
Professorship of Embedded Systems and Internet of Things
Department of Electrical and Computer Engineering
Technical University of Munich

W-ADE: Timing Performance Prediction in Web of Things

Verena Eileen Schlott

Master's Thesis

W-ADE: Timing Performance Prediction in Web of Things

Master's Thesis

Supervised by Prof. Dr. phil. nat. Sebastian Steinhorst
Professorship of Embedded Systems and Internet of Things
Department of Electrical and Computer Engineering
Technical University of Munich

Advisor Ege Korkan
Author Verena Eileen Schlott
Dachauer Straße 110F
80636 Munich

Submitted on February 5, 2020

Declaration of Authorship

I, Verena Eileen Schlott, declare that this thesis titled "W-ADE: Timing Performance Prediction in Web of Things" and the work presented in it are my own unaided work, and that I have acknowledged all direct or indirect sources as references.
This thesis was not previously presented to another examination board and has not been published.

Signed:

Date:

Abstract

As the number of devices participating in the Internet of Things (IoT) rapidly grows, the challenge of interoperability across IoT platforms becomes more apparent. In order to limit fragmentation of IoT development and improve compatibility, web mechanisms and technologies can be applied, forming the Web of Things (WoT). The World Wide Web Consortium (W3C) supports the standardization of WoT by providing a platform-independent specification called Thing Description (TD). It is a machine-readable document that semantically describes metadata, interactions and interfaces of a device, indicating its functionality. However, it does not provide any information about timing performance, which is crucial for the design of optimal system compositions. This thesis presents W-ADE, a development environment for WoT and TD that facilitates manual generation, and automated prediction of timing performance of Things, merely with a TD available. Performance is guaranteed systematically, hence allowing optimization during the design phase of Thing mashups. The evaluation shows that with 99.9% confidence, W-ADE can predict average timing behavior within a range of +/- 5%. Further, it can provide approximate static network-independent process timing behavior of interaction affordances to 99.93%. To enable the design of heterogeneous IoT applications based upon these timing requirements, a proposal on how to annotate a TD based on the measured performance data is made.

Contents

1

Introduction

The *Internet of Things* (IoT) is a system of physical devices, such as sensors or actuators, which are able to communicate over various IP-level networking interfaces and eventually can be connected to the Internet. These smart things, also referred to as *Things*, enable us to monitor and interact with the physical world in a fine-grained spatial and temporal resolution. [1]
However, limitations become visible as soon as multiple Things from diverse vendors are integrated into one system. As universal application protocols and platform-independent standards are missing, companies come up with their own solutions, leading to fragmentation of IoT development. Thus, it requires complex integration work, technical expertise, and it is mostly mandatory to have real devices available to build heterogeneous system compositions. [2]
The World Wide Web and its associated technologies are capable of providing solutions for the fragmented progression of IoT and offer the foundation for the next steps beyond basic network connectivity. Hence, web mechanisms are nowadays used to facilitate communication with IoT platforms - independent from their underlying technologies. This approach of integrating real-world devices into the Web is called the *Web of Things* (WoT). [3, 4]
The World Wide Web Consortium (W3C) is seeking to standardize this web layer for IoT in order to enable effortless integration of heterogeneous devices. The core concept in this process is the *Thing Description* (TD) specification [5]. A TD is an abstraction of the capabilities of a Thing: It semantically describes its metadata, interfaces, and available communication protocols. It acts as exposed interface, facilitating the connection and communication with the described IoT instance.

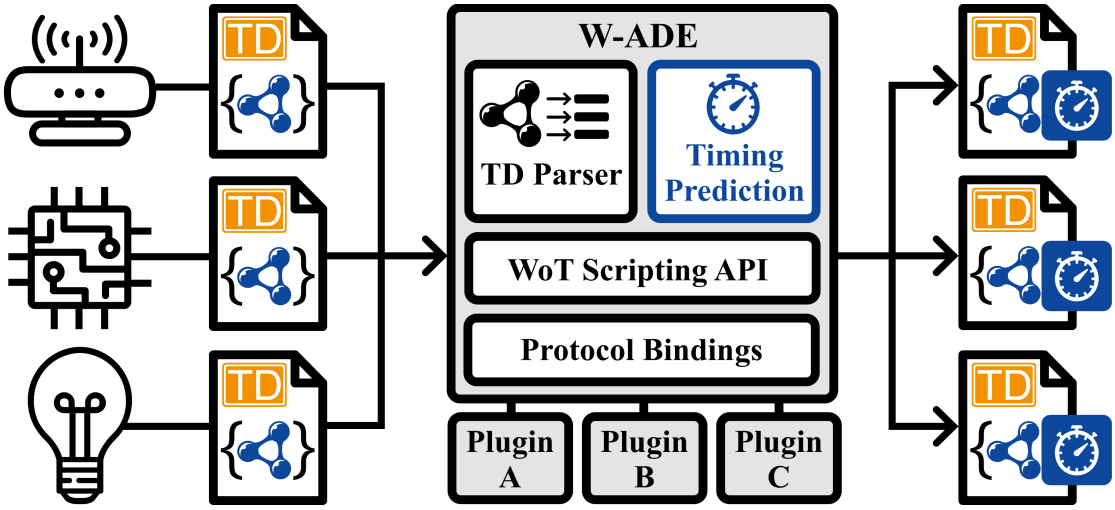

Figure 1: The Web of Things (WoT) development environment W-ADE is a standalone application based upon the W3C WoT Architecture [6]. Its core includes a WoT runtime (WoT Scripting API), protocol bindings, a Thing Description (TD) parser, and a timing performance prediction functionality. Due to its plugin-architecture, it can easily be extended with further plugins. W-ADE takes a TD as input and is able to return it, annotated with timing performance data.

1.1 Problem Statement

IoT applications are often composed of not only a single but multiple devices, also referred to as *mashups*. The TD facilitates to design them without having the actual Thing or other device documents (e.g. system specification) at disposal. To create reliable TD-based mashups, it is inevitable to have data on timing performance of the included Things. However, TDs currently do not provide information on timing. To obtain such data, a time-consuming manual process needs to be accomplished: for each protocol, a compatible service has to be started and timing behavior has to be measured manually. Although many qualitative and quantitative studies on performance of Things have been conducted, as outlined in Section 7, they are rather focusing on protocol or network than general application logic timing performance. The question on how performance can be measured with only a client-side application available, as it usually is the case when including third party IoT devices, remains unexplored.

1.2 Contribution

This work introduces *W-ADE*, an application that fills the gap of the missing foundation block of a development and testing environment for WoT, TDs and mashups, illustrated in Fig. 1. It facilitates automated timing performance measurements as well as annotating a TD with the produced data, with only the associated TD available. In particular, the following contributions are made:

- System designers are enabled to invoke single interactions of a Thing independent from its protocol, based only on its TD. This allows them to retrieve the associated timing behavior of internal application logic.
- A method to automatically generate static and dynamic timing performance predictions for device interactions is introduces and evaluated, giving estimations for worst-, best- and average-case execution with confidence interval limits.
- A vocabulary set, aligned with the TD specification, to annotate existing TDs with observed timing behavior, is proposed.

1.3 Structure of This Document

To grasp the concept of W3C Web of Things, their building blocks and concepts will be discussed in more detail in Section 2. Then the purpose, architectural design and structure of W-ADE will be explained in Section 3. This Section will give insight onto the performance prediction concept of W-ADE. After proposing the methodology and measurement implementation to give these predictions in Section 4, a recommendation how a TD can be annotated with the produced results is given in Section 5. The methodology is then evaluated in Section 6 and future tasks are identified in Section **??** The overall approach of this work is then compared to the related work in Section 7. Finally, conclusions are drawn in Section 8.

2

W3C Web of Things

The method of providing information on timing behavior evolves around the Thing Description standard. TD is one of the building blocks associated with the W3C WoT Architecture [6], which aims to prevent the further fragmentation of IoT development. The main idea is exposing device capabilities as resources in a description-oriented fashion through the WoT interface, that is, network interactions modeled as *Properties*, *Actions*, and *Events* [7]. This information can then be processed and interpreted by a *WoT Consumer*, also referred to as *Consumer*, an entity, for example another device, browser, or web application that is able to understand TDs, can consume Things and interact with them [6].
Other important building blocks are the *WoT Scripting API* [7] and *WoT Binding Templates* [8]. The Scripting API is the description of a programming interface, representing the WoT Architecture. It allows scripts to discover, operate, and expose Things. The WoT Binding Templates provide guidelines on how to define Protocol Bindings for the description of network-facing interfaces. [8]

2.1 Thing Description

In the WoT context, the TD acts as a defined representation of a Thing and can be considered the entry point for communication. As TDs are encoded in JSON-LD [9] format, they are machine-readable as well as human-understandable. The main goal is to preserve and complement existing IoT standards and solutions [6]. Thus, the TD is not a proposition for a new protocol to replace other standards but a way to represent them through syntactic and semantic information [10].
A TD document can be divided into two main blocks. The semantic metadata (including the security model) and the interaction model. The interaction model is a formal definition of mapping application intent to concrete protocol operations [6]. A TD inter-

action can therefore be understood as the description of a specific capability of a Thing, representing the data structure, access protocol and access link [10]. Consequently, a TD instance comprises a list of a Thing's interactions and how to access them. The interaction affordance is based upon the before-mentioned WoT paradigms:

- **Properties**: Exposed values of a Thing that can be *read* (e.g. sensor data), *written* (e.g. to set configuration parameters) or *observed*.
- **Actions**: Invoking them triggers physical, possibly time consuming processes (e.g. moving a robot arm) or functions inside the Thing.
- **Events**: Used for signaling asynchronous notifications that are triggered by events (e.g. a pressed button alert).

An example TD including a Action, Event, and Property is shown in List. 2.1.

2.2 Thing Description Based Mashups

As IoT systems usually consist of multiple devices, it is important to shift the focus towards mashups. Mashups in WoT are associated to digital mashups in Web 2.0. In this service-oriented web approach, a web page is not just a static markup document, but a dynamic, interactive data application that can even be consumed by other applications [11]. These describe the technology of composing modularized web applications to create entirely new services [11]. Respectively, creating WoT mashups expresses the process of aggregating WoT-enabled Things to form new applications. WoT-enabled refers to a Thing, that is accessible via its TD and can be consumed. This is done by chaining together multiple interactions, whereby the TD provides required information. A smart-home mashup could for example compose a light sensor and window shutters that are opened as the sun rises.

2.3 Mashup Limitations

When such a physical mashup is constructed, it is essential to have knowledge on the timing performance of the involved Things. When a physical process needs to be performed or internal computation needs to be completed, its execution might take time. For example, when a client has a persistent connection with a device (e.g. polling), it might send requests in a shorter time period than the device needs to process, which potentially leads to malfunctions or to a system overload. Furthermore, when interactions are dependent on each other, it is highly relevant to be able to extract data on its timing behavior from its description. Such data can give timing constraints with information on upper and lower bounds of execution times.

This becomes even more valuable when a mashup is more complex, due to including multiple interactions of different Things or interactions being dependent on other interaction's responses. Timing behavior data is especially important during the mashup's design-phase and when included Things are not available, as then, there is no opportunity to measure individual interaction request times. In order to build reliable mashups, a way to analyze, describe, and generate timing performance predictions has to be found. However, the TD does not yet include any performance related information such as timing behavior, measurement context or precision. As a remedy, this paper will introduce a way to integrate timing aspects into TDs.

```
{
    "@context": "https://www.w3.org/2019/wot/td/v1",
    "title": "Coffee-Machine",
    "securityDefinitions": {
        "basic_sc": {
            "scheme": "basic",
            "in": "header"
        },
        "security": [
            "basic_sc"
        ],
        "base": "coaps://coffee-machine.example.com:5683",
        "properties": {
            "status": {
                "forms": [
                    {
                        "href": "properties/state"
                    }
                ]
            }
        },
        "actions": {
            "brew": {
                "forms": [
                    {
                        "href": "actions/brew"
                    }
                ]
            }
        },
        "events": {
            "error": {
                "data": {
                    "type": "string"
                },
                "forms": [
                    {
                        "href": "events/error"
                    }
                ]
            }
        }
    }
}
```

Listing 2.1: A Thing Description for a smart coffee machine that exposes the machine status, a brewing action and an error event functionality, together with their URIs.

3

W-ADE: API Development Environment for WoT

In order to measure the duration of an interaction, several operations need to be performed. Depending on a Thing's implementation and the choice of protocol, a compatible service which can communicate via this protocol eventually needs to be started on the Consumer. Then, the desired endpoint has to be extracted and a request to execute it has to be sent. Subsequently, the time until the response is received has to be manually measured. To obtain representative results, this process would have to be repeated numerous times. Depending on the number of interactions, the level of their diversity and the quantity of services that need to be utilized, this can become a time-consuming and error-prone process. To minimize the susceptibility to errors and overall lighten this series of actions, *W-ADE*: *Web of Things API Development Environment*, has been developed. It simplifies the TD-based interoperation with devices, can be expanded with required protocols and facilitates timing measurement of Thing executions.

3.1 Application Features and Implementation

W-ADE can be segmented in WoT specific functionalities, typical features needed for API interaction, and the possibility to measure timing. They are respectively explained in detail in Section 3.1.1, Section 3.1.2 and Section 3.1.3.

3.1.1 WoT Features

Its core is based upon the W3C Scripting API reference implementation[1], making it compatible with WoT paradigms. It is able to parse and interpret TDs, enables users

1 Node-wot (https://github.com/eclipse/thingweb.node-wot) is based upon the JavaScript runtime Node.js (www.nodejs.org/).

to edit them and execute chosen interactions in a specific order. It acts as Consumer to communicate with virtual or physical entities over various communication protocols. Embedded Binding Templates [8] enable the incorporation of further protocols including custom ones and thus, facilitate interoperability for diverse vendors. As W-ADE supports diverse protocols existing in IoT ecosystems and web browsers are usually restricted, it is realized as a standalone Electron. Electron is JavaScript based framework, which allows to build cross platform applications and therefor eases the distribution amongst different platforms. application.
Further, W-ADE's architecture facilitates custom plugins and future adaptions, allowing it to be easily extended by numerous already existing WoT implementations that further expand the use-cases of WoT. It is a scalable application that can improve tooling around the W3C's WoT. An overview of its system architecture is presented in Figure 1.

3.1.2 ADE Features

ADEs (API Development Environment) describe software that are focused on interacting with APIs. They are utilized for designing, building, and testing APIs. Since a TD is the interface for interactions and accessing API endpoints is a main use-case, functionalities of two existing ADEs were analyzed to extract the most important features with regard to W-ADE's objectives. One ADE was the common tool *Postman*[2], an application with over 8 million users. The other was *Insomnia*[3], an ADE based on Electron. Both are HTTP Rest clients, not capable of communicating over any other protocol. To extract the most important features, a decision matrix, shown in Figure , has been created.
All features were extracted, segmented in *Basic* and *Extra* functionalities, and weighted regarding W-ADE. Features with the most weight, describing features with the most importance, were then implemented. These include sending requests with optional input values and displaying responses. Request input values are restricted based on information provided by the TD. Another substantial feature is entering, storing, and applying required security credentials. For example user-password combinations or MQTT[4] broker credentials.

3.1.3 Timing-Performance Features

Beyond that, W-ADE measures how long a target application takes to process a request and send a response. Utilizing the Node.js feature `process.hrtime()`, high-resolution real time measurements in nanoseconds are available. This and the size of the sent or received data, is then displayed.

2 https://www.getpostman.com/
3 https://insomnia.rest/
4 Message Queuing Telemetry Transport: A machine-to-machine connectivity protocol, commonly used for IoT devices.

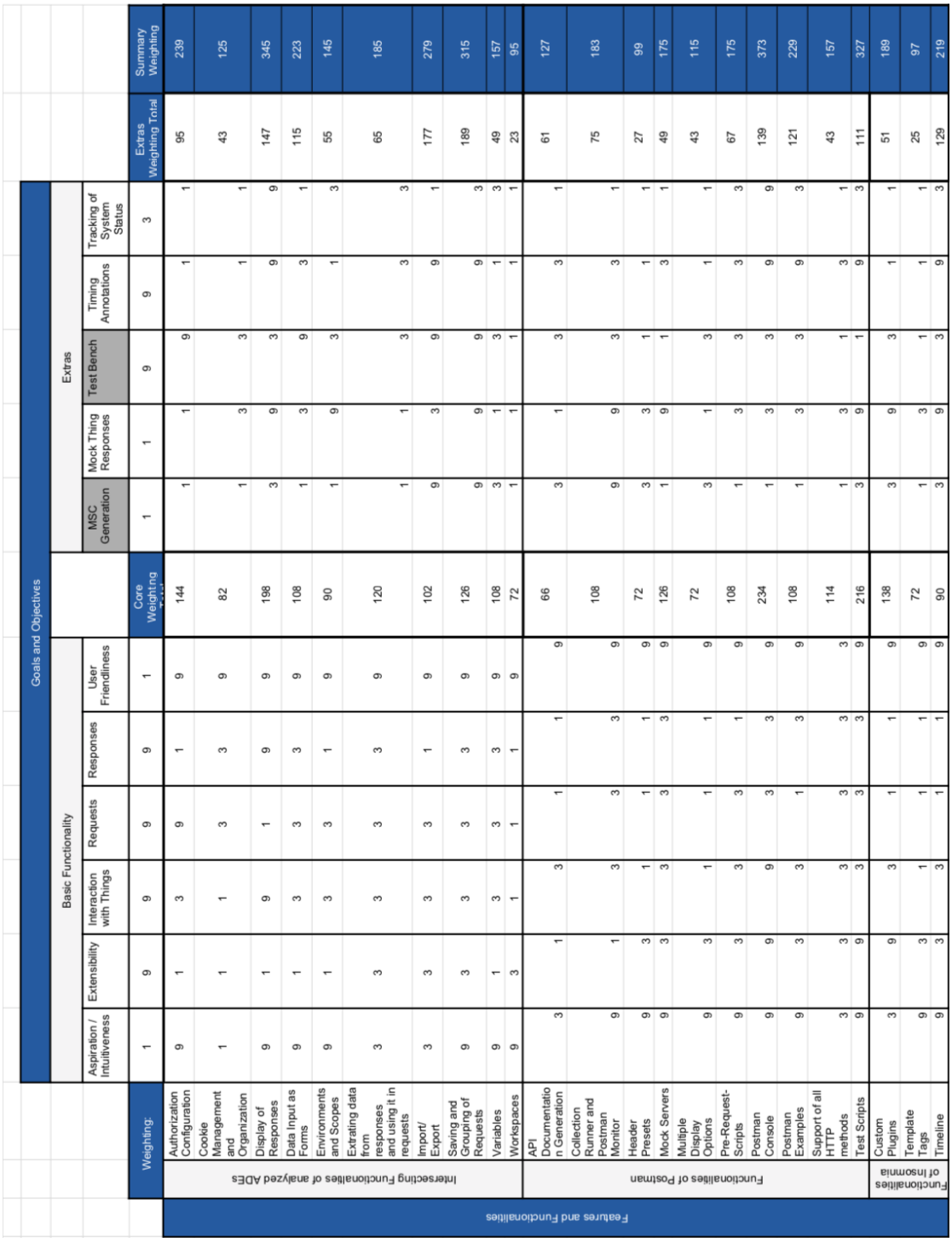

Figure 2: A decision matrix for extracting important ADE features for W-ADE's system design. Compared ADEs are Postman and Insomnia.

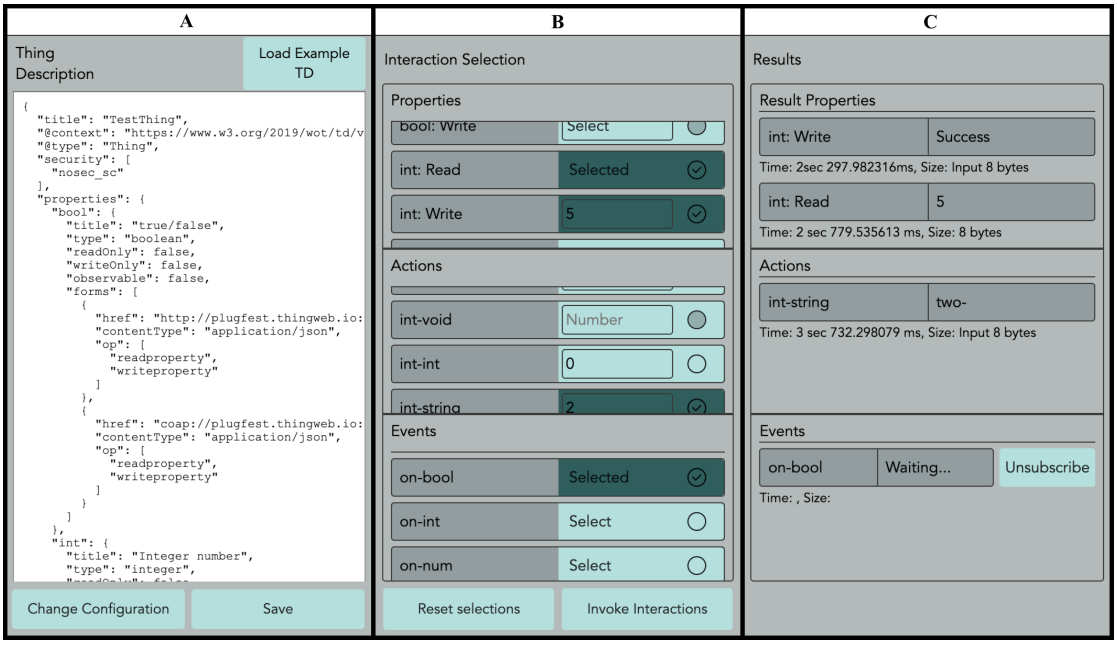

Figure 3: W-ADE's GUI. Thing Descriptions can be uploaded and edited in the Editor (A). Interactions are parsed and visualized in (B), where input values can be entered and interactions can be selected. Results, including measured communication time and payload size of input or output are displayed in (C).

3.2 Workflow

To retrieve timing information on a Thing in W-ADE, first a TD has to be inserted by the user. A new TD element needs to be created (see Figure 6). It can either be pasted, uploaded, or fetched via a URI. This TD is then consumed, parsed and all available Properties, Actions, and Events with according input fields or, alternatively, input drop-downs are generated. Interactions can then be selected and invoked. If needed required input fields have to be filled. If applicable, security credentials, can be stored beforehand (see Figure 4).

Then, a request to the associated endpoint will be sent. A sequence diagram of this workflow is displayed in Figure 5.

Simultaneously, the internal measurement is started and stopped as soon as W-ADE receives the Thing's response. The elapsed time in milliseconds and if available, the size of the received data in bytes will be shown together with the size of the request payload. The according user interface is illustrated in Figure 3.

W-ADE provides the framework for timing performance testing, eases the work with TDs and due to its plugin architecture, is able to improve tooling around the WoT ecosystem. Nonetheless, to give reliable and reusable assertions on timing behavior a more elaborate performance analysis technique is introduced in Section 4.

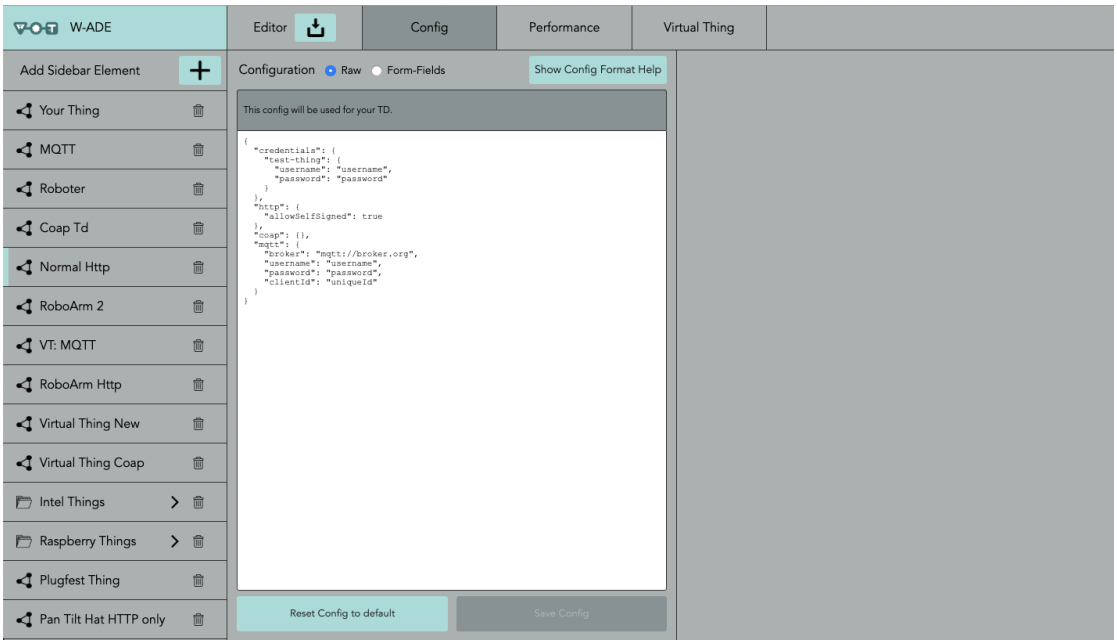

Figure 4: W-ADE's user credential GUI. If applicable users can enter security information, such as user-password combinations or for example MQTT broker information.

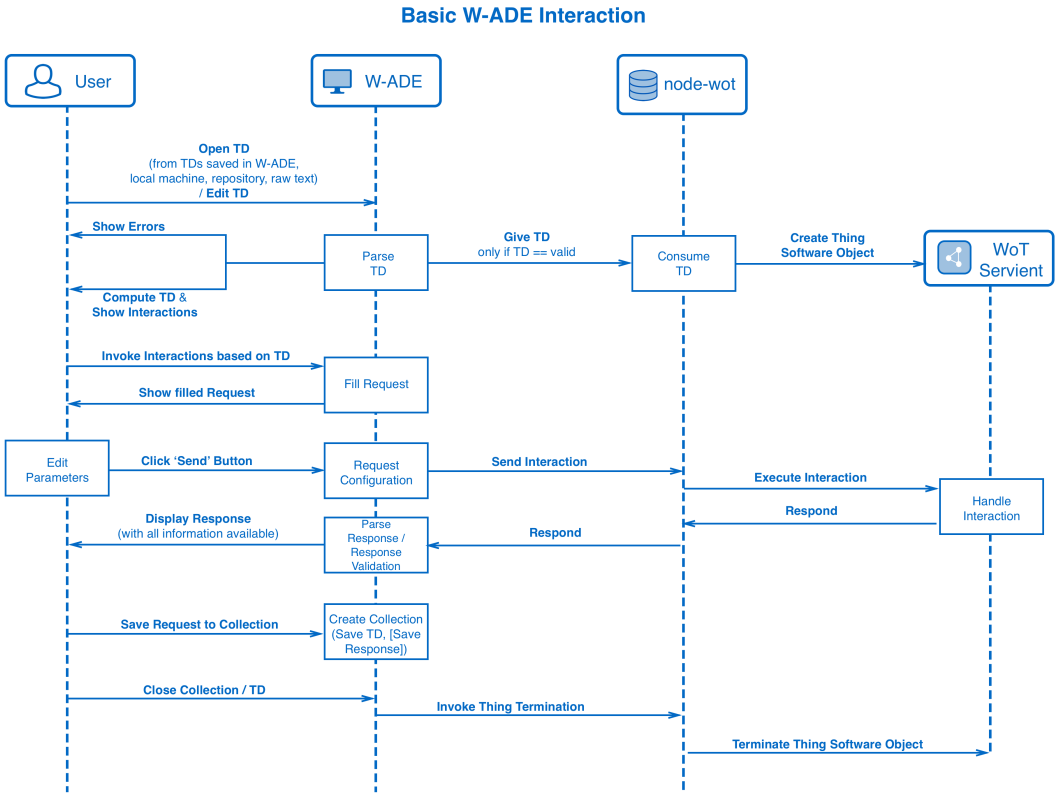

Figure 5: W-ADE's main workflow. A user pastes, fetches or uploads a TD. It is then consumed and parsed by W-ADE, presenting all interactions as GUI elements. If the TD contains any errors it cannot be validated and an error is shown to the user. Then, input parameters and security credentials can be entered and the selected interactions will be invoked. Responses are computed and results are shown to the user.

Figure 6: W-ADE's GUI of its start screen. The bar on the left contains all existing TDs and folders. TDs can be stored on their own or be aggregated into folders. By pressing the button in the middle, a new TD or folder can be created.

4

Automated Timing Performance Prediction Methodology

IoT systems are heterogeneous and their components are typically massively distributed. To validate communication and computation capabilities of an IoT device and to ensure to meet imposed requirements, its performance needs be evaluated [12]. W-ADE aims to facilitate automated timing performance assertion with merely a TD available. Premised that the specific device is WoT-enabled and consumable. The scope of performance is defined in Section 4.1, the methodology is elaborated in Section 4.2 and its implementation is presented in Section 4.3.

4.1 Timing Performance Prediction Possibilities in W-ADE

The time it takes to communicate with a Thing is dependent on factors like connection throughput, available bandwidth, network workload, loss rate, software characteristics, hardware architecture, latency of outgoing packets, packet size or used communication protocols [12, 13]. To predict timing performance, common approaches are measuring the latency of outgoing packets or the throughput rate, while tweaking network conditions, manipulating bandwidth, or switching protocols [3, 12, 14]. In test scenarios, it is often feasible to vary system configurations and retrieve data on the network environment or target devices. However, in real-world scenarios, mashup designers do not necessarily have access to manipulate third party devices or the opportunity to analyze internal application processes. Target Things could be connected via gateways and information on network conditions might be missing. As the objective is the facilitation of automated timing-predictions on any capable machine, in any network environment, without knowledge about the target application, and while using any IoT-protocol, network-, protocol-

and machine influences are not considered individually.
For this reason, W-ADE's timing prediction is based on measurements of the overall latency. Latency indicates the total delay between endpoints [13]. In the case of W-ADE this relates to the total time elapsed, from the moment the data is sent until the response is received, expressed by Eq. 4.1.

$$T_{\text{total}}(x) = T_{\text{client}}(x) + T_{\text{transfer}}(x) + T_{\text{process}}(x) \tag{4.1}$$

- $\mathbf{T_{total}}$: Total *dynamic* time needed for transferring a message from the Consumer to a Thing and receiving a response for an interaction x.
- $\mathbf{T_{client}}$: Client-side (W-ADE's) process time.
- $\mathbf{T_{transfer}}$: Time needed for sending/ receiving messages over the network.
- $\mathbf{T_{process}}$: Internal process time of a target Thing. It depends on the path of the internal execution and the time spent in the instructions on this path on this particular hardware [15]. It also covers physical interaction times.

Although the influences of T_{client} and $T_{transfer}$ cannot be isolated reliably, it is possible under certain conditions to differentiate between Action, hence time-consuming physical or virtual internal processes, and Property execution times. With the fulfilled prerequisites that an IoT device offers a Property and Action interaction, it solely sends responses after a physical or internal process is finished, and the application logic is comparable for all existing interactions, the following assumption are made:

$$T_{\text{static}}(x) = |T_{\text{total}}(x) - T_{\text{total}}(y)| \tag{4.2}$$

$\mathrm{T}_{\text{static}}(\mathrm{x})$ is the supposed *static* time of an Action x that does not change with network or client alternations. $\mathrm{T}_{\text{total}}(\mathrm{x})$ defines the *dynamic* time of the invoked Action x, $\mathrm{T}_{\text{total}}(\mathrm{y})$ the same for reading a Property y.

4.2 Methodology

To provide meaningful performance assertions, timing constraints on measurements and reliable average timing values need to be determined. For this purpose, bounds on execution times have to be identified. This can be achieved by executing an interaction for several repetitions or a specific amount of time, while simultaneously measuring the elapsed time. Results are combined to estimate execution time bounds. These are specified by deriving the overall maximum and minimum observed execution time. This is commonly called worst-case execution time (WCET) and best-case execution time (BCET) [15]. Generally, the BCET is overestimated and the WCET underestimated, as the actual values are almost impossible to derive. Since it cannot be guaranteed that

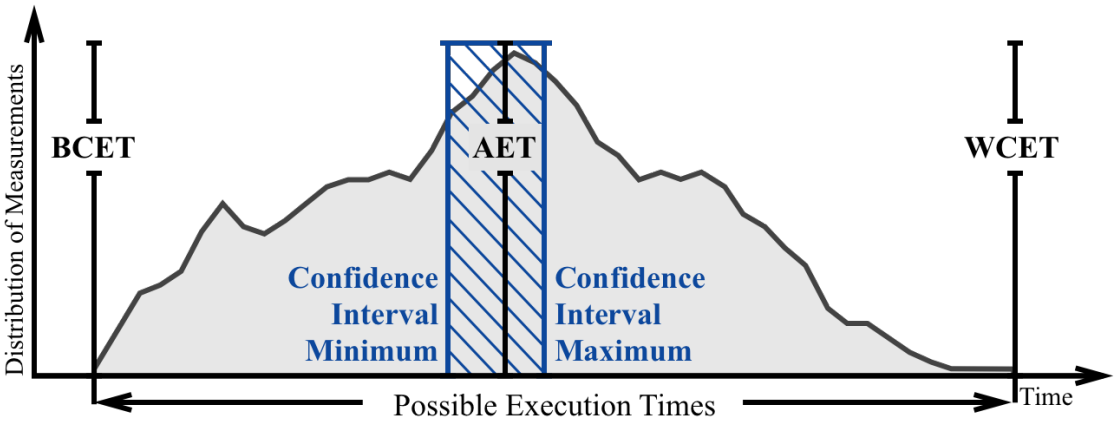

Figure 7: Basic concept of the timing prediction methodology of this work. The curve depicts sample values of interaction latency. Its minimum indicates the estimated best-case execution time, its maximum the estimated worst-case execution time. The middle line represents the Average Execution Time, surrounded by its confidence interval.

Things can be analyzed, they are treated as *black-box* components and this methodology utilizes estimated WCETs. Moreover, the average execution time (AET) for all measurements is computed.

In order to offer reliable predictions, confidence intervals (CI)[1] are used to propose a range of plausible values for the AET. The CI is calculated as seen in Eq. 4.3, where $\overline{\mathrm{x}}$ is the sample mean, σ the standard deviation, n the sample size and z^* represents the appropriate z^*-value from the standard normal distribution of the chosen Confidence Level (CL). The CL indicates the probability that the unknown parameter lies in the stated interval.

$$\overline{\mathrm{x}} \pm z^* \frac{\sigma}{\sqrt{n}} \tag{4.3}$$

To compute *static* timing, measurements of both a Property and Action are required. The *static* AET is computed from the difference of the *dynamic* AET values of the included Action and Property measurements (see Eq. 4.2). Its CI is calculated with Eq. 4.4: Elements have the same meaning as in Eq. 4.3, whereby subscript a represents Action and subscript p Property values.

$$\overline{\mathrm{x}}_\mathrm{a} - \overline{\mathrm{x}}_\mathrm{p} \pm z^* \sqrt{\frac{{\sigma^2}_\mathrm{a}}{n_\mathrm{a}} + \frac{{\sigma^2}_\mathrm{p}}{n_\mathrm{p}}} \tag{4.4}$$

Figure 7 shows how concept applied to measurements looks like. Certain factors influence execution times and need to be considered when predicting timing performance. One factor is the context dependence of execution times [15]. The execution time of individual instructions may vary by several orders of magnitude depending on the state of the processor. Thus, an execution time B can heavily depend on the state produced

1 A CI, in statistics, refers to the probability that an unknown value will fall between a specific range of values, calculated from observed data [16].

by execution A, e.g. for initial connection establishment, regarding memory or caching [15]. A task can also show variations depending on the payload or divergent environment behavior. To minimize this impact, the option of using measurements after a certain time has passed and an indefinite amount of interactions have been executed, is available and measurements can be repeated multiple times to reduce context errors. To remove the impact of initial connection establishment, the first measured values are removed. Further, timing anomalies, counter-intuitive influences on the local execution time of one instruction on the global execution time of the entire task [15], need to be considered and optionally be removed. The common approach of detecting outliers with the help of the interquartile range (IQR) that represents the width of the box in the box-and-whisker plot [17] is utilized.

$$IQR = Q_3 - Q_1 \tag{4.5}$$

$$min = Q_1 - 1.5 \times IQR \quad \text{and} \quad max = Q_3 + 1.5 \times IQR \tag{4.6}$$

The IQR, shown in Eq. 4.5, indicates how spread out middle values are, and defines outliers as values that are more than one and a half times the length of the middle-value box, away. Eq. 4.6 defines how the minimum and maximum threshold can be computed. Identified outliers can then be removed.

4.3 Implementation

To put this methodology into practice, a timing performance feature is implemented in W-ADE, depicted as plugin in Figure 1. The corresponding graphical user interface of this features is shown in Figure 8.

Users need to select interactions, enter required inputs, and choose performance analysis settings, including the type of measurement - either *Iteration* or *Duration* - with the desired amount. Iteration indicates that measurements are executed a certain number of times, whereas Duration executes them for the entered time-period. Then, a delay before the beginning of overall measurements or before the beginning of each execution can be scheduled. To finally start the execution and computation process, the desired CL to calculate the AET's CI has to be entered. Selectable options are 80%, 85%, 90%, 95%, 99%, 99.5% and 99.9%. To take into account potential environmental impacts, results are subdivided into *Possible* and *Realistic*. They respectively contain computed WCET, BCET, AET, CI limits for the AET, and the first measurement value. *Possible* results are computed considering all measurements except the first measured, whereas for *Realistic* results neither the first measured, nor detected outliers are included. Additionally, settings, general information, and all measured values in chronological order are displayed. To compute *static* timing, one Property, one Action, and the option `static timing` must be selected. As WCET and BCET are not applicable, they are not present in *static* results. To receive the entire timing behavior data, W-ADE enables users to

Figure 8: W-ADE's timing performance prediction interface. Selection of interactions and entering of input happens in (A). Measurement settings are chosen in (B), where also selected interactions are listed and execution can be started. Results, are displayed in (C).

save them as JSON file to their local machine. Figure 9 shows the workflow of this performance measuring feature.

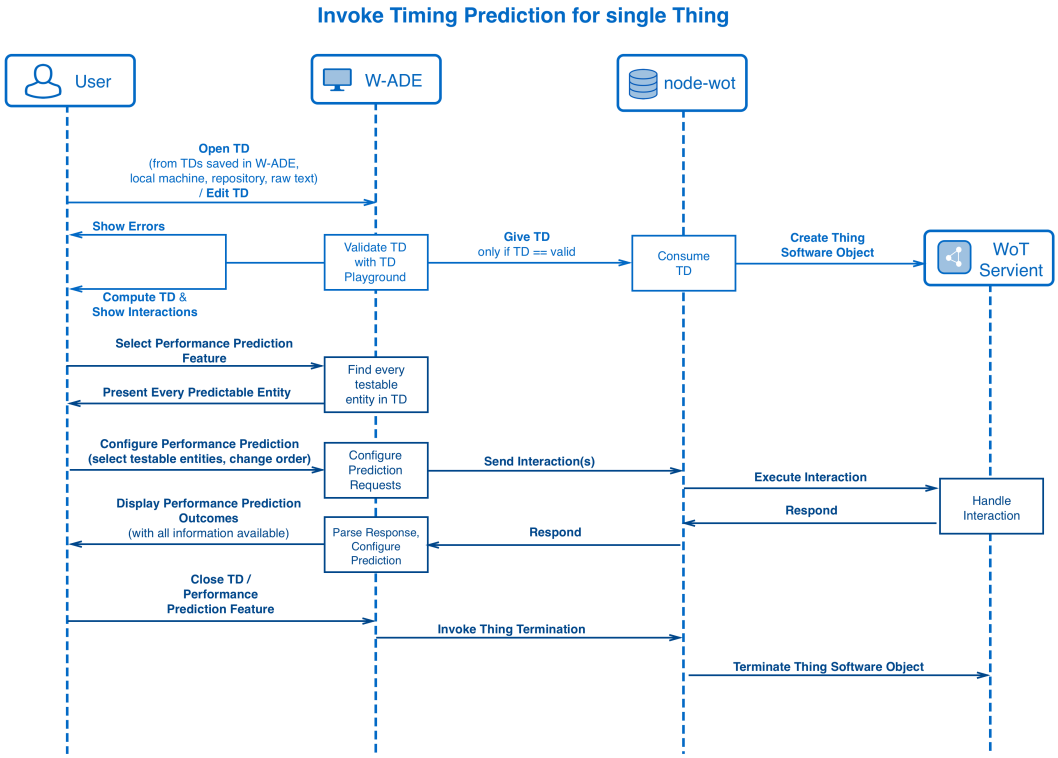

Figure 9: W-ADE's timing performance prediction interface. Selection of interactions and entering of input happens in (A). Measurement settings are chosen in (B), where also selected interactions are listed and execution can be started. Results, are displayed in (C).

5

Timing Performance Annotation

To be able to apply predicted timing behavior, results needs to be integrated into the TD, in reference to the appropriate interaction. Annotating interactions with generated information is a generic technique to support applied automatic prediction methods [15]. For this purpose, a suitable vocabulary set, referred to as *InteractionTiming*, that describes timing behavior of Things is proposed. It is compatible with present TD vocabularies, as it is based upon similar semantics and principles. Current existing TD vocabularies are independent and extensible. They each define a collection of terms that can be interpreted as objects denoting WoT Things and their Interaction Affordances [5].
As not all collected performance data discussed in Section 4 is significant for describing timing performance, only specific elements are included in the *InteractionTiming* vocabulary. Figure 10 gives an overview of the main *InteractionTiming* vocabulary terms.

The proposed term names are only suggestions and can be easily adjusted if they conflict with other TD terms. `staticTiming` characterizes *static* timing measurements and is added to the TD on the same level as `forms`. Other measurements are respectively summarized in `dynamicTiming` and are added to the particular `forms` element of the interaction. By default, both of them include `measurementContext` information, `confidence` data, and results categorized in `possible` and `realistic`. As `measurementContext` implies only contextual data, it is linked to by a JSON Pointer [18] and can be stored in another document. This default term set was elaborated with the aim of providing most information without including the entire set of results. An example of an annotated TD comprising *dynamic* and *static* performance data is depicted in List. 5.1. For the sake of readability, only *Possible* measurement data is added and confidence information of the *dynamic* annotation is absent. A complete annotated TD is provided externally[1].

1 Available at www.ei.tum.de/fileadmin/tueifei/esi/WADE-files/td_annotated.json

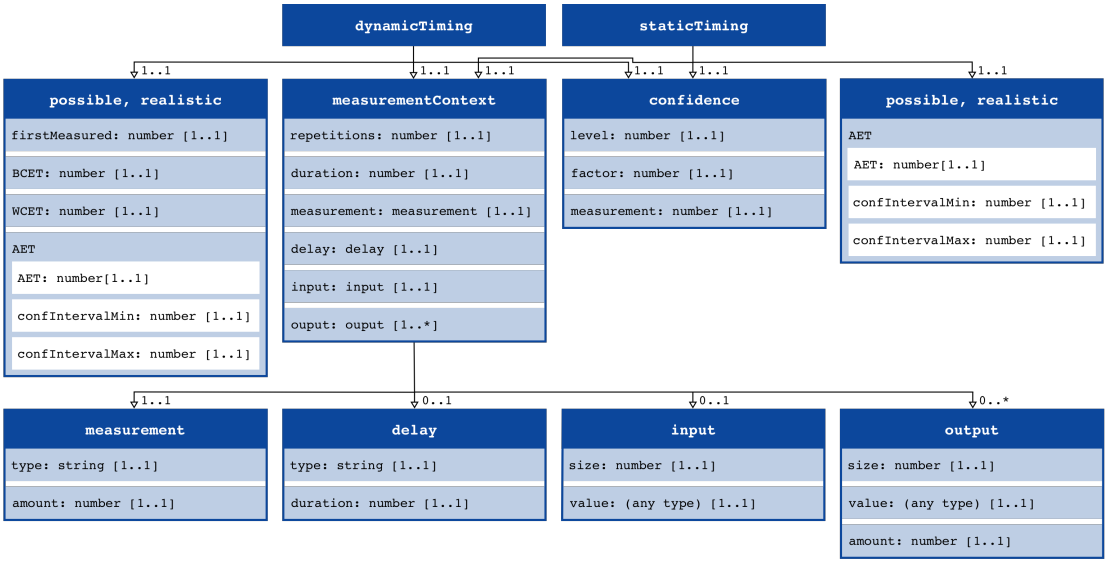

Figure 10: The proposed *InteractionTiming* vocabulary with term keys and values, including child elements and their data types. It is intended for annotating a Thing Description, in order to characterize timing behavior of its associated Thing.

The distribution of annotated TDs is not determined in this thesis, since not even the W3C WoT working group has fully explored TD distribution possibilities so far.
As a TD can always be presented as a JSON-LD file, its specification [5] also provides a JSON Schema [19] to syntactically validate TD instances. To extend this Schema, an *InteractionTiming* JSON Schema, including descriptions for all terms was created. It is shown in Appendix A.2. Additionally a TD JSON Schema extended with it, is available externally[2]. An annotated TD can be generated in W-ADE after results are computed. If desired, annotations can be further revised in the TD editor, before saving. An enhanced TD can be interpreted by a mashup designer and an automated system. It offers detailed semantics for describing how a Thing behaves regarding timing.

2 Available here: www.ei.tum.de/fileadmin/tueifei/esi/WADE-files/TD_Interaction_Timing_schema.json

```
{
  ...
  "brew": {
      "staticTiming": {
          "possible": {
              "AET": {
                  "AET": 40078.83,
                  "confIntervalMin": 40072.32,
                  "confIntervalMax": 40085.35
              }
          }
      },
      "forms": [
          {
              "href": "actions/brew",
              "dynamicTiming": {
                  "measurementContext":
    ↪ "#/measurementContext/dynamicTimingContext_action/brew",
                  "possible": {
                      "firstMeasured": 40221.06,
                      "BCET": 40118.66,
                      "WCET": 40962.02,
                      "AET": {
                          "AET": 40238.33,
                          "confIntervalMin": 40185.93,
                          "confIntervalMax": 40290.74
                      }
                  },
                  "confidence": {
                      "level": 99.9,
                      "factor": 3.291,
                      "numMeasurements": 100
                  }
              }
          }
      ]
  }
}
```

Listing 5.1: A snippet of an annotated Thing Description, including *dynamic* and *static* timing data for the Action `brew`. *Dynamic* annotations are added to the particular `form` element, whereby *static* annotations are located on the same level as `forms`.

6

Evaluation

To prove the correctness and validate the quality of the presented method, three experiments and one use-case scenario with a physical IoT device were conducted. The objective of the software-based simulations was on the one hand to confirm that computed results are able to anticipate the actual timing behavior of interactions, including the prediction of *static* timing behavior of an Action. On the other hand, W-ADE's timing prediction ability was verified by matching it to an ADE, capable of measuring latency. The focus of the physical use-case was to demonstrate the applicability of measuring timing performance of a physical IoT device with W-ADE. Test instructions for each experiment can be found in the Appendix A.3.

6.1 Experiment 1, Validity Test

To validate W-ADE's timing-prediction functionality and to show that processed results are consistent, a readable Property that returns a 14 byte string of an externally hosted TD and its Thing[1] was selected. Then, the according HTTP request was executed with a chosen CL of 99.9% for 1000 iterations, 10 times. As the TD's target is not connected to a physical Thing, different network conditions applied for the Consumer and Thing. It was then examined how results differed and to what percentage the CI fluctuates around the AET. In addition, it was verified whether AETs are in the range of other CIs. Figure 11 shows that computed AETs and their corresponding CI limits are consistent for *Possible*, as well as *Realistic* results. Only some negligible deviation, with a max. range of 10ms in *Possible* and 4ms in *Realistic* average AET values, were observed. Moreover, CI limits always lied within a range of $+/-5\%$ of AETs and AETs values lied in their associated CI, including all other measured CIs. Results in numbers is shown in Table 6.1.

1 This TD is provided by the W3C WoT working group for testing purposes.

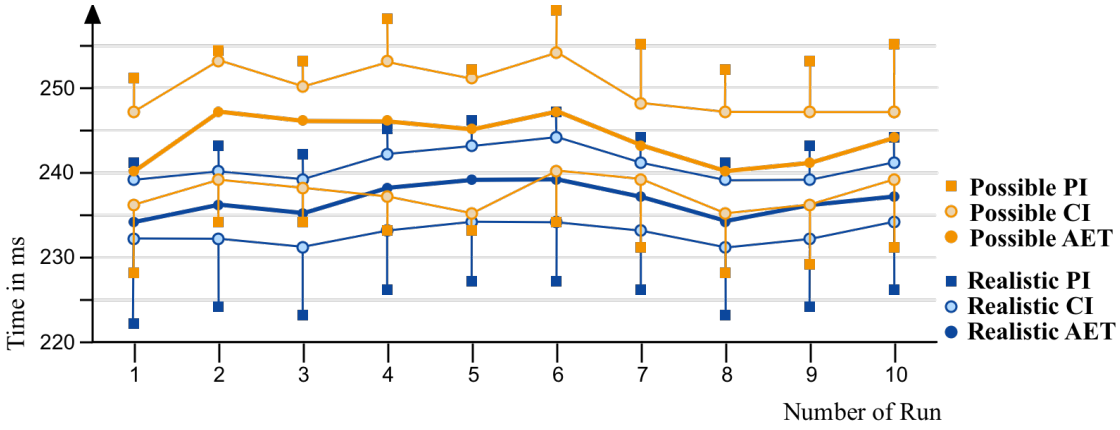

Figure 11: W-ADE's measured, rounded *Realistic* (blue) and *Possible* (orange) average execution times (AET). Confidence intervals (CI) always lied within a 5% range (PI) around the average values. AETs always lied in their own and all other CIs.

Table 6.1: Results of conducted timing measurements within W-ADE. *Run* indicates number of repetition, showing the average results of 1000 measurements each. *AET*, *CI*, and *PI* indicates the average execution time, the computed confidence interval, and a 5% range around the average execution time, respectively for *Realistic* and *Possible* values.

Run	R:AET	P:AET	R:CI	R:PI	P:CI	P:PI
1	234	240	232 - 239	222 - 246	236 - 247	228 - 252
2	236	247	232 - 240	224 - 248	239 - 254	234 - 259
3	235	246	231 - 239	223 - 247	238 - 250	234 - 258
4	238	246	233 - 242	226 - 250	237 - 258	233 - 258
5	239	245	234 - 243	227 - 251	240 - 251	233 - 257
6	239	247	234 - 244	227 - 252	240 - 253	234 - 259
7	237	243	233 - 241	226 - 249	239 - 248	231 - 255
8	234	240	231 - 239	223 - 246	235 - 247	228 - 252
9	236	241	232 - 239	224 - 248	236 - 247	229 - 253
10	237	243	234 - 241	226 - 249	239 - 247	231 - 255

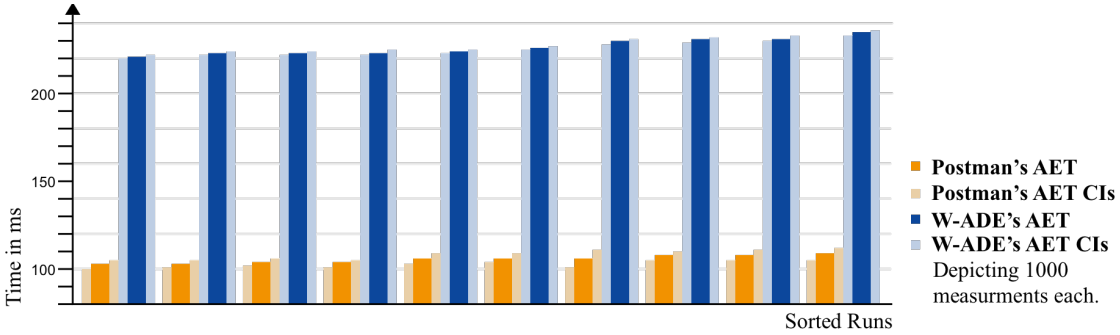

Figure 12: Measured average execution times of W-ADE (blue) are compared to Postman's (yellow). Bars denote the average values for 1000 measurements each. Enclosing thinner bars respectively indicate confidence interval limits. Both systems produced consistent values, whereas W-ADE's expected overhead to Postman was also consistent.

6.2 Experiment 2, Sanity Check

To evaluate W-ADE's credibility, measurements were matched with measurements of an ADE[2] called *Postman*[3], which acted as a control entity. Using the same test Thing from Exp. 1, an HTTP Property-write request with a 12 byte string-input, no output, and a CL of 99.9% was executed for 1000 times, repeating it 10 times with W-ADE and analogously Postman. To make result sets comparable, the methodology was applied onto Postman's results and processed the WCET, BCET, and AET with its CI in the same way as W-ADE. W-ADE's *Possible* results were used as outliers were not removed from Postman, and rounded them to integers.

Results, presented in Figure 12, show that Postman's and W-ADE's results are proportional to each other and consistent within themselves. WADE's AET values showed a max. deviation of $5,98\%$ (min. 221ms, max. 235ms). Postman's values showed a similar max. deviation of $6,42\%$ (min. 103ms, max. 109ms).

W-ADE added a significant average overhead (see $T_{client}(x)$ in Eq. 4.1) of about 121ms and $111,01\%$ in comparison to Postman, which is expected, as the implemented WoT Scripting API completes several internal processes (amongst others, the request has to be mapped to the specific protocol handler) first, before sending requests and is measuring on the Scripting API level, not on sockets. This is the anticipated way of writing a Consumer application.

2 ADE stands for API Development Environment and describes software that focuses on designing, building, and testing APIs.

3 Postman is an API development environment, capable of measuring latency (`www.getpostman.com/`). For this experiments, Postman Version v7.16.1 was used.

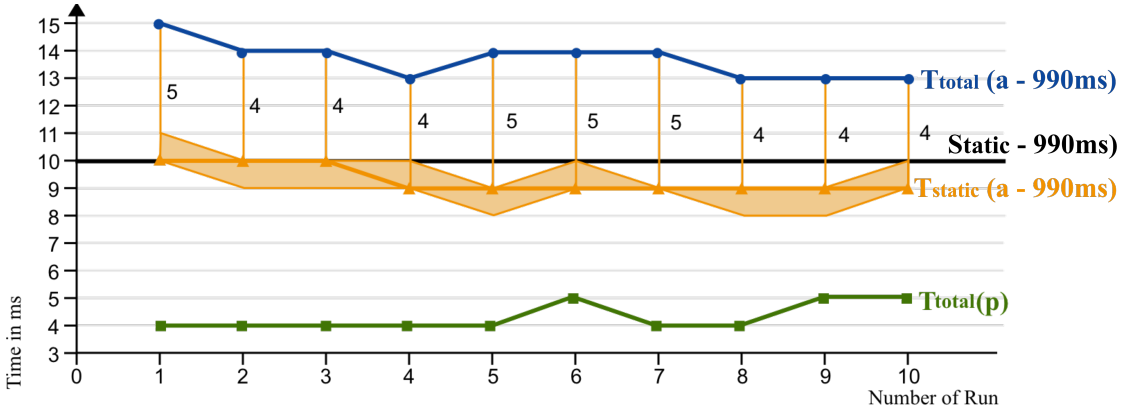

Figure 13: Computed *static* timing behavior of an Action. The upper curve indicates *dynamic* measurements (see Eq. 4.1) of Action a, the lower measurements of a Property p. The middle line shows the *static* timing with its confidence interval. For the sake of comprehension **990ms are subtracted from both Action values**.

6.3 Experiment 3, Static Timing

To confirm the validity of the *static* timing approach, introduced in Section 4.1, a script that simulates a Thing was created, providing a readable Property that returns a 14 byte string and an Action, that returns the same variable after a predetermined delay of exactly 1000ms, simulating a physical process of a device. It was then exposed on the same machine running W-ADE, communicating over the same local network. 100 Action and Property HTTP requests were sent to the before-mentioned interaction with a chosen CL of $99,9\%$ and measured with W-ADE, 10 times each. Due to network anomalies and other outliers adding a noticeable effect to timing, rounded *Realistic* values were used for the evaluation. As shown in Figure 13, the computed average *static* AET of 999,30ms matched the artificial delay of 1000ms to $99,93\%$, whereby *Realistic* AET measurements resulted either 999ms or 1000ms. The min. lower limit of the CI was 998ms and the max. 1001ms. Giving a $-0,2\%/+0,1\%$ range around the actual delay. This confirms that that computed results are able to anticipate the actual timing behavior of interactions.

6.4 Use-Case with a Physical Device

To demonstrate the practicability of timing performance prediction with a real Thing, *static* measurements were conducted analog to Exp. 3. Included components were W-ADE as client and a WoT-enabled[4] Pan-Tilt HAT module (PTH) attached to a Raspberry Pi. The PTH is a set of horizontal and vertical motion servos, that can be moved individually, as seen in Figure 14. The Raspberry Pi is a small single-board computer, here a model `3B+` running `Raspbian 2019-09-26`. Both Thing the PTH

4 A WoT-enabled Pan-Tilt HAT: `www.wotify.org/library/Pan-Tilt%20HAT/general`.

Figure 14: The device used in our physical scenario: a Pan-Tilt HAT module attached to a Raspberry Pi, that is able to move its robot arm to diverse positions.

attached to the Raspberry Pi) and Consumer were connected to the same network. The evaluated Action `scan` moves the robot arm from the outer-left to the outer-right position and the used Property `panPosition` returns the position of the pan module. Both HTTP interactions were executed 10 times with 1000 iterations each and a 99,9% CL, before the *static* timing of `scan` was calculated. Analog to Exp. 3, the evaluation was based on *Realistic* results. An average *static* AET of 40055ms with a CI of 40053ms - 40057ms was measured for `scan`. Whereby the average *dynamic* AET of all measured Actions was 40063ms, with a CI of 40049ms - 40077ms. Property measurements revealed an average AET of 8ms and CI limits of 7,6ms - 8,8ms.
This proves that W-ADE enables the conduction of *static* timing performance on an actual physical IoT device. To once again validate that the *static* AET matches the actual time of a `scan` movement, manual, possibly error-sensitive chronometer based measurements, would be necessary.

6.5 Limitations and Outlook

This evaluation proves that the methodology presented in this work is a validate approach of predicting timing-performance of IoT devices. To further enrich this application, more possibilities of performance-testing need to be included.
One thing that was noticed while evaluating the presented methodology, is the dependence to other interactions and their impact on measured timing behavior. For example, if a Property interaction is executed on its own measurements are different compared to when this same interaction is executed together with others. It was observed that the measurement time can massively change due to a change of context interactions. This is especially relevant for the creation and design of mashups. As mashups often also include

multiple interactions of one Thing, it is important to gain knowledge about different behavior together with different interactions. As requests are executed synchronously in W-ADE, testing context dependence of interaction is not feasible at the moment. To improve the prediction for interactions regarding their context, an additional asynchronous timing measurement feature needs to be implemented.
It was also noticed, that measurements can differ in a broad range for runs with a time difference. For example if an execution was repeated after a short period of time (e.g. 20 minutes), measurements differed and values were not included in the confidence interval of other measurements. One reason for this might be changes in the network environment. The network could be more crowded in another time period. Another reason might be the current performance of the system where W-ADE is executed. Possible other CPU-intensive computations might have a negative influence towards the executions made in W-ADE As these influences cannot be reliably removed in W-ADE, a way to measure network and other environmental factors needs to be found in future work. The difficulty here is W-ADE's goal of working on any machine.
Another valuable feature would be the possibility to test Things while asynchronously connecting to it via multiple client instances, as this is a common use-case. This would also enable stress-testing of the device. In real-world use-cases a Thing might be accessed from multiple Clients (depending on its usage). Timing performance might significantly decrease the more Consumers are trying to communicate with it. Giving reliable performance data for this scenario would be valuable.

7

Related Work

Numerous approaches on how the Web of Things can look, have been introduced to the world of IoT [20, 21, 22]. As the TD standard and associated WoT approaches not only present a well conceived concept, but also actively offer solutions to counteract the fragmentation of IoT, this work is based upon them. Many W3C WoT tools and services have already been contributed, nevertheless, there is little scientific work on them available. This might be due to the TD standard being rather new and hardly distributed. Therefore, no service or approach to enhance a TD with timing information has been released.

While performance evaluation in IoT is the topic of many studies, they often focus on comparison of diverse protocol performance under specific circumstances, whereby the experiment environment is mostly controlled [23, 14, 3]. Other studies target network-performance regarding an IoT device [22, 24], the general performance spectrum, or stress testing of Things [12]. In general, studies in the field of IoT performance do not deal with *dynamic* or *static* timing performance of Things and do not offer solutions on how to predict such behavior. Choosing the best fitting communication protocol and determining network performance makes sense when setting up IoT platforms, but not when third-party Things need to be integrated into mashups or their general timing behavior needs to be predicted. In contrary to existing studies, the proposed method in this thesis enables developers to actively use timing performance data for the design of real-world IoT systems and use cases.

8

Conclusion

Motivated by the problem of missing opportunities to easily measure timing behavior of IoT devices based on their TD, a methodology which facilitates prediction of timing performance, while considering environmental influences was introduced. Thereupon, W-ADE, an API development environment and platform for the WoT ecosystem that implements the proposed method and additionally enables manual timing measurement of device interactions, was developed. To validate and demonstrate the applicability in practice, this method was tested with a physical IoT device and three virtual simulations were conducted. It was proven that W-ADE reliably predicts *static* timing behavior of interactions and offers accurate timing performance insights. Combined with the proposed *InteractionTiming* vocabulary to annotate TDs, mashup designers are now able to estimate interaction timing behavior; thus, optimize system compositions during design time.

A

Appendix

A.1 TD Annotated Example

```
1
2{
3    "@context": "https://www.w3.org/2019/wot/td/v1",
4    "id": "urn:dev:ops:32473-CoffeeMachine-1234",
5    "title": "Coffee-Machine",
6    "securityDefinitions": {
7        "basic_sc": {
8            "scheme": "basic",
9            "in": "header"
10       }
11   },
12   "security": [
13       "basic_sc"
14   ],
15   "base": "coaps://coffee-machine.example.com:5683",
16   "properties": {
17       "status": {
18           "type": "string",
19           "forms": [
20               {
21                   "href": "properties/state",
22                   "dynamicTiming": {
23                       "measurementContext":
     ↪ "#/measurementContext/dynamicTimingContext_properties/state",
24                       "possible": {
25                           "firstMeasured": 165.94,
26                           "BCET": 25.92,
27                           "WCET": 1193.19,
28                           "AET": {
29                               "AET": 159.50,
30                               "confIntervalMin": 105.75,
31                               "confIntervalMax": 213.25
32                           }
33                       },
34                       "realistic": {
```

```
                            "firstMeasured": 165.94,
                            "BCET": 25.92,
                            "WCET": 315.75,
                            "AET": {
                                "AET": 126.30,
                                "confIntervalMin": 105.34,
                                "confIntervalMax": 147.27
                            }
                        },
                        "confidence": {
                            "level": 99.9,
                            "factor": 3.291,
                            "numMeasurements": 100
                        }
                    }
                }
            ]
        },
        "fillLevel": {
            "type": "integer",
            "forms": [
                {
                    "href": "properties/fillLevel",
                    "dynamicTiming": {
                        "measurementContext":
    ↪ "#/measurementContext/dynamicTimingContext_properties/fillLevel",
                        "possible": {
                            "firstMeasured": 245.56,
                            "BCET": 212.61,
                            "WCET": 223.76,
                            "AET": {
                                "AET": 216.31,
                                "confIntervalMin": 214.95,
                                "confIntervalMax": 217.66
                            }
                        },
                        "realistic": {
                            "firstMeasured": 245.56,
                            "BCET": 212.61,
                            "WCET": 223.76,
                            "AET": {
                                "AET": 216.31,
                                "confIntervalMin": 214.95,
                                "confIntervalMax": 217.66
                            }
                        },
                        "confidence": {
                            "level": 99.9,
                            "factor": 3.291,
                            "numMeasurements": 47
                        }
                    }
                }
            ]
        }
    },
    "actions": {
```

```
        "brew": {
            "input": {
                "type": "string",
                "enum": [
                    "espresso",
                    "cappuccino",
                    "americano"
                ]
            },
            "forms": [
                {
                    "href": "actions/brew",
                    "dynamicTiming": {
                        "measurementContext":
    ↪ "#/measurementContext/dynamicTimingContext_action/brew",
                        "possible": {
                            "firstMeasured": 40221.06,
                            "BCET": 40118.66,
                            "WCET": 40962.03,
                            "AET": {
                                "AET": 40238.33,
                                "confIntervalMin": 40185.93,
                                "confIntervalMax": 40290.74
                            }
                        },
                        "realistic": {
                            "firstMeasured": 40221.06,
                            "BCET": 40225.24,
                            "WCET": 40249.36,
                            "AET": {
                                "AET": 40243.91,
                                "confIntervalMin": 40241.71,
                                "confIntervalMax": 40246.11
                            }
                        }
                    },
                    "confidence": {
                        "level": 99.9,
                        "factor": 3.291,
                        "numMeasurements": 100
                    }
                }
            ],
            "staticTiming": {
                "measurementContext": [
                    "#/measurementContext/dynamicTimingContext_action/brew",
                    "#/measurementContext/dynamicTimingContext_properties/state"
                ],
                "possible": {
                    "AET": {
                        "AET": 40078.83,
                        "confIntervalMin": 40072.32,
                        "confIntervalMax": 40085.35
                    }
                },
                "realistic": {
                    "AET": {
```

```
                        "AET": 40117.61,
                        "confIntervalMin": 40114.71,
                        "confIntervalMax": 40120.50
                    }
                },
                "confidence": {
                    "level": 99.9,
                    "factor": 3.291,
                    "numMeasurements": 100
                }
            }
        }
    },
    "events": {
        "error": {
            "data": {
                "type": "string"
            },
            "forms": [
                {
                    "href": "events/error"
                }
            ]
        }
    },
    "measurementContext": [
        {
            "dynamicTimingContext_action/brew": {
                "repetitions": 100,
                "duration": 4023742,
                "measurement": {
                    "type": "iteration",
                    "amount": 100
                },
                "input": {
                    "size": 16,
                    "value": "espresso"
                }
            }
        },
        {
            "dynamicTimingContext_properties/state": {
                "repetitions": 100,
                "duration": 15969,
                "measurement": {
                    "type": "iteration",
                    "amount": 100
                },
                "delay": {
                    "type": "none",
                    "duration": null
                },
                "output": [
                    {
                        "size": 10,
                        "value": "error",
                        "amount": 100
```

```
                    }
                ]
            }
        },
        {
            "dynamicTimingContext_properties/fillLevel": {
                "repetitions": 47,
                "duration": 11000,
                "measurement": {
                    "type": "duration",
                    "amount": 1000
                },
                "delay": {
                    "type": "beforeBeginning",
                    "duration": 1000
                },
                "output": [
                    {
                        "size": 8,
                        "value": 1,
                        "amount": 47
                    }
                ]
            }
        }
    ]
}
```

Listing A.1: An example Thing Description, annotated with measured *Static* and *Dynamic* timing-performance predictions.

A.2 InteractionTiming Vocabulary

```
{
    "title": "InteractionTiming",
    "description": "JSON schema InteractionTiming extension for Thing
    ↪ Description.",
    "$schema ": "http://json-schema.org/draft-07/schema#",
    "definitions": {
        "dynamicTiming": {
            "type": "object",
            "description": "Describes dynamic timing behaviour of one
    ↪ interaction of an IoT device.",
            "properties": {
                "measurementContext": {
                    "oneOf": [
                        {
                            "type": "array",
                            "description": "Includes information on applied
    ↪ settings and general context data for arbitrary interactions.",
                            "items": {
                                "$ref": "#/definitions/measurementContext"
                            }
                        },
                        {
```

```
                        "type": "string",
                        "description": "An URI reference, pointing to a
↪ JSON document that contains the measurement context.",
                        "format": "uri-reference"
                    }
                ]
            }
        },
        "possible": {
            "$ref": "#/definitions/dynamicMeasurementOptions"
        },
        "realistic": {
            "$ref": "#/definitions/dynamicMeasurementOptions"
        },
        "confidence": {
            "$ref": "#/definitions/confidence"
        }
    },
    "staticTiming": {
        "type": "object",
        "description": "Describes static timing behavior of one Action of
↪ an IoT device, calculated from dynamic timing behavior of this Action
↪ and one Property.",
        "properties": {
            "measurementContext": {
                "oneOf": [
                    {
                        "type": "array",
                        "description": "Includes information on applied
↪ settings and general context data for arbitrary interactions.",
                        "items": {
                            "$ref": "#/definitions/measurementContext"
                        }
                    },
                    {
                        "type": "string",
                        "description": "An URI reference, pointing to a
↪ JSON document that contains the measurement context.",
                        "format": "uri-reference"
                    }
                ]
            },
            "possible": {
                "$ref": "#/definitions/staticMeasurementOptions"
            },
            "realistic": {
                "$ref": "#/definitions/staticMeasurementOptions"
            },
            "confidence": {
                "$ref": "#/definitions/confidence"
            }
        }
    },
    "measurementContext": {
        "type": "object",
        "description": "Measurement context data of one interaction.",
        "properties": {
```

```
72                "repetitions": {
73                    "type": "integer",
74                    "description": "Overall repetitions of measurements."
75                },
76                "duration": {
77                    "type": "number",
78                    "description": "Measured overall duration in milliseconds."
79                },
80                "measurement": {
81                    "$ref": "#/definitions/measurementContextItemMeasurement"
82                },
83                "delay": {
84                    "$ref": "#/definitions/measurementContextItemDelay"
85                },
86                "input": {
87                    "$ref": "#/definitions/measurementContextItemInput"
88                },
89                "output": {
90                    "$ref": "#/definitions/measurementContextItemOutput"
91                }
92            }
93        },
94        "measurementContextItemMeasurement": {
95            "type": "object",
96            "description": "Contains type of measurement and the corresponding
    ↪ amount.",
97            "properties": {
98                "type": {
99                    "description": "Type of applied measurement.",
100                   "type": "string",
101                   "enum": [
102                       "iteration",
103                       "duration"
104                   ]
105               },
106               "amount": {
107                   "description": "Amount of applied measurements, either
    ↪ number of iterations or duration of measurement in milliseconds.",
108                   "oneOf": [
109                       {
110                           "type": "number",
111                           "minimum": 0
112                       },
113                       {
114                           "type": "integer",
115                           "minimum": 0
116                       }
117                   ]
118               }
119           }
120       },
121       "measurementContextItemDelay": {
122           "type": "object",
123           "description": "Includes type and duration of delay.",
124           "properties": {
125               "type": {
126                   "type": "string",
```

```
127                     "description": "Type of applied delay.",
128                     "enum": [
129                         "beforeEach",
130                         "beforeBeginning"
131                     ]
132                 },
133                 "duration": {
134                     "description": "Duration of applied delay in milliseconds.",
135                     "oneOf": [
136                         {
137                             "type": "number",
138                             "minimum": 0
139                         },
140                         {
141                             "type": "null"
142                         }
143                     ]
144                 }
145             }
146         },
147         "measurementContextItemInput": {
148             "type": "object",
149             "description": "Includes request payload size and content.",
150             "properties": {
151                 "size": {
152                     "$ref": "#/definitions/size"
153                 },
154                 "value": {
155                     "$ref": "#/definitions/value"
156                 }
157             }
158         },
159         "measurementContextItemOutput": {
160             "type": "array",
161             "description": "List of different response payloads, each including
    ↪ payload size, content and amount of occurrences.",
162             "items": {
163                 "type": "object",
164                 "properties": {
165                     "size": {
166                         "$ref": "#/definitions/size"
167                     },
168                     "value": {
169                         "$ref": "#/definitions/value"
170                     },
171                     "amount": {
172                         "type": "integer",
173                         "description": "Amount of occurrences of this specific
    ↪ payload."
174                     }
175                 }
176             }
177         },
178         "confidence": {
179             "type": "object",
180             "description": "Provides information about applied confidence
    ↪ level, confidence factor and number of utilized measurements for the
```

```
      ↪ computation of the confidence interval.",
181           "properties": {
182               "level": {
183                   "type": "number",
184                   "description": "Chosen confidence level of user.",
185                   "enum": [
186                       80,
187                       85,
188                       90,
189                       95,
190                       99,
191                       99.5,
192                       99.9
193                   ]
194               },
195               "factor": {
196                   "type": "number",
197                   "description": "Confidence factor used for calculation of
      ↪ confidence interval.",
198                   "enum": [
199                       1.282,
200                       1.440,
201                       1.645,
202                       1.960,
203                       2.576,
204                       2.807,
205                       3.291
206                   ]
207               },
208               "numMeasurements": {
209                   "type": "integer",
210                   "description": "Number of measurements utilized to measure
      ↪ confidence interval.",
211                   "exclusiveMinimum": 0
212               }
213           }
214       },
215       "dynamicMeasurementOptions": {
216           "type": "object",
217           "description": "Elements that were calculated to describe dynamic
      ↪ timing behavior of an interaction.",
218           "properties": {
219               "firstMeasured": {
220                   "$ref": "#/definitions/firstMeasured"
221               },
222               "BCET": {
223                   "$ref": "#/definitions/BCET"
224               },
225               "WCET": {
226                   "$ref": "#/definitions/WCET"
227               },
228               "AET": {
229                   "$ref": "#/definitions/AET"
230               }
231           }
232       },
233       "staticMeasurementOptions": {
```

```
234             "type": "object",
235             "description": "Elements that were calculated to describe static
    ↪ timing behavior of an action.",
236             "properties": {
237                 "AET": {
238                     "$ref": "#/definitions/AET"
239                 }
240             }
241         },
242         "firstMeasured": {
243             "type": "number",
244             "description": "First measured execution time in milliseconds.",
245             "minimum": 0
246         },
247         "BCET": {
248             "type": "number",
249             "description": "Estimated Best-Case Execution time of all
    ↪ measurements.",
250             "minimum": 0
251         },
252         "WCET": {
253             "type": "number",
254             "description": "Estimated Worst-Case Execution time of all
    ↪ measurements.",
255             "minimum": 0
256         },
257         "AET": {
258             "type": "object",
259             "description": "Includes AET and confidence interval minimum/
    ↪ maximum for AET.",
260             "properties": {
261                 "AET": {
262                     "type": "number",
263                     "description": "Calculated Average Execution Time
    ↪ considering all measurements.",
264                     "minimum": 0
265                 },
266                 "confIntervalMin": {
267                     "type": "number",
268                     "description": "Confidence interval minimum for this AET.",
269                     "minimum": 0
270                 },
271                 "confIntervalMax": {
272                     "type": "number",
273                     "description": "Confidence interval maximum for this AET.",
274                     "minimum": 0
275                 }
276             }
277         },
278         "size": {
279             "description": "Payload size in byte or null in case payload does
    ↪ not exist.",
280             "oneOf": [
281                 {
282                     "type": "null"
283                 },
284                 {
```

```
285                     "type": "number"
286                 }
287             ]
288         },
289         "value": {
290             "description": "Payload content. Null in case payload does not
    ↪ exist, else data type of payload.",
291             "oneOf": [
292                 {
293                     "type": "number"
294                 },
295                 {
296                     "type": "boolean"
297                 },
298                 {
299                     "type": "string"
300                 },
301                 {
302                     "type": "null"
303                 },
304                 {
305                     "type": "object"
306                 },
307                 {
308                     "type": "array"
309                 }
310             ]
311         }
312     }
313 }
```

Listing A.2: The proposed InteractionTiming vocabulary JSON schema with descriptions for each element.

A.3 Test Instructions

Test-Number	1 - W-ADE	
Goal	Proof the validity of timing measurement within W-ADE.	
Description	The measured 99,9% confidence interval of the average execution time of a request should not differ more than +/- 5 percent of the AET within the local network of the system on the same system.	
Test environment	CI: 99,9%. No delay. HTTP-Property-read request of String. Output = 14 byte String "servus"	
Test entities	**Client**: W-ADE **Thing**: Virtual Test Thing: http://plugfest.thingweb.io:8083/TestThing	
Runs	**10** different runs with 1000 measurements each	
Test Settings	**Test Type**	Iterations
	Iterations	1000
	Delay	No Delay
	Num Measurements	10

Figure 15: Test instructions for the first experiment, see Sec. 6.1.

Test-Number	2 - W-ADE vs. Postman	
Goal	Calculate how much overload W-ADE actually adds to request and check whether results are proportional to Postman results.	
Description	Compare outcomes of W-ADEs performance measurements and Postman's performance test with Postman Collection Runner with local virtual Thing.	
Test environment	CI: 99,9%. No delay. HTTP-Property-write request with String. Input: 14 byte String "servus".	
Test entities	**Client #1**: W-ADE. **Client #2**: Postman v7.16.1 (Postman Collection Runner). **Thing**: Virtual Test Thing: http://plugfest.thingweb.io:8083/TestThing	
Runs	10 runs with 1000 measurements respectively for W-ADE and Postman.	
Processing	Retrieve timing values of Postman results and compute them as the same way as wade. (But no differentiation between Static / Realistic).	
Test Settings	**Test Type**	Iterations
	Input	"Servus"
	Iterations	1000
	Delay	No Delay
	Num Measurements	10

Figure 16: Test instructions for the second experiment, see Sec. 6.2.

Test-Number	3 - Static Timing	
Goal	Proof that measurements result are able to anticipate the actual/ static timing behaviour of requests.	
Description	We will measure action request with an artificial delay of 1000ms before sending a response, then measure property requests, then compute Static Timing behavior with them.	
Test environment	CI: 99,9%. No Delay HTTP Property-read request returning a 12 byte String “success”. HTTP Action-read-request returning the same String after 1000ms. No output.	
Test entities	**Client**: W-ADE **Thing**: Virtual Thing exposed by Node-WoT **node-wot-thing1**	
Runs	Run Property and Action 1000 each for 10 times.	
Processing	After retrieving all measurements, Static-Timing can be computed respectively for one Property and one Action measurement.	
Run	1 - HTTP Property-Read	
Test Settings	**Test Type**	Iterations
	Iterations	1000
	Delay	No Delay
	Num Measurements	10
Run	2 - HTTP Action	
Test Settings	**Test Type**	Iterations
	Iterations	1000
	Delay	Inside Thing
	Delay Duration	1000 ms
	Num Measurements	10

Figure 17: Test instructions for the third experiment, see Sec. 6.3.

Test-Number	4 - Virtual Use-Case	
Goal	Proof that timing-measuring is applicable for physical IoT devices.	
Description	Same approach as test 3, only that now the Test Thing is replaced with a real IoT Device	
Test environment	CI: 99,9%. No Delay HTTP Property-read request returning a number. HTTP Action--request performing a movement of the Test Thing.	
Test entities	**Client**: W-ADE **Thing**: WoT-enabled PanTilt HAT module for Raspberry Pi.	
Runs	Run Property and Action 1000 each for 10 times.	
Processing	After retrieving all measurements, Static-Timing can be computed respectively for one Property and one Action measurement.	
Run	1 - HTTP Property-Read	
Test Settings	**Test Type**	Iterations
	Iterations	1000
	Delay	No Delay
	Num Measurements	10
Run	2 - HTTP Action	
Test Settings	**Test Type**	Iterations
	Iterations	1000
	Delay	No artificial Delay
	Delay Duration	around ~4000ms
	Num Measurements	10

Figure 18: Test instructions for the use-case scenario with an IoT device, see Sec. 6.4.

Bibliography

[1] D. Guinard and V. Trifa, *Building the Web of Things: With Examples in Node.js and Raspberry Pi*. Manning Publications Co., 2016. cited on p. 1

[2] D. Guinard, V. Trifa, T. Pham, and O. Liechti, "Towards Physical Mashups in the Web of Things," in *Proc. of INSS*, vol. 9, pp. 17–19, 2009. cited on p. 1

[3] Z. B. Babovic, J. Protic, and V. Milutinovic, "Web Performance Evaluation for Internet of Things Applications," *IEEE Access*, vol. 4, pp. 6974–6992, 2016. cited on p. 1, 17, 33

[4] D. Guinard and V. Trifa, "Towards the Web of Things: Web Mashups for Embedded Devices," in *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in Proc. of WWW*, vol. 15, 2009. cited on p. 1

[5] T. Kamiya, S. Käbisch, M. Kovatsch, M. McCool, and V. Charpenay, "WoT Thing Description," tech. rep., W3C, 2019. www.w3.org/TR/2019/CR-wot-thing-description-20191106/. cited on p. 1, 23, 24

[6] R. Matsukura, M. Lagally, M. Kovatsch, K. Toumura, and T. Kawaguchi, "WoT Architecture," tech. rep., W3C, 2019. www.w3.org/TR/2019/CR-wot-architecture-20191106/. cited on p. 2, 5

[7] D. Peintner, K. Nimura, Z. Kis, J. Hund, and K. Nimura, "WoT Scripting API," tech. rep., W3C, 2019. www.w3.org/TR/2019/WD-wot-scripting-api-20191028/. cited on p. 5

[8] M. Koster, "WoT Protocol Binding Templates," tech. rep., W3C, 2018. www.w3.org/TR/2018/NOTE-wot-binding-templates-20180405/. cited on p. 5, 10

[9] P. Champin, G. Kellogg, and D. Longley, "JSON-LD 1.1," tech. rep., W3C, 2019. https://www.w3.org/TR/2019/CR-json-ld11-20191212/. cited on p. 5

[10] E. Korkan, S. Kaebisch, M. Kovatsch, and S. Steinhorst, "Safe Interoperability for Web of Things Devices and Systems," in *Languages, Design Methods, and Tools for Electronic System Design*, pp. 47–69, Springer, 2020. cited on p. 5, 6

[11] X. Liu, Y. Hui, W. Sun, and H. Liang, "Towards Service Composition Based on Mashup," in *2007 IEEE Congress on Services*, pp. 332–339, IEEE, 2007. cited on p. 6

[12] J. Esquiagola, L. C. de Paula Costa, P. Calcina, G. Fedrecheski, and M. Zuffo, "Performance Testing of an Internet of Things Platform," in *IoTBDS*, pp. 309–314, 2017. cited on p. 17, 33

[13] G. Huston, "Measuring IP Network Performance," *The Internet Protocol Journal*, vol. 6, no. 1, pp. 2–19, 2003. cited on p. 17, 18

[14] Y. Chen and T. Kunz, "Performance Evaluation of IoT Protocols under a Constrained Wireless Access Network," in *2016 Int. Conf. on Selected Topics in MoWNeT)*, pp. 1–7, IEEE, 2016. cited on p. 17, 33

[15] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, *et al.*, "The Worst-Case Execution Time Problem – Overview of Methods and Survey of Tools," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 3, p. 36, 2008. cited on p. 18, 19, 20, 23

[16] J. Neyman, "X–outline of a Theory of Statistical Estimation Based on the Classical Theory of Probability," *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, vol. 236, no. 767, pp. 333–380, 1937. cited on p. 19

[17] J. W. Tukey, *Exploratory Data Analysis*, vol. 2. Reading, Mass., 1977. cited on p. 20

[18] P. Bryan, K. Zyp, and M. Nottingham, "JavaScript Object Notation (JSON) Pointer," *RFC 6901 (Proposed Standard)*, 2013. cited on p. 23

[19] A. Wright, H. Andrews, and G. Luff, "JSON Schema Validation: A Vocabulary for Structural Validation of JSON," *IETF Standard*, 2016. cited on p. 24

[20] D. Guinard, V. Trifa, M. Friedemann, and E. Wilde, "From the Internet of Things to the Web of Things: Resource-oriented Architecture and Best Practices," in *Architecting the Internet of Things*, pp. 97–129, Springer, 2011. cited on p. 33

[21] T. Käfer, S. R. Bader, L. Heling, R. Manke, and A. Harth, "Exposing Internet of Things Devices via REST and Linked Data Interfaces," in *Proc. 2nd Workshop Semantic Web Technol. Internet Things*, pp. 1–14, 2017. cited on p. 33

[22] S. Duquennoy, G. Grimaud, and J. Vandewalle, "The Web of Things: Interconnecting Devices with High Usability and Performance," in *2009 Int. Conf. on Embedded Software and Systems*, pp. 323–330, IEEE, 2009. cited on p. 33

[23] T. Yokotani and Y. Sasaki, "Comparison with HTTP and MQTT on Required Network Resources for IoT," in *2016 Int. Conf. on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, pp. 1–6, IEEE, 2016. cited on p. 33

[24] R. Morabito, I. Farris, A. Iera, and T. Taleb, "Evaluating Performance of Containerized IoT Services for Clustered Devices at the Network Edge," *IEEE Internet of Things Journal*, vol. 4, no. 4, pp. 1019–1030, 2017. cited on p. 33