud servers via Internet. In most cases, direct communication with users is restricted to ensure privacy and security needs of the interacting systems. More details on the security and privacy requirements of the server come later on after the internal components.

The internal organization of the server and the interaction of the components is kept simple. From the perception layer side, there is a single Transport Handler component that initializes and manages the various network interfaces supported by the gateway. Any incoming request through the network interfaces is submitted to the Transport Handler which puts the messages in an incoming message queue. Similarly, after handling the requests, response messages are submitted to the Transport Handler in an outgoing message queue. This component is created by the main Server and handles messages asynchronously. Figure 4 visualizes this interaction and position of the Transport Handler. The second component, Thing Factory, deals with the abstraction of the physical things that send or receive messages to the server. It keeps list of known thing descriptions for different smart objects and depending on the namespace or definition used, it parses the description from local store to create a virtual representation. The abstractions created by the Thing Factory are given to the Thing Registry to keep track of subsequent interaction. Other components include Thing Space and Event Store, which are mostly linked to the dynamic behavior of the Server in handling activities of virtual things. The Thing Space is responsible for horizontal communication with other gateways or the cloud via its RESTful services. The uniform interface provided by REST and its lightweight nature coupled with the fact that similar interface is developed for the communication with the perception layer makes REST preferable compared to other options such as SOAP (Simple Object Access Protocol). Moreover, an external component, Redis, is used to manage the publishing and subscription of events as well as the persistence of the virtual things by working with the support of Event Store module. Redis is an in memory data store and message broker [40]. Its performance to handle the large amount of events and interaction in IoT added to the small footprint makes it perfect for our server. It is shown outside the server boundary in Figure 4 indicating the scope of the work.

In addition to addressing the integration requirements of our WoVT server, the system architecture should be able to ensure the privacy and security means already in place by the protocols, devices and platforms are not compromised. It is also necessary to revisit the stakeholders of IoT systems, shown in Figure 1, to see the role of each stakeholder, possible security threats and associated risks categorized by application domain. For instance, a Healthcare IoT system which collects remote patient data is more critical than a smart city system collecting traffic information. Regardless of the application domain, there are three main areas where the server interact with the underlying infrastructure; during communication, storage and processing of the data. The exposed interfaces from the server that are used for communication with the bottom layer or between servers use existing communication protocols, such as Bluetooh, Zigbee or Http, to transport the message. Therefore, the security of messages during communication is as good as the the protocol underneath. In the current version of the Server, storage and analysis is limited to only the most recent data. In addition, a separate module is needed to control the permission of one virtual thing to access the data of another virtual device. It also needs a means to authenticate and authorize each device. Finally, this work is targeted to be generic and application domain independent. As such detailed analysis of security considerations for each application domain go beyond the main focus of this work.

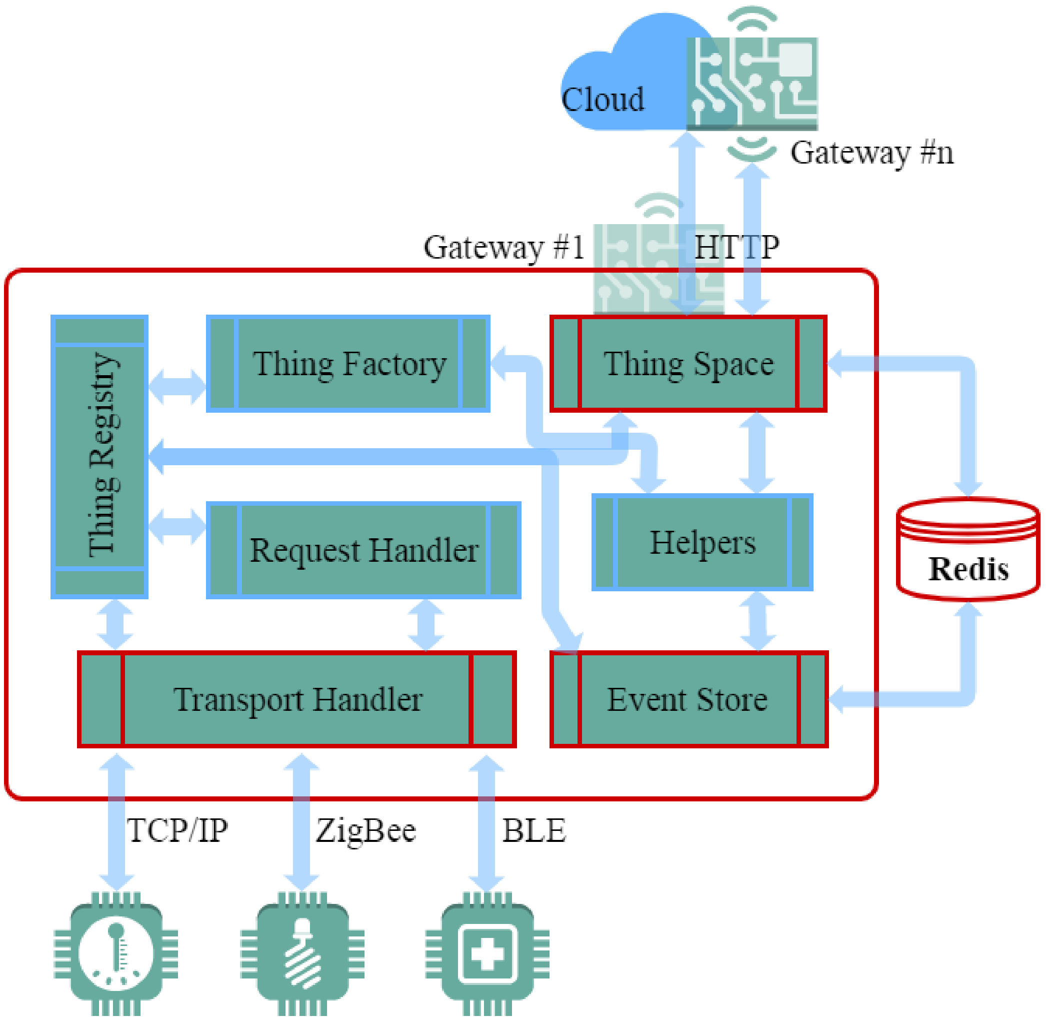

Figure 4: Components and their interactions in the WoVT Server. Threads are shown in red color and the communication between internal and external components is shown in blue arrows.

5.2. WoVT in Action

The server is implemented using Python for ease of development and test across different platforms. The components identified in the previous section are implemented using different Object Oriented design patterns. The virtual things server is a collaboration of four threads. The outer boundary in Figure 4 is the main server logic that initialize the remaining components. Traversing the diagram from bottom to top, we find the **Transport Handler**, which runs in a separate thread and monitors incoming messages from the set of network interfaces. These interfaces have an abstract base class, *Network Interface*, that defines the required methods that need to be implemented by any supported protocol. These methods include initialization routine, checking availability of new message, sending and receiving. For the sake of simplifying the simulation for different protocols, there are also concrete methods in this demo version to create different requests.

Three simulated network interfaces inherit the base class and are initialized. These include, an nRF wireless radio, a BLE and a TCP/IP. Objects of each of these child classes are created and passed to the Transport Handler. When a request arrives through any of these interfaces, the Transport Handler initializes a **Request** object, builds a message from the interface name and Request object and puts it in incoming message queue. The structure of the Request class or the incoming message format is shown in Figure 5. The first byte includes the messaging format version, the request type or verb (GET or POST), payload format (JSON is the only one in this version) and if the request (in case of the GET request) is a subscription for notification of certain events. The second byte is for the length of the body, followed by protocol and namespace (thing description choice - one of the three descriptions). The resource url is a unique name given to the smart object, which is also defined in thing descriptions. Finally, the body contains the property update or subscription information sent from the device.

When there is a message in the out going message queue of the Transport Handler, a **Response** object is created. The format of the Response object is also shown in Figure 5. These responses can be script files broken into pieces to fit in the size of the payload, notification messages or commands. The response code shows the success or failure of the request. Subsequent bytes ensure the proper delivery of the message with remaining packets and checksum before the body. A sample response is shown in Listing 1, when a domain specific script is sent to a smart object.

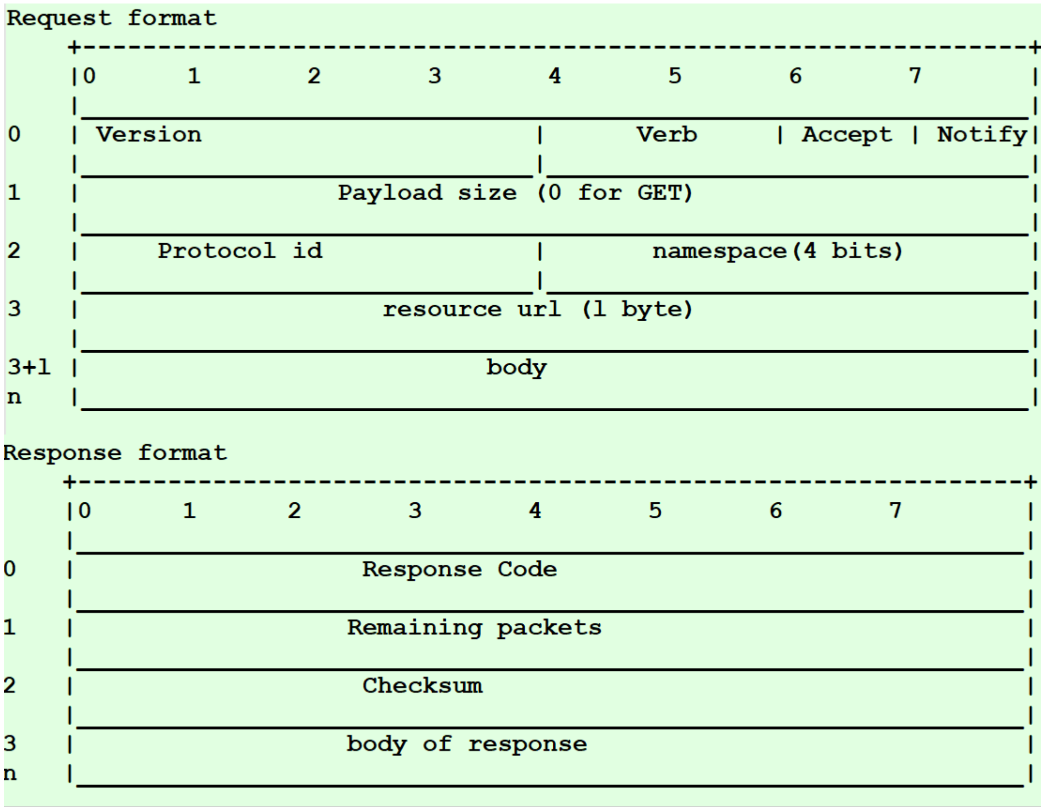

Figure 5: Request and response message formats. Request messages have only 4 bytes of header and responses have 3 byte headers carrying important information about the sender and the messaging format used.

The incoming messages are first handled by an instance of **Request Handler** class. It checks the version of the message format, checks the request method (GET or POST) and pass it to the correct handler method. In addition, the request URL is checked if it exists in the **Thing Registry** of the server. If it exists in the registry, the *Last Update* time is modified in the registry so that the *Garbage Collector* leaves the virtual thing in the registry. If the URL is not found in the registry, it is passed to the **Thing Factory** that creates the virtual representation of the Physical Thing according to the data model specified in the request. The Thing Factory has internal child classes for the creation of virtual things according to their specifications. In the current version W3C, IPSO and OCF models are supported. The created virtual thing is inserted in the registry, which also passes it to a callback that updates the **Thing Space**.

The Thing Space is a permanent (relative to the Registry) store backed by an in memory data structure store - Redis. Thing Space is a separate thread that listens to requests coming through RESTful interfaces, either from the Cloud or other peer gateways. The port 8080 is used to listen to the HTTP request and a request to the root URL results in the list of IDs of all the things registered. Appending the ID of a thing to the root url gives the detail structure of the thing. The server is backed by the handy pub-sub module of Redis, modifications to the virtual objects triggers one of the three events - **Created, Updated or Removed**. These events are dispatched from the **Event Store** thread that also allows subscribing and unsubscribing of events. These four threads work in collaboration to keep the physical and virtual things in sync while addressing user functional requirements.

5.3. Thing Abstraction

The virtual representation of the things in the digital world operates on behalf of its physical counter part. As such, the process of abstraction to design a common virtual thing model shared among devices involves identification of similarities in smart objects. All of the existing thing description proposals contain properties that identify the object and actions performed on or by the object. It also contains schema information for semantic interoperability. In addition, there exist various information useful for the execution of the server but not for users of the system. These include network interface information and last known contact time. These characteristics are organized in three main groups: metadata, properties and actions. Metadata contains read only information, whereas properties and actions can be modified (read and write); properties are changed by requests from physical thing and actions are written through HTTP POST requests. Whenever a property or action is updated, an *Updated* event fires notifying all the subscribers in the Event Store. If the update is an action, it has to be pushed to the physical thing to alter the physical behavior. A virtual thing has a configurable lifetime that starts counting down since it is created. Unless it has an activity, either GET or POST request to the server with meaningful information or a simple ping, it will be removed from the registry. This is to allow the system identify mobile clients changing their server and notify users accordingly.

5.4. Distributed execution of WoVT

The current implementation and experiment assumes a single gateway communicating only vertically down to smart objects or up to the Cloud. However, the up link can be used to communicate with peer gateways (as in Figure 4) to form a mesh network of gateways or hierarchy of gateways as defined by the OpenFog Reference Architecture for Fog Computing [41]. This creates the support of more protocols and objects from different application domains that can be easily integrated via their virtual representations forming the foundations for the next Web of Things. For the communication between gateways as well as a gateway and the Cloud, complementing the HTTP interface, additional Web Socket connections will support pushed messages in parallel to the request-response communication model.

Listing 1: Script fragmentation for response. Response code 100 shows there is still more message and 200 shows it is last message.

```
Path: dss/common/SampleWear.dsil
[100, 20, '52', '!Version 0.1!\r\nwhile ( dev']
[100, 19, '9d', 'ice ->ready());\r\n\tdevice -']
[100, 18, 'e8', '>every(12){\r\n\t\tsnumber _si']
[100, 17, 'f3', 'd:=4;\r\n\t\tsnumber _aid:=2;\r']
[100, 16, '1c', '\n\t\tsnumber _eid:=16;\r\n\t\tsf']
[100, 15, 'f6', 'ormat _format := {\r\n\t\t\t\t\ti']
[100, 14, 'e8', 'd,\r\n\t\t\t\t\ttimestamp,\r\n\t\t\t\t\t']
[100, 13, 'cf', 'unit,\r\n\t\t\t\t\tvalue,\r\n\t\t\t\t\tt']
[100, 12, 'f4', 'ype\r\n\t\t\t\t};\r\n\t\tsval_measur']
[100, 11, '40', 'e := sensor ->read(sid);\r\n']
[100, 10, '82', '\t\tif ( measure ->Value >= ']
[100, 9, '58', '12.35)\r\n\t\t{\r\n\t\t\tactuator -']
[100, 8, 'b0', '>rotate(aid,90);\r\n\t\t}\r\n\t\te']
[100, 7, '43', 'lse\r\n\t\t{\r\n\t\t\twhile ( event']
[100, 6, '2c', ' ->getState(eid));\r\n\t\t\tif ']
[100, 5, 'ef', '( event ->args(eid) < 0)\r\n']
[100, 4, '4a', '\t\t\t{\r\n\t\t\t\tdevice ->accumul']
[100, 3, 'e4', 'ate(measure,format);\r\n\t\t\t}']
[100, 2, '32', '\r\n\t\t\telse\r\n\t\t\t{\r\n\t\t\t\tdevic']
[100, 1, '82', 'e ->sleep();\r\n\t\t\t}\r\n\t\t}\r\n}']
[200, 0, 'c9', '\r\n\t\r\n']
```

6. WoVT Analysis and Evaluation

The introduced WoVT server was tested extensively to assess its performance and usability. Before delving into the details of the WoVT server, we first discuss shortly our initial, simple WoVT server whose simulation results inspired us to further continue the development.

6.1. WoVT: the first round

The initial version only serves scripts and handles GET requests from smart objects, with no interface from the Cloud. The evaluation was conducted on a hardware environment where the communication occurs over nRF radio. Sensors were connected to an Arduino Uno board with an nRF24L01 radio as a communication interface to represent a smart object. For the gateway simulation, an intel Galileo Gen2 board running a Linux distribution and a similar nRF24L01 module connected was used. As shown in the message format in Figure 5, the Arduino Uno sends GET request with its name in the *resource url* area. This url is queried from an ontology that contains the domain description of the system the device belongs to. When the path to the script is resolved, the script is broken into multiple payloads, depending on the capacity of the network interface, and sent to the client node. Listing 1 shows how a script is sent over a small bandwidth communication interface. It shows a script sent to a device broken into 21 separate messages. The formatting is also sent over the interface without any optimization or compression to remove space, tab and new line characters. The average round trip time for a script request was found to be 42 ms. Parsing RDF store to search for the path of the script based on the resource name took longer time initially but dropped sharply in subsequent requests [33].

6.2. WoVT: the second round

The second round of the evaluation is more extended and performed using single computer (a virtual machine running Ubuntu Linux). The simulated system consisted of multiple network interfaces and a large number of smart objects. We evaluated the following three points: **A1**- performance of handling incoming requests from smart objects, **A2**- internal resource requirement and **A3**- performance of handling RESTful requests from other gateways or the Cloud.

To collect better results during profiling of the server, we run it for fixed number of seconds to see how many requests are handled in the given time. For analysis A1, 1327 requests were made by 129 different smart objects (each using one of the three thing description models) over 3 network interfaces for a period of 100 seconds. These requests are either GET or POST requests. The server handled on average 1325 requests, of which 129 were unique GET requests that resulted in virtual thing creation and the remaining 1196 are either ping requests to notify the server the thing is alive or POST requests of property update. To record the activities and statistics from the server,

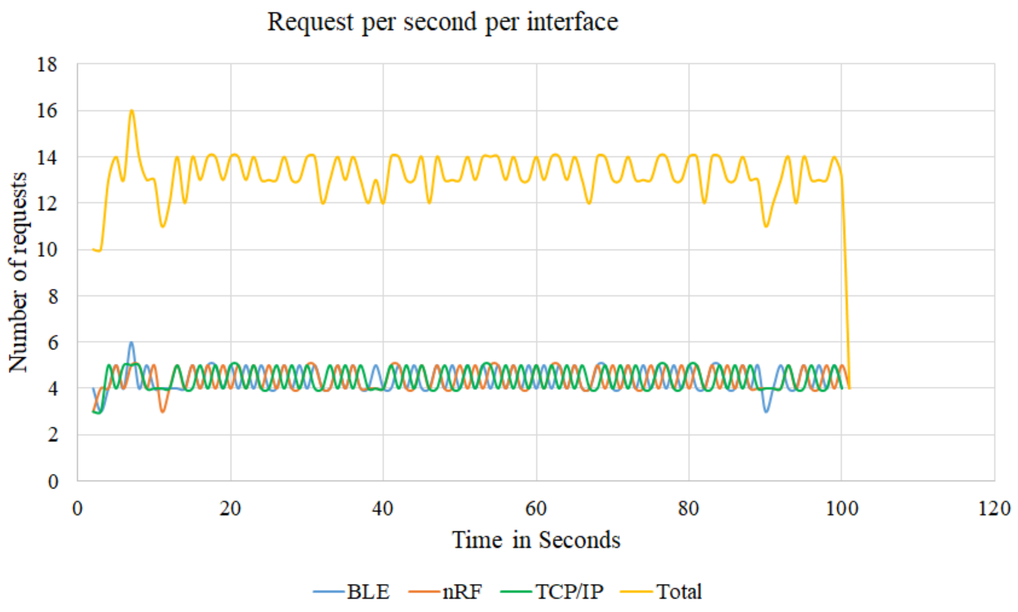

Figure 6: Requests received per interface and total requests made from perception layer. The requests are evenly distributed over the three interfaces with small fluctuation on the total request per second or incoming message queue write speed.

logging of debug information was activated during the test. Even though this has a negative effect on the performance, the log is used to get the following performance results shown in Figure 6 and Figure 7.

6.2.1. Evaluation Point: A1

The simulated network interfaces send requests at a uniform rate (average 4 per second per interface) of 13 requests per second as shown in Figure 6. The corresponding response graph in Figure 7 shows almost the same average throughput as the input. The most interesting event of the result occurs around 5 seconds. First, almost none of the incoming requests are handled before 5 seconds creating many unhandled messages in the incoming message queue; at the average incoming message speed of 13, in 5 seconds there will be at least 65 messages. This delay is caused by the initialisation of the server. After the initialisation is completed, the system processes all the messages in the queue, which shows that it is capable of serving a greater amount of devices and incoming messages than was used in this simulation. The maximum throughput that was measured was 51 messages per seconds,

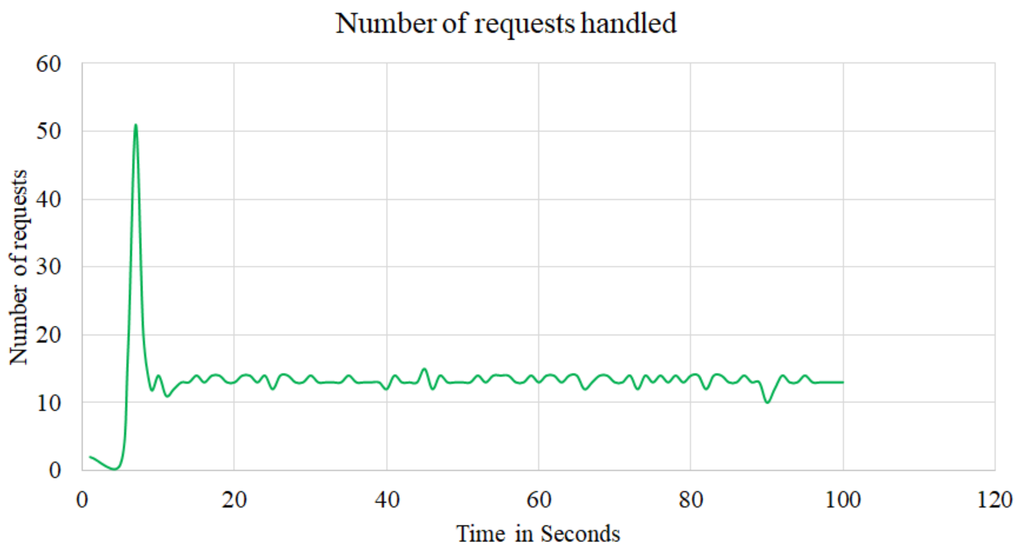

Figure 7: Response per second for smart object interface or communication with perception layer. This is the speed at which the incoming message queue is read.

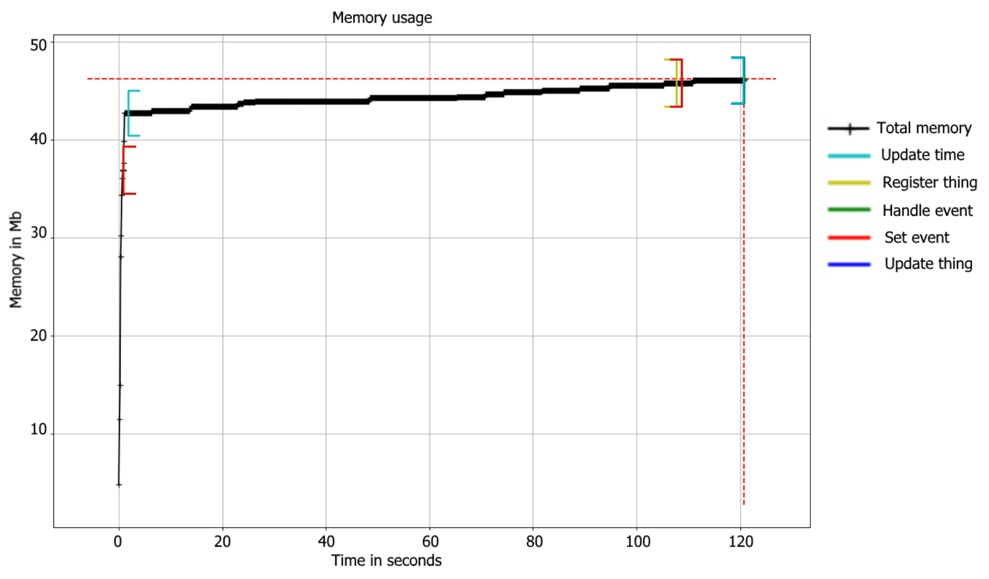

Figure 8: Memory profile of the WoVT Server. Main methods of the server are shown in different colors indicating the time it starts executing with open square brackets and ends with close brackets. Legend and graph simplified for clarity by removing small sections of execution times of functions.

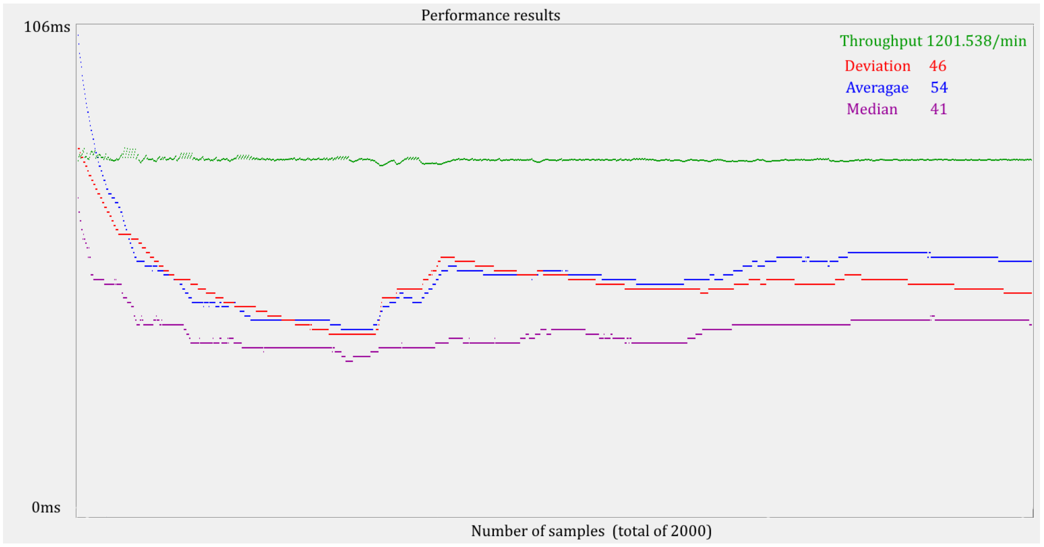

Figure 9: Performance of WoVT as measured by JMeter. The number of transactions per second, the average, mean, median and deviation of the response time are shown in different colors. It is also zoomed on the relevant features for clarity.

which is an average of 19.6ms per message including the creation of the thing's virtual image and releasing it in to the Thing Space. In summary, at the maximum speed observed (19.6ms per message) our server can handle 2000 messages, which is sufficient for a use case having hundreds of devices per gateway.

6.2.2. Evaluation Point: A2

The second part of the analysis is done on the internal resource requirements of the server. As discussed in Section 5, the main server thread initiates three child threads to execute in parallel. The memory usage of the server shown in Figure 8 is an aggregate of all the core functions called from all the threads. The graph is simplified to show the main functions with the first and last execution times in different colors on the graph. This analysis is carried out using a thread aware python *memory_profiler* module. The result is collected by executing the server for 120 seconds. During this period, the server created 129 virtual things. For this, the server needed a maximum of 45 Mb of memory, which is efficient enough for a range of devices used as IoT gateways, such as Raspberry Pi and Intel Galileo.

In addition to memory profiling, the server is also analyzed to find long running functions for future optimization work; especially unique tasks of the server, such as virtual thing creation, registration, update, event triggering and event handling. This profiling is conducted using a thread aware python profiler called *yappi* and the results are rendered in browser via *snakeviz* python module. Filtering out of 1890 entries, the *CreateThing* function from the Thing Factory module was called 793 times (including calls to generate request for simulation). It took a total of 640.3 ms for all the calls that is 0.807 ms per call. In addition, the Event Store module has two main functions, Set - trigger event and handle - broadcast event to listeners, that are called 129 times each (equal to the number of things created). Update and Delete events were not activated during this test and the number of call to this functions is only to Thing Created event. Set function has a cumulative time of 63.4ms or 0.49ms per single call and handle function took a total time of 0.6ms. The reason for this is the simple test where only one event listener subscribed to a single event. The time used by the handle function grows proportionally to the number of event subscriptions in the server. The Thing Space module also has two main functions that were called 129 times (which is equal to the created virtual things) and 664 times (equal to the number of POST requests). These numbers correspond to *NewThing* and *UpdateThing* functions that contributed a total time of 132 ms and 333.9 ms, which is equivalent to 1.02 ms and 0.50 ms per each call of the functions respectively. In summary, the observed performance of the server is promising enough for further development and enhancement. Moreover, these function calls are not happening in one thread sequentially; our server takes advantage of the parallel execution.

6.2.3. Evaluation Point: A3

Third part of the analysis covers the RESTful interface exposed for peer gateways and Cloud communication. This performance evaluation is checked using a load test tool called Apache JMeter [42]. We simulated 500 nodes each sending 4 requests over a period of 100 seconds. The sent requests were all GET requests to the root URL of the server on port 8080. A request to the root url returns the list of all the registered things in the requested server in the JSON format. As the execution of the server progresses, the size of the response increases as more things get registered. The results is shown in Figure 9, a partial graph rendered by the load test tool. The figure shows the average, median, deviation of the response time and throughput of the server over the HTTP interface. The server has a throughput of 1201 requests per

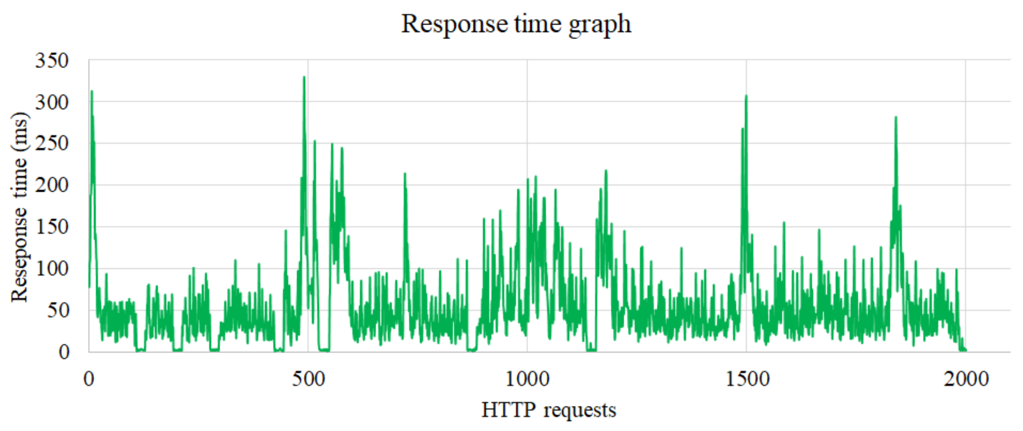

Figure 10: Response time of the Server. These are requests made through the Http interface by simulated gateways, human users or from the Cloud. The test was conducted over 2000 requests using JMeter.

minute (shown in green), which is also shown as an average response time of 54.6 ms (blue line). The median response time is 41 ms (violet line) with a deviation of 46 ms (red line).

In addition to the aggregated graphical results, the collected data is exported to a CSV file and the response time graph is shown in Figure 10. There are outlier values that can be easily identified in this graph; response times as high as 330 ms and as low as 1 ms. However, the majority of the requests, around 65%, were handled below the average time.

6.2.4. Evaluation Point: Redis

As a final part of this section, Redis performance analysis was carried. Redis has a command line interface tool to benchmark the performance in a convenient way. Using this tool over 100,000 commands, have shown an average of 90,000 and 86,500 requests per second for SET and GET commands, respectively, on the virtual machine used for simulation; the two mostly used commands in our server, SET (insert or update values by key) and GET (retrieve value by key). These values show that the use of Redis for storing the virtual things has very small performance drawback to the overall server throughput and it can scale even more with the same back-end. Moreover, the replication feature of Redis helps in bulk transfer of the virtual things from a gateway to another gateway or the Cloud.

6.2.5. Privacy and security considerations

With the wide adoption of the Internet of Things, a crucial concern users raise is the issue of privacy and security. This section covers how our implementation of a web of virtual things server address this concern. It considers three areas that can potentially be compromised. The first area is during communication with both the devices in the perception layer and with other gateways or the Cloud. As described in Section 5.1, the security of the server interface is as good as the underlying communication protocol. For instance, using Https to expose the RESTful services for communication with other servers is recommended than plain Http. In addition, the server helps in isolating the physical devices at the perception layer from any potential attack from the Internet since only the virtual counterpart is visible. The second area considered is during storage of the data in the WoVT server. As mentioned earlier, for this version only the latest information is stored. However, using existing data stores helps minimize the vulnerability of the data. The last area is during analysis of the data in the server. To address this, our implementation used Linux as a platform and core Python modules to manage the server processes. Finally, further work in this area is mandatory, as presented in Section 7, to ensure that the privacy and security of the systems integrated over the WoVT is not compromised.

7. Conclusion and Discussion

A major part of the value of the Internet of Things is still unavailable due to the lack of interoperability. Variations in connectivity protocols, platforms and data model for the things are challenging the integration work of IoT at different levels. The resource constraints in the majority of the IoT devices makes the challenge more restrictive. We presented an interoperability approach through a Fog layer server that listens to requests from the perception layer things through multiple network interfaces. It formats the incoming messages in a uniform way creating a virtual representation of the things in memory. This virtual representation, an abstract thing, represents always a physical thing - also when the physical thing is sleeping. This means that the abstract thing receives subscribed events and responds to incoming requests. If a physical thing is sleeping, it is updated when it wakes up. Virtual things are registered to a WoVT server and accessed through it over an exposed RESTful API.

The performance profiling and analysis of the server has shown promising results for further development and use in a real environment. We analyzed our server implementation from three point of views; the perception layer where smart things communicate, internal resource utilization and the interface with other servers or the Cloud. The first analysis show a pick performance of 19.6 ms per message and an average of 13.5 messages per second throughput. The server also consumed a maximum of 45 Mb of memory for a 129 virtual thing use case representing 3 types of thing models. The last part of the anlysis was for the HTTP interface which showed an average of 54.6 ms processing time per request and a throughput of 20 transactions per second.

Based on our analysis, our server has shown promising performance results that inspire more experiments in a real world scenario with performance tuning. We need also test the system with real-time requirements to push the WoVT server to its limits and beyond. Another future perspective of the server is handling of historical data of the things. In the current version, only the latest update is refelected on the virtual thing. Moreover, as mentioned before, additional modules are needed to authenticate and authorize the access of data by a virtual thing. Security and privacy concerns associated with the introduction of WoVT server will also be fine tuned for different application domains. Our model showed that other smart objects or users of a specific thing of interest do not have to wait for the object to wake up and reply to requests or continuously check if the thing has woke up. The virtual thing representation in the server can handle any required communication needs while the physical device is sleeping. Similar results could be achieved to get the latest values of the devices by using a simple data store. However, mobile devices might have already changed their location while the data store still maintains the information. Moreover, event handling per virtual device makes our approach a better candidate than a simple data store. Compared to other approaches where an embedded web server is running in smart objects to reply requests coming from other things, our model is more aware of the resource constraints of IoT devices. Finally, we have demonstrated that interoperability can be achieved across various network protocols and data models while respecting the design constraints of IoT devices, there by building an open web of virtual things at the Fog layer and scale beyond. The source code of the server and the profile results are hosted on Github for open collaboration with anyone interested [43].

References

[1] Roberto Minerva, Abyi Biru, and Domenico Rotondi. Towards a definition of the internet of things (IoT). Technical report, IEEE, 2015.

[2] Timothy Chou. *Precision: Principles, Practices and Solutions for the Internet of Things.* CrowdStory Publishing, 2016.

[3] T. Berners-Lee. Www: past, present, and future. *Computer*, 29(10):69–77, Oct 1996.

[4] Dominique Guinard. *A Web of Things Application Architecture – Integrating the Real-World into the Web.* PhD thesis, ETH Zurich, Zurich, Switzerland, August 2011.

[5] MCKINSEY Global Institute. The internet of things: Mapping the value beyond the hype. Technical report, McKinsey, 2015.

[6] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao. A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications. *IEEE Internet of Things Journal*, 4(5):1125–1142, Oct 2017.

[7] S. Kraijak and P. Tuwanut. A survey on internet of things architecture, protocols, possible applications, security, privacy, real-world implementation and future trends. In *2015 IEEE 16th International Conference on Communication Technology (ICCT)*, pages 26–31, Oct 2015.

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

[9] S. Nalbandian. A survey on internet of things: Applications and challenges. In *2015 International Congress on Technology, Communication and Knowledge (ICTCK)*, pages 165–169, Nov 2015.

[10] H. Tschofenig, J. Arkko, D. Thaler, and D. McPherson. Architectural considerations in smart object networking. RFC 7452, RFC Editor, March 2015.

[11] C. Bormann, M. Ersue, and A. Keranen. Terminology for constrained-node networks. RFC 7228, RFC Editor, May 2014. http://www.rfc-editor.org/rfc/rfc7228.txt.

[12] H. Tschofenig and J. Arkko. Report from the smart object workshop. RFC 6574, RFC Editor, April 2012.

[13] Z. Shelby, K. Hartke, and C. Bormann. The constrained application protocol (coap). RFC 7252, RFC Editor, June 2014. http://www.rfc-editor.org/rfc/rfc7252.txt.

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

[15] Carles Gomez, Joaquim Oller, and Josep Paradells. Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology, 2012. http://www.mdpi.com/1424-8220/12/9/11734.

[16] Inc ZigBee Alliance. Zigbee-pro stack profile: Platform restrictions for compliant platform testing and interoperability. Technical report, ZigBee Alliance, 2008.

[17] N. Kushalnagar, G. Montenegro, and C. Schumacher. Ipv6 over low-power wireless personal area networks (6lowpans): Overview, assumptions, problem statement, and goals. RFC 4919, RFC Editor, August 2007. https://www.rfc-editor.org/info/rfc4919.

[18] PrithviRaj Narendra, Simon Duquennoy, and Thiemo Voigt. BLE and IEEE 802.15.4 in the iot: Evaluation and interoperability considerations. In *Internet of Things. IoT Infrastructures - Second International Summit, IoT 360 degree 2015, Rome, Italy, October 27-29, 2015, Revised Selected Papers, Part II*, pages 427–438, 2015.

[19] Behailu Negash, Amir M. Rahmani, Tomi Westerlund, Pasi Liljeberg, and Hannu Tenhunen. LISA 2.0: lightweight internet of things service bus architecture using node centric networking. *Journal of Ambient Intelligence and Humanized Computing*, 7(3):305–319, 2016.

[20] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke. Middleware for internet of things: A survey. *IEEE Internet of Things Journal*, 3(1):70–95, Feb 2016.

[21] V. M. Tayur and R. Suchithra. Review of interoperability approaches in application layer of internet of things. In *2017 International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*, pages 322–326, Feb 2017.

[22] C. H. Lee, Y. W. Chang, C. C. Chuang, and Y. H. Lai. Interoperability enhancement for internet of things protocols based on software-defined network. In *2016 IEEE 5th Global Conference on Consumer Electronics*, pages 1–2, Oct 2016.

[23] J. Jimenez, M. Koster, and H. Tschofenig. Ipso smart objects. Technical report, IPSO Alliance, May 2015.

[24] Open Connectivity Foundation. Ocf smart home device specification. Technical report, Open Connectivity Foundation, June 2017.

[25] World Wide Web Consortium. Thing description. https://www.w3.org/WoT/IG/wiki/Thing_Description.

[26] J. Strassner and W. W. Diab. A semantic interoperability architecture for internet of things data sharing and computing. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pages 609–614, Dec 2016.

[27] A. Mazayev, J. A. Martins, and N. Correia. Semantic web thing architecture. In *2017 4th Experiment@International Conference (exp.at'17)*, pages 43–46, June 2017.

[28] F. Paganelli, S. Turchi, and D. Giuli. A web of things framework for restful applications and its experimentation in a smart city. *IEEE Systems Journal*, 10(4):1412–1423, Dec 2016.

[29] T. Sakamoto, H. Ito, and K. Nimura. Dynamically exposing and controlling physical devices by expanding web of things scheme. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 329–335, July 2017.

[30] J. A. Martins, A. Mazayev, and N. Correia. Hypermedia apis for the web of things. *IEEE Access*, 5:20058–20067, 2017.

[31] F. Van den Abeele, E. Dalipi, I. Moerman, P. Demeester, and J. Hoebeke. Improving user interactions with constrained devices in the web of things. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pages 153–158, Dec 2016.

[32] K. Kumar, J. Bose, and S. Tripathi. A unified web interface for the internet of things. In *2016 IEEE Annual India Conference (INDICON)*, pages 1–6, Dec 2016.

[33] Behailu Negash, Tomi Westerlund, Pasi Liljeberg, and Hannu Tenhunen. Rethinking 'Things' – Fog Layer Interplay in IoT: a Mobile Code Approach". *Springer Lecture Notes in Business Information Processing (LNBIP)*, 2017.

[34] Behailu Negash, Tomi Westerlund, Amir M. Rahmani, Pasi Liljeberg, and Hannu Tenhunen. Dos-il: A domain specific internet of things language for resource constrained devices. *Procedia Computer Science*, 109:416 – 423, 2017.

[35] M. Bermudez-Edo, T. Elsaleh, P. Barnaghi, and K. Taylor. Iot-lite: A lightweight semantic model for the internet of things. In *2016 Intl. IEEE Conferences on UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld*, pages 90–97, July 2016.

[36] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* PhD thesis, University of California, Irvine, 2000.

[37] A. Fuggetta, G. P. Picco, and G. Vigna. Understanding code mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, May 1998.

[38] Dewayne E. Perry and Alexander L. Wolf. Foundations for the study of software architecture. *SIGSOFT Softw. Eng. Notes*, 17(4):40–52, October 1992.

[39] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang. A survey on the edge computing for the internet of things. *IEEE Access*, PP(99):1–1, 2017.

[40] RedisLabs. Why redis. https://redislabs.com/why-redis/.

[41] OpenFog Consortium. OpenFog Reference Architecture for Fog Computing. Technical report, OpenFog Consortium, 2017.

[42] The Apache Software Foundation. Apache JMeter. http://jmeter.apache.org/.

[43] Behailu Negash. WoVTS python implementation source code. https://github.com/behailus/WOVT.