International Journal on Advances in Internet Technology

2019 vol. 12 nr. 1&2

The *International Journal on Advances in Internet Technology* is published by IARIA.
ISSN: 1942-2652
journals site: http://www.iariajournals.org
contact: petre@iaria.org

International Journal on Advances in Internet Technology
Volume 12, Number 1 & 2, 2019

International Journal on Advances in Internet Technology
Volume 12, Numbers 1 & 2, 2019

CONTENTS

Towards Service Level Guarantee within IoT Sensing Layer

Ahmad Khalil, Nader Mbarek, Olivier Togni
LIB, University of Bourgogne Franche-Comté
Dijon – France
emails: Ahmad.Khalil@u-bourgogne.fr, Nader.Mbarek@u-bourgogne.fr, Olivier.Togni@u-bourgogne.fr

Abstract — **Enabling service level guarantee within IoT (Internet of Things) environments is an important and a challenging task in order to enhance user experience while using IoT applications. The corresponding user service level expectations could be specified in a Service Level Agreement (SLA) that we have to conclude with the IoT Service Provider for each IoT service. As a consequence, several QoS (Quality of Service) mechanisms must be deployed within the IoT architecture layers (Sensing, Network, Cloud) to guarantee the agreed on IoT service level. We present in this paper a new QoS mechanism concerning the IoT Sensing layer. It is an adaptation of the slotted Carrier-Sense Multiple Access with Collision Avoidance (CSMA/CA) method used in the Media Access Control (MAC) layer of the IEEE 802.15.4 standard. This adaptation provides IoT smart objects with a differentiated wireless access according to the QoS class of their generated traffic in order to respect the requirements of the corresponding IoT SLA. The proposed method ensures a service level guarantee for a Low Rate Wireless Personal Area Network (LR-WPAN) in an IoT environment. Our adaptation offers a minimal delay for real time traffic along with higher Packet Delivery Ratio (PDR) for all traffics comparing to the standard slotted CSMA/CA. It consists in creating different Contention Access Periods (CAP); each will be specific for a traffic type and so for a specific QoS class. To do so, we propose firstly a QoS based wireless access method to be used by the coordinator, known as the gateway. Secondly, we propose an algorithm used by the IoT smart objects. This method, called QBAIoT (QoS Based Access for IoT environments), enables the coordinator to configure different contention periods with a specific number of slots. Consequently, the IoT objects of the same QoS class will access the channel only during their respective contention periods without collision with nodes belonging to other classes.**

Keywords - IoT; Service Level; QoS; QBAIoT; Slotted CSMA/CA; IoT Gateway; IoT objects.

I. INTRODUCTION

The Internet of Things (IoT) is currently an evidence in our daily lives. This paper extends the work conducted in [1] to show the importance of QoS guarantee in the IoT environment. In fact, by 2020, more than 20 billion digital and electronic devices will be connected resulting in an average of 2 devices per human being on Earth [2]. Thus, the impact of the IoT on human life will be important and should improve the quality of life by changing how people interact with connected objects and use IoT applications. The future growth of IoT environments will lead to an advanced technology usage enabling to facilitate the daily tasks of humans. Therefore, the improvement of the corresponding services is a major challenge within the IoT. In order to expand the usage of the IoT environment, a better user experience is expected. Consequently, QoS mechanisms should be implemented within the IoT environment [3] and especially the communication technologies used in the sensing layer of the IoT architecture such as the IEEE 802.15.4 standard [4]. The latter specifies the physical (PHY) and the Media Access Control (MAC) layers and provides an important foundation for other standards. Indeed, IEEE 802.15.4 standard is used by 6LowPAN [5] and ZigBee [6] for their lower layers implementation.

In this context, we specify QBAIoT as a novel QoS based wireless access method for IoT environments. It is an enhancement of the slotted Carrier-Sense Multiple Access with Collision Avoidance (CSMA/CA) technique, used by the IEEE 802.15.4 standard. The objective of QBAIoT is to ensure a differentiation between traffics while using the wireless channel of the IoT sensing layer. Thus, QBAIoT allows serving different IoT generated traffics while respecting the requirements of each traffic type (i.e., reduced delay for Real Time traffic). In this paper, we aim to present the design details of our proposed QoS based access method, as well as the corresponding simulation results. The reminder of the paper is organized as follows. We present in Section II the state of the art concerning the IoT environment, as well as the related technologies and we introduce the important characteristics of the IEEE 802.15.4 standard. Section III presents QoS motivations in the IoT, some related research works along with a description of an IoT Service Level Agreement (iSLA) achieved between an IoT Service Provider (IoT-SP) and an IoT Client (IoT-C). Then, we specify in Section IV our proposed method enabling QoS based access for IoT environments. Section V presents a detailed performance evaluation of our access method as well as a comparison with the standard access method. Finally, we conclude the paper in Section VI and present future works.

II. STATE OF THE ART

A. IoT environment

The important impact of the IoT on our society has led the international organizations to present several definitions and architectures and to create specific working and study groups focusing on IoT environments. The International Telecommunication Union - Telecommunication sector (ITU-T) presented different recommendations for the IoT such as Y.2060 [7] and Y.2066 [3] documents. Furthermore, the International Organization for standardization / International Electrotechnical Commission (ISO/IEC) presented a preliminary report about IoT in 2014 [8]. Moreover, the Internet Engineering Task Force (IETF) took an interest in the IoT environment by presenting different drafts concerning the emerging challenges for the IoT [9] [10]. Based on definitions and concepts presented by the different standardization organizations and international research projects, we can propose the following IoT definition: IoT is a system of systems interconnected via standard and interoperable communication technologies. This interconnection allows creating a considerable network of communicating objects, each addressed uniquely, in order to offer new services for improving the quality of human life. Also, self-management capabilities are essential within IoT environment in order to offer autonomous self-managed objects. In the context of the IoT, we use external resources such as cloud computing and fog computing for the processing and the storage of huge amount of data. Indeed, cloud computing functionalities enhance reliability and efficiency of IoT service provision [11]. On the other hand, fog computing decentralizes the computing capacities and distributes the operations on network extremities [12].

Different application domains with a variety of services are provided in the IoT environment. These application domains cover a wide variety of everyday services like health services, industry services, transportation services, city management services, etc. They had drawn the attention of several international organizations in order to work on standards used in the mentioned domains. For example, the ISO/IEC focuses on the standardization of underlying technologies useful in different IoT application areas. Thus, the Working Group 9 of ISO/IEC Technical Committee 1 (JTC 1/WG9) focuses on the standardization of Big Data technologies in the areas of IoT [13]. In addition, each IoT application domain attracts specific international organizations. For the e-health services, the World Health Organization (WHO) and the Program for Appropriate Technology in Health (PATH) have signed a partnership to accelerate the evolution of digital health worldwide [14]. As for the smart city domain, ISO/IEC through the technical subcommittee JTC1/SC25, standardizes microprocessor systems and interconnection mediums associated with equipment for commercial and residential environments. IoT services has attracted also, the attention of a large number of manufacturers and industrial companies like Ericsson and its partners that had offered portable prototypes for the e-health domain with long battery life [15]. In addition, Nokia offered several services and technologies on the market to manage video surveillance, sensors' networks, smart parking, etc [16].

In order to offer the IoT services, various communication technologies interconnect IoT objects and gateways within IoT environments. Each technology is suitable for a specific scenario based on different criteria such as energy consumption, CPU utilization, range of the technology, etc. IoT communication technologies correspond to an adaptation of an existing technology or to a new specifically specified technology. IoT can use wireless cellular technologies [17] (LTE, 4G, NB-IoT, 5G, etc.) or wireless non-cellular technologies (IEEE802.15.4 [4], LoRaWAN [18], ZigBee [6], 6LoWPAN [5], etc.). We describe in the following section the IEEE 802.15.4 wireless non-cellular technology, which is the foundation of our proposed QoS based access method.

B. IEEE 802.15.4

The IEEE 802.15.4 standard is an IEEE proposed standard for Low Rate Wireless Personal Area Networks (LR-WPAN). It defines the physical and the MAC layers to provide a basic format. This format will be used by other technologies and protocols by adding their own specificities through the specification of the higher layers. The IEEE 802.15.4 physical layer specifies different essential parameters: 250 Kbit/s of data rate for a 2.4 GHz band, control functions like the activation or deactivation of the radio module, the test of the channel occupation and the choice of the transmission channel. On the other hand, the MAC layer defines the data management format and specifies the usage of different access methods for the wireless shared channel (i.e., Unslotted CSMA/CA, Slotted CSMA/CA, TSCH CCA, TSCH CSMA/CA, CSMA/CA with PCA, DSME, etc.). As for data encryption, the IEEE 802.15.4 standard uses AES-128 (Advanced Encryption Standard) to ensure data confidentiality [4]. Different standards use IEEE 802.15.4 as a foundation for their lower layers. We can mention as an example the 6LowPAN standard that combines IPv6 with low power WPAN networks. Another example is ZigBee, a specification for a series of high-level, low-power communication.

IEEE 802.15.4 supports a beacon-enabled mode using a superframe structure, which is the base of our contribution. The superframe (see Fig. 1) consists of an active part known as the Superframe Duration (*SD*) and can be followed by an inactive period. The active part is formed by 16 equally sized time slots partitioned into a Contention Access Period (*CAP*) where nodes compete to gain the access to the channel; and an optional Contention Free Period (*CFP*) where nodes are allocated guaranteed time slots.

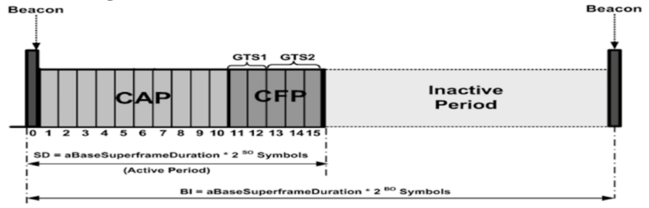


Figure 1. IEEE 802.15.4 beacon enabled mode superframe structure

In beacon-enabled mode, the coordinator sends periodically a beacon frame on the network including all the superframe specifications. The beacon, sent at the Beacon Interval (*BI*) time, allows the coordinator to identify its WPAN and ensures that all the objects are synchronized. The Beacon Order (*BO*) and Superframe Order (SO) parameters determine the Beacon Interval (*BI*) and *SD*, respectively as mentioned in (1) and (2). The Base Superframe Duration (*BSFD*) corresponds to the minimum duration of the superframe (*SO* = 0).

$$BI = BSFD * 2^{BO} \quad (1)$$

$$SD = BSFD * 2^{SO} \quad (2)$$

BSFD is fixed to 960 symbols of 4 bits or 15.36 ms assuming the data rate of 250 Kbit/s for the 2.4 GHz band. In addition, *BO* and SO should respect the inequality $0 \leq SO \leq BO \leq 14$ [4].

Three variable are used in the slotted CSMA/CA algorithm (see Fig. 2): the Backoff Exponent (*BE*), the Contention Window (*CW*) and the Number of Backoffs (*NB*). To compute the backoff delay, that an object has to observe before performing the Clear Channel Assessment (*CCA*), the algorithm chooses a random value for the backoff delay between 0 and ($2^{BE} - 1$). CW is the number of backoff periods during which the channel must be idle before accessing the channel. By default, the value of *CW* is fixed to 2. *NB* is the number of backoff executed for channel access. This value is initialized to 0 and is compared to a maximum value, *macMaxCSMABackoffs* by default equal to 5. In case the *NB* value is greater than this maximum value, a failure occurs.

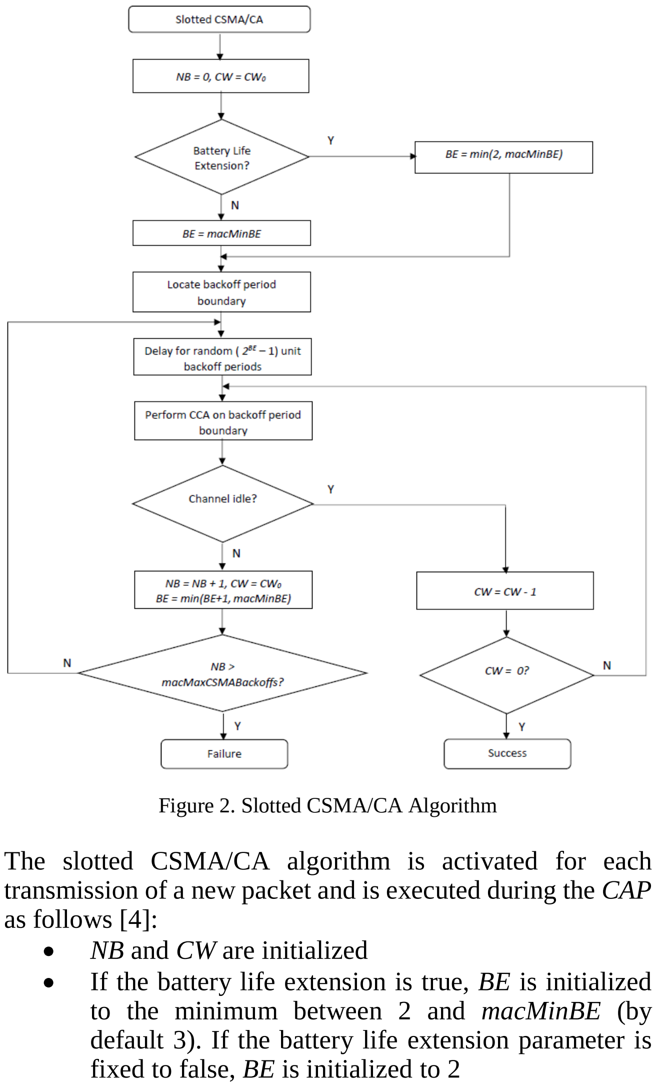


Figure 2. Slotted CSMA/CA Algorithm

The slotted CSMA/CA algorithm is activated for each transmission of a new packet and is executed during the *CAP* as follows [4]:

- *NB* and *CW* are initialized
- If the battery life extension is true, *BE* is initialized to the minimum between 2 and *macMinBE* (by default 3). If the battery life extension parameter is fixed to false, *BE* is initialized to 2
- The node using the algorithm waits the backoff delay, and then performs *CCA*
- If the channel is busy, *CW* is re-initialized to 2, *NB* and *BE* are incremented. *BE* must not exceed *aMaxBE* (by default 5). If *macMaxCSMABackoffs* is reached, the algorithm reports a failure to the higher layer. If *NB* < *macMaxCSMABackoffs*, the backoff operation is restarted and the *CCA* should be performed again
- If the channel is sensed idle and *CW* > 0, the *CCA* is repeated and *CW* decremented. Otherwise, the node attempts to transmit if the remaining time in the current *CAP* is sufficient to transmit the frame and receive the acknowledgement. If not, the process is deferred to the next superframe.

III. QoS GUARANTEE IN THE IoT

A. *Motivations and challenges for QoS guarantee in the IoT*

The ITU-T E.800 [19] has defined QoS as the totality of the characteristics of a telecommunication service to satisfy in order to meet the user requirements. In this context, a QoS requirement is expressed in terms of QoS parameters (Delay, Jitter, Packet Delivery Ratio, Effective Data Rate, etc.). QoS guarantee in the IoT environment requires an effective and optimized management of the corresponding resources to improve users' experience. In order to provide predictable services, QoS mechanisms in the IoT environment handle delays, jitter, bandwidth and packet loss ratio by classifying traffic. As the IoT environment is made of different technologies and heterogeneous networks, different types of data and streams exist on a single system. Hence, it is important to provide the IoT environment with QoS guarantee mechanisms to meet the requirements of each type of traffic [9]. QoS guarantee is a critical challenge in the IoT, as the number of connected objects increases considerably leading to a greater amount of created and transported data with different characteristics. Consequently, the performance of the IoT system will be affected and especially QoS constrained data traffic due to congestion periods. Deploying QoS mechanisms within IoT environment will enhance the performance by identifying and differentiating traffic in order to allow a reduced cost and a better scalability [10].

The importance of the QoS guarantee in the IoT has been put forward by various international organizations The ITU-T describes the importance of QoS integration in the IoT through various documents such as Y.2066 [3] where it was mentioned that service priority is an important requirement. In addition, Y.2066 indicates that the prioritization functionality satisfies different service requirements of IoT users. On the other hand, LinkLabs, an American company developing technologies for computer networks, indicates that integrating QoS into IoT allows a better management of the corresponding capabilities and resources in order to provide a reliable and optimized infrastructure for connecting objects. According to LinkLabs, QoS mechanisms enables predictable IoT services thanks to better delay, jitter, bandwidth and Packet Delivery Ratio (PDR) by classifying traffic and offering services according to systems' resources [22].

In order to provide QoS within an IoT architecture, the requirements of each layer (Sensing layer, Network layer and Cloud Layer) should be addressed through one or several mechanisms. The Sensing layer includes all the IoT objects along with the gateways allowing their interconnection and management. Thus, the QoS provision at this layer should meet the IoT objects and gateways requirements. An essential challenge for this layer is traffic differentiation and prioritization. It can be offered by classifying the different flows according to their criticality and applying prioritization through different adapted QoS mechanisms. Thus, it is important to classify IoT applications according to specific criteria in order to propose an appropriate QoS mechanism while respecting their traffics characteristics. Each set of applications will have mechanisms well adapted to their requirements. In addition, at this layer the optimization of the systems resources usage should be applied in order to offer the best performances. The network layer of the IoT architecture includes all network features such as routing, handoff, and path management (path selection and recovery) through a multi-path infrastructure. This layer acts as a network infrastructure interconnecting the Sensing layer to the Cloud layer. The integration of QoS mechanisms in this layer should consider the large number of requests and data transiting from the Sensing layer to the Cloud layer of the IoT architecture. The data processing must be differentiated in the Network layer. It must prioritize requests according to their importance. As a result, the QoS requirements of the IoT Network layer correspond to the traditional QoS requirements of a network infrastructure while adapting these needs to the characteristics of the IoT environment. Finally, the IoT Cloud layer includes computing and storage capabilities. In addition, this layer hosts IoT applications enabling processing data for useful purposes. The QoS guarantee in the Cloud layer is an emerging discipline with several research challenges. This is due to the lack of standardized end-to-end approaches for QoS assurance and the existence of various constraints and QoS parameters specific to each cloud service. Indeed, QoS requirements in the Cloud layer depend on the provided service (Infrastructure as a Service - IaaS, Software as a Service - SaaS, Platform as a Service - PaaS) by the Cloud Service Provider (CSP). Finally, it is necessary to specify the needs and mechanisms ensuring end-to-end QoS guarantee across the different layers of the IoT architecture. This end-to-end QoS provision allows customers to perceive the requested service level without distinguishing the declination of this QoS according to the IoT architecture several layers.

In the next sections, we present related research work concerning QoS offer in IoT environments and we describe the IoT Service Level Agreement.

B. Related research work

Different international projects and research works had studied the Quality of Service in the IoT environment and its impact on the service provision. The European project OpenIoT [23] specified different QoS parameters and metrics for the IoT. These metrics include utility metrics related to sensors and other metrics related to the network and application. As an example of utility metrics, OpenIoT indicated the Quality of sensors that determines the accuracy of measurement, the energy consumption, data volume, and bandwidth. For the other metrics, system lifetime is taken into consideration. In addition, traditional QoS parameters are used such as latency, jitter, delay, throughput, etc. On the other hand, this project presented a high level architecture based on a QoS Manager that keeps track of the following parameters: quality of sensors, energy consumption, trustworthiness, bandwidth and data volume.

The research work carried out in [24], concerning the guarantee of QoS in IoT, proposes to classify various IoT applications according to 3 service models (i.e., Open Service Model, Supple Service Model, Complete Service Model).. It maps each class to a physical topology for sensors' implementation. Open Service Model corresponds to interactive, non-real-time and non-critical applications. Supple Service Model corresponds to interactive, Soft Real Time and critical applications. Complete Service Model corresponds to interactive, Hard Real time and critical applications. Thus, the authors classified the IoT applications belonging to different domains according to these 3 models. In addition, this work has matched the proposed service models with physical topologies (star topology and random topology) at the device layer to meet the needs of each model. Indeed, the applications belonging to the Complete model must be provided through a physical star topology to obtain better delays. On the other hand, applications belonging to the Open model must be provided through a random physical topology for better energy consumption.

Furthermore, other research works had focused on the QoS in the lower layer of the IoT architecture (sensor layer). For example, the research work conducted in [25], tried to use different queues and a scheduler to ensure a certain priority for QoS constrained flows. Moreover, different research work tried to adapt the slotted CSMA/CA algorithm to ensure QoS guarantee. Thus, the authors present in [26] a contribution that allows the delivery of critical data with a highest priority during the CFP. In [27], the authors describe the usage of different values for *CW*, *minBE* and *maxBE* to differentiate services thanks to three different priority levels. However, these research works did not take into consideration the existence of real time applications in the IoT environment requiring a reduced delay that does not exceed milliseconds range. For this matter, our proposed QoS based access method aims to provide a differentiation between IoT objects' flows based on different QoS classes' characteristics.

C. IoT Service Level Agreement

In this research work we consider four types of traffics corresponding to four QoS classes as specified in a previous work [28]: Real Time Mission Critical (RTMC), Real Time Non Mission Critical (RTNMC), Streaming and Non Real Time (NRT). Each QoS class corresponds to several requirements regarding performance parameters such as delay, jitter, etc. For example, our specified Real Time QoS classes are more sensitive to delay and jitter variation. The Streaming class is more sensitive to jitter variation while the Non Real Time class is a non-constrained QoS traffic class.

In order to specify the concrete requirements of each QoS class (IoT-C's expected value of each performance parameter), we had presented in our previous work [28] a specific Service Level Agreement (SLA) for IoT environments, called iSLA, in order to allow an IoT-SP and an IoT-C to negotiate and agree on the expected service level. The expectations are described through different measurable parameters according to the IoT type of service (i.e., QoS class). We specify for each QoS class a set of measurable parameters that are critical for the type of data concerned by that QoS class. In addition, the IoT-SP uses a cloud infrastructure, a network infrastructure and a sensing infrastructure to provide the IoT service. In this context, our proposed iSLA considers the characteristics of each sub-infrastructure needed by the provided IoT service. Thus, the corresponding sub-SLAs, forming the global iSLA, are concluded with a CSP (i.e., cloud SLA: cSLA) and a Network Service Provider (NSP) (i.e., network SLA: nSLA). For the sensing infrastructure, the IoT- SP dispose of two kinds of gateways; High Level Gateways (HL-Gws) used for self-management provision and Low Level Gateways (LL-Gws) used to collect data from IoT objects. The IoT-SP concludes another internal sub-SLA called the gateway SLA (gSLA) to specify the characteristics of the gateways for the corresponding IoT Service. The gSLAs (stored on the HL-Gw) allow the HL-Gw to have detailed information concerning the characteristics of the underlying infrastructure for self-management consideration. After concluding the cSLA, nSLA and gSLA, the IoT-SP is able to conclude the global iSLA with the IoT-C. In order to describe the iSLA establishment process accomplished by the IoT-SP, we specify a Finite State Machine (FSM) diagram with several states illustrating the behavior of the IoT service Provider (see Fig. 3).

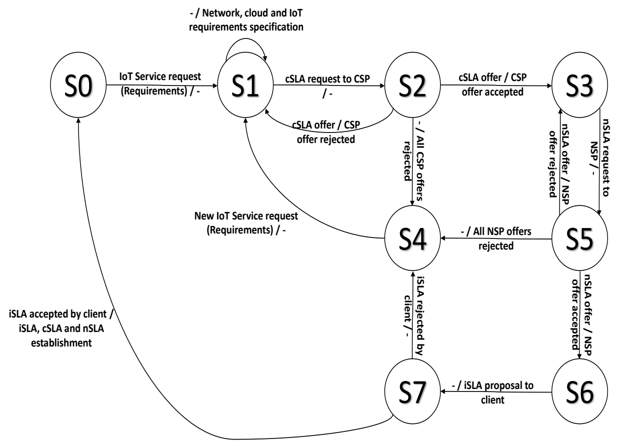

- / Network, cloud and IoT requirements specification
S0
IoT Service request (Requirements) / -
S1
cSLA request to CSP / -
S2
cSLA offer / CSP offer accepted
S3
cSLA offer / CSP offer rejected
- / All CSP offers rejected
nSLA offer / NSP offer rejected
nSLA request to NSP / -
New IoT Service request (Requirements) / -
S4
- / All NSP offers rejected
S5
iSLA accepted by client / iSLA, cSLA and nSLA establishment
iSLA rejected by client / -
nSLA offer / NSP offer accepted
S7
- / iSLA proposal to client
S6

Figure 3. Finite State Machine of iSLA establishment

In state S0, the IoT-SP waits for the service requirements from the client to start the process of iSLA establishment. After receiving these requirements, the IoT-SP classifies the requirements in state S1 and changes to state S2 when it sends the cSLA request to the CSP in order to conclude a cloud SLA. The CSP sends to the IoT-SP a cSLA offer. If the offer is rejected, then the IoT-SP state changes again to state S1 but if the offer is accepted, the IoT-SP reaches state S3. If all CSP offers are rejected, the IoT-SP will be at state S1 after reaching state S4 to wait for a new set of requirements as the older set cannot be satisfied and the process restarts. The same process is executed with the NSP. If the NSP offer is accepted, the IoT-SP will be at the state S6. If all NSP offers are rejected, the IoT-SP will be at state S1 after reaching state S4 to wait for a new set of requirements as the older set cannot be satisfied and the process restarts. After accepting the nSLA, the IoT-SP at state S6 sends an iSLA proposal to the client and reaches state S7. If the client rejects the iSLA, the IoT-SP passes to state S4 and a new round of negotiation with the service provider should be achieved in order to build a new iSLA. If the iSLA proposal is accepted, the IoT-SP concludes the sub-SLAs with the corresponding NSP and CSP and concludes the iSLA with the IoT-C while reaching the initial state S0.

We specify in the next section our proposed QoS mechanism called QBAIoT. It is a wireless access method based on the the four QoS classes mentioned above and ensures a differentiation in traffic processing for QoS integration within the sensing layer of the IoT architecture.

IV. QoS Based Access for IoT

We describe in the following our QoS based access method for IoT environments called QBAIoT. The specification of our novel access method is based on a new superframe structure, as well as algorithms implemented within the IoT Gateway and IoT objects enabling Class based Contention Free Periods.

A. Class based Contention Free Period Access

Our proposed access method consists in using an IEEE 802.15.4 superframe that respects the requirements of the four QoS classes. For achieving our QoS guarantee according to the requirements of the different traffics, we adapt the structure of the IEEE 802.15.4 superframe in order to include a CAP (called QoS CAP) for each traffic corresponding to a specific QoS class. Moreover, there are no CFP and inactive periods in our adapted superframe.

We had removed the inactive period to reduce the delay of Real Time generated data. In this context, we can find up to four QoS CAPs in our superframe in case the IoT gateway (Coordinator or LL-Gw) is configured with four QoS classes (see Fig. 4).

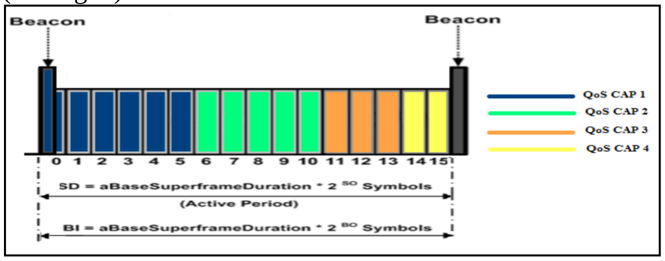


Figure 4. QBAIoT superframe structure

During each QoS CAP, only objects belonging to the corresponding QoS class can try to use the slots in order to send their data. The slots configuration and the number of QoS CAPs in the superframe is based on the number of QoS classes available in the IoT gateway environment. Different configurations for the superframe based on the existence of

Real Time applications and the number of QoS classes in the considered IoT environment are possible. If the network includes one QoS class, a single CAP will exist in the superframe and the normal IEEE 802.15.4 slotted CSMA/CA algorithm is used. If there are multiple QoS classes with a minimum of one Real Time class in the network, BO and SO will be configured with the value 2 in order to minimize the latency of Real Time traffic thanks to a reduced Superframe Duration among others. Consequently, based on (1) and (2), BI and SD correspond to 61.44 ms with a slot time of 3.84 ms. If multiple QoS classes exist with no Real Time classes, BO and SO are set to 3 fixing BI and SD to 122.88 ms with a slot time of 7.68 ms. We specify for each QoS CAP a fixed number of slots. This configuration differs according to the number of existing QoS classes in the IoT Gateway environment. For example, in the case of 4 QoS classes the superframe slot configuration is as follows: RTMC class QoS CAP is allocated 6 slots, RTNMC class QoS CAP is allocated 5 slots, Streaming class QoS CAP is allocated 3 slots and NRT class QoS CAP is allocated 2 slots. So, slots configuration and the number of QoS CAP in the superframe is based on the number of existing QoS classes.

B. IoT Gateway QoS based access method design

For the coordinator part (i.e., IoT Gateway) of our proposed QBAIoT access method, we specify Algorithm 1 (see Fig. 5) among with the corresponding variables described in Table I.

Algorithm 1 Gateway QBAIoT Access Method Algorithm

```
Input:  Nb_QoS_Classes, RT_Classes
1:  N ← 1
2:  if (Nb_QoS_Classes = 1) then
3:          BO, SO ← 14
4:          MAC ← Slotted_CSMA
5:          While true do
6:                  Send_Beacon (BO, SO, CAP)
7:                  Receive_Data ()
8:          end while
9: else
10:         if (RT_Classes = 0) then
11:                 BO, SO ← 3
12:                 MAC ← QBAIoT
13:                 Initial_Slots_Configuration ()
14:                 While true do
15:                         Send_Beacon (BO, SO, QoS CAPs)
16:                         While(N<=Nb_QoS_Classes) do
17:                                 Receive_Data (QoS CAP)
18:                                 N ← N + 1 // Next QoS CAP
19:                         end while
20:                 end while
21:         else
22:                 BO, SO ← 2
23:                 MAC ← QBAIoT
24:                 Initial_Slots_Configuration ()
25:                 While true do
26:                         Send_Beacon (BO, SO, QoS CAPs)
27:                         While(N<=Nb_QoS_Classes) do
28:                                 Receive_Data (QoS CAP)
29:                                 N ← N + 1 // Next QoS CAP
30:                         end while
31:                 end while
32:         end if
33: end if
```

Figure 5. Gateway QBAIoT Access Metthod Algorithm

TABLE I. VARIABLE SPECIFICATION OF ALGORITHM 1

Name of the variable	Description
Nb_QoS_Classes	Number of QoS classes
RT_Classes	Number of Real Time classes
N	Index of QoS classes
MAC	Channel access algorithm
QoS CAP; CAP	Configuration of the CAP (CAPStart and CAPEnd)
Initial_Slots_Configuration()	Algorithm that computes the slots configuration based on the Number of QoS classes and Number of Real Time classes.

As shown in Fig. 6, **t**he IoT Gateway using our QoS based access method (i.e., QBAIoT gateway) will receive data from objects during the corresponding QoS CAPs.

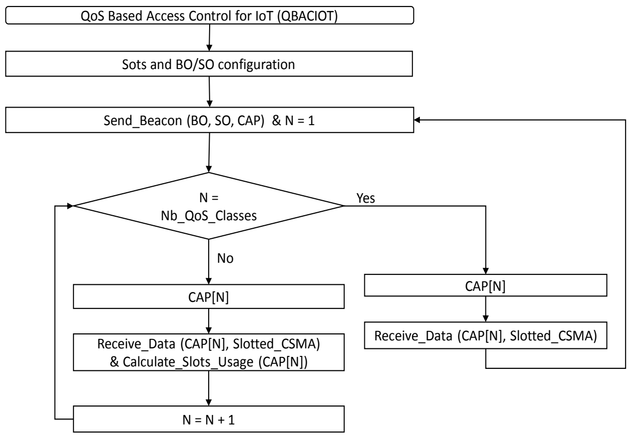

QoS Based Access Control for IoT (QBACIOT)
Sots and BO/SO configuration
Send_Beacon (BO, SO, CAP) & N = 1
N = Nb_QoS_Classes
Yes
No
CAP[N]
Receive_Data (CAP[N], Slotted_CSMA) & Calculate_Slots_Usage (CAP[N])
N = N + 1
CAP[N]
Receive_Data (CAP[N], Slotted_CSMA)

Figure 6. Gateway QBAIoT Access method

At each Beacon Interval, the gateway sends the beacon including the information regarding the values of *BO*, *SO* and the first and final slot for each QoS CAP. These values are used by the IoT objects to calculate the slot time and to determine during which time they are allowed to compete for the channel. A QBAIoT gateway should include also self-management capabilities.

A self-configuring capability enables the gateway to adapt the superframe slots configuration according to the existing number of QoS classes within its environment. A self-optimizing capability is performed in case of unused slots in a QoS CAP thanks to a slot reallocation mechanism covering the entire superframe. The self-management capabilities design is out of the scope of this paper.

C. *Class based access for IoT objects*

For the IoT object part of our proposed QBAIoT access method, we specify Algorithm 2 (see Fig. 7) among with the corresponding variables described in Table II.

Algorithm 2 Object QBAIoT Access Method Algorithm

```
1: Receive_Beacon (BO, SO, QoS CAPs)
2: Configuration (BO, SO, QoS CAPs)
3: while (Slot ∈ [CAPStart, CAPEnd] and Data = true) do
4:        if (Slotted_CSMA (Slot) = Success) then
5:                Send_Data (Success, PAN Coordinator)
// slotted CSMA/CA returns a success state
6:        else
7:                Send_Data (Failure, PAN Coordinator)
// slotted CSMA/CA returns a failure state
8:        end if
9: end while
10: if (Slot < CAPStart) then
11:        Wait_until (Slot ∈ [CAPStart, CAPEnd])
12: else
13:        Wait_Until (Beacon) // Wait until next superframe
14: end if
```

Figure 7. Object QBAIoT Access Method Algorithm

TABLE II. VARIABLE SPECIFICATION OF ALGORITHM 2

Name of the variable	Description
QoS CAP	Configuration of the CAP (CAPStart and CAPEnd)
CAP_Start_Slot	The first slot for the corresponding QoS CAP assigned to the object
CAP_End_Slot	The last slot for the corresponding QoS CAP assigned to the object

Any object in the IoT Gateway environment receives the beacon. According to the QoS class it belongs to, the object will determine during which QoS CAP it can compete to access the shared medium. When an IoT object generates data, it should test if it has the right to compete in order to send its traffic. If the corresponding QoS CAP of the object has not started, it waits until its CAP time and then competes to send the data according to our adapted slotted CSMA/CA algorithm. If the object QoS CAP had passed, it should wait until the corresponding QoS CAP in the next SuperFrame.

Fig. 8 shows the adapted CSMA/CA algorithm adopted by the IoT Objects that communicate using our QBAIoT method.

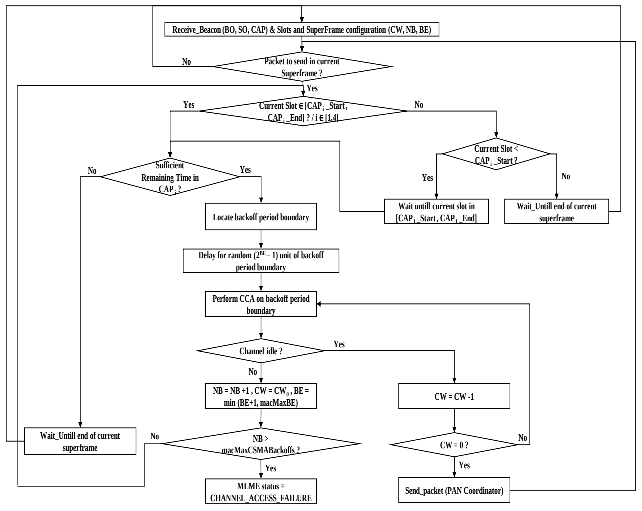

Receive_Beacon (BO, SO, CAP) & Slots and SuperFrame configuration (CW, NB, BE)
Packet to send in current Superframe ?
No
Yes
Current Slot ∈ [CAP_i _Start, CAP_i _End] ? / i ∈ [1,4]
Yes
No
Current Slot < CAP_i _Start ?
Yes
No
Wait untill current slot in [CAP_i _Start, CAP_i _End]
Wait_Untill end of current superframe
Sufficient Remaining Time in CAP_i ?
No
Yes
Locate backoff period boundary
Delay for random (2^BE – 1) unit of backoff period boundary
Perform CCA on backoff period boundary
Channel idle ?
Yes
No
NB = NB +1 , CW = CW_0 , BE = min (BE+1, macMaxBE)
CW = CW -1
Wait_Untill end of current superframe
NB > macMaxCSMABackoffs ?
No
Yes
CW = 0 ?
No
Yes
MLME status = CHANNEL_ACCESS_FAILURE
Send_packet (PAN Coordinator)

Figure 8. Object QBAIoT Access Method

V. PERFORMANCE EVALUATION AND RESULTS

A. *Simulation environment*

In order to evaluate our proposed QBAIoT access method, we conduct a simulation study using OMNeT++ based on the IEEE 802.15.4 model [29] including all the necessary features like the beacon, the superframe structure, etc. We had adapted this model to take into consideration our proposed QoS based access method thanks to a superframe with no CFP and different QoS CAPs. In our simulation scenario, we simulated four QoS classes (RTMC, RTNMC, Streaming and NRT). We used a star topology with a single coordinator (i.e., IoT Gateway) where all devices (i.e., IoT objects) are in each other's radio range. Each device transmits data to the coordinator. The data packets are generated periodically but are transmitted during the corresponding QoS CAP. Table III shows the used simulation parameters.

In the first simulation scenario, we fixed the Data Generation Interval (DGI) to 0.25 seconds and we increased the number of IoT objects from 4 (1 per QoS class) to 12 (3 per QoS class). The IoT objects are sending data simultaneously as they start generating data at the same time with the same interval of packet generation. As for the second set of simulations, we used a DGI of 0.125s allowing generating a double amount of packets comparing to the first set of simulations.

TABLE III. SIMULATION PARAMETERS

Parameter	Value
Carrier Frequency	2.4 GHz
Transmitter Power	1 mW
Bit rate	250 Kbps
Simulation Time	100 s
Max Frame Retries	3
Mac Payload Size	50 Bytes

B. *Performance evaluation*

The evaluation of our proposed QoS based access method is based on different performance parameters concerning the traffic of our QoS classes. The importance of these parameters depends on the characteristics of the corresponding traffic. Indeed, the average delay is very important and critical for the RTMC and RTNMC traffic whereas it is less important for Streaming traffic and not important for NRT traffic. In this context, we considered the following performance parameters.

- Average Delay: It refers to the average time experienced by a generated packet to be received by the destination. It is computed by dividing the total delay experienced for all the packets by the number of packets as shown in equation (3).

$$Average\ Delay = \frac{\Sigma\ Delay}{Number\ of\ received\ packets} \quad (3)$$

- PDR: It expresses the degree of reliability achieved by the system for successful transmissions. It is obtained by dividing the number of received packets by the number of generated packets as shown in equation (4). Non received packets are either lost due to a collision or still in the sender buffer waiting for channel access.

$$PDR = \frac{Number\ of\ received\ packets}{Number\ of\ sent\ packets} \quad (4)$$

- Mean Packet Delivery Ratio (MPDR): It expresses the degree of reliability achieved by the system for successful transmissions of all traffic types. It is obtained by computing the mean value of the PDRs of the different traffic types as shown in equation (5).

$$MPDR = \frac{\Sigma\ PDR}{Number\ of\ traffic\ types} \quad (5)$$

- Effective data rate (EDR): It evaluates the link bandwidth utilization. It is computed by multiplying the number of received packets by their sizes to obtain the total length of the frame, which is divided by the simulation time as shown in equation (6).

$$EDR = \frac{\text{Number of received packets} * \text{Packet Size}}{\text{Simulation Time}} \quad (6)$$

Table IV presents the delay evaluation for 4 QoS classes traffic while using our proposed QBAIoT access method and the traditional IEEE 802.15.4 slotted CSMA/CA method for the first set of simulations (using the 0.25s DGI). The Delay QoS parameter is very sensitive for RTMC and RTNMC traffic. The obtained results in Table IV shows that for 4 objects (1 object per QoS CAP), our proposed method enables better delay for the RTMC traffic (10 ms less than the standard) and the RTNMC traffic (7 ms less than the standard). This difference becomes greater while increasing the number of objects. For 8 objects in the IoT environment (2 objects per QoS CAP), we can observe a 35 ms better delay for RTMC traffic and 26 ms better delay for RTNMC traffic. The better delays that we obtain for Real Time traffic with our proposed method are owing to the fact of giving the Real Time classes a more important number of slots in which they can send their data without any collision with other objects belonging to other non-real time QoS classes. Consequently, data packets do not need to wait in buffer for a long time. They are served faster than other traffic types.

Although it is not critical for NRT traffic, we notice important delays for this traffic when the total number of objects is equal to 12 (3 objects per QoS CAP). This delay comes from the fact that this traffic is served during 2 slots in each superframe and that each traffic class generates the same number of packets in our scenario at the same time; all packets of the different QoS classes are generated at the same time. So, when the number of objects in the NRT class increases, the delay will increase because the generated traffic is greater than the allocated capacity of 2 slots resulting in a great number of packets in the sending buffer.

TABLE IV. AVERAGE DELAY EVALUATION FOR DIFFERENT TRAFFIC TYPES USING QBAIOT AND IEEE 802.15.4 STANDARD

	Delay in seconds							
QoS class / Number of object per class	RTMC Standard	RTMC QBAIoT	RTNMC Standard	RTNMC QBAIoT	Streaming Standard	Streaming QBAIoT	NRT Standard	NRT QBAIoT
1	0.062	0.052	0.063	0.056	0.062	0.063	0.067	0.067
2	0.1	0.065	0.1004	0.074	0.102	0.104	0.104	0.67
3	0.115	0.09	0.123	0.106	0.0129	0.124	0.131	30.61

Table V shows the Packet Delivery Ratio for 4 QoS classes traffic while using our proposed QoS based access method and the IEEE 802.15.4 standard for the first set of simulations. Our QBAIoT access method is giving, for all QoS classes three times better PDR with one object by class, four times better PDR with two objects by class and 6 times better PDR (except NRT class 1,5 times) with 3 objects by class than IEEE 802.15.4 standard method. We obtain a better PDR with our approach thanks to an optimized channel access per class avoiding collisions between different QoS classes. Indeed, for each QoS CAP, only objects of the corresponding QoS class can compete to access the channel. For example, with 1 object per QoS class, there is no competition between objects to gain access to the channel during each slot with QBAIoT comparing to a competition between 4 objects while using IEEE 802.15.4. Consequently, with QBAIoT a lower number of objects are competing for accessing the channel for a given slot. Packets will not run the slotted CSMA/CA algorithm for several times and there is no need to drop packets after several attempts when macMaxCSMABackoffs is reached.

TABLE V. PDR EVALUATION FOR DIFFERENT TRAFFIC TYPES USING QBAIOT AND IEEE 802.15.4 STANDARD

	Packet Delivery Rate							
QoS class / Number of object per class	RTMC Standard	RTMC QBAIoT	RTNMC Standard	RTNMC QBAIoT	Streaming Standard	Streaming QBAIoT	NRT Standard	NRT QBAIoT
1	0.31	1	0.36	1	0.35	1	0.33	1
2	0.18	0.99	0.21	0.99	0.21	0.97	0.19	1
3	0.20	0.98	0.18	0.96	0.15	0.90	0.16	0.26

As for the effective data rate, Table VI compares the obtained results using our proposed QBAIoT method and the traditional slotted CSMA/CA of the IEEE 802.15.4. The obtained results show that QBAIoT allows always better effective data rate than the traditional approach, as the PDR of QBAIoT is always higher. A lower number of collisions offer a higher number of received packets. Consequently, the number of bits served is higher during the simulation time allowing a greater EDR with QBAIoT. We can note an average of 4 times better EDR with QBAIoT comparing to IEEE 802.15.4 for all QoS classes with 4, 8 and 12 objects in the IoT environment (except for the NRT traffic with 3 objects per QoS class in the environment where the EDR with QBAIoT is only 1.7 time better).

TABLE VI. EDR EVALUATION FOR DIFFERENT TRAFFIC TYPES USING QBAIOT AND IEEE 802.15.4 STANDARD

	Effective Data Rate in bits per second							
QoS class / Number of object per class	RTMC Standard	RTMC QBAIoT	RTNMC Standard	RTNMC QBAIoT	Streaming Standard	Streaming QBAIoT	NRT Standard	NRT QBAIoT
1	508	1600	588	1600	564	1600	528	1600
2	604	3190	680	3180	692	3120	612	3200
3	972	4710	904	4620	728	4330	722	1240

In the second set of simulations, we used the same environment as for the first set but with a Data Generation Interval of 0.125s allowing generating 800 packets per objects during the simulation time.

Fig. 9 presents the comparison of the different average delay results concerning RTMC traffics while using QBAIoT and IEEE 802.15.4 standard with a DGI of 0.125s, as well as with 1, 2 and 3 objects per QoS class. We can note that with QBAIoT, better average delays are observed in all cases. By incrementing the number of objects, the results of average delay turn into greater values as more important number of packets should be served during the same QoS CAP. Comparing to Table IV, the observed average delay by RTMC traffic becomes more important by decreasing the DGI. Indeed, lower DGI values correspond to a more important number of generated packets each second.

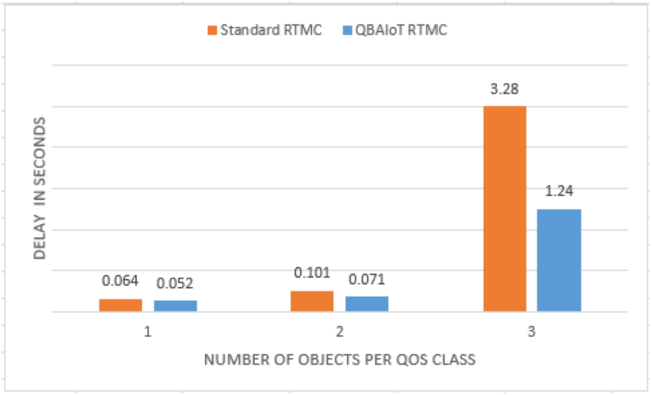


Figure 9. Average delay evaluation for RTMC traffic using QBAIoT and IEEE 802.15.4 for a DGI of 0.125s

Fig. 10 presents a comparison of QBAIoT RTMC traffic average delay for a DGI of 0.125s and 0.25s for 1, 2 and 3 objects per QoS class. We can note that the generation of the same number of packets by a single object allows observing a lower average delay comparing to the same number of packets generated by two or more objects. For instance, the generation of 800 RTMC packets by a single object (1 object per QoS class with a DGI of 0.125s) induces a 0.052 ms average delay for RTMC traffic. Whereas, the generation of 800 RTMC packets by two objects (2 objects per QoS class with a DGI of 0.25s) induces an average delay of 0.065 ms for RTMC traffic. The 13 ms higher average delay with two objects generating the 800 packets is due to the collisions that can occur between the two objects of the same QoS class during the contention for accessing the channel.

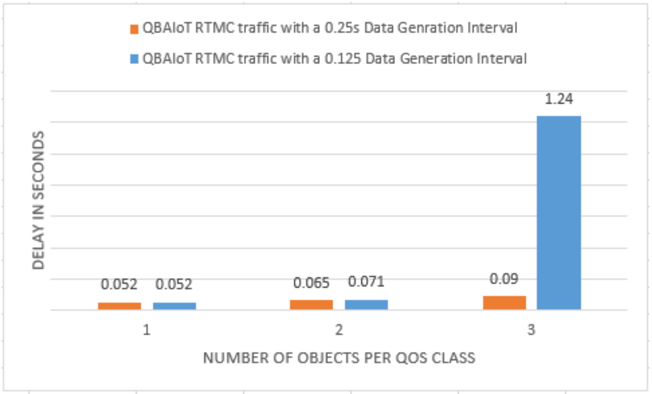


Figure 10. Average delay evaluation for QBAIoT RTMC traffic with different DGI

Fig. 11 presents the comparison of the different average delay results of RTNMC traffics while using QBAIoT and IEEE 802.15.4 standard with a DGI of 0.125s, as well as 1, 2 and 3 objects per QoS class. We can note that with QBAIoT a better dealy is observed by RTNMC traffic in all cases thanks to the fact of minimizing the collisions and organizing the time during which each object can compete to gain access to the shared medium.

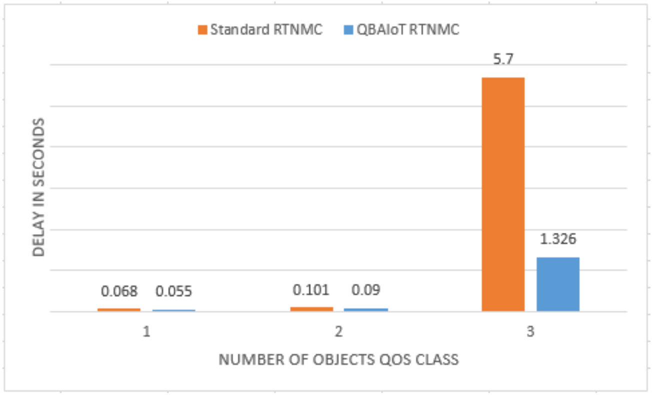


Figure 11. Average delay evaluation for RTMC traffic using QBAIoT and IEEE 802.15.4 for a DGI of 0.125s

Fig. 12 presents the comparison between the different MPDR values of the different traffics with the DGIs of 0.25s and 0.125s while using QBAIoT. We can observe that with 1 object per QoS class, as there is no collisions between packets of different objects and the maximum capacity of the medium has not been reached yet for each QoS CAP, the PDR mean value is equal to the maximum value of 1. As for the case with 2 and 3 objects per QoS class, the MPDR value is lower with a DGI of 0.125s as the number of competition executed to access the channel is higher than the case with a DGI of 0.25s. Consequently, the probability of having a collision is higher resulting in lower MPDR values.

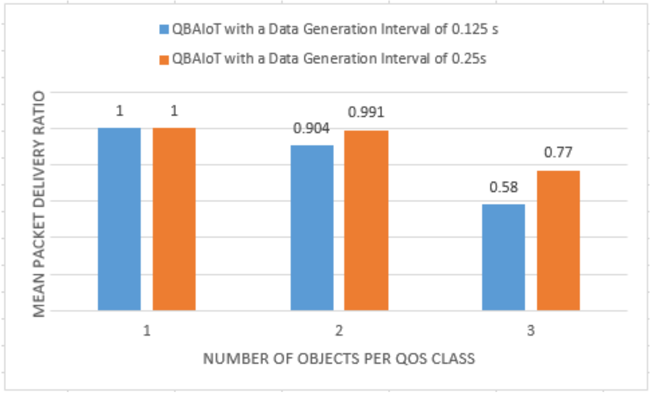


Figure 12. Mean Packet Delivery ration for different DGIs

VI. Conclusion

To ensure better user experience in the IoT environment, researchers try to optimize the delivered services while guaranteeing the QoS. Different access technologies could be used in the sensing layer of the IoT architecture. Several of these technologies are based on the IEEE 802.15.4 standard but the latter does not provide any QoS guarantee for the traffic generated by objects using this standard to access the IoT infrastructure. Therefore, we proposed the QBAIoT access method as an enhancement of the IEEE 802.15.4 slotted CSMA/CA mechanism in order to take into consideration QoS requirements of 4 different kinds of QoS traffic classes generated in the IoT environment. QBAIoT allows to respect the service level negotiated between the IoT-C and the IoT-SP during the establishment of the iSLA. In particular, QBAIoT QoS provision within the lower layer of the IoT architecture (Sensing Layer). We compared our proposed access method to the IEEE 802.15.4 standard and we showed that we obtain better results while using our QoS based access method to guarantee a reduced delay for Real Time traffic, as well as a greater PDR and effective data rate for all QoS classes with different DGIs.

As ongoing work, we aim to provide the IoT environment with a self-configuring capability allowing activating the minimum needed number of objects per QoS class in an autonomic manner while optimizing energy consumption. To do so, we will use the Fuzzy Logic theory in order to let the system choose autonomously the best objects in order to minimize the number of communications and so to expand the system lifetime by conserving the energy of non-activated objects.

Acknowledgment

This research was funded by the Conseil Régional de Bourgogne-Franche Comté through the “plan d’actions regional pour l’innovation (PARI)” and the European Union through the “PO FEDER-FSE Bourgogne 2014/2020 programs”.

References

[1] A. Khalil, N. Mbarek, and O. Togni, “QBAIoT: QoS Based Access for IoT Environments,” The Fourteenth Advanced International Conference on Telecommunications (AICT 2018), 2018, pp. 38–43, ISBN: 978-1-61208-650-7.

[2] A. Nordrum, “Popular IoT Forecast of 50 Billion Devices by 2020 Is Outdated”, IEEE Spectrum, August 2016.

[3] ITU-T Y.2066, “Next Generation Networks – Frameworks and functional architecture models”, 32 pages, 2014.

[4] IEEE Standard for Local and metropolitan area networks, Low-Rate Wireless Personal Area Networks, IEEE Computer Society, 311 pages, September 2011.

[5] P. Thubert, C. Bormann, L. Toutain, and R. Cragie, “Pv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing Header”, IETF RFC, 37 pages, April 2017.

[6] S. Nath, S. Aznabi, N. Islam, A. Faridi, and W. Qarony, “Investigation and Performance Analysis of Some Implemented Features of the ZigBee Protocol and IEEE 802.15.4 Mac Specification”, International Journal of Online Engineering (iJOE), vol.13, pp. 14-32, Nov.2017, ISSN: 1861-2121, doi:10.3991/ijoe.v13i01.5984

[7] ITU-T Y.2060, “Y.2060: Overview of the IoT, ITU-T”, 22 pages, 2012.

[8] ISO/IEC JTC 1, “IoT (IoT) Preliminary Report 2014”, 17 pages, 2015.

[9] J. Jimenez, H. Tschofenig and D. Thaler, “Report from the IoT (IoT) Semantic Interoperability (IOTSI) Workshop 2016”, Internet Draft, 17 pages, July 2018.

[10] O. Garcia-Morchon, S. Kumar and M. Sethi, “State of the Art and Challenges for the IoT Security”, Internet Draft, 47 pages, December 2018.

[11] P. Mell and T. Grance, “The NIST Definition of Cloud Computing”, NIST, 2 pages. version 15, July 2009.

[12] A. Banafa, “Definition of fog computing”, IBM, August 2014, https://www.ibm.com/blogs/cloud-computing/2014/08/fog-computing/, (Last access 17 March 2018).

[13] International Electrotechnical Commission, “IEC role in the IoT”, 20 pages, 2017.

[14] World Health Organization, "WHO and PATH partner to globalize digital health", September 2018, https://www.who.int/ehealth/events/WHO-PATH-partnership/en/, (Last Access 14 January 2019).

[15] Ericsson, "Ericsson and partners demonstrate battery life improvements in Massive IoT e-health wearable prototype", September 2018, https://www.ericsson.com/en/news/2018/8/connected-e-health-IoT, (Last Access 14 January 2019).

[16] Nokia, " Enabling the human possibilities of smart cities", https://networks.nokia.com/smart-city, (Last Access 14 January 2019).

[17] 4G Americas, "Cellular Technologies Enabling the IoT", 2015, http://www.5gamericas.org/files/6014/4683/4670/4G_Americas_Cellular_Technologies_Enabling_the_IoT_White_Paper_-_November_2015.pdf, (Last Access 14 January 2019).

[18] Lora Alliance, "A Technical Overview of LoRa® and LoRaWAN™", https://www.tuv.com/media/corporate/products_1/electronic_components_and_lasers/TUeV_Rheinland_Overview_LoRa_and_LoRaWANtmp.pdf (Last Access 14 January 2019).

[19] ITU-T E.800, "Definitions of terms related to quality of service", 30 pages, 2008.

[20] J.Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "IoT (IoT): A vision, architectural elements, and future directions", Future Generation Computer Systems, vol. 29, pp. 1645-1660, 2013, doi: 10.1016/j.future.2013.01.010

[21] R. Bhaddurgatte and V. Kumar, "Review: QoS Architecture and Implementations in IoT Environment", Research & Reviews: Journal of Engineering and Technology, ISSN: 2319-9873, pp. 6-12, 2015.

[22] B. Ray, "Benefits of Quality of Service (QoS) in LPWAN for IoT", LinkLabs, December 2016

[23] M. Serrano, "OpenIoT D.4.6 Quality of Service (QoS) for IoT services", OpenIoT Consortium, Project Number 287305, 51 pages, 2014.

[24] M. A. Nef, L. Perleps, S. Karagiorgou, and G. I. Stamoulis, "Enabling QoS in the IoT", The Fifth International Conference on Communication Theory, Reliability, and Quality of Service, May 2012.

[25] S. Ezdiani, I. S. Acharyya, S. Sivakumar, and A. Al-Anbuky "An IoT Environment for WSN Adaptive QoS", 2015 IEEE International Conference on Data Science and Data Intensive Systems (DSDIS 2015), , 2015, pp. 586-593, ISBN: 978-1-5090-0214-6, doi:10.1109/DSDIS.2015.28

[26] S. Sarode and J. Bakal, "A Slotted CSMA/CA of IEEE 802.15.4 Wireless Sensor Networks: A Priority Approach", International Journal of Computer Trends and Technology (IJCTT), vol. 44, pp. 33-38, Feb. 2017, ISSN: 2231-2803, doi: 10.14445/22312803/IJCTT-V44P106

[27] F. Xia, J. Li, R. Hao, X. Kong, and R. Gao, "Service Differentiated and Adaptive CSMA/CA over IEEE 802.15.4 for Cyber-Physical Systems", The Scientific World Journal, vol. 2013, Article ID 947808, 12 pages, 2013, doi:10.1155/2013/947808

[28] A. Khalil, N. Mbarek, and O. Togni, "Service Level Guarantee Framework for IoT environments", International Conference on IoT and Machine Learning (IML 2017), 2017, ISBN: 978-1-4503-5243-7, doi: 10.1145/3109761.3158393

[29] M. Kirsche, IEEE 802.15.4-Standalone, https://github.com/michaelkirsche/IEEE802154INET-Standalone (Last Access 17 March 2018)

Improving the Effectiveness of Web Application Vulnerability Scanning

Marc Rennhard

School of Engineering
Zurich University of Applied Sciences
Winterthur, Switzerland
Email: rema@zhaw.ch

Damiano Esposito, Lukas Ruf, Arno Wagner

Consecom AG
Zurich, Switzerland
Email: Damiano.Esposito,Lukas.Ruf,
Arno.Wagner@consecom.com

***Abstract*—Using web application vulnerability scanners is very appealing as they promise to detect vulnerabilities with minimal configuration effort. However, using them effectively in practice is often difficult. Two of the main reasons for this are limitations with respect to crawling capabilities and problems to perform authenticated scans. In this paper, we present JARVIS, which provides technical solutions that can be applied to a wide range of vulnerability scanners to overcome these limitations and to significantly improve their effectiveness. To evaluate JARVIS, we applied it to five freely available vulnerability scanners and tested the vulnerability detection performance in the context of seven deliberately insecure web applications. A first general evaluation showed that by using the scanners with JARVIS, the number of detected vulnerabilities can be increased by more than 100% on average compared to using the scanners without JARVIS. A significant fraction of the additionally detected vulnerabilities is security-critical, which means that JARVIS provides a true security benefit. A second, more detailed evaluation focusing on SQL injection and cross-site scripting vulnerabilities revealed that JARVIS improves the vulnerability detection performance of the scanners by 167% on average, without increasing the fraction of reported false positives. This demonstrates that JARVIS not only manages to greatly improve the vulnerability detection rate of these two highly security-critical types of vulnerabilities, but also that JARVIS is very usable in practice by keeping the false positives reasonably low. Finally, as the configuration effort to use JARVIS is small and as the configuration is scanner-independent, JARVIS also supports using multiple scanners in parallel in an efficient way. In an additional evaluation, we therefore analyzed the potential and limitations of using multiple scanners in parallel. This revealed that using multiple scanners in a reasonable way is indeed beneficial as it further increases the number of detected vulnerabilities without a significant negative impact on the reported false positives.**

Keywords–Web Application Security; Vulnerability Scanning; Vulnerability Detection Performance; Authenticated Scanning; Combining Multiple Scanners.

I. Introduction

This paper is an extended and revised version of our conference paper [1] that was published at ICIMP 2018 (the thirteenth International Conference on Internet Monitoring and Protection). Compared to the original version, this paper contains a much more elaborate evaluation to further demonstrate the effectiveness and usefulness of the presented approach.

Security testing is of great importance to achieve security and trustworthiness of software and systems. Security testing can be performed in different ways, ranging from completely manual methods (e.g., manual source code analysis), to semi-automated methods (e.g., analyzing a web application using an interceptor proxy), to completely automated ways (e.g., analyzing a web service using a vulnerability scanner).

Ideally, at least parts of security testing should be automated. One reason for this is that it increases the efficiency of a security test and frees resources for those parts of a security test that cannot be easily automated. This includes, e.g., access control tests, which cannot really be automated as a testing tool does not understand which users or roles are allowed to perform what functions. Another reason is that automating security tests enables performing continuous and reproducible security tests, which is getting more and more important in light of short software development cycles.

There are different options to perform automated security testing. The most popular approaches include static and dynamic code analysis and vulnerability scanning. Vulnerability scanners test a running system "from the outside" by sending specifically crafted data to the system and by analyzing the received response. Among vulnerability scanners, web application vulnerability scanners are most popular, as web applications are very prevalent, are often vulnerable, and are frequently attacked [2]. Note also that web applications are not only used to provide typical services such as information portals, e-shops or access to social networks, but they are also very prevalent to configure all kinds of devices attached to the Internet, which includes, e.g., switches, routers and devices in the Internet of Things (IoT). This further underlines the importance of web application security testing.

At first glance, using web application vulnerability scanners seems to be easy as they claim to uncover many vulnerabilities with little configuration effort – as a minimum, they only require the base URL of the application to test as an input. However, the effective application of web application vulnerability scanners in practice is far from trivial. The following list summarizes some of the limitations:

1) The detection capabilities of a scanner are directly dependent on its crawling performance: If a scanner cannot find a specific resource in a web application, it cannot test it and will not find vulnerabilities associated with this resource. Previous work shows that the crawling performance of different scanners varies significantly [3], [4].
2) To test areas of a web application that are only reachable after successful user authentication, the scanners must authenticate themselves during crawling and testing. While most scanners can be configured so they can perform logins, they typically do not support

all authentication methods used by different web applications. Also, scanners sometimes log out themselves (e.g., by following a logout link) during testing and sometimes have problems to detect whether an authenticated session has been invalidated. Overall, this makes authenticated scans unreliable or even impossible in some cases.

3) To cope with these limitations, scanners usually provide configuration options, which can increase the number of detected vulnerabilities [5]. This includes, e.g., specifying additional URLs that can be used by the crawler as entry points, manually crawling the application while using the scanner as a proxy so it can learn the URLs, and specifying an authenticated session ID that can be used by the scanner to reach access-protected areas of the application if the authentication method used by the web application is not supported. However, using these options complicate the usage of the scanners and still does not always deliver the desired results.
4) With respect to the number and types of the reported findings, different vulnerability scanners perform differently depending on the application under test [6]–[10]. Therefore, when testing a specific web application, it is reasonable to use multiple scanners in parallel and combine their findings. However, the limitations described above make this cumbersome and difficult, as each scanner has to be configured and optimized differently.

In this paper, we present JARVIS, which provides technical solutions to overcome limitations 1 and 2 in the list above. Using JARVIS requires only minimal configuration, which overcomes limitation 3. And finally, JARVIS and its usage are independent of specific vulnerability scanners and can be applied to a wide range of scanners available today, which overcomes limitation 4 and which provides an important basis to use multiple scanners in parallel in an efficient way.

To demonstrate the effectiveness and usefulness of JARVIS, to quantify how much it can improve the vulnerability detection performance of scanners, and to learn more about the potential and limitations of combining multiple scanners, this paper also includes a detailed evaluation. In this evaluation, JARVIS was applied to the five freely available scanners listed to Table I.

TABLE I. ANALYZED WEB APPLICATION VULNERABILITY SCANNERS

Scanner	*Version/Commit*	*URL*
Arachni	1.5-0.5.11	http://www.arachni-scanner.com
OWASP ZAP	2.5.0	https://www.owasp.org/index.php/ OWASP_Zed_Attack_Proxy_Project
Skipfish	2.10b	https://code.google.com/archive/p/ skipfish/
Wapiti	r365	http://wapiti.sourceforge.net
w3af	cb8e91af9	https://github.com/andresriancho/w3af

The choice for using freely available scanners was mainly driven by the desire to evaluate the performance of using multiple scanners in parallel. This is a much more realistic scenario with freely available scanners as commercial ones often have a hefty price tag. Also, several previous works concluded that freely available scanners do not perform worse than commercial scanners [3], [4], [11], [12]. Arguments for using the scanners in Table I instead of using others include our previous experience with these scanners, that these scanners are among the most popular used scanners in practice, and that they perform well in general according to [4], [11], [12].

The main contributions of this paper are the following:

- Technical solutions to improve the crawling coverage and the reliability of authenticated scans of web application vulnerability scanners. In contrast to previous work (see Section II), our solutions cover both aspects, can easily be applied to a wide range of scanners available today, and require only minimal, scanner-independent configuration.
- A general evaluation that shows that by using these technical solutions, the vulnerability detection performance of the scanners included in the evaluation can be improved by more than 100% on average. Many of the additionally reported vulnerabilities are security-critical, which means that JARVIS provides a true security benefit.
- A more detailed evaluation focusing on SQL injection and cross-site scripting vulnerabilities that demonstrates that the vulnerability detection performance of the scanners with respect to these two types of highly relevant vulnerabilities can be increased by 167% on average, without increasing the fraction of reported false positives.
- A final evaluation that shows that using multiple scanners in a reasonable way is beneficial as it further increases the number of detected vulnerabilities without a significant negative impact on the reported false positives.

The remainder of this paper is organized as follows: Section II covers relevant related work. Section III describes the technical solutions to overcome the limitations of today's web application vulnerability scanners. Section IV contains the general evaluation results and Section V provides a more detailed evaluation focusing on SQL injection and cross-site scripting vulnerabilities. The final part of the evaluation is provided in Section VI, where the benefits and limitations of using multiple scanners in parallel are analyzed. Section VII concludes this work.

II. RELATED WORK

Several work has been published on the crawling coverage and detection performance of web application vulnerability scanners. In [3], more than ten scanners were compared, with the main results that good crawling coverage is paramount to detect many vulnerabilities and that freely available scanners perform as well as commercial ones. The same is confirmed in [4], which covers more than 50 free and commercial scanners. The works by Chen [11], which covers about 20 scanners and which is updated regularly, and by El Idrissi et al. [12], which includes 11 scanners in its evaluation, also result in the conclusion that free scanners perform as well as commercial ones. In [5], Suto concludes that when carefully training or configuring a scanner, detection performance is improved, but this also significantly increases the complexity and time effort needed to use a scanner. Furthermore, Bau et al. demonstrate that the eight scanners they used in their analysis have different strengths, i.e., they find different vulnerabilities [6]. The same is confirmed by Vega et al. [7], which in addition compare the vulnerabilities detected by the four scanners in their evaluation with alerts reported by the intrusion detection system (IDS) Snort [13]. Qasaimeh et al. conclude that the five scanners used in their evaluation not only perform differently with respect to the number of findings detected, but also with respect to the number of false positives [8]. Smaller studies by using two and three scanners were done in [9] and [10], respectively, which confirm that different scanners have different strengths with respect to detection capabilities.

Other work specifically aimed at improving the coverage of vulnerability scanning. In [14], it is demonstrated that by considering the state changes of a web application when it processes requests, crawling and therefore scanning performance can be improved. In [15], van Deursen et al. present a Selenium WebDriver-based crawler called Crawljax, which improves crawling of Ajax-based web applications. The same is achieved by Pellegrino et al. by dynamically analyzing JavaScript code in web pages [16]. In [17], Zulla discusses methods to improve web vulnerability scanning in general, including approaches to automatically detect login forms on web pages.

Our work presented in this paper builds upon this previous work, in particular on the observations that freely available scanners perform similarly as commercial ones, that different scanners have different strengths with respect to detection capabilities, and that good crawling coverage is paramount to detect many vulnerabilities. Besides this, however, our work goes significantly beyond existing work. First of all, the presented solution – JARVIS – not only addresses crawling coverage but also the reliability of authenticated scans, which has a significant impact on the number of vulnerabilities that can be detected. In addition, JARVIS is scanner-independent, which means it can easily be applied to most vulnerability scanners available today. Furthermore, we provide a detailed evaluation using several scanners and several test applications that truly demonstrates the benefits and practicability of our technical solutions. And finally, to our knowledge, our work is the first one to quantitatively evaluate the benefits and limitations when combining multiple scanners.

III. Technical Solutions to Improve Web Application Vulnerability Scanning

One way to improve the vulnerability detection performance of web application vulnerability scanners is to directly adapt one or more scanners that are available today. However, the main disadvantage of this approach is that this would only benefit one or a small set of scanners and would be restricted to scanners that are provided as open source software. Therefore, a proxy-based approach was chosen. The advantages of this approach are that it is independent of any specific scanner, that it does not require adaptation of any scanner, and that it can be used with many scanners that are available today and most likely also with scanners that will appear in the future. The basic idea of this proxy-based approach is illustrated in Figure 1.

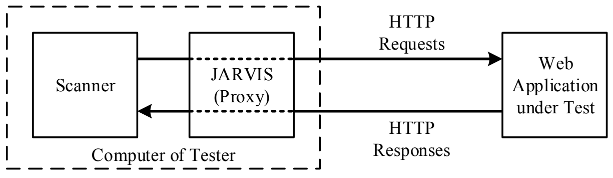


Figure 1. Proxy-based Approach of JARVIS.

A proxy-based approach means that JARVIS, which provides the technical solutions to overcome the limitations of web application vulnerability scanners, acts as a proxy between the scanner and the web application under test. This gives JARVIS access to all HTTP requests and responses exchanged between the scanner and the web application, which enables JARVIS to control the entire crawling and scanning process and to adapt requests or responses as needed. This proxy-based approach is possible because most scanners are proxy-aware, i.e., they support configuring a proxy through which communication with the web application takes place. Note that JARVIS can basically be located on any reachable host, but the typical scenario is using JARVIS on the same computer as the web application vulnerability scanner (e.g., on the computer of the tester).

As a basis for JARVIS, the community edition version 1.7.19 of Burp Suite [18] is used. Burp Suite is a tool that is intended to assist a tester during web application security testing. It is usually used as a proxy between the browser of the tester and the web application under test and supports recording, intercepting, analyzing, modifying and replaying HTTP requests and responses. Therefore, Burp Suite already provides many basic functions that are required to implement JARVIS. In addition, Burp Suite provides an application programming interface (API) so it can be extended and JARVIS makes use of this API.

JARVIS consist of two main components. The first is described in Section III-A and aims at improving the test coverage of scanners. This component should especially help scanners that have a poor crawling performance. The second component, described in Section III-B, aims at improving the reliability of authenticated scans and should assist scanners that have limitations in this area. Finally, Section III-C gives a configuration example when using JARVIS to demonstrate that the configuration effort is small.

A. Improving Test Coverage

Improving test coverage could be done by replacing the existing crawler components of the scanners with a better one (see, e.g., [14]–[16]). While this may be helpful for some scanners, it may actually be harmful for others, in particular if the integrated crawler works well. Therefore, an approach was chosen that does not replace but that assists the crawling components that are integrated in the different scanners. The idea is to supplement the crawlers with additional URLs (beyond the base URL) of the web application under test. These additional URLs are named *seeds* as they are used to seed the crawler components of the scanners. Intuitively, this should significantly improve crawling coverage, in particular if the integrated crawler is not very effective. To get the additional URLs of a web application, two different approaches are used: endpoint extraction from the source code of web applications and using the detected URLs of the best available crawler(s).

Endpoint extraction means searching the source code (including configuration files) of the web application under test for URLs and parameters. The important benefits of this approach are that it can detect URLs that are hard to find by any crawler and that it can uncover hidden parameters of requests (e.g., debugging parameters). To extract the endpoints, ThreadFix endpoint CLI [19] was used, which supports many common web application frameworks (e.g., JSP, Ruby on Rails, Spring MVC, Struts, .NET MVC and ASP.NET Web Forms). In addition, further potential endpoints are constructed by appending all directories and files under the root directory of the source code to the base URL that is used by the web application under test. This is particularly effective when scanning web applications based on PHP.

Obviously, endpoint extraction is only possible if the source code of the application under test is available. If that is not the case, the second approach comes into play. The idea here is to use the best available crawler(s) to gather additional URLs. As will be shown later, the scanner Arachni provides good crawling performance in general, so Arachni is a good starting point as a tool for this task. Of course, it is also possible to combine both approaches to determine the seeds: extract the endpoints from the source code (if available) *and* get URLs with the best available crawler(s).

Once the seeds have been derived, they must be injected into the crawler component of the scanners. To do this, most scanners provide a configuration option. However, this approach has its limitations as such an option is not always available and usually only supports GET requests but no POST requests. Therefore, the seeds are injected by JARVIS. To do this, four different approaches were implemented based on *robots.txt*, *sitemap.xml*, a landing page, and the index page.

Using *robots.txt* and *sitemap.xml* is straightforward. These files are intended to provide search engine crawlers with information about the target web site and are also evaluated by most crawler components of scanners. When the crawler component of a scanner requests such a file, JARVIS supplements the original file received from the web application with the seeds (or generates a new file with the seeds in case the web application does not contain the file at all). Both approaches work well but are limited to GET request.

The other two approaches are more powerful as they also support POST request. The landing page-based approach places all seeds as links or forms into a separate web page (named *landing.page*) and the scanner is configured to use this page as the base URL of the web application under test (e.g., http://www.example.site/landing.page instead of http://www.example.site). When the crawler requests the page, JARVIS delivers the landing page, from which the crawler learns all the seeds and uses them during the remainder of the crawling process. One limitation of this approach is that the altered base URL is sometimes interpreted as a directory by the crawler component of the scanners, which means the crawler does not request the landing page itself but tries to fetch resources below it. This is where the fourth approach comes into play. The index page-based approach injects seeds directly into the first page received from the web application (e.g., just before the </body> tag of the page *index.html*). Overall, these four approaches made it possible to successfully seed all scanners in Table I when used to test the web applications in the test set (see Section IV-A).

As an example, the effectiveness of the landing page-based approach is demonstrated. To do this, WIVET version 4 [20] is used, which is a benchmarking project to assess crawling coverage. Table II shows the crawling coverage that can be achieved with OWASP ZAP (in headless mode) and Wapiti when they are seeded with the crawling results of Arachni via a landing page.

TABLE II. CRAWLING COVERAGE

Scanner	*Raw Coverage*	*Coverage when Seeded with the Crawling Results of Arachni*
Arachni	92.86%	
OWASP ZAP	14.29%	96.43%
Wapiti	48.21%	96.43%

Table II shows that the raw crawling coverage of Arachni is already very good (92.86%), while Wapiti only finds about half of all resources and OWASP ZAP only a small fraction. By seeding OWASP ZAP and Wapiti with the crawling results of Arachni, their coverage can be improved drastically to 96.43%. This demonstrates that seeding via a landing page indeed works very well.

B. Improving Authenticated Scans

Performing authenticated scans in a reliable way is challenging for multiple reasons. This includes coping with various authentication methods, prevention of logouts during the scans, and performing re-authentication when this is needed (e.g., when a web application with integrated protection mechanisms invalidates the authenticated session when being scanned) to name a few. It is therefore not surprising that many scanners have difficulties to perform authenticated scans reliably.

To deal with these challenges, several modules were implemented in JARVIS. The first one serves to handle various authentication methods, including modern methods based on HTTP headers (e.g., OAuth 2.0). The module provides a wizard to configure authentication requests, can submit the corresponding requests, stores the authenticated cookies received from the web applications, and injects them into subsequent requests from the scanner to make sure the requests are interpreted as authenticated requests by the web application. The main advantages of this module are that it enables authenticated scans even if a scanner does not support the authentication method and that it provides a consistent way to configure authentication independent of a particular scanner.

Furthermore, a logout prevention module was implemented to make sure a scanner is not doing a logout by following links or performing actions which most likely invalidate the current session (e.g., change password or logout links). This is configured by specifying a set of corresponding URLs that should be avoided during the scan. When the proxy detects such a request, it blocks the request and generates a response with HTTP status code 200 and an empty message body. In addition, a flexible re-authentication module was developed. Re-authentication is triggered based on matches of configurable literal strings or regular expressions with HTTP response headers (e.g., the *location* header in a redirection response) or with the message body of an HTTP response (e.g., the occurrence of a keyword such as *login*).

C. Configuration Example

To give an impression of the configuration effort needed when using JARVIS, Table III lists the parameters that must be configured when scanning the test application BodgeIt (see Section IV-A). In this example, the seeds are extracted from the source code.

TABLE III. Example Configuration when Scanning BodgeIt

Parameter	*Value(s)*
Base URL	http://bodgeit/
Source code	~/bodgeit/
Authentication mode	POST
Authentication URL	http://bodgeit/login.jsp
Authentication parameters	username=test@test.test password=password
Out of scope	http://bodgeit/password.jsp http://bodgeit/register.jsp http://bodgeit/logout.jsp
Re-auth. search scope	HTTP response body
Re-auth. keywords	Login, Guest, user
Re-auth. keyword interpretation	Literal string(s)
Re-auth. case-sensitive	True
Re-auth. match indicates	Invalid session
Seeding approach(es)	Landing page, robots.txt, sitemap.xml

The entries in Table III are self-explanatory and show that the configuration effort is rather small. In particular, the configuration is independent of the actual scanner, which implies that when using multiple scanners in parallel (see Section VI), this configuration must only be done once and not once per scanner.

IV. General Evaluation

This section starts with a description of the evaluation setup. Then, it is analyzed how many vulnerabilities are reported when the scanners are used with and without the technical improvements described in Section III. Next, these vulnerabilities are analyzed in more detail to check how many unique vulnerabilities are detected and how severe they are. Finally, it is analyzed whether all vulnerabilities that can be detected by the scanners without using JARVIS are always also detected when JARVIS is used.

A. Evaluation Setup

Table IV lists the web applications that were used to evaluate the scanners (Cyclone Transfers and WackoPicko do not use explicit versioning).

TABLE IV. Web Applications used for the Evaluation

Application	*Version*	*URL*
BodgeIt	1.4.0	https://github.com/psiinon/bodgeit
Cyclone Transfers	–	https://github.com/thedeadrobots/bwa_cyclone_transfers
InsecureWebApp	1.0	https://www.owasp.org/index.php/Category:OWASP_Insecure_Web_App_Project
Juice Shop	2.17.0	https://github.com/bkimminich/juice-shop
NodeGoat	1.1	https://github.com/OWASP/NodeGoat
Peruggia	1.2	https://sourceforge.net/projects/peruggia/
WackoPicko	–	https://github.com/adamdoupe/WackoPicko

All these applications are deliberately insecure and well suited for security training and to test vulnerability scanners. The main reason why the applications in Table IV were chosen is because they cover various technologies, including Java, PHP, Node.js and Ruby on Rails.

The evaluation uses four different configurations that are identified as *-/-*, *S/-*, *-/A* and *S/A*. Basically, *S* indicates that seeding is used and *A* indicates that authenticated scans are used. The four configurations are described in more detail in Table V.

TABLE V. Configurations used during the Evaluation

Config.	JARVIS is Used	*The Scans are Executed...*
-/-	No	...without seeding and non-authenticated (i.e., using the basic configuration of the scanners by setting only the base URL)
S/-	Yes	...with seeding but non-authenticated (i.e., using the technical solution described in Section III-A)
-/A	Yes	...authenticated but without seeding (i.e., using the technical solution described in Section III-B)
S/A	Yes	...with seeding and authenticated (i.e., using both technical solutions described in Sections III-A and III-B)

As the source code of all the test applications is available, the endpoint extraction approach described in Section III-A is used for seeding in configurations *S/-* and *S/A*.

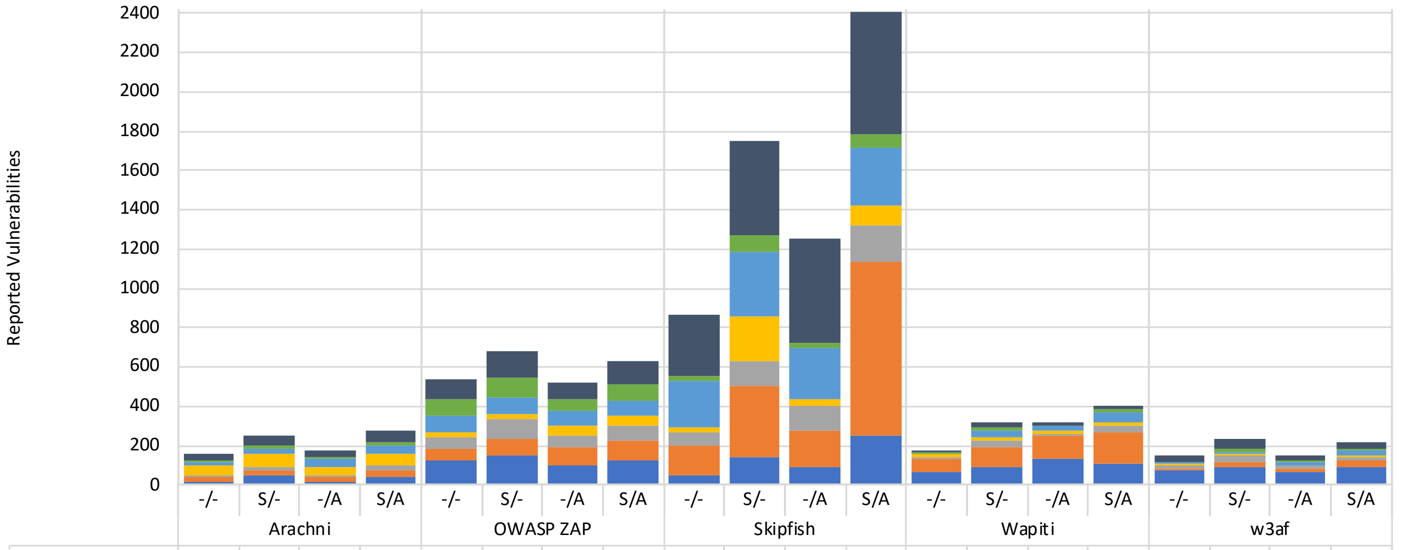
The test applications were run in a virtual environment that was reset to its initial state before each test run to make sure that every run is done under the same conditions and is not influenced by any of the other scans.

B. Total Number of Reported Vulnerabilities

The first evaluation analyzes the total number of vulnerabilities that are reported by the scanners when using the four different configurations described in Table V. Figure 2 illustrates the evaluation results. The height of the bars represents the number of vulnerabilities reported over all seven test applications and the different colors of the bars represent the number of reported vulnerabilities per test application. The table in the lower part of the figure also contains the number of vulnerabilities reported per test application.

The first observation when looking at Figure 2 is that some scanners identify many more vulnerabilities than others. For example, Skipfish reports about ten times as many findings as Arachni or w3af. However, this does not mean that Skipfish is the best scanner, because Figure 2 depicts the "raw number of vulnerabilities" reported by the scanners and does not consider whether the vulnerabilities include false positives or duplicate findings, or how severe the findings are. For instance, as will be seen in Section IV-C, about 75% of the vulnerabilities reported by Skipfish are rated as *info* or *low* (meaning they have only little security impact in practice), while the other scanners report a much smaller fraction of such findings.

More importantly, Figure 2 allows to do a first assessment about the impact of using JARVIS. By comparing the total number reported vulnerabilities in configuration *S/-* with the one in configuration *-/-*, it can easily be seen that the technical solution to improve test coverage works well with all scanners: With every scanner, the number is always higher when seeding is used. For instance, when adding up the reported vulnerabilities of all test applications, Arachni reports 254 findings in configuration *S/-* compared to only 162 in configuration *-/-*. The same behavior can also be observed with the other four scanners. In addition, the benefit of seeding is not only obvious when looking at the combined results of all test applications, but also when looking at individual test applications: The number of vulnerabilities reported when seeding is used



	Arachni				OWASP ZAP				Skipfish				Wapiti				w3af			
	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A
⬛ WackoPicko	39	50	36	54	101	137	79	117	313	484	526	617	14	21	14	20	28	55	26	34
🟩 Peruggia	4	20	4	20	79	98	62	78	25	81	27	69	1	17	2	18	2	14	2	7
🟦 NodeGoat	22	24	44	42	83	84	79	80	235	327	262	293	3	32	23	49	9	14	19	25
🟨 Juice Shop	47	64	47	60	29	29	49	49	20	229	33	104	19	19	19	19	4	7	4	8
⬜ InsecureWebApp	11	24	7	26	59	102	58	75	66	128	130	183	9	36	8	31	19	30	15	22
🟧 Cyclone Transfers	20	23	28	32	58	85	90	103	154	359	183	886	62	102	119	158	11	25	12	31
🟦 BodgeIt	19	49	12	39	126	149	102	125	51	145	90	252	70	89	134	110	74	94	70	92

Figure 2. Total Number of Reported Vulnerabilities per Scanner and Test Application.

is nearly always higher than without seeding. For instance, Arachni reports 64 vulnerabilities in Juice Shop in configuration *S/-* compared to 47 in configuration *-/-*. Among all 35 combinations of the five scanners and seven test applications, there is an improvement in 33 cases and overall, there are just two exceptions where the number of reported vulnerabilities is not increased and remains unchanged (OWASP ZAP and Wapiti when scanning Juice Shop).

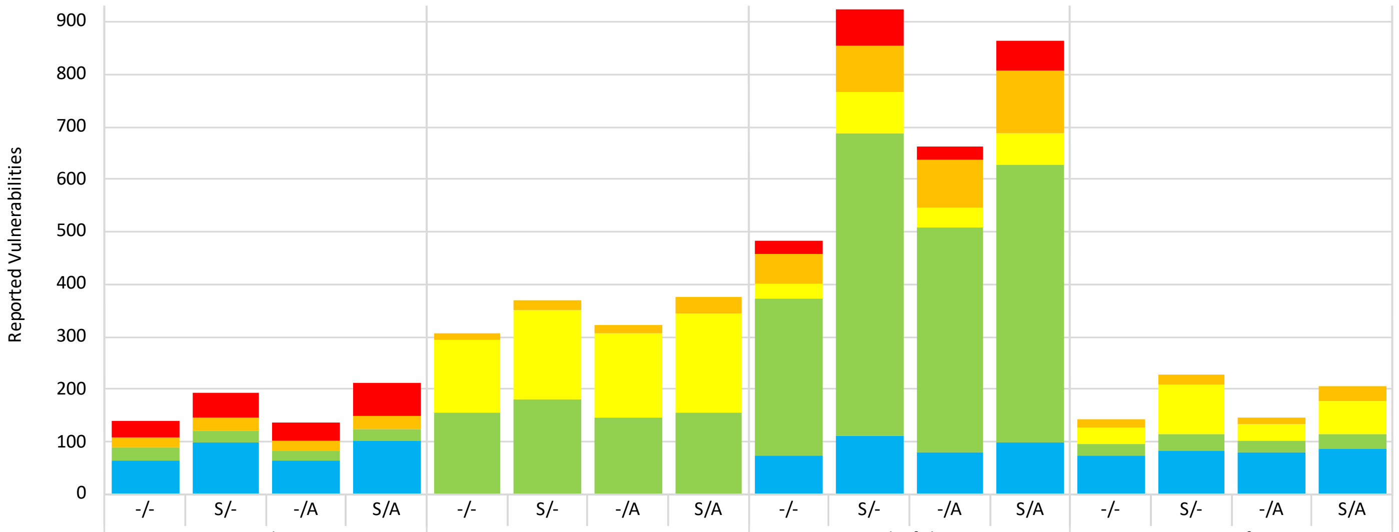
The benefit of the technical solution to improve authenticated scans is less obvious from the results in Figure 2. Using again Arachni as an example, the 178 vulnerabilities reported over all test applications in configuration *-/A* are only a small improvement compared to the 162 vulnerabilities reported in configuration *-/-*. With Wapiti, the results are much better with an improvement from 178 to 319 reported vulnerabilities. But in the case of OWASP ZAP, the numbers even get slightly lower when authenticated scans are used, from 535 to 519. When looking at individual test applications, the results vary as well. For instance, when scanning Cyclone Transfer, Wapiti reports 62 findings in configuration *-/-* and 119 findings in configuration *-/A*, which is significant improvement. But when scanning Peruggia with OWASP ZAP, 79 findings are reported in configuration *-/-*, which drops to 62 in configuration *-/A*. In general, more analysis is required to assess the impact of the technical solution to improve authenticated scans, which will follow in Sections IV-C and IV-D.

Furthermore, Figure 2 provides insights into the benefit of using both technical solutions at the same time (configuration *S/A*). Intuitively, one would expect this configuration to deliver clearly the highest number of vulnerabilities with all scanners, but this is not the case. With OWASP ZAP and w3af, the number of reported vulnerabilities over all test applications is slightly lower than in configuration *S/-*, with Arachni it is almost the same as in configuration *S/-*, and only Skipfish and Wapiti report clearly the highest number of vulnerabilities in configuration *S/A*. So just like when using only the solution to improve authenticates scans (see above), this result is currently non-conclusive and more analysis is required.

Note that to make sure that authenticated scans were carried out reliably, the involved requests and responses were analyzed after each scan. This showed that it was indeed possible to maintain authentication during all these scans, which confirms that the technical solution to improve authenticated scans is sound and works well in practice.

C. Reported Unique Vulnerabilities and Severity of Vulnerabilities

The previous evaluation in Section IV-B demonstrates that when considering just the raw number of reported vulnerabilities, JARVIS works well, in particular with respect to the technical solution to improve test coverage. However, it is not clear whether there is a true benefit in practice because it may be that the additionally found vulnerabilities are mainly duplicates of vulnerabilities that are already found in the basic configuration *-/-*, or are mainly non-critical issues. For instance, it could be that the increased number of reported vulnerabilities is mainly because the scanners report a higher number of issues related to missing HTTP response headers (e.g., missing X-Frame-Options headers), which are sometimes reported once for every requested URL (which implies many duplicate findings) and which are usually not very security-relevant.



	Arachni				OWASP ZAP				Skipfish				w3af			
	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A
🟥 Critical	33	47	36	65	0	0	0	0	23	70	25	58	0	0	0	0
🟧 High	19	27	16	25	13	20	16	30	58	88	92	118	14	19	11	27
🟨 Medium	0	0	0	0	138	169	162	188	27	78	38	60	33	94	31	63
🟩 Low	25	21	20	21	155	181	146	157	299	575	427	529	21	30	24	29
🟦 Info	64	99	65	103	0	0	0	0	75	113	80	99	75	84	79	87

Figure 3. Reported Unique Vulnerabilities per Scanner over all Test Applications, according to Severity.

To analyze this in more detail, the reports of the scanners were processed with ThreadFix [21]. ThreadFix provides the functionality to normalize reports of different scanners, to eliminate duplicate findings, and to compare the results of different scanners or different runs by the same scanner. Eliminating duplicate findings means that if a specific vulnerability such as a missing HTTP response header is reported, e.g., ten times by a scanner, then it will be only included as one vulnerability in the output generated by ThreadFix. In addition, ThreadFix maps the severity levels of vulnerabilities reported by different scanners to five standard severity levels: *critical*, *high*, *medium*, *low* and *info*. The results of this processing with ThreadFix are illustrated in Figure 3. For each scanner, it shows the number of reported unique vulnerabilities (i.e., without duplicates) over all test applications when using the four different configurations. In addition, the number of vulnerabilities is separated according to the standard severity levels.

Note that Figure 3 and also the remainder of Section IV do not include the scanner Wapiti, because at the time of writing, Wapiti was not supported by ThreadFix. In addition, not every scanner uses all five standard severity levels from *critical* to *info*, as this depends on the scanner-specific severity mappings done by ThreadFix. Specifically, ThreadFix maps the severity levels of Arachni to the standard severity levels *critical*, *high*, *low* and *info* (without using *medium*), in the case of OWASP ZAP, only three levels *high*, *medium* and *low* are used (so *critical* and *info* are not used), and in the case of w3af, level *critical* is not used. The only scanner in Figure 3 that uses all five standard levels is Skipfish.

When comparing Figure 3 with Figure 2, one can see that the absolute heights of the bars, i.e., the total number of reported vulnerabilities, are lower in Figure 3. For instance, in the case of OWASP ZAP, the total number of reported vulnerabilities went down from 535 to 306 in the basic configuration *-/-* and from 684 to 370 in configuration *S/-*. This is not surprising as duplicate findings (in this case 229 and 314, respectively) were eliminated by ThreadFix. As all the bars got lower, this also shows that all scanners tend towards reporting duplicate vulnerabilities, no matter whether JARVIS is used or not. However, the more important result that can be seen from Figure 3 is that for each scanner, the relative heights of the bars when using different configurations are still very similar as in Figure 2, which means that many of the additional vulnerabilities that are reported when JARVIS is used are indeed new vulnerabilities, and not just duplicates of vulnerabilities detected in the basic configuration *-/-*. As a side note, Figure 3 also puts the high number of vulnerabilities reported by Skipfish into perspective, as a significant portion of them have severity *low* and *info* and which are therefore typically not very security-critical.

To quantify the benefit of JARVIS in more detail, Table VI contains the numbers of reported unique vulnerabilities and the relative improvements when using JARVIS.

First, the improvement that can be achieved with the technical solution to increase test coverage is analyzed. For instance, Table VI shows that when using Arachni, 141 unique vulnerabilities are reported in configuration *-/-*, which is increased to 194 vulnerabilities in configuration *S/-*. This corresponds to an improvement of reported vulnerabilities of 38%. With the other three scanners, the improvements are 21%, 92% and 59%. In the last row of Table VI, the reported vulnerabilities of the four scanners are added up. This shows that on average, the number of reported unique vulnerabilities is increased by 60% when moving from configuration *-/-* to *S/-*, which demonstrates that the technical solution to increase

TABLE VI. Reported Unique Vulnerabilities per Scanner, and Improvement by using JARVIS

Scanner	*Config.*	*Reported Unique Vulnerabilities*	*Improvement by using JARVIS*
Arachni	-/-	141	
	S/-	194	38%
	-/A	137	-3%
	S/A	214	52%
OWASP ZAP	-/-	306	
	S/-	370	21%
	-/A	324	6%
	S/A	375	23%
Skipfish	-/-	482	
	S/-	924	92%
	-/A	662	37%
	S/A	864	79%
w3af	-/-	143	
	S/-	227	59%
	-/A	145	1%
	S/A	206	42%
All four Scanners	-/-	1'072	
	S/-	1'715	**60%**
	-/A	1'268	**18%**
	S/A	1'659	**55%**

test coverage works very well.

Next, the improvement of the technical solution to improve authenticated scans is analyzed. As already seen in Section IV-B, the improvement is much smaller. For instance, Table VI shows that when using Arachni and when using configuration *-/A* instead of configuration *-/-*, the number of reported unique vulnerabilities actually goes down, from 141 to 137, which is a reduction of 3%. With the other scanners, the improvements are 6%, 37% and 1%, and adding up the reported vulnerabilities of the four scanners shows that on average, the number of reported unique vulnerabilities is improved by 18% when moving from configuration *-/-* to *-/A*. It therefore can be concluded that the solution to improve authenticated scans results in a significantly smaller improvement with respect to the absolute number of reported unique vulnerabilities than the solution to increase test coverage, which confirms the observation made in Section IV-B.

Finally, the combined effect of using both technical solutions is analyzed, i.e., configuration *S/A*. For the four scanners, this results in improvements between 23% and 79% and on average, an improvement of 55% can be achieved compared to configuration *-/-*. As these numbers are quite similar as the numbers that can be achieved in configuration *S/-*, i.e., when using only the technical solution to increase test coverage, and as the improvement that can be achieved in configuration *-/A* (see above) is relatively small, this further underlines that the effect of the technical solution to improve authenticated scans only has a relatively small effect on the absolute number of reported unique vulnerabilities.

Another important result that can be seen by looking at Figure 3 is that the increased number of vulnerabilities when using JARVIS is not just because several additional non-security-critical issues were detected (i.e., severity levels *low* and *info*). Instead, for each of the four scanners in Figure 3, the distribution of the different severity levels appears to be more or less constant, independent of the configuration that is used. To quantify this in more detail, Table VII contains the numbers of reported unique vulnerabilities and the absolute and relative number of security-critical vulnerabilities among them. For simplicity, we assume that severity levels *critical*, *high* and *medium* are considered security-critical, while levels *low* and *info* are considered non-security-critical.

TABLE VII. Reported Unique Vulnerabilities per Scanner and Configuration, and Fraction of Security-Critical Vulnerabilities

Scanner	*Config.*	*Reported Unique Vulnerabilities*	*Number of Security-critical Vulnerabilities*	*Fraction of Security-critical Vulnerabilities*
Arachni	-/-	141	52	37%
	S/-	194	74	38%
	-/A	137	52	38%
	S/A	214	90	42%
OWASP ZAP	-/-	306	151	49%
	S/-	370	189	51%
	-/A	324	178	55%
	S/A	375	218	58%
Skipfish	-/-	482	108	22%
	S/-	924	236	26%
	-/A	662	155	23%
	S/A	864	236	27%
w3af	-/-	143	47	33%
	S/-	227	113	50%
	-/A	145	42	29%
	S/A	206	90	44%
All four Scanners	-/-	1'072	358	**33%**
	S/-	1'715	612	**36%**
	-/A	1'268	427	**34%**
	S/A	1'659	634	**38%**

To explain Table VII, the numbers of scanner Arachni are discussed in detail. For instance, in configuration *-/-*, Arachni reports 141 unique vulnerabilities, 52 of them are security-critical (i.e., severity levels *critical*, *high* or *medium*). This corresponds to a fraction of 37%. In configuration *S/-*, 74 of the 194 reported vulnerabilities are security-critical, which corresponds to 38%. In the other two configurations *-/A* and *S/A*, these fractions are 38% and 42%, respectively. This shows that in the case of Arachni, the fraction of security-critical vulnerabilities is approximately the same for all four configurations. The same can be observed for the other scanners in Table VII, with the exception of w3af, where the fractions vary a bit more. The last row in the table contains the added up numbers of all four scanners, which shows a fraction of 33% security-critical vulnerabilities in configuration *-/-* and slightly higher fractions of 36%, 34% and 38% in the other three configurations, i.e., when using JARVIS. This demonstrates that on average, JARVIS not only increases the number of reported unique vulnerabilities, but that many of the additionally reported vulnerabilities are security-critical, which means that JARVIS provides a true security benefit.

To summarize this subsection, the following can be concluded:

- The technical solution to increase test coverage significantly increases the absolute number of reported unique vulnerabilities. On average, the number of reported vulnerabilities is improved by 60% when moving from configuration *-/-* to *S/-*.
- The technical solution to improve authenticated scans only has a small positive impact on the absolute number of reported unique vulnerabilities. On average, the number of reported vulnerabilities is improved by 18% when moving from configuration *-/-* to *-/A*.
- Using both technical solutions at the same time also significantly increases the absolute number of reported unique vulnerabilities. On average, the number of

reported vulnerabilities is improved by 55% when moving from configuration *-/-* to *S/A*. As this number is similar to what is achieved in configuration *S/-*, i.e., when using only the technical solution to increase test coverage, this further underlines that the effect of the technical solution to improve authenticated scans only has a relatively small effect on the absolute number of reported unique vulnerabilities.

- JARVIS slightly improves the fraction of security-critical vulnerabilities among all reported vulnerabilities. This means the practical benefit of JARVIS is even slightly better than the figures above. So, for instance, when the number of reported vulnerabilities can be improved by 60% when moving from configuration *-/-* to *S/-* (see above), then the improvement of security-critical vulnerabilities is even a bit higher than 60%.

For completeness, Figure 4 shows the number of unique vulnerabilities reported per scanner and test application when using the four different configurations, again separated according to the severity levels. Without going into the details, Figure 4 confirms that the conclusions of this subsection are also valid when considering the test applications individually: Using JARVIS results in a higher number of detected unique vulnerabilities and the distribution of the different severity levels per scanner and test application is more or less constant, independent of the configuration that is used.

D. Re-Detection of Vulnerabilities in Advanced Configurations

Intuitively, additionally seeding a scanner and/or performing authenticated scans (i.e., using configurations *S/-*, *-/A* and *S/A*) should always also report all vulnerabilities that are detected when scanning without additional seeding and without using authentication (i.e., in configuration *-/-*). However, this is not the case. To demonstrate this, Figure 5 illustrates how many of the vulnerabilities reported in the basic configuration are also found when scanning in the other three configurations. Just like in Section IV-C, this analysis is also based on the vulnerabilities after they have been processed with ThreadFix, which means that the scanner Wapiti is again not included and which implies that the heights of the bars (i.e., the total number of reported unique vulnerabilities) are exactly the same as in Figure 3.

Once more, the results of scanner Arachni are used to explain Figure 5 in details. The leftmost bar shows that in configuration *-/-*, Arachni reports 141 unique vulnerabilities. When using configuration *S/-*, then 194 findings are reported in total. Of these 194 findings, 128 are "new" findings compared to configuration *-/-* (indicated by the green part of the bar), and 66 are "old" findings compared to configuration *-/-* (indicated by the gray part of the bar), i.e., findings that were already detected in configuration *-/-*. This means that only 66 of the 141 vulnerabilities reported in configuration *-/-* are detected again in configuration *S/-* while 75 of the 141 vulnerabilities are missing, i.e., are not detected in configuration *S/-*. The same can be observed with all scanners and with all configurations: Whenever configurations *S/-*, *-/A* or *S/A* are used, a significant portion of the vulnerabilities detected in the basic configuration *-/-* are no longer detected. This means that in general, using JARVIS delivers a significant number of new findings, but also misses several of the findings that are reported when JARVIS is not used.

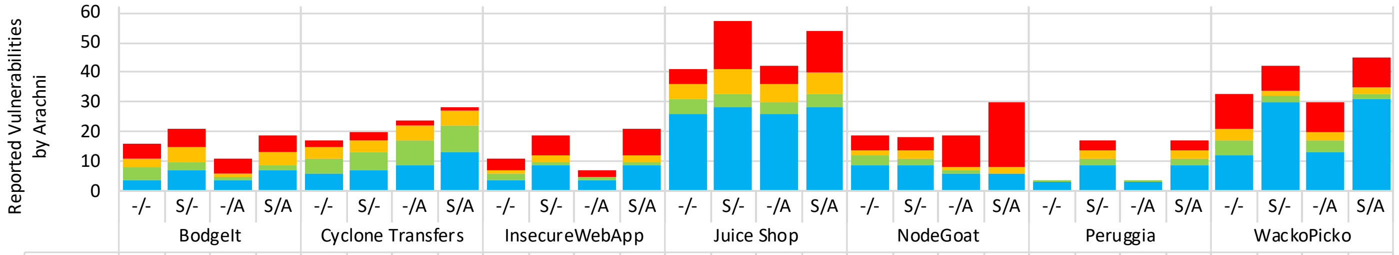
The direct consequence of this observation is that the increase of newly detected vulnerabilities is significantly higher than the increase of the absolute number of detected vulnerabilities as discussed in Section IV-C. To analyze this in detail, the relevant numbers are included in Table VIII.

TABLE VIII. REPORTED UNIQUE NEW VULNERABILITIES PER SCANNER, AND IMPROVEMENT BY USING JARVIS

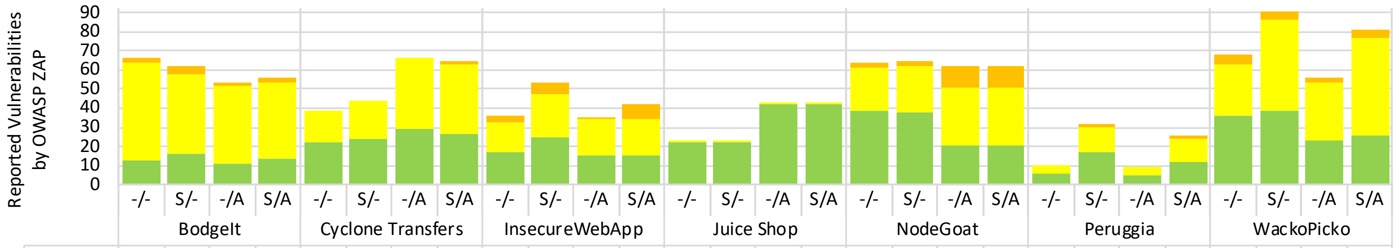
Scanner	*Config.*	*Reported Unique New Vulnerabilities*	*Improvement by using JARVIS*
Arachni	-/-	141	
	S/-	128	91%
	-/A	59	42%
	S/A	161	114%
OWASP ZAP	-/-	306	
	S/-	116	38%
	-/A	126	41%
	S/A	183	60%
Skipfish	-/-	482	
	S/-	611	127%
	-/A	419	87%
	S/A	606	126%
w3af	-/-	143	
	S/-	153	107%
	-/A	85	59%
	S/A	144	101%
All four Scanners	-/-	1'072	
	S/-	1'008	**94%**
	-/A	689	**64%**
	S/A	1'094	**102%**

The third column in Table VIII contains the number of reported new vulnerabilities per scanner and configuration and directly correspond to the "New Vulnerabilities" numbers in Figure 5. Looking at the numbers of Arachni, one can see that in configuration *S/-*, 128 new vulnerabilities are detected. Compared to the 141 vulnerabilities detected in configuration *-/-*, this corresponds to an increase of newly detected vulnerabilities of 91%. Likewise, 59 new vulnerabilities are detected in configuration *-/A*, an increase of 42% compared to configuration *-/-*. And finally, configuration *S/A* yields an increase of 114% compared to the basic configuration *-/-*. Similar results can be observed for the other three scanners. Adding up the numbers of all four scanners result in an increase of 94% in configuration *S/-*, 64% in configuration *-/A*, and 102% in configuration *S/A*.

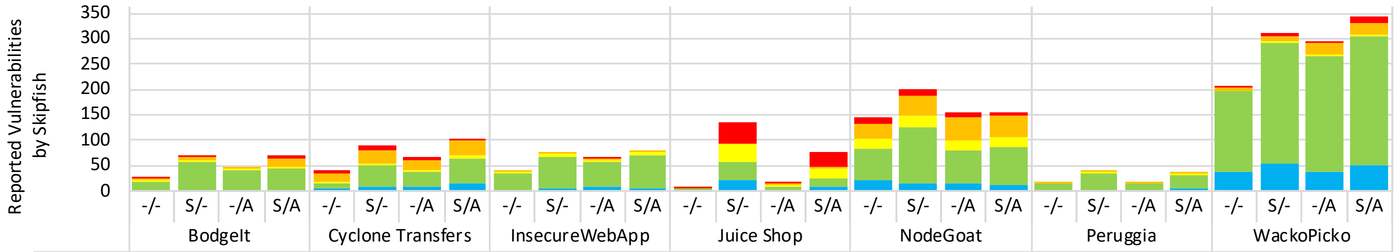
This analysis clearly shows that the effective benefit of JARVIS, i.e., the increase of new vulnerabilities that can be detected by using JARVIS, is significantly higher than the increase of the absolute number of detected vulnerabilities that was discussed in Section IV-C and listed in Table VI. For instance, in configuration *S/-*, there is an improvement of 60% on average with respect to the absolute number of vulnerabilities (see Table VI), but there is an improvement of 94% on average with respect to newly detected vulnerabilities. In addition, this analysis puts into perspective the previous conclusion that the technical solution to improve authenticated scans has only a relatively small positive impact. While the increase of the absolute number of vulnerabilities is indeed relatively small (18% on average, see Table VI), the increase of newly detected vulnerabilities is 64% on average, which is significantly higher. And in configuration *S/A*, the newly detected vulnerabilities can be increased by 102% on average, whereas the absolute increase according to Table VI is only



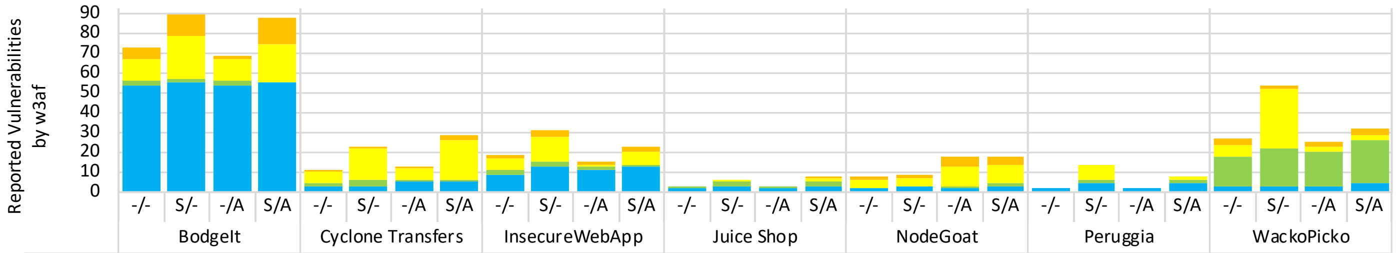
	BodgeIt				Cyclone Transfers				InsecureWebApp				Juice Shop				NodeGoat				Peruggia				WackoPicko			
	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A
🟥 Critical	5	6	5	6	2	3	2	1	4	7	2	9	5	16	6	14	5	4	11	22	0	3	0	3	12	8	10	10
🟧 High	3	5	1	4	4	4	5	5	1	2	0	2	5	8	6	7	2	3	1	2	0	3	0	3	4	2	3	2
🟨 Medium	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
🟩 Low	4	3	1	2	5	6	8	9	2	1	1	1	5	5	4	5	3	2	1	0	1	2	1	2	5	2	4	2
🟦 Info	4	7	4	7	6	7	9	13	4	9	4	9	26	28	26	28	9	9	6	6	3	9	3	9	12	30	13	31



	BodgeIt				Cyclone Transfers				InsecureWebApp				Juice Shop				NodeGoat				Peruggia				WackoPicko			
	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A
🟥 Critical	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
🟧 High	2	4	1	3	0	0	0	2	3	6	1	8	0	0	0	0	3	3	11	11	0	2	0	2	5	5	3	4
🟨 Medium	51	42	41	39	17	20	37	36	16	22	19	19	1	1	1	1	22	24	30	30	4	13	4	12	27	47	30	51
🟩 Low	13	16	11	14	22	24	29	27	17	25	15	15	22	22	42	42	39	38	21	21	6	17	5	12	36	39	23	26
🟦 Info	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

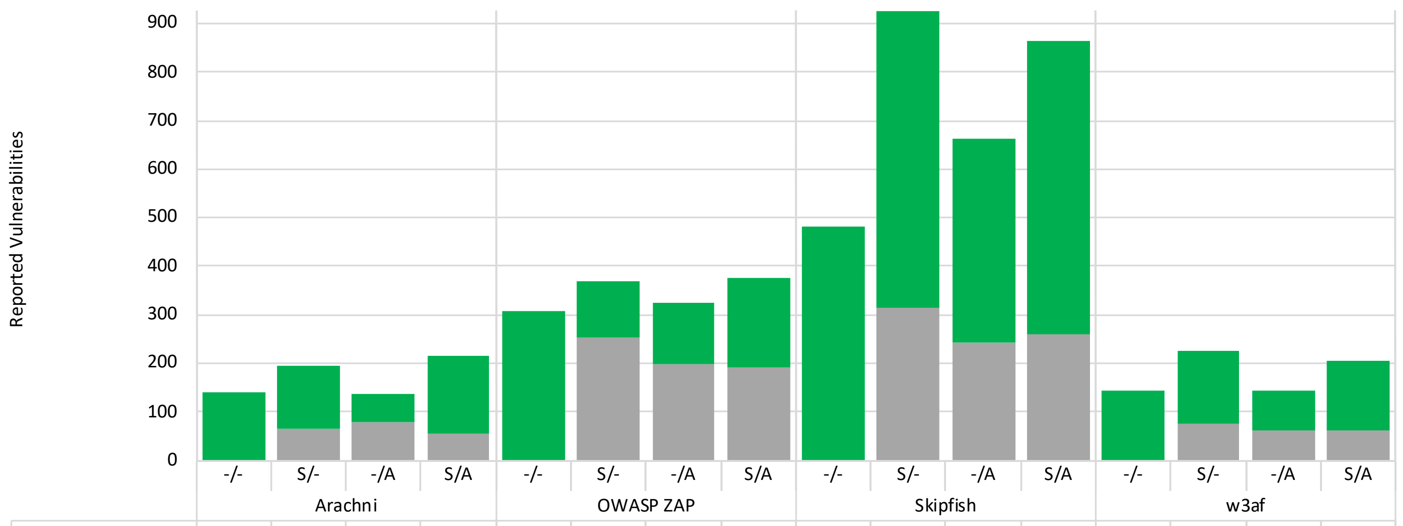


	BodgeIt				Cyclone Transfers				InsecureWebApp				Juice Shop				NodeGoat				Peruggia				WackoPicko			
	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A
🟥 Critical	1	1	0	7	6	8	8	2	0	0	1	0	2	41	3	31	13	14	12	6	0	0	0	0	1	6	1	12
🟧 High	4	6	2	16	17	27	19	29	2	3	1	2	0	0	2	1	29	39	45	43	1	3	1	2	5	10	22	25
🟨 Medium	2	3	4	3	2	3	3	7	1	5	3	6	2	37	4	20	18	23	18	20	1	4	1	2	1	3	5	2
🟩 Low	20	57	37	43	12	41	31	49	33	64	51	64	3	37	4	16	62	108	67	76	11	31	11	28	158	237	226	253
🟦 Info	0	2	3	1	5	10	8	14	3	5	8	6	2	21	4	10	23	17	14	11	3	4	4	5	39	54	39	52



	BodgeIt				Cyclone Transfers				InsecureWebApp				Juice Shop				NodeGoat				Peruggia				WackoPicko			
	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A
🟥 Critical	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
🟧 High	6	11	2	13	1	1	1	3	2	3	1	3	0	0	0	1	2	2	5	4	0	0	0	0	3	2	2	3
🟨 Medium	11	22	11	20	6	16	6	20	6	13	1	6	0	1	0	2	4	4	10	10	0	8	0	2	6	30	3	3
🟩 Low	2	2	2	0	1	3	1	1	2	2	2	1	1	2	1	2	0	0	1	1	0	2	0	2	15	19	17	22
🟦 Info	54	55	54	55	3	3	5	5	9	13	11	13	2	3	2	3	2	3	2	3	2	4	2	4	3	3	3	4

Figure 4. Reported Unique Vulnerabilities per Scanner and Test Application, according to Severity.



	Arachni				OWASP ZAP				Skipfish				w3af			
	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A
🟩 New Vulnerabilities	141	128	59	161	306	116	126	183	482	611	419	606	143	153	85	144
▨ Old Vulnerabilities	0	66	78	53	0	254	198	192	0	313	243	258	0	74	60	62

Figure 5. Reported Unique Vulnerabilities per Scanner, according to New and Old Vulnerabilities.

55% on average. To summarize, this analysis demonstrates that not only the technical solution to increase test coverage, but also the technical solution to improve authenticated scans significantly helps to uncover vulnerabilities that would not be found otherwise and therefore, it can be concluded that both technical solutions integrated in JARVIS provide a major benefit to increase the number of detected vulnerabilities.

Determining the exact reasons why several of the vulnerabilities found in configuration *-/-* are no longer detected when using the advanced configurations would require a detailed analysis of the crawling components of the scanners, of the specific behavior of the scanners when carrying out the vulnerability tests, and of the web applications in the test set, which is beyond the scope of this work. Nevertheless, it is certainly possible to give some arguments that explain that the observed behavior is reasonable:

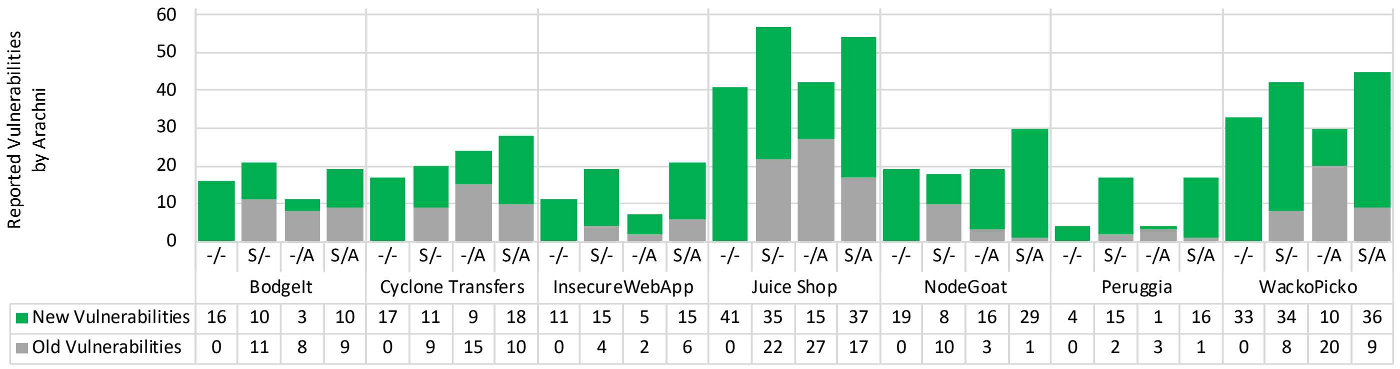
- Providing the crawler component of a scanner with additional seeds has a direct impact on the order in which the pages are requested. A different order implies different internal state changes within the web application under test [14], which typically leads to a different behavior of the web application both during crawling and during testing, and therefore to different findings.
- When doing authenticated scans, some of the resources that do not require authentication are often no longer reachable, e.g., registration, login and forgotten password pages. As deliberately insecure web applications often use such resources to place common vulnerabilities and as the evaluation of JARVIS is based on deliberately insecure applications (see Section IV-A), this most likely has a noticeable impact on the evaluation results.

An important consequence of the observation that not all vulnerabilities found in the basic configuration *-/-* are also found when using the three advanced configurations is that when testing a web application, a scanner should be used in all four configurations to maximize the total number of reported unique vulnerabilities (this will be analyzed in more detail in Section V). And obviously, although this was not analyzed in detail, an application that provides different protected areas for different roles should be scanned with users of all roles, i.e., configurations *-/A* and *S/A* should be used once per role.

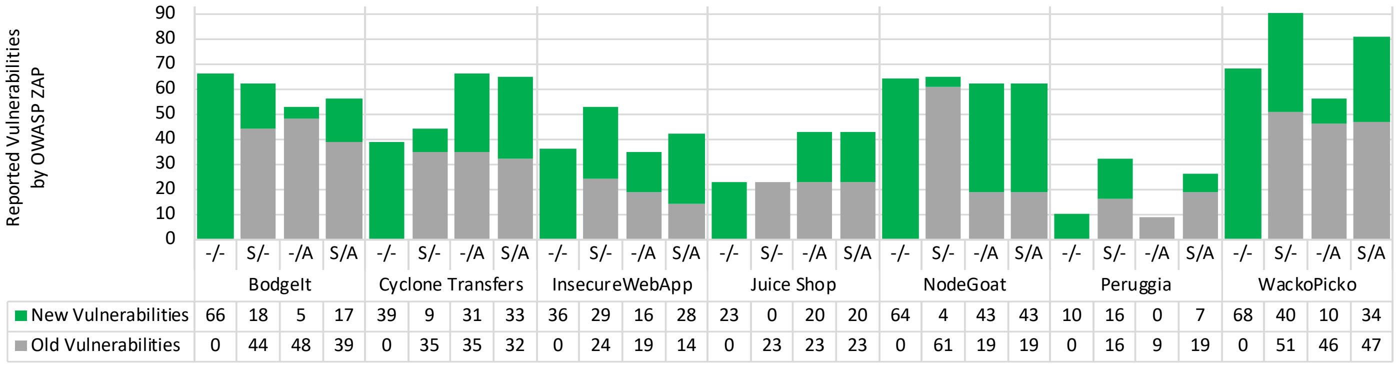
For completeness, Figure 6 shows how many of the vulnerabilities reported in the basic configuration are also found when scanning in other configurations, this time separated per scanner and per test application. Without going into the details, Figure 6 confirms that the conclusions made above are also valid when considering the test applications individually: When using the advanced configurations, several new vulnerabilities are reported and at the same time, several of the findings detected in the basic configuration are no longer reported.

V. Detailed Evaluation Focusing on SQL Injection and Cross-Site Scripting Vulnerabilities

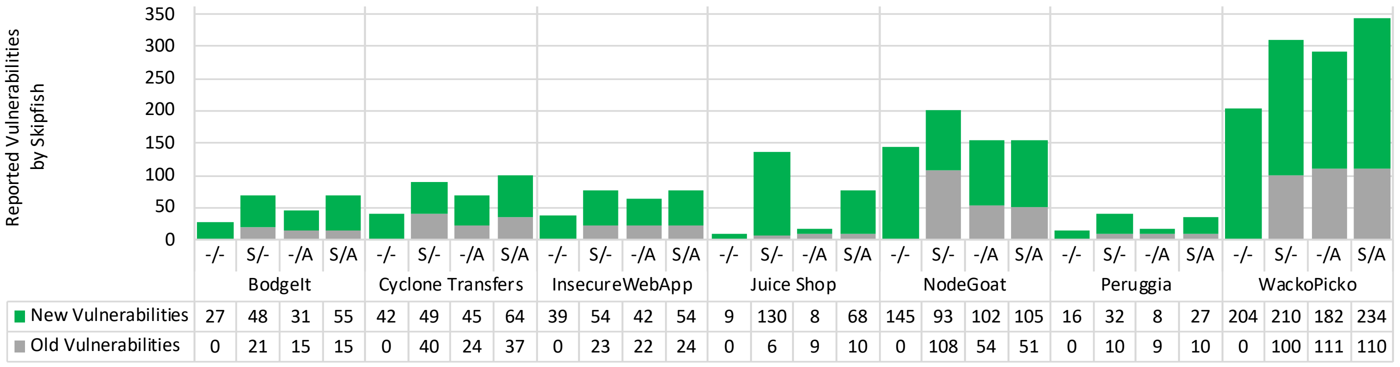
The evaluations done in Section IV demonstrate that JARVIS works very well to increase the number of detected vulnerabilities in the sense that in the advanced configurations, many additional vulnerabilities are detected and a significant fraction of them are security-critical. Two questions are still open, however. The first one is whether the additionally detected vulnerabilities are true vulnerabilities or merely false positives. In the context of a web application vulnerability scanner, a false positive is a vulnerability that is reported by the scanner, but that does not actually exist in the application under test. Conversely, a true positive is a vulnerability that is correctly identified by the scanner, i.e., one that is truly present in the tested application. The second question is whether it is indeed true that a scanner should always be used in all four configurations to maximize the total number of reported unique vulnerabilities. Based on the observations made in Section IV-D, this is most likely the case, but it should nevertheless be verified and quantified.



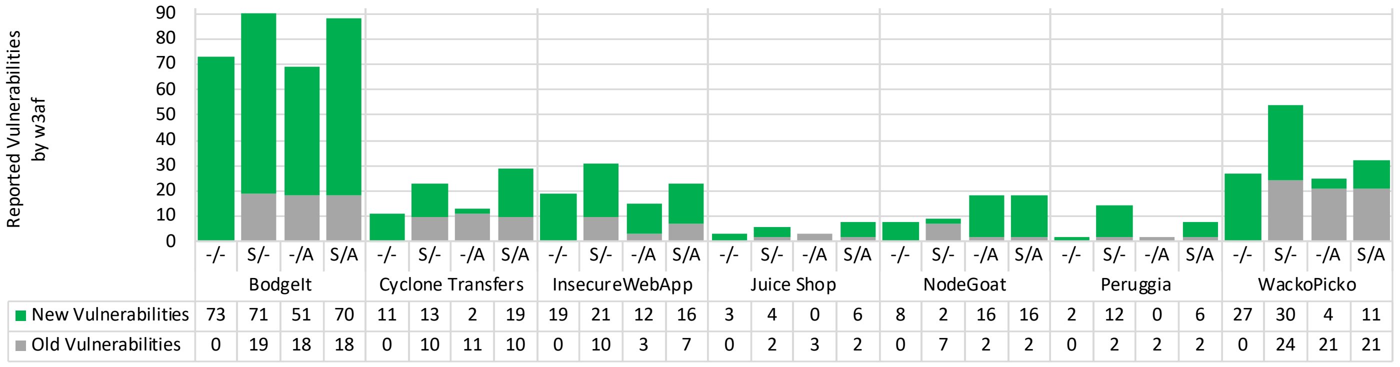
	BodgeIt				Cyclone Transfers				InsecureWebApp				Juice Shop				NodeGoat				Peruggia				WackoPicko			
	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A
🟩 New Vulnerabilities	16	10	3	10	17	11	9	18	11	15	5	15	41	35	15	37	19	8	16	29	4	15	1	16	33	34	10	36
▨ Old Vulnerabilities	0	11	8	9	0	9	15	10	0	4	2	6	0	22	27	17	0	10	3	1	0	2	3	1	0	8	20	9



	BodgeIt				Cyclone Transfers				InsecureWebApp				Juice Shop				NodeGoat				Peruggia				WackoPicko			
	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A
🟩 New Vulnerabilities	66	18	5	17	39	9	31	33	36	29	16	28	23	0	20	20	64	4	43	43	10	16	0	7	68	40	10	34
▨ Old Vulnerabilities	0	44	48	39	0	35	35	32	0	24	19	14	0	23	23	23	0	61	19	19	0	16	9	19	0	51	46	47

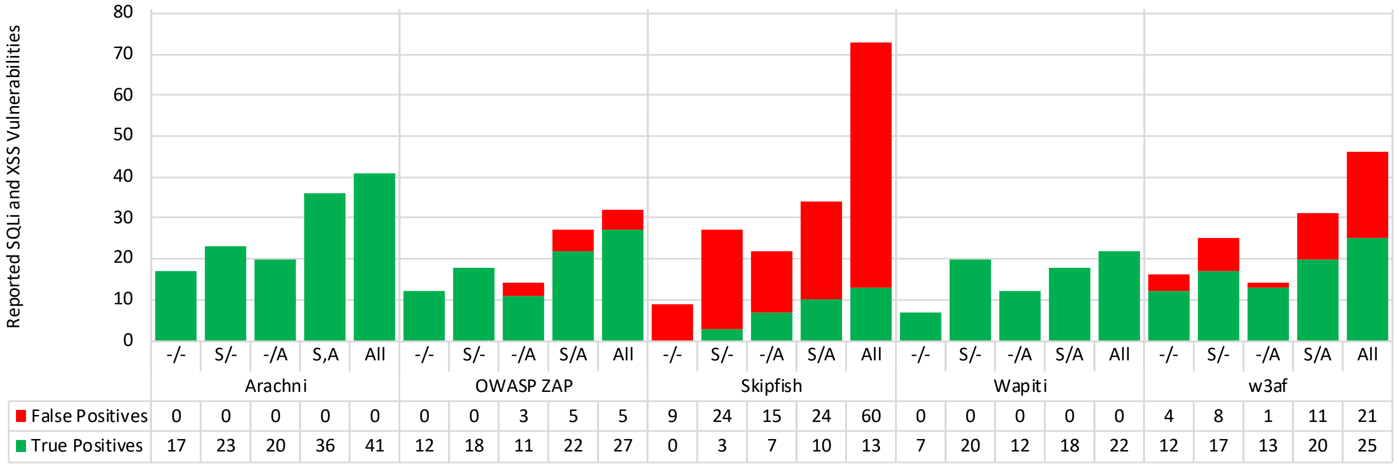


	BodgeIt				Cyclone Transfers				InsecureWebApp				Juice Shop				NodeGoat				Peruggia				WackoPicko			
	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A
🟩 New Vulnerabilities	27	48	31	55	42	49	45	64	39	54	42	54	9	130	8	68	145	93	102	105	16	32	8	27	204	210	182	234
▨ Old Vulnerabilities	0	21	15	15	0	40	24	37	0	23	22	24	0	6	9	10	0	108	54	51	0	10	9	10	0	100	111	110



	BodgeIt				Cyclone Transfers				InsecureWebApp				Juice Shop				NodeGoat				Peruggia				WackoPicko			
	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A	-/-	S/-	-/A	S/A
🟩 New Vulnerabilities	73	71	51	70	11	13	2	19	19	21	12	16	3	4	0	6	8	2	16	16	2	12	0	6	27	30	4	11
▨ Old Vulnerabilities	0	19	18	18	0	10	11	10	0	10	3	7	0	2	3	2	0	7	2	2	0	2	2	2	0	24	21	21

Figure 6. Reported Unique Vulnerabilities per Scanner and Test Application, according to New and Old Vulnerabilities.



	Arachni					OWASP ZAP					Skipfish					Wapiti					w3af				
	-/-	S/-	-/A	S,A	All	-/-	S/-	-/A	S/A	All	-/-	S/-	-/A	S/A	All	-/-	S/-	-/A	S/A	All	-/-	S/-	-/A	S/A	All
🟥 False Positives	0	0	0	0	0	0	0	3	5	5	9	24	15	24	60	0	0	0	0	0	4	8	1	11	21
🟩 True Positives	17	23	20	36	41	12	18	11	22	27	0	3	7	10	13	7	20	12	18	22	12	17	13	20	25

Figure 7. Reported Unique SQLi and XSS Vulnerabilities per Scanner, over all Test Applications, according to True and False Positives.

To answer these final two questions, a more detailed analysis focusing on SQL injection (SQLi) and cross-site scripting (XSS) vulnerabilities was done. To do this, all vulnerabilities of these types were first extracted from the original reports of the scanners. Then, the vulnerabilities were manually verified to identify them as either true or false positives. This required a lot of effort, which is the main reason why the focus was set on these two types. Nevertheless, this serves well to answer the two open questions and also to evaluate the true potential of JARVIS in general as both vulnerabilities are highly relevant in practice and highly security-critical. In addition, the test applications contain several of them, which means SQLi and XSS vulnerabilities represent a meaningful sample size. The results are illustrated in Figure 7, for each scanner and over all test applications. The green parts of the bars correspond to true positives (true vulnerabilities) and the red parts correspond to false positives (incorrectly reported vulnerabilities). Duplicates were manually removed, so the bars represent the number of unique vulnerabilities that were reported. In addition, Figure 7 not only shows the number of reported unique vulnerabilities per configuration, but also the total number of reported unique vulnerabilities when the findings of all four configurations are combined (this is identified as configuration *All*).

The first observation when analyzing Figure 7 is that the conclusions made in Section IV-C are still valid in the sense that for each scanner, the number of reported unique SQLi and XSS vulnerabilities is significantly increased when using the advanced configurations compared to the basic configuration *-/-* . This is not very surprising based on the analyses that were done so far, but it demonstrates that JARVIS not only improves the vulnerability detection performance when considering all reported vulnerabilities, but also when focusing on specific and highly relevant SQLi and XSS vulnerabilities.

In addition, Figure 7 delivers the answer to the first of the final two questions. Looking at the bars in the figure, it can be seen that using JARVIS does not have a significant impact on the number of false positives that are reported. For instance, Arachni, OWASP ZAP and Wapiti all produce no false positives when used in the basic configuration *-/-*. When using the advanced configurations, Arachni and Wapiti still do not report any false positives, while OWASP ZAP produces a relatively small fraction of false positives in configurations *-/A* and *S/A*. On the other hand, scanners that report false positives in the basic configurations (w3af and especially Skipfish, which does not report a single true positive in the basic configuration) also do so in the advanced configurations, but overall, the fraction of false positives reported in the advanced configurations remains in a similar order as in the basic configuration and is not significantly increased. As an example, the fractions of false positives reported by w3af are 25% in configuration *-/-*, 32% in configuration *S/-*, 7% in configuration *-/A*, and 35% in configuration *S/A*, so the fraction of false positives reported in any of the advanced configurations is not significantly higher than the 25% reported in configuration *-/-*. The same is true in the case of Skipfish, with the difference that the fraction of reported false positives is very high in general. Overall, the conclusion therefore is that JARVIS does not have a negative impact on the fraction of reported false positives. This is a very important finding because if using JARVIS resulted in a significantly increased fraction of reported false positives, then the value in practice would be very limited, even if the absolute number of true positives were also increased.

Furthermore, Figure 7 also answers the second open question and confirms what was already stated in Section IV-D: It is important to perform scans in all four configurations and to combine the detected vulnerabilities to maximize the number of reported unique vulnerabilities. This can easily be seen by comparing the heights of the bars: For each scanner, the height of the bar labeled with *All* is always greater than any of the other four bars, which means that the sum of the vulnerabilities detected in the four configuration (i.e., configuration *All*) is always higher than the number of vulnerabilities detected in any of the individual configurations (i.e., configurations *-/-*, *S/-*, *-/A* and *S/A*). For instance, in the case of OWASP ZAP, the four individual configurations report 12, 18, 14 and 27 unique vulnerabilities, and combining all these vulnerabilities results in 32 unique vulnerabilities, which is more than what was detected in any of the individual configurations. This is not only true when considering all vulnerabilities, i.e., true and false positives combined, but also when just considering the true positives. To analyze this in more detail, Table IX is used, which is based on the numbers in Figure 7, but which only considers the true positive vulnerabilities.

TABLE IX. Reported Unique True Positive SQLi and XSS Vulnerabilities per Scanner, and Improvement by using JARVIS

Scanner	*Config.*	*Reported SQLi and XSS Vulnerabilities*	*Improvement by using JARVIS*
Arachni	-/-	17	
	S/-	23	35%
	-/A	20	18%
	S/A	36	112%
	All	41	141%
OWASP ZAP	-/-	12	
	S/-	18	50%
	-/A	11	-8%
	S/A	22	83%
	All	27	125%
Skipfish	-/-	0	
	S/-	3	–%
	-/A	7	–%
	S/A	10	–%
	All	13	–%
Wapiti	-/-	7	
	S/-	20	186%
	-/A	12	71%
	S/A	18	157%
	All	22	214%
w3af	-/-	12	
	S/-	17	42%
	-/A	13	8%
	S/A	20	67%
	All	25	108%
All five Scanners	-/-	48	
	S/-	81	**69%**
	-/A	63	**31%**
	S/A	106	**121%**
	All	128	**167%**

From Table IX, it can be seen that for each of the five scanners, combining the results of all configurations delivers more true positives than are reported in any individual configuration. With Arachni, for instance, the best individual configuration (*S/A*) reports 36 findings, but when combining all four configurations, 41 findings are detected. The same observation can be made for the other scanners, which demonstrates that combining the vulnerabilities reported in all four configurations always results in the highest number of unique true positive vulnerabilities.

Compared to the basic configuration -/-, using configuration *All* more than doubles the number of reported unique true positive SQLi and XSS with every scanner. The smallest improvement is achieved with w3af, where the number of vulnerabilities is increased from 12 to 25 (a plus of 108%), followed by OWASP ZAP (125%), then Arachni (141%), then Wapiti (214%), and in the case of Skipfish, where not a single vulnerability (true positive) could be detected in the basic configuration, using JARVIS manages to detect 13 vulnerabilities (no %-benefit is included in Table IX with Skipfish as configuration -/- reports 0 true positives). Combining the numbers of all five scanners (see final row of Table IX) shows that on average and by combining the vulnerabilities reported in any of the four configurations, JARVIS manages to increase the number of reported true positive SQLi and XSS vulnerabilities by 167% compared to using the scanners without JARVIS.

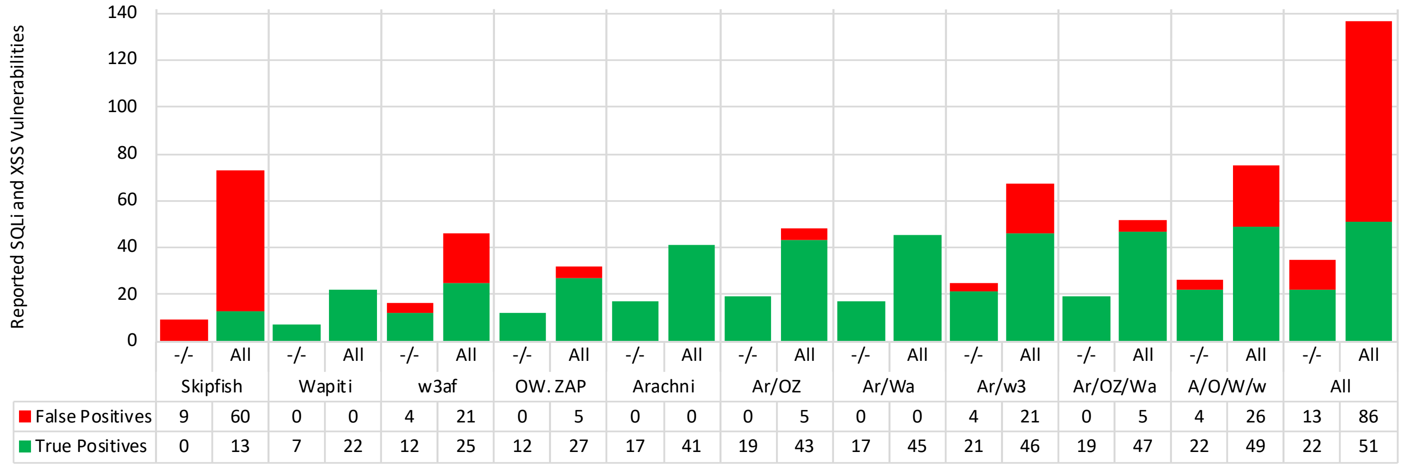
Finally, Figure 7 and Table IX also make it possible to compare the scanners. In particular, based on the test applications used in the evaluation and focusing on SQLi and XSS vulnerabilities, it shows that Arachni performs best as it finds the highest number of vulnerabilities without producing a single false positive, followed by OWASP ZAP and Wapiti. OWASP ZAP finds more true vulnerabilities than Wapiti, but also reports a few false positives. Next, there is w3af, which already reports a considerable fraction of false positives and finally, there is Skipfish, which performs quite poorly, not only with respect to true positives but especially also with respect to false positives. This once more puts into perspective the results of the first evaluation (see Figure 2), where Skipfish reported many more vulnerabilities than the other scanners.

VI. Evaluation of Combining Multiple Scanners

As the configuration effort to use JARVIS is small and the configurations are scanner-independent (see Section III-C), JARVIS makes it possible to use multiple scanners in parallel in an efficient way. Therefore, in a final evaluation, the benefits and limitations of using multiple scanners in parallel are analyzed. To do this, the same vulnerabilities as in the previous subsection are used, i.e., only SQLi and XSS vulnerabilities are considered, which makes it possible to precisely analyze the impact of using multiple scanners on the reported true and false positives. Figure 8 shows the reported unique true and false positive vulnerabilities when using individual scanners and different combinations of multiple scanners and when using the scanners in the basic configuration -/- and when combining the results of all four configurations (i.e., configuration *All*). The results are ranked from left to right in ascending order according to the number of true positives that are identified in configuration *All*.

Looking at the results in configuration *All*, the rightmost bar combines the results of all five scanners, which obviously delivers most true positives (51), but which also delivers most false positives (86). The results also show that in this test setting, Arachni performs very well on its own, as it finds 41 true positives (without a single false positive), which means that the other four scanners combined can only detect 10 true positives that are not found by Arachni. Looking at combinations of scanners, then the combination of Arachni & Wapiti (Ar/Wa) performs well and manages to identify 45 of the 51 true positives without any false positives. Combining Arachni, OWASP ZAP & Wapiti (Ar/OZ/Wa) is also a good choice as it finds 47 true positives with only a few false positives. This demonstrates that combining multiple scanners is indeed beneficial to increase the number of detected true positives without a significant negative impact on the number of reported false positives. However, blindly combining as many scanners as possible (e.g., all five scanners used here) is not a good idea in general because although this results in most true positives, it also maximizes the number of reported false positives. Finally, comparing the results in configuration *All* with the ones in configuration -/- demonstrates that even when combining multiple scanners, configuration *All* increases the number of detected true positives by more than 100% in every single case, which again underlines the benefits of JARVIS.

Note that since seven test web applications that cover several technologies were used in this evaluation, the results are at least an indication that the suitable combinations of scanners identified above (Arachni & Wapiti and Arachni, OWASP ZAP & Wapiti) should perform well in many scenarios. However, this is certainly no proof and it may be that other combinations of scanners are better suited depending on the web application under test. This means that in practice, one has to experiment



	-/-	All	-/-	All	-/-	All	-/-	All	-/-	All	-/-	All	-/-	All	-/-	All	-/-	All	-/-	All	-/-	All
	Skipfish		Wapiti		w3af		OW. ZAP		Arachni		Ar/OZ		Ar/Wa		Ar/w3		Ar/OZ/Wa		A/O/W/w		All	
🟥 False Positives	9	60	0	0	4	21	0	5	0	0	0	5	0	0	4	21	0	5	4	26	13	86
🟩 True Positives	0	13	7	22	12	25	12	27	17	41	19	43	17	45	21	46	19	47	22	49	22	51

Figure 8. Reported Unique SQLi and XSS Vulnerabilities using different Scanner Combinations, over all Test Applications.

with different scanner combinations to determine the one that is best suited in a specific scenario.

VII. Conclusion

In this paper, we presented JARVIS, which provides technical solutions to overcome some of the limitations – notably crawling coverage and reliability of authenticated scans – of web application vulnerability scanners. As JARVIS is independent of specific scanners and implemented as a proxy, it can be applied to a wide range of existing vulnerability scanners. The evaluation based on five freely available scanners and seven test web applications covering various technologies demonstrates that JARVIS works well in practice. In particular, JARVIS manages to significantly improve the number of reported vulnerabilities without increasing the fraction of false positives, and many of the additionally found vulnerabilities are security-critical. The most relevant evaluation results are summarized in the following list:

- The technical solution to increase test coverage has a major positive impact on the number of detected vulnerabilities. Compared to using the scanners without JARVIS (i.e., in the basic configuration *-/-*), the absolute number of reported unique vulnerabilities can be increased by 60% on average in configuration *S/-*. When only considering newly detected vulnerabilities, i.e., vulnerabilities that are not detected in the basic configuration *-/-*, the increase is 94% on average.
- The technical solution to improve authenticated scans has a relatively small impact on the absolute number of reported unique vulnerabilities. On average, the absolute number of reported vulnerabilities is increased by 18% when moving from configuration *-/-* to *-/A*. However, when considering the newly detected vulnerabilities, the improvement is 64% on average, which means the technical solution to improve authenticated scans also has a significant positive impact on the number of detected vulnerabilities.
- Using both technical solutions, i.e., when using configuration *S/A* instead of configuration *-/-*, the absolute number of reported vulnerabilities is increased by 55% and the number of newly detected vulnerabilities is increased by 102% on average. This means that on average, using JARVIS with both technical solutions more than doubles the newly detected vulnerabilities compared to scanning without using JARVIS.
- JARVIS slightly improves the fraction of security-critical vulnerabilities among all reported vulnerabilities. This underlines the practical benefit of JARVIS as it does not just report many additional irrelevant findings, but truly increases the number of security-critical issues that can be found
- A significant portion of the vulnerabilities that are detected when a scanner is used without JARVIS (i.e., in the basic configuration *-/-*) are not detected again when the scanner is used with JARVIS (i.e., in the advanced configurations *S/-*, *-/A* and *S/A*). A direct consequence of this observation is that the scanners should always be used in all four configurations, i.e., in configuration *-/-* without using JARVIS and in configurations *S/-*, *-/A* and *S/A* with using JARVIS to maximize the total number of detected vulnerabilities.
- A detailed analysis using SQLi and XSS vulnerabilities showed that JARVIS does not have a negative impact on the fraction of false positives that are reported. Scanners that report no false positives in configuration *-/-* deliver no or only very few vulnerabilities when using JARVIS. And scanners that report some false positives in the basic configuration also do so in the advanced configurations, but overall, the fraction of false positives remains more or less constant, independent of the configuration. This result is highly relevant for the applicability of JARVIS in practice, as otherwise, the practical benefit would be very limited.
- The same analysis demonstrated that it is indeed important to perform scans in all four configurations and to combine the detected vulnerabilities, as the sum of the vulnerabilities that are detected in the four different configurations is always greater than the number of vulnerabilities detected in any of the individual configurations. Also, this analysis showed that by using JARVIS, the effectiveness of each of the

five scanners used in the evaluation could be more than doubled and on average, the number of detected true positive SQLi and XSS vulnerabilities could be increased by 167%. This underlines that JARVIS is both an effective and truly scanner-independent solution to increase the number of detected security-critical vulnerabilities.

The configuration effort to use JARVIS is small and the configurations are scanner-independent. Therefore, JARVIS also provides an important basis to use multiple scanners in parallel in an efficient way. The provided analysis shows that combining multiple scanners is indeed beneficial as it increases the number of true positives, which is not surprising as different scanners detect different vulnerabilities. However, it was also demonstrated that blindly combining as many scanners as possible is not a good idea in general because although this results in most true positives, it also delivers the sum of all false positives reported by the scanners. In the evaluation, the combination of Arachni & Wapiti or Arachni, OWASP ZAP & Wapiti yielded the best compromise between a high rate of true positives and a low rate of false positives. As a representative set of web application technologies was used in the evaluation, it can be expected that these combinations work well in many scenarios, but this is no proof and in practice, one has to experiment with different combinations to determine the one that is best suited in a specific scenario.

Acknowledgment

This work was partly funded by the Swiss Confederation's innovation promotion agency CTI (project 18876.1 PFES-ES).

References

[1] D. Esposito, M. Rennhard, L. Ruf, and A. Wagner, "Exploiting the Potential of Web Application Vulnerability Scanning," in Proceedings of the 13th International Conference on Internet Monitoring and Protection (ICIMP). Barcelona, Spain: IARIA, 2018, pp. 22–29.

[2] WhiteHat Security, "2018 Application Security Statistics Report," Tech. Rep., 2018, URL: https://www.whitehatsec.com/blog/2018-whitehat-app-sec-statistics-report/ [accessed: 2019-05-03].

[3] A. Doupé, M. Cova, and G. Vigna, "Why Johnny can't Pentest: An Analysis of Black-Box Web Vulnerability Scanners," in Proceedings of the 7th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, ser. DIMVA'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 111–131.

[4] S. Chen, "SECTOOL Market," 2016, URL: http://www.sectoolmarket.com/price-and-feature-comparison-of-web-application-scanners-unified-list.html [accessed: 2019-05-03].

[5] L. Suto, "Analyzing the Accuracy and Time Costs of Web Application Security Scanners," Tech. Rep., 2010, URL: http://www.think-secure.nl/pdf/Accuracy_and_Time_Costs_of_Web_App_Scanners.pdf [accessed: 2019-05-03].

[6] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell, "State of the Art: Automated Black-Box Web Application Vulnerability Testing," in Proceedings of the 2010 IEEE Symposium on Security and Privacy, 2010, pp. 332–345.

[7] E. A. A. Vega, A. L. S. Orozco, and L. J. G. Villalba, "Benchmarking of Pentesting Tools," International Journal of Computer, Electrical, Automation, Control and Information Engineering, vol. 11, no. 5, 2017, pp. 602–605.

[8] M. Qasaimeh, A. Shamlawi, and T. Khairallah, "Black Box Evaluation of Web Application Scanners: Standards Mapping Approach," Journal of Theoretical and Applied Information Technology, vol. 96, no. 14, 2018, pp. 4584–4596.

[9] Y. Makino and V. Klyuev, "Evaluation of Web Vulnerability Scanners," in Proceedings of the IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), vol. 1, Warsaw, Poland, 2015, pp. 399–402.

[10] N. I. Daud, K. A. A. Bakar, and M. S. M. Hasan, "A Case Study on Web Application Vulnerability Scanning Tools," in 2014 Science and Information Conference, London, UK, 2014, pp. 595–600.

[11] S. Chen, "Security Tools Benchmarking: WAVSEP 2017/2018 - Evaluating DAST against PT/SDL Challenges," 2017, URL: http://sectooladdict.blogspot.com/2017/11/wavsep-2017-evaluating-dast-against.html [accessed: 2019-05-03].

[12] S. El Idrissi, N. Berbiche, F. Guerouate, and S. Mohamed, "Performance Evaluation of Web Application Security Scanners for Prevention and Protection against Vulnerabilities," International Journal of Applied Engineering Research, vol. 12, no. 21, 2017, pp. 11 068–11 076.

[13] SNORT, "Network Intrusion and Prevention System," URL: https://www.snort.org [accessed: 2019-05-03].

[14] A. Doupé, L. Cavedon, C. Kruegel, and G. Vigna, "Enemy of the State: A State-aware Black-Box Web Vulnerability Scanner," in Proceedings of the 21st USENIX Security Symposium (USENIX Security 12). Bellevue, WA: USENIX, 2012, pp. 523–538.

[15] A. v. Deursen, A. Mesbah, and A. Nederlof, "Crawl-based Analysis of Web Applications: Prospects and Challenges," Science of Computer Programming, vol. 97, 2015, pp. 173–180.

[16] G. Pellegrino, C. Tschürtz, E. Bodden, and C. Rossow, "jäk: Using Dynamic Analysis to Crawl and Test Modern Web Applications," in Research in Attacks, Intrusions, and Defenses, H. Bos, F. Monrose, and G. Blanc, Eds. Cham: Springer International Publishing, 2015, pp. 295–316.

[17] D. Zulla, "Improving Web Vulnerability Scanning," DEF CON, 2012, URL: https://www.defcon.org/images/defcon-20/dc-20-presentations/Zulla/DEFCON-20-Zulla-Improving-Web-Vulnerability-Scanning.pdf [accessed: 2019-05-03].

[18] PortSwigger, "Burp Suite," URL: https://portswigger.net/burp [accessed: 2019-05-03].

[19] ThreadFix, "ThreadFix Endpoint CLI," URL: https://github.com/denimgroup/threadfix/tree/master/archived/threadfix-cli-endpoints [accessed: 2019-05-03].

[20] B. Urgun, "WIVET: Web Input Vector Extractor Teaser," URL: https://github.com/bedirhan/wivet [accessed: 2019-05-03].

[21] ThreadFix, "Application Vulnerability Correlation with ThreadFix," URL: https://threadfix.it [accessed: 2019-05-03].

Applying Quality Requirements Framework to an IoT System and its Evaluation

Tsuyoshi Nakajima
Department of Computer Science and Engineering
Shibaura Institute of Technology
Tokyo, Japan
e-mail: tsnaka@shibaura-it.ac.jp

Toshihiro Komiyama
Software Engineering Division
NEC Corporation
Tokyo, Japan
e-mail: t-komiyama@bk.jp.nec.com

Abstract—**Modern information and communication technology systems are more focused on their quality requirements since they have been increasing their complexity. This paper shows how the quality requirements framework of the ISO/IEC 25030 can be applied to an Internet of things application. The results of this application are qualitatively evaluated to show the usefulness of the framework for defining quality requirements, and also its problems to be solved.**

Keywords—Quality requirements; SQuaRE; IoT.

I. INTRODUCTION

Information and Communication Technology (ICT) systems are increasingly used to perform a wide variety of organizational functions and personal activities. The quality of these products enables and impacts various business, regulatory and information technology stakeholders. High-quality ICT systems are hence essential to provide value, and avoid potential negative consequences, for the stakeholders.

To develop such high-quality ICT systems, it is important to define quality requirements, because finding the right balance of quality requirements, in addition to well-specified functional requirements, is a critical success factor to meet the stakeholders' objectives.

Furthermore, the complexity of ICT systems has grown exponentially with the advent of modern digital technologies like Internet of Things (IoT). This has also led to focus on more and more quality requirements that are critical to modern ICT systems.

ISO/IEC 25030 quality requirements was published in 2007, and its revision process has been going on to expand its scope from software to ICT systems [2]. The standard belongs to ISO/IEC 25000 series: Systems and software Quality Requirements and Evaluation (SQuaRE) has been developed as the successor of the other standards on product-related quality, including ISO/IEC 9126.

The quality requirements framework is applied to an IoT system in our previous work [1]. This paper fleshes out the contents to provide detailed discussion and evaluation of the framework. Section II explains the quality requirements framework and Section III describes the target IoT system, and then the framework is applied to the system in Section IV, and results of the application are qualitatively evaluated in Section V. Section VI reviews the related works, and finally, Section VII concludes this study.

II. QUALITY REQUIREMENTS FRAMEWORK

A. Architecture of the SQuaRE series

The SQuaRE series consists of five main divisions and one extension division. The main divisions within the SQuaRE series are:

- **ISO/IEC 2500n - Quality Management Division**. The standards that form this division define all common models, terms and definitions used by all other standards in the SQuaRE series. The division also provides requirements and guidance for the planning and management of a project.
- **ISO/IEC 2501n - Quality Model Division**. The standards that form this division provide quality models for system/software products, quality in use, data, and IT services. Practical guidance on the use of the quality model is also provided.
- **ISO/IEC 2502n - Quality Measurement Division**. The standards that form this division include a system/software product quality measurement reference model, definitions of quality measures, and practical guidance for their application. This division presents internal measures of software quality, external measures of software quality, quality in use measures and data quality measures. Quality measure elements forming foundations for the quality measures are defined and presented.
- **ISO/IEC 2503n - Quality Requirements Division**. The standard that forms this division helps specifying quality requirements. These quality requirements can be used in the process of quality requirements elicitation for a system/software product to be developed, designing a process for achieving necessary quality, or as inputs for an evaluation process.
- **ISO/IEC 2504n - Quality Evaluation Division**. The standards that form this division provide requirements, recommendations and guidelines for system/software product evaluation, whether performed by independent evaluators, acquirers or developers. The support for documenting a measure as an Evaluation Module is also presented.

B. Quality requirements and quality models/measures

Quality in use is the extent to which the influence (behavioral and attitudinal outcomes and consequences) of use of an ICT product or service meets the needs of users or other stakeholders in specific contexts of use (Figure 1). Therefore, quality in use requirements (QIURs) specify the required levels of quality from the stakeholders' point of view. These requirements are derived from the needs of various stakeholders. QIURs relate to the outcomes and consequences when the product is used in a particular context of use, and QIURs can be used as the target for validation of the product.

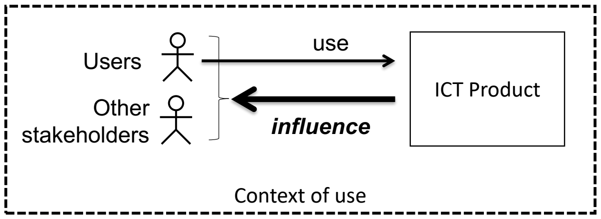


Figure 1. Quality in use

QIURs can be specified using quality in use model (ISO/IEC 25010 [3]) and measures (ISO/IEC 25022 [5]). Figure 2 describes characteristics and subcharacteristics of quality in use model.

Effectiveness	Efficiency	Satisfaction	Freedom from risk	Context coverage
Effectiveness	Efficiency	Usefulness Trust Pleasure Comfort	Economic risk mitigation Health and safety risk mitigation Environmental risk mitigation	Context completeness Flexibility

Figure 2. Quality in use model [3]

Product/Data quality is the capability of an ICT product/data that enables stakeholders to meet their needs (Figure 3).

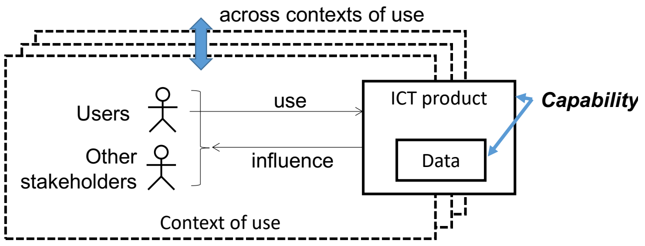


Figure 3. Product quality and data quality

Product quality requirements (PQRs) specify levels of quality required from the viewpoint of the ICT product. Most of them are derived from stakeholder quality requirements including QIURs, which can be used as targets for verification and validation of the target ICT product.

PQRs can be specified using product quality model (ISO/IEC 25010 [3]) and measures (ISO/IEC 25023 [6]). Figure 4 describes characteristics and subcharacteristics of product quality model.

Functional suitability	Performance efficiency	Compatibility	Usability
Functional completeness Functional correctness Functional appropriateness	Time-behavior Resource utilization Capacity	Co-existence Interoperability	Appropriateness recognisability Learnability Operability User error protection Use interface aesthetics Accessibility

Reliability	Portability	Maintain-ability	Security
Maturity Availability Fault tolerance Recoverability	Adaptability Installability Replaceability	Modularity Reusability Analysability Modifiability Testability	Confidentiality Integrity Non-repudiation Accountability Authenticity

Figure 4. Product quality model [3]

Data quality requirements (DQRs) specify levels of quality required for the data associated with the ICT product. These can be derived from related QIURs and PQRs. DQRs can be used for verification and validation from the data side.

DQRs can be specified using data quality model (ISO/IEC 25012 [4]) and measures (ISO/IEC 25024 [7]). Figure 5 describes 15 characteristics of data quality model, which are categorized by inherent and/or system dependent.

Inherent	Inherent & System dependent	System dependent
Accuracy Completeness Consistency Credibility Currentness	Accessibility Compliance Confidentiality Efficiency Precision Traceability Understandability	Availability Portability Recoverability

Figure 5. Data quality model [4]

C. *System hierarchy and scope of quality requirements*

Figure 6 describes the system hierarchy the SQuaRE series suppose and the scope for each type of quality requirements.

An information system, as the scope of QIURs, includes at least one ICT product, one user and relevant environments, and also can include other stakeholders such as developers, acquirers, regulatory bodies and society at large.

An ICT product, includes software, and also can include data, hardware, communication facilities, and other ICT products as its ICT components. PQRs are defined for the ICT product or its constituents (including sub-ICT products, hardware, communication facilities, software, and in some case software components), and DQRs are defined for the data inside the ICT product.

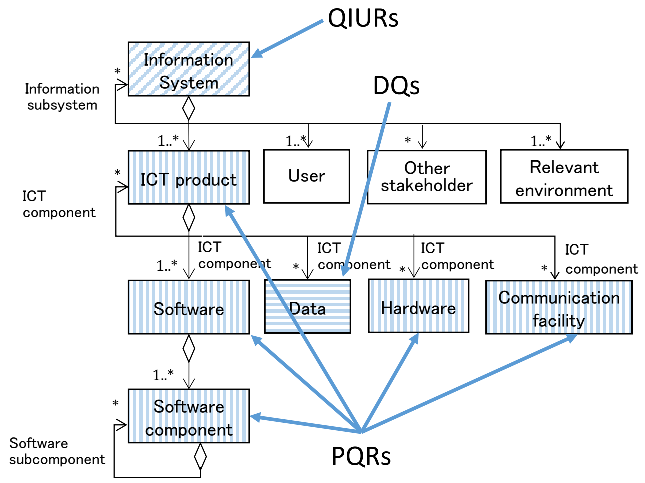


Figure 6. System hierarchy and scope of quality requirements

Figure 7 shows an example of mapping of a small IoT system, named Room open/close monitoring system, to the SQuaRE system hierarchy. The system judges whether the room is open or close based on luminance data measured under the room light, and user can know it through web.

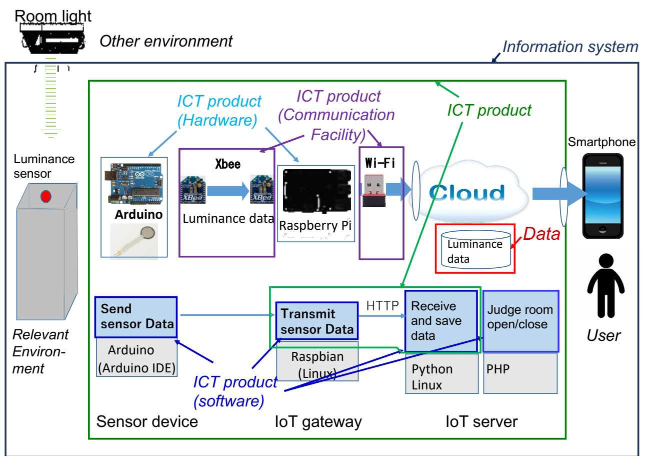


Figure 7. Mapping of Room open/close monitoring system to the SQuaRE system hierarchy

The *information system* includes a target *ICT system*, its *users* and *relevant environment*, which includes smartphones (as non-target ICT products) and the physical layout and phenomena in the room. The target *ICT product* includes a sensor device and an IoT gateway as *hardware*, Xbee and Wi-Fi as *communication facilities*, several software components (such as "send sensor data," "transmit sensor data," "receive and save data" and "judge room open/close") as *software*, and "luminance data" as *data*. If an ICT product is to be developed, the quality of all the target entities must be addressed and managed.

D. Quality requirements framework

The revision of ISO/IEC DIS 25030 [2] will provide a framework for quality requirements, which consists of concept of the quality requirements, and processes and methods to elicit, define, use and govern them.

There are three important points:

- To elicit quality requirements, not only direct users of the ICT product but also indirect users (using results of the product) and other stakeholders, such as developers, regulatory body, and society at large should be taken into account. TABLE I shows which type of stakeholders is a source of, a user of and relevant to which type of quality requirements.

TABLE I. STAKEHOLDERS AND TYPES OF QUALITY REQUIREMENTS

Stakeholder		Quality requirements		
		QIUR	PQR	DQR
User	Primary User	S	S	S
	Secondary User	S	S	S
	Indirect User	S		S
Other stakeholder	Developer	U	S, U	S, U
	Acquirer	U	U	U
	Regulatory body	S	S	S
	Society	R		

S: a source of / U: a user of / R: relevant to

- QIURs should be considered first because most of PQRs are derived from QIURs, and they should be deployed into PQRSs and DQRs of its sub-products (smaller ICT products, software, data, hardware and communication facilities) to meet them. Figure 8 describes how quality requirements derive others in the system hierarchy.

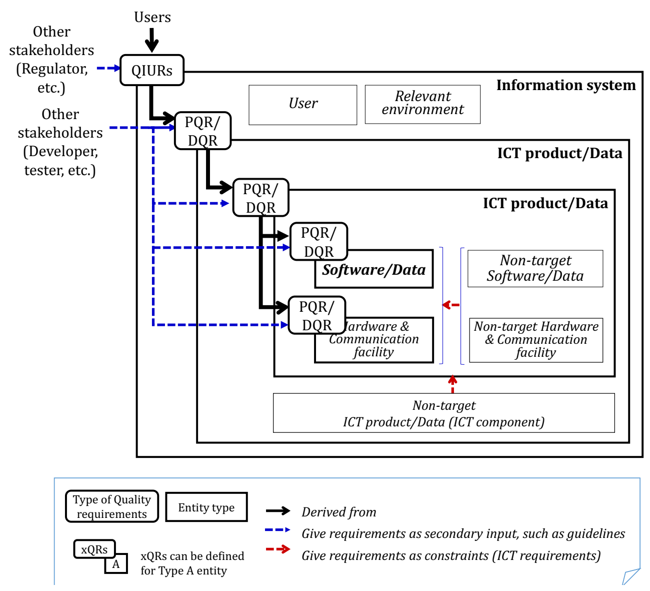

Users
Other stakeholders (Regulator, etc.)
Other stakeholders (Developer, tester, etc.)
QIURs
Information system
User
Relevant environment
PQR/ DQR
ICT product/Data
PQR/ DQR
ICT product/Data
PQR/ DQR
Software/Data
Non-target Software/Data
PQR/ DQR
Hardware & Communication facility
Non-target Hardware & Communication facility
Non-target ICT product/Data (ICT component)
Type of Quality requirements
Entity type
xQRs
A
xQRs can be defined for Type A entity
Derived from
Give requirements as secondary input, such as guidelines
Give requirements as constraints (ICT requirements)

Figure 8. Derivation of quality requirements [2]

- Quality requirements should be defined quantitatively, in order not to be vague and unverifiable requirements that depend on subjective judgement for their interpretation. To specify the quality measure, ISO/IEC 25022 for QIUR, ISO/IEC 25023 for PQR and ISO/IEC 25024 for DQR should be used.

III. IoT SYSTEM AND TARGET SYSTEM

A. *Characteristics of IoT systems*

The IoT envisages the future in which digital and physical things or objects can be connected by means of suitable information and communication technologies, to enable a range of applications and services. The IoT's characteristics include [8]:

- many relevant stakeholders involvement
- device and network heterogeneity and openness
- resource constrained
- spontaneous interaction
- increased security attack-surface

These characteristics will make development of the diverse applications and services a very challenging task.

B. *Target system*

The target IoT system, to which SQuaRE's quality requirements framework is applied, is Elderly monitoring system. Figure 9 shows its system architecture.

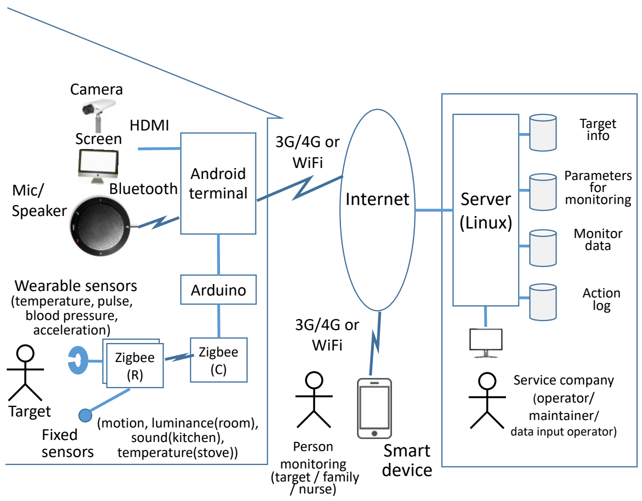

Camera
HDMI
Screen
Mic/ Speaker
Bluetooth
Android terminal
Arduino
Wearable sensors (temperature, pulse, blood pressure, acceleration)
Zigbee (R)
Zigbee (C)
Target
Fixed sensors
(motion, luminance(room), sound(kitchen), temperature(stove))
3G/4G or WiFi
Internet
3G/4G or WiFi
Person monitoring (target / family / nurse)
Smart device
Server (Linux)
Target info
Parameters for monitoring
Monitor data
Action log
Service company (operator/ maintainer/ data input operator)

Figure 9. Elderly monitoring system [9]

The sensor devices of the system gathers sensor data of the target elderly living alone. The sensors include motion, luminance, temperature, sound (microphone) and vision (camera) sensors at fixed points in the elderly's house, and wearable sensors to measure body temperature, pulse, blood pressure, and acceleration of the target. These sensor data are sent to the server of the service company, which monitors and analyzes actions and body conditions of the target to provide several services, such as informing the designated persons (persons to monitor) of the dangerous situation about the target, directly give advices to the target through the speakers, and so on.

TABLE II shows the important data for this system in the site of the service company.

TABLE II. IMPORTANT DATA FOR THE SYSTEM

Data	Description	Data items
Target Info	Personal data about targets, including their medical history	Target ID, Name, Birthday, Medical history, Physical info, Place
Parameters for monitoring	Parameters and rules about what and how to monitor	Target ID, Sensor data (type, range, accuracy, unit), Sensor configuration, Abnormity: (data, range)->action Persons monitoring the target
Monitor data	Time series of data for targets and system components monitored from sensors	Target ID, Sensor data with time, Status of system components with time
Action log	Time series of targets' actions the system guesses	Target ID, Sensor data with time, Status of system components with time

Figure 10 describes all the use cases of elderly monitoring system.

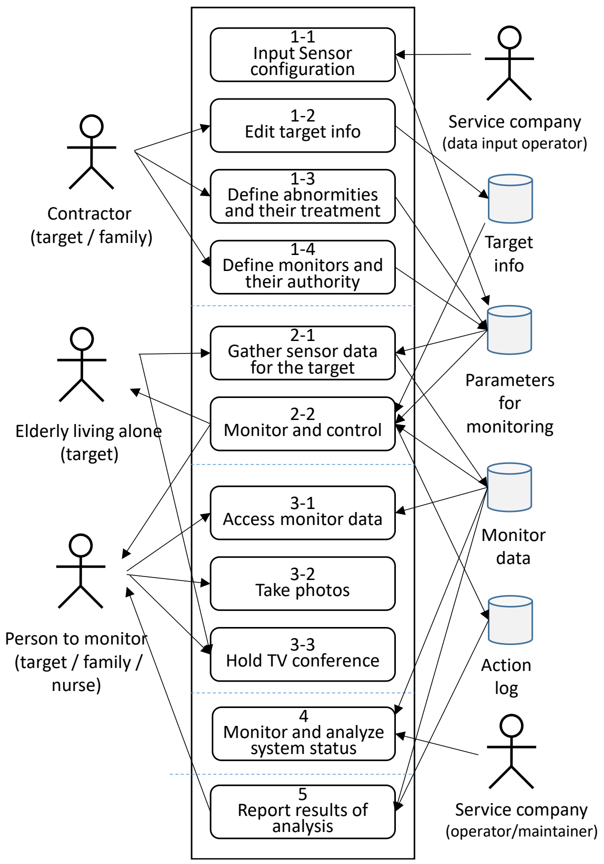

1-1 Input Sensor configuration
1-2 Edit target info
1-3 Define abnormities and their treatment
1-4 Define monitors and their authority
2-1 Gather sensor data for the target
2-2 Monitor and control
3-1 Access monitor data
3-2 Take photos
3-3 Hold TV conference
4 Monitor and analyze system status
5 Report results of analysis
Service company (data input operator)
Contractor (target / family)
Elderly living alone (target)
Person to monitor (target / family / nurse)
Target info
Parameters for monitoring
Monitor data
Action log
Service company (operator/maintainer)

Figure 10. Use cases of the elderly monitoring system (written by the author)

IV. Application of the Framework

A. *Stakeholder identification and selection of important QIURs*

In the first step, stakeholders of the target system are to be identified, in which the quality requirements framework provides categories of stakeholders: direct users, indirect user and other stakeholders. Other stakeholders include users of quality requirements (developers, acquirers and independent evaluators), regulatory bodies, and society at large.

The stakeholders identified for the elderly monitoring system are:

- Direct user: contractor, elderly living alone, family, nurse, and service company's operators
- Indirect user: service company's managers
- Other stakeholder: Developer, Ambulance

In the second step, the outcomes and consequences the target ICT product is required to provide should be identified. First, for all stakeholders, their goals to achieve through using the target system are extracted. In case of the direct users, there must be some use cases of the system (Figure 10), in which they are involved to achieve their goal. In case of indirect users, who uses not the product itself but the outputs of the product, and other stakeholders, which do not use it but may get influenced from it, there are no use cases relevant to their goals.

Since the quality in use model (its characteristics/ subcharacteristics), shown in Figure 2, categorizes outcomes and consequences that the ICT product provide, this step is simplified into selecting important quality in use characteristics/ subcharacteristics for achieving stakeholders' goals (and use cases).

TABLE III shows an example of selecting important QIURs for direct users. One example is about contractor, which has the goal to inform the service company of what he/she wants them to do. This goal corresponds to Use case 1-2, 1-3 and 1-4, and therefore, efficiency (of operation for input) and freedom from risks (of inputting wrong parameters) are selected as important subcharacteristics of quality in use. Another example is about the elderly living alone has two goals: to detect the designated abnormalities on himself/ herself to take the designated actions, and to obtain useful information on his/her current body conditions and behavioral patterns. The former goal corresponds to Use case 2-2 "Monitor and control," and therefore effectiveness (early medical treatment) and trust (on getting correct results on proper timing) are selected as important subcharacteristics of quality in use. The latter goal corresponds to Use case 5 "Report results of analysis," and therefore effectiveness (obtaining useful information on current body conditions and behavioral patterns to provide objective insights) is selected.

These QIURs, which consist of selected subcharacteristics and their brief description, are a starting point for further enhancement to detailed quality requirements and for derivation of PQRs and DQRs, which is described in Section B.

TABLE III. QIURs SELECTION FOR DIRECT USERS

Stakeholder	Goal	Use case	QIUR (with target outcomes and consequences)
Elderly living alone (**direct user**)	Detect designated abnormalities for the target, and take actions.	2-2	**Effectiveness**: early medical treatment **Trust**: correct results on proper timing
	Obtain his/her own current body condition and behavioral pattern.	5	**Effectiveness**: obtain info on current body condition and behavioral pattern to provide objective insights.
Family (**direct user**)	Confirm target's normality.	3-1 3-2	**Effectiveness**: see target's condition anytime and anywhere
	Be informed of target's serious abnormalities.	2-2	**Trust**: correct results on proper timing
			Freedom from risks: prevention from * overlook of serious abnormalities * unnecessary notice on trivial abnormalities
Nurse (**direct user**)	Confirm target's normality.	3-1 3-2	**Effectiveness**: remote nursing **Efficiency**: early notice of patient's abnormalities
	Be informed of target's all abnormalities.	2-2	**Effectiveness**: early treatment **Trust**: correct results on good timing
			Freedom from risks: prevention from overlook of serious abnormalities
	Create reports for asking doctors to diagnose abnormalities.	5	**Efficiency**: automatic reporting
Service company's operator (**direct user**)	Monitor all equipment, and take actions if something wrong with them.	4	**Efficiency**: system monitor and control **Effectiveness**: preventive actions before disfunction or malfunction
	Maintain and update system and equipment.	1-1	**Efficiency:** maintenance activities
Contractor (**direct user**)	Inform the service company of what he/she wants them to do.	1-2 1-3 1-4	**Efficiency**: operation for input **Freedom from risks**: prevention from wrong input

TABLE IV shows an example of selecting important QIURs for indirect users and other stakeholders.

TABLE IV. QIURs SELECTION FOR INDIRECT USERS AND OTHER STALEHOLDERS

Stakeholder	Goal	QIUR (with target outcomes and consequences)
Service company's manager (**indirect user**)	Customer satisfaction	**Usefulness** **Trust**
	Prevention from incidents	**Freedom from risks:** prevention from * incidents by system faults or malfunctions * incidents by normal operation * privacy leakage * malfunction by malicious attack
Developer (**Other stakeholder**)	Achieve QCD goal	**Efficiency**: development activities
	Update the system to implement new functions periodically	**Efficiency**: maintenance activities
Ambulance (**Other stakeholder**)	Dispatch ambulance cars on demand (by nurse's call)	**Freedom from risks**: prevention from unnecessary dispatches of ambulance cars

For instance, the service company's manager has two goals: to get customer satisfaction, and to prevent from incidents that may affect the company's business. To achieve the former goal, usefulness (of the product) and trust (on getting good services) are selected as important subcharacteristics of quality in use. To achieve the latter, freedom from risks (of system faults, security incidents and so on) is selected.

Other stakeholders, such as developers and regulatory bodies, also give some quality requirements on the target entities.

B. Derivation of PQRs and DQRs

As described in previous section, QIURs for each stakeholder have been elicited and documented. In the next step, they recursively evolve into PQRs and DQRs for the target entities at the lower level of the system hierarchy.

Figure 11 is a mapping of Elderly monitoring system to the SQuaRE system hierarchy after the concept design finished, in which half-tone processed IoT devices, data and software components are the target entities whose quality must be managed.

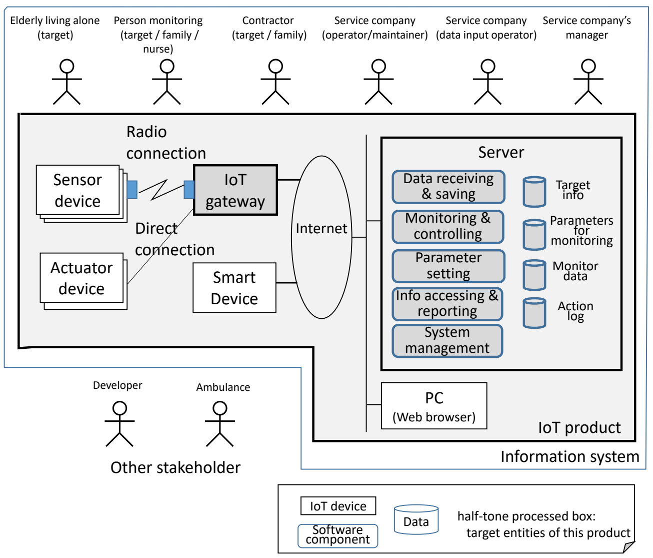


Figure 11. Target entities for quality requirements for Elderly monitoring system

The other entities, which are non-target ones, may influence and give some constraints to the target entities that include or interact with them.

To meet the corresponding QIURs, important product quality characteristics/ subcharacteristics (shown in Figure 4) for PQRs, and data quality characteristics/ subcharacteristics (shown in Figure 5) for DQRs are selected. Some PQRs for the target ICT product may be deployed into subcomponents to meet them (denoted with ->). DQRs are identified for the data files or data base used in the product.

TABLE V exemplifies how to derive PQRs and DQRs from QIURs of the service company's manager. From freedom from risks of incidents by system faults or malfunctions, three PQRs of availability for server, and maturity and time-behavior of the whole ICT product are identified. Availability of server (PQRs) entails recoverability of all the data on server (DQRs). Maturity of the whole ICT product (PQRs) is deployed into maturity of all the subcomponents (PQRs), including IoT devices, and software components, and accuracy, completeness, and consistency of all the data (DQRs). Time-behavior of the whole ICT product (PQRs) is deployed into throughput of server (PQRs), which entails efficiency and accessibility of monitor data (DQRs).

From freedom from risks of privacy leakage (QIURs), confidentiality of monitor data, target info and action log (DQRs) is derived, and then, one of monitor data entails confidentiality of all the devices and communication facilities from sensors to server, and one of target info and action log entails confidentiality of server and web.

TABLE V. DERIVATION OF PQRs AND DQRs FROM QIURs OF SERVICE COMPANY'S MANAGER

Stakeholder	QIUR	PQR	DQR
Service company's manager (**indirect user**)	**Freedom from risks:** prevention from		
	＊incidents by system faults or malfunctions	**Availability** of server	**Recoverability** of all the data on server
		Maturity ->**Maturity** of all sub-components	**Accuracy, Completeness and Consistency** of all data
		Time-behavior ->Throughput of server	**Efficiency** and **Accessibility** of Monitor data
	＊incidents by normal operation	**Maturity**:	**Accuracy, Consistency** and **Currentness** of Monitor data
	＊privacy leakage	**Confidentiality** of all the devices and communication facilities from sensors to server	**Confidentiality** of Monitor data
		Confidentiality of server and web	**Confidentiality** of Target info and Action log
	＊malfunction by malicious attack	**Integrity**: IoT devices, network	**Traceability** of Parameters for monitoring

TABLE VI exemplifies how to derive PQRs and DQRs from QIURs of the contractors (direct user). Use case 1-2, 1-3 and 1-4 are associated with "parameter setting," which is a software component on the server, and therefore, the derived PQRs and DQRs are respectively for the component and for "parameters for monitoring" as its input/output data. From efficiency (of operation for input), operability and accessibility of parameter setting as PQRs are derived, which entails understandability of parameters for monitoring (DQRs). From freedom from risks (of inputting wrong parameters) (QIURs), learnability and user error protection of parameter setting (PQRs) are derived, which entails understandability of parameters for monitoring (DQRs).

When considering freedom from risks about an IoT system, it is necessary to consider not only the risks relating to the integrity of the system and the confidentiality of its important data, but also the risks that the system gives some damage to the other systems; e.g., some IoT devices in the system infect malware to contributes to distributed denial-of-service (DDoS) attacks [10][11]. This means that the product quality requirements for IoT devices connecting to the Internet should include general internet security requirements.

TABLE VI. DERIVATION OF PQRs AND DQRs FROM QIURs OF CONTRACTOR

Stakeholder	Use case	QIUR	PQR	DQR
Contractor (**direct user**)	1-2 1-3 1-4	**Efficiency**: operation for input	**Operability** and **Accessibility** of Parameter setting (web)	**Understandability** of Parameters for monitoring
		Freedom from risks: prevention from wrong input	**Learnability** and **User error protection** of Parameter setting (web)	**Accuracy**, **Completeness** and **Consistency** of Parameters for monitoring

C. *Specifying quality requirements*

Quality requirements framework requires to quantitatively specify all the QIURs, PQRs and DQRs specified by using the quality requirements structure, shown in Figure 12.

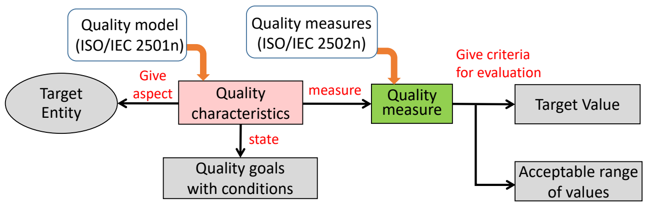

Quality model (ISO/IEC 2501n)
Quality measures (ISO/IEC 2502n)
Target Entity
Give aspect
Quality characteristics
measure
Quality measure
Give criteria for evaluation
Target Value
state
Quality goals with conditions
Acceptable range of values

Figure 12. Quality requirements structure

Selected important quality subcharacterisrics selected and derived in Sections A and B are enhanced through it into complete quality requirements.

The following example describes a PQR for "User error protection of Parameter setting" in TABLE VI.

- **Target entity:** Parameter setting
- **Selected characteristic:** User error protection
- **Quality goal with conditions:**
  Parameter setting assist contractor to correctly input parameters for monitoring through web.
- **Quality measure:** Avoidance of user operation error (Uep-1-G [6])
- **Target value:** 1
- **Acceptable range of values:** 0.98 - 1.00

V. QUALITATIVE EVALUATION

The following evaluation results are obtained from the application of the quality requirement framework defined in ISO/IEC DIS 25030 [2] to an IoT product of Elderly monitoring system:

A. *Stakeholder identification and selection of important QIURs*

- Merits:
  - ➢ The categorization and examples of roles for stakeholders makes it easy to identify stakeholders, especially not overlooking indirect users and other stakeholders other than direct users. In addition, provided categories and roles ease to guess stakeholders' essential goals.
  - ➢ Knowing the goals of stakeholders and their use cases, it is easy to find the quality sub-characteristics related to them.
- Issues:
  - ➢ There may be a high possibility that relationship patterns between the stakeholder's roles and quality sub-characteristics can be developed.

B. *Derivation of PQRs and DQRs*

- Merits:
  - ➢ Because extracting the QIURs first and then associating them with PQRs and DQRs, the necessity and the priority of PQRs and DQRs are much more visible than extracting them alone. These would be useful in the steps of prioritizing of and resolving conflicts between quality requirements, which are in the quality request framework but not applied this time.
  - ➢ Mapping the target information system and the ICT product to the SQuaRE system hierarchy provides two advantages:
    - ✧ to ease to clarify the target entities whose quality should be managed, and
    - ✧ to support to derivate PQRs and DQRs recursively along with the hierarchy.
- Issues:
  - ➢ It is difficult to check whether PQRs and DQRs it are comprehensively derived from QUIRs. The framework does not support how to check it. There may be a high possibility to develop patterns of mapping from the types of ICT products to a set of important quality sub-characteristics.

C. *Specifying quality requirements*

- Merits:
  - Because the quality requirement structure provides a list of items required to quantitatively describe the quality requirements and a list of quality measures corresponding to the quality sub-characteristics, it is very smooth to refine the quality requirements if an appropriate measures can be found.
- Issues:
  - The set of measures provided in ISO/IEC 25022-25024 are not enough to find the right ones for this application. Especially for engineering purposes, measures for a function and component are needed, but such measure are very few.

VI. Related work

There are few reports on application of quality requirements standards to somewhat large and complex systems. Doerr et al. [10] reports their experience with using the ISO 9126 [13] and IEEE-830 [14] as quality requirements methods in three different settings, concluding that the methods led to more complete quality requirements. Jardim-Goncalves et al. [15] propose a test and evaluation framework to assess quality of ICT product in the architectural design, supported by the SQuaRE and Generalized Net.

Elicitation for of quality requirements is one of the most important issues [16]. Robertson et al. [17] address that use case is a good but not-always-useful method to elicit quality requirements because some quality requirements can be linked directly to a functional requirement, while some apply to the product. To help elicit quality requirements, they classified quality requirements into eight types: look and feel, usability and humanity, performance, operational, maintainability, security, cultural and political, and legal. Plosch et al. [18] propose an elicitation method for quality requirements using goal-oriented approach, which consist of four steps: identify goals, specify quality aspects, derive measurable factors and derive quality requirements. The quality requirements framework of ISO/IEC 25030 provides all the aspects which the above approaches have.

It is important to develop and update the quality requirements techniques in order to deal with new technologies. Noorwali et al. [19] propose an approach for specifying quality requirements in the context of big data. Knass et al. [20] propose a knowledge management framework for knowledge about quality requirements, so that a development team in agile can properly establish, share and maintain them. The quality requirements framework of ISO/IEC 25030 will be continuously updated so that it can be applicable to new technologies.

VII. Conclusion and Future work

Modern ICT systems like IoT systems should put more focus on their quality requirements. This paper provides the brief introduction of ISO/IEC 25000 (SQuaRE) series, which define quality models and measures, and how to define quality requirements and evaluate quality of the ICT products.

And then, the IoT systems' unique characteristics compared to the other information systems are mentioned, including many relevant stakeholders' involvement, device and network level heterogeneity and openness, resource constrained, spontaneous interaction, and increased security attack-surface, which may make development of the diverse applications and services a very challenging task.

To solve this problem, we apply the quality requirements framework of the ISO/IEC 25030 revision to an IoT system, Elderly monitoring system [1], and this paper fleshes out the contents and provide detailed discussion. The results of this application make us understand the usefulness and limitations (some issues to impede its smooth use) of the framework. The usefulness of the framework includes: the stakeholder categorization makes it easy to identify stakeholders; extracting the QIURs first, from which PQRs and DQRs are derived, makes their necessity and priority visible; the SQuaRE system hierarchy clarifies the target entities whose quality should be managed, and supports to derivate PQRs and DQRs recursively along with it; the quality requirement structure makes it smooth to quantitatively refine the quality requirements.

More application of the framework to a variety of IoT systems and much larger scale ones and some quantitative evaluation should be needed to ensure its usefulness and to clarify its limitations and problems.

References

[1] T. Nakajima. "Applying Quality Requirements Framework to an IoT System," The Fourth International Conference on Fundamentals and Advances in Software Systems Integration, Venice (Italy), September 16 - 20, 2018.

[2] ISO/IEC DIS 25030, Systems and Software engineering — Quality requirements framework.

[3] ISO/IEC 25010:2011, Systems and Software engineering — System and software quality models.

[4] ISO/IEC 25012:2008, Systems and Software engineering — Data quality model.

[5] ISO/IEC 25022:2016, Systems and Software engineering — Measurement of quality in use.

[6] ISO/IEC 25023:2016, Systems and Software engineering — Measurement of system and software product quality.

[7] ISO/IEC 25024:2015, Systems and Software engineering — Measurement of data quality.

[8] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for internet of things, a survey," IEEE Internet of Things Journal, Vol. 3, No. 1, pp. 70-95, 2016.

[9] S. Okazaki et al., "An Intelligent Space System and its Communication Method to Achieve the Low Energy Consumption," IEEJ-C Vol. 136, No. 12, pp. 1804-1814, 2016 (in Japanese).

[10] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas. "DDoS in the IoT: Mirai and other botnets," Computer, Vol. 50, No. 7, pp. 80-84., 2017.

[11] E. Bertino and N. Islam. "Botnets and internet of things security," Computer, Vol. 50, No. 2, pp. 76-79, 2017.

[12] J. Doerr, D. Kerkow, T. Koenig, T. Olsson, and T. Suzuki. "Non-functional requirements in industry-three case studies adopting an

experience-based NFR method." 13th IEEE International Conference on Requirements Engineering (RE'05). IEEE, 2005.

[13] ISO/IEC 9126, Software engineering — Product quality -- Part 1: Quality model.

[14] IEEE 830-1998 - IEEE Recommended Practice for Software Requirements Specifications.

[15] R. Jardim-Goncalves, Ricardo, and V. Taseva. "Application of SQuaRE and Generalized Nets for extended validation of CE systems." 2009 IEEE International Technology Management Conference (ICE). IEEE, 2009.

[16] S. Ullah, M. Iqbal, and A. M. Khan. "A survey on issues in non-functional requirements elicitation." International Conference on Computer Networks and Information Technology. IEEE, 2011.

[17] S. Robertson, and J. Robertson. Mastering the requirements process: Getting requirements right. Addison-wesley, 2012.

[18] R. Plosch, A. Mayr, and C. Korner. "Collecting quality requirements using quality models and goals." 2010 Seventh International Conference on the Quality of Information and Communications Technology. IEEE, 2010.

[19] I. Noorwali, D. Arruda, and N. H. Madhavji. "Understanding quality requirements in the context of big data systems." Proceedings of the 2nd International Workshop on BIG Data Software Engineering. ACM, 2016.

[20] E. Knauss, G. Liebel, K. Schneider, J. Horkoff, and R. Kasauli. "Quality requirements in agile as a knowledge management problem: more than just-in-time." 2017 IEEE 25th International Requirements Engineering Conference Workshops (REW). IEEE, 2017.

Comparative Evaluation of Database Read and Write Performance in an Internet of Things Context

Denis Arnst*,
University of Passau, Passau, Germany
Email: *arnst@fim.uni-passau.de
Thomas Herpich†, Valentin Plenk‡, Adrian Wöltche§
Institute of Information Systems at Hof University, Hof, Germany
Email: †thomas.herpich@hof-university.de, ‡valentin.plenk@iisys.de, §adrian.woeltche@iisys.de

Abstract—**In the context of the Internet of Things (IoT), there is the need to manage huge amounts of time series sensor data, if high frequency device monitoring and predictive analytics are targeted for improving the overall process quality in production or supervision of quality management. The key challenge here is to be able to collect, transport, store and retrieve such high frequency data from multiple sensors with minimum resource usage, as this allows to scale such systems with low costs. For evaluating the performance impact of such an IoT scenario, we produce 1000 datasets per second for five sensors. We send them to three different types of popular database management systems (i.e., MariaDB, MongoDB and InfluxDB) and measure the resource impacts of the writing and reading operations over the whole processing pipeline. These measurements are CPU usage, network usage, disk performance and usage, and memory usage results plus a comparison of the difficulty for the developers to engineer such a processing pipeline. In the end, we have a recommendation depending on the needs, which database management system is best suited for processing high frequency sensor data in an IoT context.**

Keywords–performance; benchmark; nosql; relational; database; industry 4.0; mariadb; mongodb; influxdb; internet of things; high frequency data acquisition; time series.

I. Introduction

Internet of Things (IoT), Industry 4.0 (I4.0), ...these current buzzwords and many more refer to data-based management strategies, i.e., a new way of processing big and smart data. While many papers propose data-mining algorithms to extract commercial value from a database or a data lake (e.g. [2]–[4]), less address the computing requirements of such algorithms in combination with the systems writing or reading the data. The need for such an evaluation arises, because of the urge to become more and more precise in the technological advancement, the quality management has to keep up for being able to minimize defective products. We call this the industrial data analytics process. Without proper systems for managing high frequency data of dozens or hundreds of different sensors, it is for example nearly impossible to detect electrical distortions in the power supply of precision tools for producing highest quality engine parts. Having possible candidates of systems for managing such data without having to pay enormous sums of money, allows to incorporate a new level of quality management and even predictive analytics in new fields of technological systems, e.g., in production, surveillance, smart home, security business or economics that would otherwise be too expensive or too slow.

In this paper, we therefore evaluate the computing requirements on all parts of an IoT and I4.0 sensor system in a benchmark scenario for being able to recommend one or more systems depending on the needs of the industrial data analytics process [5]. The benchmark scenario is based on one of our research projects, where we collect and store $\approx 4\frac{\mathrm{GB}}{\mathrm{day}}$ of sensor data. This does not sound much, but within a year of measurement, this can grow to $\approx 1.5\frac{\mathrm{TB}}{\mathrm{year}}$, which is a lot for a traditional database system. This is why we need to focus on a small impact of resource usage for being able at all to accomplish the goal of high frequency data management.

In our scenario, for simulating this big picture, we first store generated time series sensor data to a database, which simulates the acquisition, and then we retrieve parts of the data for simulating the analytics part. We think that typical sensor acquisition computing resources have only low performance when sitting nearby the sensor (i.e., integrated circuits only made for reading and sending the sensor data). For the data to reach the database server, we believe that there might be cases where no cable connection is set up but wireless data transportation could be installed. Although the server itself is normally well suited concerning its computational capacity, the low resource impact on writing is an important goal. With reading the data, we think most work is still on the database server, which has to find and accumulate the needed data points. The client that reads the data might be a normal desktop computer or laptop, but also could be a smartphone or tablet, so the performance impact on the reading client side also is not to be left out. As the database server is the primary key in performance here, since all the writing and reading work is done there, our benchmark mainly focuses on the different database system servers.

Of course, the typical database servers can be tuned towards high performance reading or writing of data, but often not towards both at once. This is especially the case, when a fast retrieval is more important than a fast storage, for example with time series data in predictive analytics. When comparing different sensor readings at different points in time, relational databases rely on B-tree indexes that allow a fast search for data. These indexes are a huge performance bottleneck if frequent updates are made. This stems from B-trees being optimized for random fills and not for updates only coming from one side of the tree. [6] propose structures like the B(x)-tree to overcome this problem. Nevertheless, standard databases do not implement specialized index structures in most cases. Instead, specialized ”time-series” databases for this use case exist (e.g. [7]–[10]).

To verify whether these databases are more suitable for our application, we use the benchmark scenario presented in

Section II that generates a standard load on all subsystems of the setup, to compare relational, NoSQL and specialized time-series databases. Section III presents our test candidates.

Moreover, our experience is that companies rely on systems they already know and that have been proven to work stably. Additionally, the developers often have a long experience in standard database systems such as relational databases but not in specialized databases like the mentioned time-series databases. We believe that there are many installations of traditional databases that are considered for the industrial data analytics process instead of choosing a specialized tool, because of the risk that comes with a software that has not been tested and validated by the company yet. Therefore, we also consider the implementation difficulty of specialized databases in comparison to traditional systems, and we also develop "sophisticated" algorithms for getting more performance out of these systems, which would not be available with rather "naive" implementations.

In Section IV, we describe different implementations we developed for writing to the databases and reading from them. We evaluated several ideas from [11], such as time series grouping, which is such a more sophisticated approach.

To evaluate the database performance, we measure the load on the involved infrastructural components, i.e., CPU, memory, network and hard disk, and perform the benchmarking, as described in Section V. We believe that the infrastructural impact is most important for deciding which database is best suited for a specific IoT or I4.0 scenario. Section VI discusses and explains the findings. Section VII summarizes the paper and gives recommendations for different needs in an industrial data analytics process.

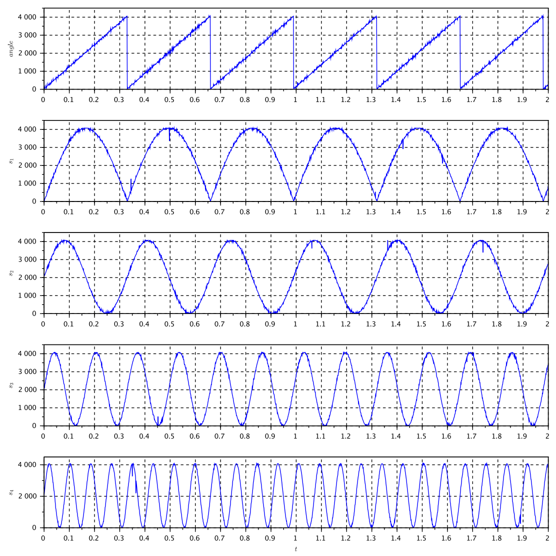


Figure 1. Simulated Test Data: Machine Angle (top) and four Data channels

II. Benchmark Application

One of our current research projects is using predictive maintenance for analyzing data stemming from a complex tool operating within an industrial machine tool. The tool is equipped with 13 analog and 37 digital sensors recording mechanical parameters during operation. The machine opens and closes the individual tool components $\approx$ 3 times per second, i.e., 3 working cycles per second.

Our data-gathering application records ≈ 300 samples per cycle from the sensors and stores them in a database for later analysis. Basically, it stores $1000 \frac{\text{samples}}{\text{sec}}$. This keeps the software structure simple and universal and requires few computing resources on the system writing the data.

For our analysis on the client side we need to retrieve all samples in one cycle. This does not correspond to the structure of the database. Our client software maps the time-series data to machine cycles by using one of the analog input channels as abscissa. This channel, shown as top channel in Figure 1, represents the rotatory angle of the machine tool's main drive. One rotation corresponds to one machine cycle. The time-series data of this channel is a sawtooth wave. The period of this wave is equal to the cycle time.

We use this scenario of writing and reading sensor data as a base idea for this paper to benchmark the industrial data analytics process. For the tests in this paper, we substitute the actual instrumentation and signal conversion with a small program that creates the sawtooth wave and four sine waves. For more realistic data, we add some random noise. This simulates the uncertainty of the sensor readings due to the sensor resolution and electrical distortion. This prevents the optimazation of the algorithm by hard-coding a machine cycle duration, which would be possible if the data was completely deterministic. Our reading algorithm, which searches for the measured machine cycle by comparing the noisy abscissa data, can be seen as very realistically usable this way.

Figure 1 shows the simulated data. In total, we simulate 5 analog channels with a resolution of 12 bit (represented using 2 bytes) and a sample rate of $1000 \frac{\text{samples}}{\text{sec}}$. This corresponds to a data rate of $10000 \frac{\text{bytes}}{\text{sec}}$ of simulated data at the sensor. Later, we add timestamps for each sample with millisecond resolution, which increases the amount of data sent to the database server.

Figure 2 shows the flow of the data through our setup. The reason for this data flow is that we think this exactly matches a typical industrial environment, where sensors gather measurements (Data-Source), a piece of small software transmits this information to a database server (Database Writer), and a monitoring service reads the data from the database (Database Reader).

In our setup, the Banana Pi single board computer is running two separate applications: the first simulates sensor data for replacement of real sensors. The second receives the data and writes it to the database on our server. These applications are linked via a Linux message queue. If the second application is not reading fast enough to keep the buffered data in the queue below ≈ 16 kByte, data is lost. This is similar to reading a real sensor which does not cache the data or waits for another application to read the data before it overwrites its internal memory with new measurements.

Figure 2 then shows the database specific applications with gray background. These writer and reader applications are implemented each for InfluxDB, MariaDB and MongoDB, because every database has it's own application programming interface. They use high-level libraries as far as possible to access the database.

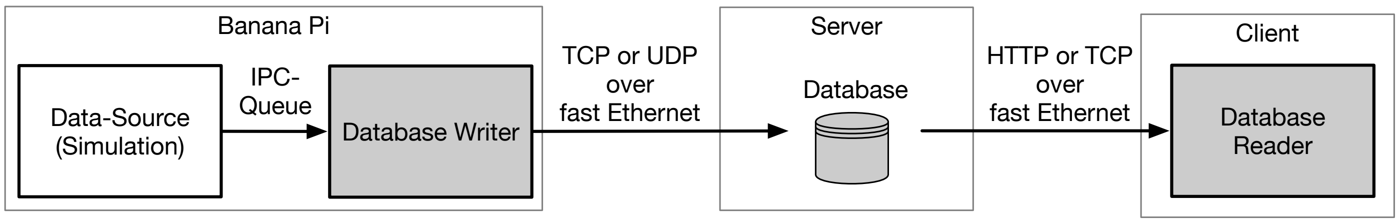


Figure 2. Block Diagram of the Test setup

For the transmission of the data, in the case of InfluxDB, the writing application uses a very fast and easy to access UDP interface. The advantage of this interface is that it is sufficient to send a simple concatenated string that is then interpreted by the InfluxDB server system for commitment of the data. For reading InfluxDB, we use the HTTP interface with an high-level library, which is based on TCP packets. This is because the UDP interface is solely for writing and the HTTP interface is the recommended way of querying data from InfluxDB. MariaDB and MongoDB both use specific TCP connections for the writing and reading of the data.

Concerning the hardware used in our setup, the single board-computer is a Banana Pi M3. This system uses an ARM Cortex A7 (8 x 1.8 GHz) with 2 GB DDR3-RAM and has Gigabit Ethernet on board.

The three databases are run on a dedicated server with an Intel Core i5-4670 CPU (4 x 3.4 GHz), 16 GB DDR3-RAM (4 x 4 GB) and a 256 GB SSD on SATA 3.1 (6.0 Gb/s), also with Gigabit Ethernet. This server is different from the server used for the measurements in [1], because, unfortunately, the old hardware broke.

The database reader is run on a dedicated desktop computer with an Intel Core i7-4785T (4 x 2.2 GHz with SMT), 16 GB DDR3-RAM (4 x 4 GB) and Gigabit Ethernet.

Concerning the software, the simulation application as well as the InfluxDB and MariaDB writing and reading applications are written in C, the MongoDB writing and reading applications are written in C++. We decided to use C-based languages for performance reasons, so that the application benchmark can be run with native platform speed.

All systems are running a Linux CentOS 7 without X.org server and with the same level of system updates. They are linked via a Gigabit Ethernet switch for non-blocking network IO.

III. Choice of databases

Various publications like [8] or [12] list a huge number of different databases. They distinguish three categories relevant for us: Relational Database Management Systems (RDBMS), NoSQL Database Management Systems (DBMS), and the more specialized Time Series Databases (TSDB). For our benchmark, we chose one system for each category. For the selection we focused on mature (stable releases available for at least three years) and free software with options for enterprise support. We mainly consulted the database ranking website [12] as a basis for selecting databases for our comparison.

As a representative RDBMS we selected the open source database MariaDB [13]. It is a fork of the popular MySQL database and widely used in web applications and relational scenarios. [14] lists MySQL and its more recent fork MariaDB combined as top RDBMS.

We selected MongoDB [15] as a DBMS advertised expressly for its usefulness in an IoT context with a lot of sensor data. It is also the most promising document store [16].

As TSDB we chose InfluxDB [17], which claims to be highly specialized in sensor data. This claim is confirmed by the score in [18].

We believe that our choice stands for all major and currently important and well-known types of database systems in the IoT and I4.0 industrial data analytics process.

IV. The different Implementations

As seen in Figure 2 the database writers commit the sensor data to the databases (see Section IV-A). The database readers (see Section IV-B) read the written sensor data and then calculate a sum for one machine cycle of sensor data, thus simulating light analytical processing.

Each writer and reader is written for each database system, so that we have at least three writing and three reading applications for comparison purposes. Moreover, because of the architecture of a database server (NoSQL vs. RDBMS vs. TSDB), we also had to implement different ways of storing the data. Additionally, during early development and because of recommendations at MongoDB, we decided to write an additional two variants in data storage and transport for both MariaDB and MongoDB to optimize a degradation behavior that occurs with what we call a ”naive” implementation. This sums up to five different writing and five different reading applications for our benchmark. Later, we will introduce a sixth and seventh reading variant of the InfluxDB application for being able to compare the reading better to the two variants of the other database systems.

The difference is that in theoretical database lecture, we are taught to normalize data for being able to freely filter and combine without having redundant entries. This typical (or ”naive”) way of implementing database architectures in our case leads to a performance degradation because of the high frequency of values. The trouble lies in the frequent updates of index structures and files on the hard disk. As a typical hard disk drive (HDD) is able to perform $\approx$ 100 input or output operations per second (IOPS), a traditional, normalized approach would not be able to handle such high frequency inserts of 1000 sensor reads per second. For still being able to measure and compare this traditional approach, we had to use solid state drives (SSD), which can reach about $\approx$ 100K of IOPS today, which in our scenario is enough for not being the bottleneck.

Nevertheless, by using what we call ”advanced” implementation strategies, optimized towards this special problem of high frequency inserts of data, we can reduce the number of necessary IOPS of the database server and improve the performance for having more capacity left to address increasing workloads.

The ”advanced” approach caches all the sensor data of one second (in our case, which could be more in reality, if needed) and then only writes our $5 * 1000$ data points in one operation to MariaDB and MongoDB. Interestingly, InfluxDB does the same on the server side and caches the data until a larger block is filled, before it writes the data down to the disk. Because InfluxDB does this itself (as it is specialized for time series data), we only had to write one version for this database server.

The aggregation of the sensor data in case of MariaDB und MongoDB, which theoretically should be a lot faster than the single insertion of data, unfortunately, has one big flaw that has to be considered before following this approach: As the data has to be cached in the writing application, it is not available on the reading side, because it is not stored in the database, yet. So when the monitoring application needs live view of data, an additional pipeline from the sensor read to the live monitoring system has to be established. As we write the data to a database first, which allows advanced database querying techniques not available in client software without additional libraries, but does not allow live monitoring, we decided that this flaw has no impact in our case. Moreover, as our reading algorithm needs complete machine cycles for analysis, our scenario is not in the need of having live access to the sensor data. As we also believe that many analytical applications have no advantage of live data and still work fine when the data is available only one or more seconds later, this flaw is not further discussed in this paper.

Each implementation itself is optimized concerning runtime complexity for reduced influence on the benchmarks by using memory usage techniques (i.e., stack memory allocation), database specific techniques (i.e., prepared statements), and general algorithmic design principles. This way, we are able to achieve optimal database performance results. This is also a reason for the usage of C and C++ as underlying languages, because this allowed us to tune our algorithms towards optimal performance, which would not have been possible, for example, with a language that has no pointer arithmetic or that uses a garbage collector. It could, of course, also be possible, to compare the optimized implementations in C and C++ against implementations without optimizations like prepared statements, and others, but as this paper is about different database processing pipelines and wants to max out the performance, not lay out all possible code optimization techniques, this is not covered here.

A. Database Writer and Database Structure

Now, every millisecond, the simulator (i.e., sensor) writes a new measurement value into its local memory, overriding the previous measurement. The sensor uses an internal clock for this, which wakes up every millisecond. Our simulator uses the `clock_nanosleep` function for simulating this behavior of updating a measurement each millisecond, like a real sensor could do.

The database writer application running on the single-board computer now reads that new measurement from the simulator (i.e., sensor) over the IPC queue, five values per millisecond, so that we have received 1000 measurements per second with five sensor measurements each, at the end of a second. As the database writer is running on a computer that has a system clock being synchronized to the real time (via network time protocol), compared to a real sensor that might not have a synchronized system time or no time at all, the timestamp for the datapoint is added by the database writer. Listing 1 shows the structure of the datapoint that is then sent to the database: It contains the added timestamp and the set of the five digital values read from the sensor (i.e., simulator). The timestamp added has a resolution of one nanosecond (for further adjustments to even higher frequencies than 1000 values per second) and uses $8 + 4 = 12$ bytes of memory for representing the second as `long` and the relative nanosecond part of the corresponding second as `int`. The digital values are represented as 16-bit (2 byte) integers. Thus one datapoint uses $12 + 5 * 2 = 22$ bytes of memory in sum.

Listing 1. One datapoint

```
struct data_point
{
  int64_t s;
  int32_t ns;
  uint16_t measurements[5];
};
```

1) MariaDB – Individual datapoints: This is a straightforward implementation of the data structure (we called it ”naive” earlier). We sequentially store each datapoint as five rows in the database table, so we have a normalized table structure (i.e., each measurement gets its own row). This results in a high rate of operations on the database ($1000 \frac{\text{writes}}{\text{second}}$). The impact could be even higher, if we had written each measurement in a single commit, but we aggregated each sensor readout (with five values each) in one commit, so that five rows are inserted per commit. This means that we have 5000 $\frac{\text{rows}}{\text{second}}$ of sensor data. Moreover, it means that the index also is updated $1000 \frac{\text{times}}{\text{second}}$ and the disk probably has a four to five times higher load, because it has to write the database log, the data itself, the index update, file system table updates and maybe also file access and or modification times. For this reason, this normalized approach is not suited for a traditional hard disk drive.

Table I shows the structure of the data, which is based on the `data_point` structure. A compound index is set on *second* and *nanosecond* for later retrieval of single measurements in our reading benchmark. *number* describes the index of the sensor, so in our test, a value from 0 to 4 for the five different sensor values, *measurement* is the corresponding sensor value.

TABLE I. MariaDB - Table structure of individual datapoints

Field	Type
second	bigint(20)
nanosecond	int(11)
number	smallint(5) unsigned
measurement	smallint(5) unsigned

Our C implementation of the algorithm based on `libmariadb` uses prepared statements, struct data binding and a single commit for five rows per sensor read for higher performance. These performance optimizations, the explicit transaction preparation and commitment and the manual creation of tables necessary for a relational database (not needed in the other database systems) make the MariaDB code the largest and most complicated of all our implementations.

2) MariaDB – Bulk Datapoints: As mentioned in Section IV and in the preceding paragraph, this implementation collects all datapoints for one second in memory (i.e., same `int64_t s` value), creates one JSON document per second and writes this document out as one row per second. Thus, we can store the data in bigger units, which reduces the load dramatically. Instead of $1000\frac{\text{writes}}{\text{second}}$ with 5000 rows per second, we now only have $1\frac{\text{write}}{\text{second}}$ with 1 row per second and only 1 index update per second.

For the storage of the JSON document, the table structure is a little bit different. In MariaDB, the JSON field is an alias for longtext field. Yet, the specialized JSON query commands in MariaDB work for such fields, which could allow to later query within the JSON data directly, though we did not use this approach in our reading benchmark, because of other reasons discussed later. Table II shows the used structure with this approach. *second* has an index, again for faster retrieval of the data later in our reading, *size* contains the length of the text in the JSON field, and *measurements* is the mentioned JSON document, built in linear time according to the example in Listing 2. The JSON document now contains the nanoseconds with the related measurements according to the `data_point` structure. We did not save the amount of measurements (five) within the JSON document, because we have a defined `data_point.measurements` size of five. This means that we can save this space in our scenario, as we will never have fewer or more than five measurements per sensor read per millisecond in our datapoints.

TABLE II. MariaDB - Table structure of datapoints in bulk

Field	Type
second	bigint(20)
size	int(10) unsigned
measurements	json

Listing 2. MariaDB - JSON Documents

```
{
 "measurements": [
   {"ns":346851124,"m":[389,792,602,315,552]},
   {"ns":346933204,"m":[516,794,634,317,559]}
   ...
]}
```

The difficulty of this adaption is similar to the original, individual approach, but in one detail is quite complicated: As it is theoretically impossible to know how many measurements one cycle will have (most of the time the stated 5000 measurements per second in our case, but this is not guaranteed in a real-world scenario), we needed to implement a dynamically growing character field for the JSON data. We also needed to change the struct binding in the transaction commitment for honoring the dynamical length of the JSON data. As dynamic arrays copy their memory contents multiple times while growing, this theoretically reduces performance. But as we use a global string variable and as the length of the datapoints as string is always very similar, the dynamic string normally only grows during the first iteration and then is not reallocated anymore in subsequent calls. This is why we can state that the JSON document is built in linear time during the benchmark.

3) MongoDB – Individual Datapoints: As a document-orientated database, MongoDB allows for flexible schemata, which allows us to leave out any schema creation. Data is internally organized in BSON (Binary JSON) documents, which are in turn grouped into collections.

Saving the individual datapoints according to Listing 1, each measurement would be a document with the time of measurement and the values organized as a JSON-array. This is like a mixture of the individual MariaDB and the bulk MariaDB approach, just with a JSON document per sensor read, so with 1000 JSON documents per second.

The database supports setting an index on a field of a document, so to support further searching of measurements, we set an index on time as we have in MariaDB. With such a structure, similar to the individual MariaDB approach, numerous documents are created per second. After each document has been added, the index also needs to be updated, which results in a similarly high computational effort as with the individual MariaDB approach.

The software for the MongoDB database writer is written in C++ and uses `mongocxx` in conjunction with the `bsoncxx` library. The document orientated approach of MongoDB makes designing data structures very flexible. However, the freedom leads to more work on the initial programming approach, as there is no schema for clear orientation. Also the need to link two libraries creates additional effort.

4) MongoDB – Bulk Datapoints: As already stated in Section IV-A2, we can store a collection of datapoints at once. In MongoDB, we can implement this with the structure shown in Listing 3.

Listing 3. Datapoints in bulk

```
{
  "time" : ISODate("2018-02-12T19:56:49Z"),
  "measurements" : [
    { "time" : ISODate("2018-02-12T19:56:49.13
        5Z"), "sensors" : [ 0, 0, 0, 9, 347 ] }
        ,
    { "time" : ISODate("2018-02-12T19:56:49.13
        6Z"), "sensors" : [ 0, 2, 4, 10, 351 ]
        },
    ...
  ]
}
```

The time value of the top-level document has again a precision of one second. This document holds all datapoints sampled during this second in an array, similar to the bulk MariaDB variant. Every nested document contains the exact time of its measurement and the actual sensor-values. This is similar to the MariaDB JSON document, though the time has no extra field and the nanoseconds have a defined format (ISODate) that bloats the document. Of course, we thought about using the nanosecond solely, like with MariaDB, but MongoDB recommended the general use of ISODate, so we followed this recommendation.

With this approach, similar to MariaDB, the index has to be updated only once per second resulting in optimized write performance. Nevertheless, it must be considered that in this case only a whole second but no parts of it can be retrieved efficiently. Although MongoDB can retrieve values directly within a JSON document, similar to MariaDB, the document first has to be loaded and parsed, before the database server can find the queried value. However, because of the high increase in write throughput, we accept this drawback.

Similar to the MariaDB approach, the application creates a document for a whole second and fills it until the second has passed. Then it sends the document to the database server once per second.

The documentation for MongoDB provides examples for the use of streams and basic builders consisting of function calls. We followed these examples as well as possible with our implementation. Yet the use of nested structures and the nature of C++-streams is poorly documented in the doxygen-based manuals, increasing the implementation effort.

5) InfluxDB: As a time-series database InfluxDB has a strict schema design we have to follow. Every series of data consists of points. Each point has a timestamp, the name of the measurement, an optional tag, and one or more key-value pairs called fields. Timestamps have an accuracy of up to one nanosecond and are indexed by default. The name of the measurement should describe the data stored. The optional tags are also indexed and used for grouping data. Data is retrieved with InfluxQL, a SQL-like query language. Data is written using the InfluxDB Line Protocol (Listing 4). This is how the protocol is built up: The first string is the name of the measurement, here simply *measurement*. Subsequently follow the key-value pairs with five measurements and finally a timestamp in nanosecond precision.

Listing 4. InfluxDB Line-Protocol example

```
measurement m0=0, m1=0, m2=0, m3=9, m4=347
    1518465409001000000
```

The database writer for InfluxDB is written in C. The default API for InfluxDB is HTTP. For our high-frequency write access however, we chose the UDP protocol, which is also supported. We believe that the overhead is smaller when using the UDP protocol, because HTTP is a very verbose protocol, especially when sending small requests very frequently like in our case. So in this case, the data is composed into the Line Protocol with simple C-String functions and sent with the Unix function `sendto`. Since no external code is required and a custom design of the data structure is not possible, using the database is straightforward and fast to implement.

Additionally, InfluxDB also offers built-in functions to process data statistically and a client library is not necessary, which is a benefit for software developers using it, when small overhead is desired.

Of course, the choice of UDP has the probability of data loss, which is acceptable in our use case, because we have very high frequency data and the loss of single measurements could be compensated, for example by interpolation, or just be ignored, when reading. For enabling the UDP service of InfluxDB, the OS was configured corresponding to the information provided by InfluxData [19]. Because we use dedicated computers linked together with a nonblocking switch, we had no measurable UDP loss in our tests. When using a wireless link between database writer and database server, maybe the HTTP protocol should be preferred despite the large overhead.

With InfluxDB, additional implementation variants were not needed, because the scheme is fixed and InfluxDB itself caches, accumulates in bigger units and even compresses the data, before it is written to disk. This is why InfluxDB is the only database system that works out of the box for time series data without additional effort in optimizing the datapoint storage.

B. Database Reader

In our reading part of the benchmark, we want to simulate an interactive monitoring application. We use the database reader application variants to retrieve the data for three randomly selected machine cycles per second. The rate of $3\,\frac{\text{reads}}{\text{sec}}$ is quite high for an interactive application, where a user selects individual machine cycles for further analysis, but we assume the user clicks very fast and often. A non-interactive application, e.g., condition monitoring for predictive analytics, will process the consecutive cycles at the end of the data set, not randomly selected cycles. As the most recent data might still be cached inside the database or page cache of the operating system, and could also be selected by just jumping to the end of the data set minus a selected range, our random selection is an adequate usage scenario between worst and best case. As we select by time and always have set indexes on the time, the selection should not trigger sequential scans of the data but make usage of our selected database structures.

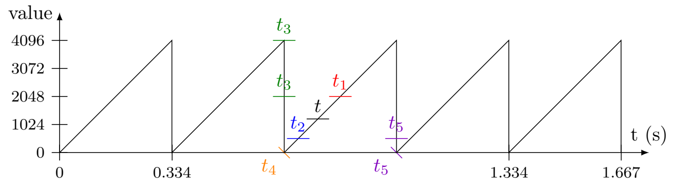


Figure 3. Strategy to find start and end time for a machine cycle

Figure 3 explains our implementation of the reading algorithm. We start at t, which is randomly selected in our benchmark. From there we search forward to t_1, the first point after t where the data of the abscissa-track is bigger than half the amplitude. This value is fixed, as we know the range of the abscissa amplitude. From t_1 we search backward to t_2, the first point before t_1 where the data of the abscissa-track is smaller than an eighth of the amplitude. From t_2 we search backward to t_3, the first point before t_2 where the data of the abscissa-track is (again) bigger than half the amplitude. The start point of our cycle is at t_4 the minimum value of the abscissa-track in the range $t_3 \leq t_4 \leq t_2$. The end point of the cycle is at t_5. We find t_5 by searching forward from t_1 for the first point where the data of the abscissa-track is (again) smaller than an eighth of the amplitude.

This strategy can be implemented by issuing several `SELECT` statements to the databases. In the following we refer to this strategy as ”individual”.

Alternatively we can simply retrieve data for a longer time span around the time t in question, e.g, $\lfloor t - 1\,\text{sec} \ldots \lfloor t$ for

$t - \lfloor t < 0.5$ and $\lfloor t \ldots \lfloor t + 1\,\mathrm{sec}$ for $t - \lfloor t \geq 0.5$, and then perform the same search operation on the retrieved data in memory. This alternative way is necessary for the write strategies in sections IV-A2 and IV-A4 where the database structure does not allow to retrieve individual samples with high performance. We said that the database allows individual selection with JSON query commands, but as the data has to be loaded and parsed in either way, we could just send it to the client application and work with the aggregated second on the reading side, again, just as when we aggregated on the writing side. This leads to the five (or seven with the InfluxDB alternatives) different reading applications, so that we have a direct comparison to the five writing applications. Moreover, we believe that the direct queries are slower, for example in MariaDB (see Section IV-A2), because for the selection of several time points in one document, the data has to be loaded several times in sequence for each select statement. When we copy the whole JSON block to the client side, parse it in linear time O(n) and search solely in-memory, we can guarantee a single read on the database side and reduce the possible load on the server, as well as guarantee the linear searching time on the client side. In the following we refer to this strategy as ”bulk”.

For InfluxDB, while we can retrieve individual datapoints, we also had a bulk variant that read the whole block with all measurements as with the MariaDB und MongoDB bulk variants, and we implemented a ”bulk-1” variant of this strategy, which reads only the data in the column corresponding to the abscissa track, i.e, the machine angle, as a block, and reads the data for the other four columns in a second read operation spanning $t_1 \ldots t_5$. The reason for this is that a machine cycle in our benchmark is ≈ 0.33 sec and when we read a block of a whole second, many measurements (of ≈ 3 machine cycles) are transmitted, though only around one third of the data (one machine cycle) is really needed. With our MariaDB and MongoDB bulk variants, we had to transmit the whole block to prevent the database from parsing the data multiple times, but InfluxDB has the flexibility to load blocks of different datapoints. We were able to investigate the transmission bandwidth with this behavior, at least for InfluxDB, too. This is why we have seven reading applications.

V. Testing

Most applications in our context face limitations in terms of computing power and network bandwidth. For example, IoT sensor devices in the field of smart home are often battery powered and wirelessly connected. In I4.0 context, sensors have no computing power for analytical data processing and they often are connected via proprietary protocols and interfaces. Although servers are often better equipped, the client systems like tablets, smartphones or notebooks have limited computing capacity in comparison. Consequently, when writing data from the sensor to the server, in our scenario, we measure the load on the single board computer, the load on the server, and the network load. When reading data, we measure the load on the server, the load on the client, and the network load as well. With these measurement parameters, we have a good overview of all critical components of the industrial data analytics process.

For the concrete measurement process, we defined the following: The system load on the computers is measured in terms of CPU and memory usage. For this, we created a script, which runs the specified application for 15 minutes, after a warm-up phase of 5 minutes for filtering out cold-start phenomenona like caching data in the operating system page cache, or CPU clock changes due to heat or power usage (especially on the single board computer, which reduces its CPU clock under heavy load).

Before the test run stops the application, it uses two Linux-System commands to gather the following parameters: L_{CPU} indicates the processor usage. We obtain this value with the Linux command `ps -p <pid> -o %cpu`, which returns a measure for the percentage of time the process <pid> spent running over the measurement time.

The maximum value for one core is always 100%. Therefore, on our 8-core single-board computer, the absolute maximum value would be $L_{CPU} = 800\%$. On the server with 4 cores, the absolute maximum value is 400%. The client has simultaneous multithreading enabled, so its 4 cores are doubled to 8 threads in the operating system, for an absolute maximum value of 800% instead of 400%.

L_{mem} indicates the memory usage in kByte. We use the amount of memory used by the process <pid> as the sum of active and paged memory as returned by the command `ps aux -y | awk '{if ($2 == <pid> ) print $6}'`. It outputs the *resident set size* (RSS) memory, the actual memory used, which is held in RAM.

L_{disk} shows the the amount of disk used by a database server. To determine this parameter, we first empty the respective database completely by removing its data folder. Then we start the database and measure the disk space of the folder before we test. After the test we measure again the used disk space. L_{disk} is then calculated as the difference between the folder size after the test and the folder size before the test with an empty database folder. `du -sh <path>` is used to get the disk consumption of the respective data folder.

To put the results in perspective: Our benchmark application gathers and transmits ≈ 26.4 MByte of raw data during the 20 minutes of our test. This is calculated as the following: We have $5 * 2$ bytes of sensor data plus 12 bytes of timestamp data per sensor read. We read each sensor 1000 times per second. This sums up to $(5*2+12)*1000 = 22000$ bytes per second, or 22 kBytes per second. We measure 20 minutes (of which 15 minutes are used for the CPU, memory and network measurement). So we have $22\ \mathrm{kBytes} * 60\ \mathrm{s} * 20\ \mathrm{min}/1000 = 26.4\ \mathrm{MBytes}$, if no sensor value is omitted (which can happen in the UDP InfluxDB writing test).

L_{IO} then shows the average disk input-/output in $\frac{\mathrm{kBytes}}{\mathrm{s}}$ caused by the database writing operation being measured using the `pidstat` command. As we use a SSD on the server, we have raised the hardware limit of the IOPS a lot, compared to a traditional HDD.

L_{net} finally shows the average bandwidth used on the network. We obtain that value with the command `nload`. We run our test in the university network and therefore have additional external network load (for example DHCP packets, ARP requests, discovery services, ...). However before each test, we observe the additional network load for some time, and as it was always smaller than $1\frac{\mathrm{kBytes}}{\mathrm{sec}}$, we neglect it.

To put L_{IO} and L_{net} in perspective: In our benchmark we transfer $22\frac{\mathrm{kBytes}}{\mathrm{sec}}$ from the simulator to the database writers, but as the raw data has to be packed into UDP or TCP packets, which itself are packed into ethernet frames, the network load has to be larger than the raw data. Especially in case of the RDBMS and NoSQL database, we have an additional communication protocol overhead (like SQL in case of MariaDB), which adds more data to transmit than, for example, the InfluxDB Line Protocol does. Our network bandwidth results show the data plus the overhead for being able to see what the individual applications really need as underlying network throughput.

As our network connection between the tested systems is always $1\frac{\text{Gbit}}{\text{sec}}$, our hardware network limit is high enough for our benchmark. Nevertheless, when using slower or weaker connections (i.e., wireless network), the network bandwidth, that may also differ in time, has to be considered as a hard limit.

Before each test, we reboot the operating systems used in the test. In case of a write test, we then erase the database folder before starting the database. Then we turn on the simulator of the sensor data (in case of a write test), the database server (always), log in to the single-board computer (in case of a write test) or the client (in case of a read test) and start the database writer or reader software for the currently active database. The actual benchmark begins with restarting the database reader or writer after the warm up phase. The database folder of course is not deleted after the warm up phase and is measured over the whole 20 minutes, because this could interfere with the other measurement parameters like CPU, memory and disk IOPS. The reason for this is that index structures like B-trees are never completely filled, but at least 50%. This is for faster insertion in consecutive insert operations, as the tree does not have to be updated for each insert operation. When the database is empty though, the tree is small and has to expand fast during the first minutes, as there are, in absolute numbers, not enough empty buckets for holding all the high frequency values. After the warm-up phase, we believe that the index tree is big enough (as $\frac{1}{4}$ of the data is already written and the tree has enough space left for approximately another fourth, when it is filled between 50% to 75%), so that tree rebalancings do not occur often, and have no noticeable impact on the other measurement parameters anymore. Of course, this example calculation of the tree structure is not exactly precise, but explains very well why we do not clear the database folder again after the warm-up phase.

So in the end, we let the system gather the CPU, network, memory, and disk IOPS data only for the 15 minutes phase and the results from the 5 minutes warm-up phase are thrown away, except for the disk usage, as stated. The performance data detailed in Section VI is gathered by two scripts running on the computers used for the test, which start and stop the applications and measure the resource usages. We test each database server and each writing-reading application bundles sequentially, as testing everything in parallel would interfere with each other's measurements.

VI. Results

Tables IV and V show our results for writing, respectively reading. Figures 4 and 5 then visualize the data in relation to the maximum values for each criterion.

To directly compare all our candidates, we calculate a combined score by weighing the parameters. We think that there are parameters that are more important in an IoT and I4.0 context, than others. We distinguish between critical (weight of 3), important (weight of 2) and normal (weight of 1).

Since we find that the CPU is the most critical and limiting parameter, we give it a weight of 3 on the server. On the client it might even be more limiting due to the Banana Pi's low power design, which also justifies a weight of 3.

In absolute terms, the RAM usage on server and client was very small compared to the available RAM, and therefore we give it a normal weight (with 1).

As already stated, we used an SSD for our benchmarks, which would not be a limiting factor in our tests. Nevertheless, as it is possible to have a server with an HDD, which in that case would be critical, we value the L_{IO} parameter with a mixture of both scenarios as important (so 2).

As the disk usage already correlates with L_{IO} (i.e., both disk parameters), we weight it with 1, so that the impact of the disk results is in a decent relation to the other component's results. Additionally, the disk usage is not critical, as disk space is easy to expand, but for example a high-performance CPU can not easily be doubled to increase processing power for non-parallel algorithms.

In wide areas, network-bandwidth could be a limiting factor, especially when we have wireless connections, for example in smart home scenarios. As in our main context of IoT and I4.0, where we believe it is easier to connect the machines with cables (as they need a large power supply, too), we give the network an important weight (of 2). Wireless scenarios would require an even bigger weight.

Finally, we take the subjective difficulty of our implementations into account. We grade on a scale from 5 (i.e., most difficult), to 1 (i.e., easy), and weight this parameter with a normal weight. The individual rating is determined by the experience with the client implementation described in Section IV. We know that this parameter is not objective, but as we explained in Section I, the developer experience is an important argument in deciding which database is selected. Nevertheless, because we do not want to give a subjective parameter an important weight, we only weight it with 1, as stated.

Based on the gathered result data, we calculate a score according to the following formula, where
$i = \{net, CPU, mem, IO, \ldots\}$ and
$\text{imp} = \{MongoDB_{individual}, MongoDB_{bulk}, \ldots\}$:

$$Score_{\text{imp}} = \left[1 - \frac{\sum_i \left(\frac{L_i}{max_i(L_i)} \cdot weight_i\right)}{\sum_i weight_i}\right] \cdot 100\%$$

In this formula, we first normalize the resource usage to the maximum value for each column in Tables IV and V. Then we sum up the weighted normalized values, and normalize again to the sum of all weights. Lastly we "invert" the value by subtracting it from 1. Thus, the best score is 100%.

Table III shows the aggregated scores for writing and reading. The total score is the average between write- and read-score. For writing, we only have one InfluxDB application, which is the reason for the same score in all three InfluxDB-Writing cells. The ranking and data differs from our

TABLE III. Scored Ranking

	Writing		Reading		Total	
Implementation	Score	Rank	Score	Rank	Score	Rank
MariaDB Individual	15%	7	71%	5	43%	6
MariaDB Bulk	59%	1	83%	2	71%	1
MongoDB Individual	35%	6	50%	7	43%	7
MongoDB Bulk	54%	2	60%	6	57%	5
InfluxDB Individual	39%	3	93%	1	66%	2
InfluxDB Bulk	39%	3	80%	4	60%	4
InfluxDB Bulk-1	39%	3	82%	3	61%	3

previous benchmark in [1], as we repeated all benchmarks in a completely new environment, due to the fact that the previous setup and hardware was not available anymore. In the previous paper, we also used analog circuitry and a signal-generator for generating our sensor data, but as this device was no longer available, for this benchmark, we use a simulator to generate our data. Moreover, we adapt the weights to be more objective, which also has an impact on the ranking. Nevertheless, the order of the ranks is the same as last time, so we think our new setup is quite comparable.

A. Write Benchmark Results

In the write benchmarks, all five implementations show little CPU usage with $L_{CPU_{Server}} \leq 5\%$ on the server and with $L_{CPU_{Client}} \leq 14\%$ a significant but not critical CPU usage on the Banana Pi. MongoDB individual, MongoDB bulk and MariaDB bulk are the least demanding implementations with respect to the server's CPU. MariaDB bulk also is least demanding on the client's CPU. We can see that the server CPU usage is in all cases far from critical, with the client CPU usage being much higher.In cases of InfluxDB and MariaDB Individual, as the CPU usage is a lot higher, this can limit the amount of sensor information or the frequency, that the database is able to process, earlier, than in the other scenarios.

The parameters $L_{mem_{Server}}$ and $L_{mem_{Client}}$ are fairly uniform and non critical. We can see that the server's memory usage is a lot higher in all scenarios, compared to the client's memory usage. This is because of the database servers being complex software products, requiring some memory to operate. Nevertheless, with typical servers having often more than 16 GBytes of memory nowadays, the memory usage is always uncritical. As the client's memory usage in the write benchmarks is only a few megabytes per test run, we can say that this also has no real impact. It is interesting that the InfluxDB UDP Line Protocol writing application uses by far the least amount of memory on the client. This does make sense, because we had no additional libraries involved in this writing application.

L_{net} is also similar for all systems. With $L_{net} \approx 400$ kBit/s MariaDB bulk needs less network bandwidth. We think, this is because the data that is sent to MariaDB in the bulk variant, is transferred as binary data (i.e., bulk data binding) attached to the insert SQL statement. All other variants have either more individual transport operations, or transport the data with more describing variables. For example, InfluxDB always sends the full measurement name and the timestamp as full timestamp with nanosecond precision, while MariaDB bulk splits the timestamp into the second (transmitted once) and the nanoseconds since the last full second. MongoDB also is more verbose because of the JSON-format with the ISODate. We believe that this explains, why MariaDB bulk has the least network demand in writing.

In terms of disk usage, the compressing databases (i.e., InfluxDB) have a clear advantage. With $L_{disk} = 5$ MByte InfluxDB is the best in the test. Nevertheless, the bulk variants also need significantly less disk space than the individual variants. This is easy to explain, because in the individual data point rows, informations like the second are redundantly stored for each row. Moreover, the index structures are larger, as they have to reference more rows than in the bulk variants.

Concerning the implementation difficulty, InfluxDB was the easiest, needed the fewest lines of code, no additional libraries, and no schema to define. MongoDB Individual and bulk was much more difficult, because it needed two additional libraries (with BSON instead of JSON not being a well known data structure) and more effort in creating a schema for the document storage. MariaDB was most difficult in both cases, as in the individual case, the schema and index structure was more complicated (i.e., combined index, which needs attention in the definition because it is sensitive to the order of the definition of the columns), while in the bulk case, the schema and index were easier to define, but additional effort was needed for the buffering of the data up to one second, before the commit. Both MariaDB variants needed the most lines of code and had advanced techniques like prepared statements and bulk data binding applied for optimal performance, which increased the difficulty for the developer.

The way we wrote our data was relatively simple concerning its structure and insertion since we made no preprocessing for our analytical reading algorithm. Especially for InfluxDB this made the implementation very easy and straightforward because we could used the given schema. This led to a more difficult implementation on the reading side.

With our weighting, however, MariaDB bulk is the best ranked database. It needed the least resources concerning the critical CPU usage and the important network usage. Its disk usage was average, though the disk IOPS were worse than MongoDB and InfluxDB. Although its implementation was more difficult, it scored a little bit higher than MongoDB bulk, especially because of the much higher CPU usage of MongoDB. The InfluxDB writer only performed third, because it needed a lot more CPU resources on the server side, due to its buffering and compression mechanism.

B. Read Benchmark Results

In the read benchmarks, MongoDB individual could not keep up with 3 reads per second, as we defined in our scenario. This means that MongoDB individual has a noticeable latency in the interactive application use case. With $L_{CPU_{Server}} \approx 87\%$ (single thread, so maximum is 100% here) it almost blocked our database server, which would further delay parallel queries from other reading clients.

MongoDB bulk was better concerning the CPU usage on the server side, but caused a high load on the client with $L_{CPU_{Client}} \approx 16\%$. We attribute this to the JSON-formatted entries that the client had to parse. As we used a library for parsing, we could not optimize the parsing process towards this special JSON format, like we could do in MariaDB bulk.

This is why MariaDB bulk has the least CPU impact, as it only refers to loading blocks of data and sending them over without further processing, because the client application parses the JSON data. This might have been different if we had used the server side JSON query mechanisms that we wanted to avoid because of the impact on the CPU usage. MariaDB Individual obviously causes a higher CPU load as it has to look up more individual table entries, and InfluxDB also uses more CPU power than MariaDB, as it has to decompress the data it previously compressed while saving.

For all approaches but MariaDB Individual, $L_{IO_{Server}}$ is below the data rate of $22\ \mathrm{kByte/s}$ that is theoretically required for reading 3 machine cycles per second (with each being $\approx 0.33\ \mathrm{sec}$), i.e., the same data rate as in the write benchmarks. We attribute this to the databases or operating system caching data in memory and the relatively small amount of data stored during our test ($\approx 26.4\ \mathrm{MBytes}$). With several hundred gigabytes or even terabytes of data, we might have overcome this problem when randomly selecting machine cycles. In a real-world scenario, where sensor information for several hundreds of sensors is logged over many years, we believe the outcome for the disk IOPS in reading will be different to our benchmark.

The memory footprint $L_{mem_{Server}}$ is comparable to the writing benchmark, which is obvious in case of MariaDB and MongoDB being complex database systems. Interestingly, InfluxDB had much lower memory usage, which we believe lies in the fact that when writing, InfluxDB buffered and compressed the incoming data, while when reading, does not have to buffer anything and uses decompression methods normally being less memory intensive, as no code book has to be built up and held in memory.

Concerning the client memory, InfluxDB individual and bulk loading with all data were best in the memory usage.What surprised us was the memory footprint of InfluxDB, when bulk loading only the data of the abscissa track and then loading the machine-cycle in a second run (bulk-1) was tested. Although the CPU usage was lower, as less data had to be processed on the client side, the memory usage was clearly higher than in the other scenario. Unfortunately, we detected a memory leak in the code that was used for this benchmark, after the testing environment was already shut down and disassembled. We believe that the memory footprint should have been similar to the other two InfluxDB reading benchmarks.

The network bandwidth was much higher when reading than while writing. We think that this lies in the fact that in case of the individual queries, we sent six queries to the databases that each had to run three times per second over the network, and in case of the bulk queries, we always had to load two seconds of data (six machine cycles) three times per second for being able to select one machine cycle in the client application then. InfluxDB performed best in its individual case, having less impact than when writing, which does make sense, as the protocol needed to transmit only the filtered data.

The difficulty rating is that MongoDB was the easiest in this part, because the defined scheme used on writing as well as the ability of MongoDB to help us with the JSON (or BSON) parsing by its libraries, made it adequately difficult to implement. InfluxDB in its individual variant also was comparably easy to implement due to the given scheme and simple InfluxQL language. The bulk variants were more difficult in InfluxDB, because InfluxDB itself did not bulk-store the data, so we had to manually select more data, though InfluxDB could have filtered it for us server-side. We wanted to compare the resource impact of bulk variant reading in InfluxDB, but to sum up, this is not advisable, as the benefit compared to individual loading is small in the resource usage, but the implementation is a lot more difficult. MariaDB was by far the most difficult part, because even sending out the individual queries needed prepared statements and data binding mechanisms, which bloated the code a lot. The MariaDB bulk variant was the most difficult to implement, because we had to write a complete JSON parser ourselves (to be fair, only a highly optimized JSON parser for the underlying data structure was built, no general use parser) for guaranteeing linear parsing time for six in-memory queries, because we believe the that the database itself would have parsed the data multiple times.

With our weighting, InfluxDB individual is the best database for reading. This is because even though the overall CPU usage was higher than in case of MariaDB bulk, the other resource usages (like memory, disk IOPS and network bandwidth) were all lower. Moreover, it was relatively easy to implement, compared to the other variants.

We cannot recommend MongoDB individual, but MongoDB itself already recommends to bulk-write and -load the data. So at least, we can confirm this recommendation. MongoDB bulk, however, was still not very fast. We believe that this lies in the high CPU usage on the client, which is caused by the BSON library we used, which parses the document data in a more general and thus not optimized for this specific use case way.

MariaDB Individual on the hand end is normalized, like theory says is a good practice, but for this specific use case, the memory, CPU and disk usage on the server are quite high in comparison. MariaDB bulk is definitely the recommended way, despite its difficult implementation, as it causes nearly no load on the server and client, especially because we also implemented a highly optimized JSON parser for our own document structure.

C. Summary

In total we find MariaDB Bulk as the best implementations altogether with a score of 75%. However, the MariaDB implementations are quite complex and in our case, they were written by an experienced developer, who knew how to optimize the workload. This means that RDBMS are still capable of working with the types of data we investigated.

With 66% the InfluxDB individual implementations are not far behind, while being much easier to implement. We believe that if the developer is willing to learn a new database system for this use case of time series data management in the context of IoT and I4.0, InfluxDB might be the right choice.

VII. Conclusion

We introduced a complete benchmark set that was inspired by real world scenarios from IoT and I4.0 scenarios and benchmarked a set of three different types of database systems with one or more variants of writing and reading applications to simulate the behavior of high frequency monitoring and predictive analytics in the context of the industrial data analytics process.

While we presented MongoDB as a good candidate in [1], we had to observe a lack of performance in the read benchmarks. This might be caused by our high-frequency, low data volume read application. MongoDB obviously is better for larger documents with different schemata than for relational structured time series data.

MariaDB bulk scored best in the write benchmarks with a low resource requirement, and it performed well in the read benchmarks, especially because we were able to optimize a lot, like custom JSON parsing. This demonstrates that 'classic' RDBMS are able to keep up with more modern architectures like the one we benchmarked, although they put some strain on L_{IO}, L_{disk} and the developer.

The TSDB InfluxDB individual showed reasonably good write and read performance while requiring the least amount of database know-how. The naive read and write implementations scored quite well, but we think that in very large installations, the compression and server-side buffering mechanisms can lead to an earlier exhaust of resources than when using a RDBMS. If the server is more than capable enough and if the developer wants an easier implementation, then InfluxDB can be recommended.

With our results, individual developers and companies have a base for deciding which database system, and how much effort and resources are needed to implement high frequency monitoring and analytical processing techniques for improving their overall product.

ACKNOWLEDGMENT

This work was supported by the European Union from the European Regional Development Fund (ERDF) and the German state of Bavaria.

TABLE IV. Test Results: Write Benchmarks

	Server			Banana Pi		Infrastructure		Difficulty
	$L_{CPU_{Server}}$	$L_{mem_{Server}}$	$L_{IO_{Server}}$	$L_{CPU_{Client}}$	$L_{mem_{Client}}$	L_{net}	L_{disk}	
MariaDB Individual	5%	170.835 kByte	1840 kByte/sec	9,4%	2.096 kByte	715 kBit/s	88 MByte	5
MariaDB Bulk	1%	160.834 kByte	417 kByte/sec	6,5%	2.232 kByte	395 kBit/s	32 MByte	5
MongoDB Individual	1%	257.166 kByte	71 kByte/sec	13,9%	3.561 kByte	710 kBit/s	58 MByte	4
MongoDB Bulk	1%	137.701 kByte	56 kByte/sec	8,9%	3.607 kByte	617 kBit/s	15 MByte	4
InfluxDB	5%	147.444 kByte	37 kByte/sec	11,7%	284 kByte	785 kBit/s	5 MByte	1
Weight	3	1	2	3	1	2	1	1

TABLE V. Test Results: Read Benchmarks

	Server			Client		Infrastructure	Difficulty
	$L_{CPU_{Server}}$	$L_{mem_{Server}}$	$L_{IO_{Server}}$	$L_{CPU_{Client}}$	$L_{mem_{Client}}$	L_{net}	
MariaDB Individual	7%	208.896 kByte	46,9 kByte/s	0,4%	4.427 kByte	1025 kBit/s	4
MariaDB Bulk	0%	154.544 kByte	5,2 kByte/s	0,6%	3.979 kByte	3520 kBit/s	5
MongoDB Individual	87%	188.737 kByte	3,5 kByte/s	1,4%	5.487 kByte	6750 kBit/s	3
MongoDB Bulk	11%	121.252 kByte	1,6 kByte/s	15,9%	2.564 kByte	3520 kBit/s	3
InfluxDB Individual	8%	40.605 kByte	0,9 kByte/s	1,3%	1.533 kByte	530 kBit/s	3
InfluxDB Bulk	12%	37.272 kByte	1,1 kByte/s	7,0%	1.556 kByte	1898 kBit/s	4
InfluxDB Bulk-1	7%	26.970 kByte	1,2 kByte/s	3,4%	28.979 kByte	1025 kBit/s	4
Weight	3	1	2	3	1	2	1

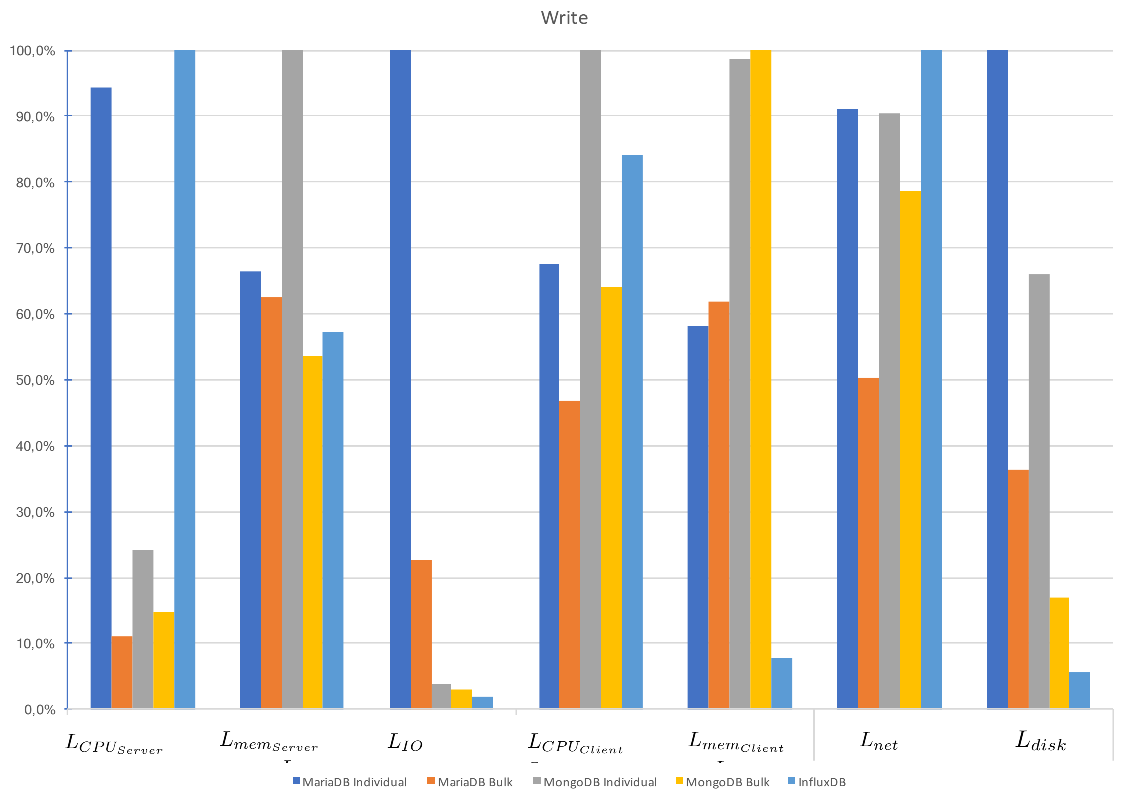


Figure 4. Overview of all Write Benchmark Values (normalized to respective Maximum)

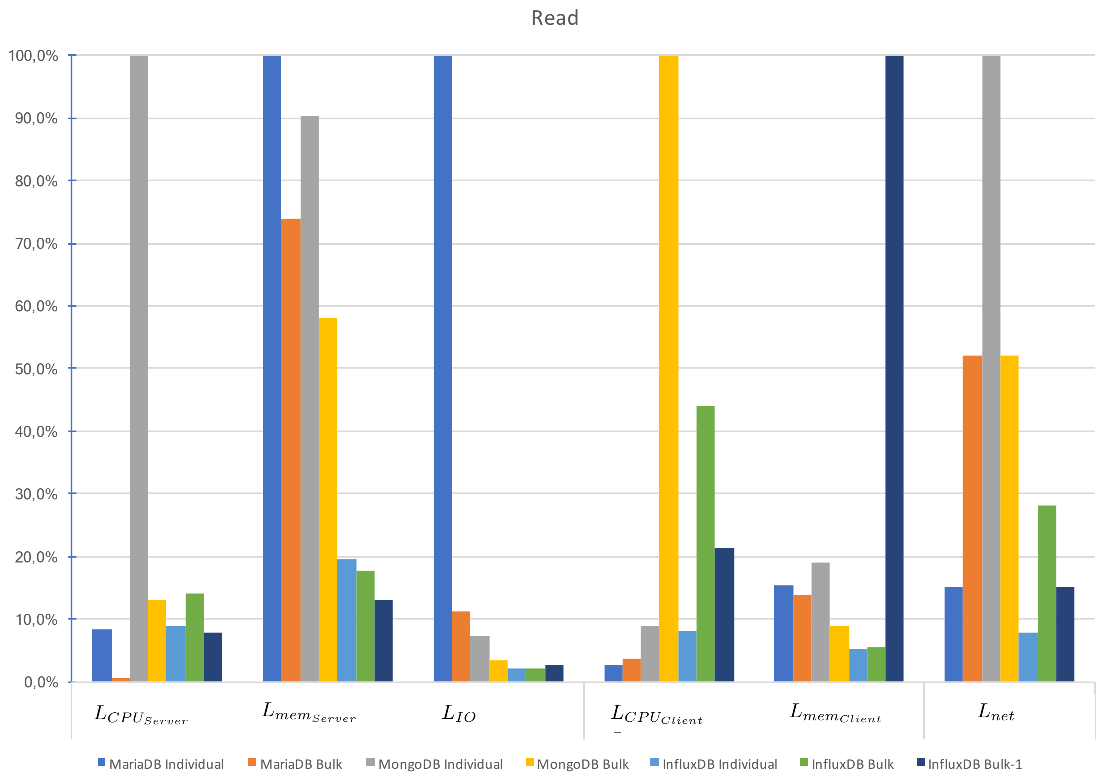


Figure 5. Overview of all Read Benchmark Values (normalized to respective Maximum)

References

[1] D. Arnst, V. Plenk, and A. Wöltche, "Comparative Evaluation of Database Performance in an Internet of Things Context," in Proceedings of ICSNC 2018 : The Thirteenth International Conference on Systems and Networks Communications, Nizza, October 2018, pp. 45 – 50.

[2] D. Wang, J. Liu, and R. Srinivasan, "Data-driven soft sensor approach for quality prediction in a refining process," IEEE Transactions on Industrial Informatics, vol. 6, no. 1, Feb 2010, pp. 11–17, URL: https://dx.doi.org/10.1109/TII.2009.2025124 [retrieved: 2018-08-14].

[3] G. Köksal, İ. Batmaz, and M. C. Testik, "A review of data mining applications for quality improvement in manufacturing industry," Expert Systems with Applications, vol. 38, no. 10, 2011, pp. 13 448 – 13 467, URL: http://www.sciencedirect.com/science/article/pii/S0957417411005793 [retrieved: 2018-08-14].

[4] F. Chen, P. Deng, J. Wan, D. Zhang, A. V. Vasilakos, and X. Rong, "Data mining for the internet of things: Literature review and challenges," International Journal of Distributed Sensor Networks, vol. 11, no. 8, 2015, p. 431047, URL: https://doi.org/10.1155/2015/431047 [retrieved: 2018-08-14].

[5] J. Lee, H. D. Ardakani, S. Yang, and B. Bagheri, "Industrial big data analytics and cyber-physical systems for future maintenance & service innovation," Procedia CIRP, vol. 38, 2015, pp. 3 – 7, URL: http://www.sciencedirect.com/science/article/pii/S2212827115008744 [retrieved: 2018-08-14].

[6] C. S. Jensen, D. Lin, and B. C. Ooi, "Query and update efficient b+-tree based indexing of moving objects," in Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30, ser. VLDB '04. VLDB Endowment, 2004, pp. 768–779, URL: http://dl.acm.org/citation.cfm?id=1316689.1316756 [retrieved: 2018-08-14].

[7] S. Acreman, "Top 10 time series databases," URL: https://blog.outlyer.com/top10-open-source-time-series-databases [retrieved: 2018-08-14].

[8] A. Bader, O. Kopp, and M. Falkenthal, "Survey and Comparison of Open Source Time Series Databases," Datenbanksysteme für Business, Technologie und Web - Workshopband, 2017, pp. 249 – 268, URL: http://btw2017.informatik.uni-stuttgart.de/slidesandpapers/E4-14-109/paper_web.pdf [retrieved: 2018-08-14].

[9] D. Namiot, "Time series databases," in DAMDID/RCDL, 2015, URL: https://www.semanticscholar.org/paper/Time-Series-Databases-Namiot/bf265b6ee45d814b3acb29fb52b57fd8dbf94ab6 [retrieved: 2018-08-14].

[10] S. Y. Syeda Noor Zehra Naqvi, "Time series databases and influxdb," Studienarbeit, Université Libre de Bruxelles, 2017, URL: http://cs.ulb.ac.be/public/_media/teaching/influxdb_2017.pdf [retrieved: 2018-08-14].

[11] A. M. Castillejos, "Management of time series data," Dissertation, School of Information Sciences and Engineering, 2006, URL: http://www.canberra.edu.au/researchrepository/file/82315cf7-7446-fcf2-6115-b94fbd7599c6/1/full_text.pdf [retrieved: 2018-08-14].

[12] solidIT consulting & software development gmbh, "DB-Engines Ranking," URL: https://db-engines.com/en/ranking [retrieved: 2018-08-14].

[13] "MariaDB homepage," URL: https://mariadb.org/ [retrieved: 2018-08-14].

[14] solidIT consulting & software development gmbh, "DB-Engines Ranking of Relational DBMS," URL: https://db-engines.com/en/ranking/relational+dbms [retrieved: 2018-08-14].

[15] "MongoDB homepage," URL: https://www.mongodb.com/what-is-mongodb [retrieved: 2018-08-14].

[16] solidIT consulting & software development gmbh, "DB-Engines Ranking of Document Stores," URL: https://db-engines.com/en/ranking/document+store [retrieved: 2018-08-14].

[17] "InfluxDB homepage," URL: https://www.influxdata.com/time-series-platform/influxdb/ [retrieved: 2018-08-14].

[18] solidIT consulting & software development gmbh, "DB-Engines Ranking of Time Series DBMS," URL: https://db-engines.com/en/ranking/time+series+dbms [retrieved: 2018-08-14].

[19] "UDP Configuration of InfluxDB," URL: https://github.com/influxdata/influxdb/tree/master/services/udp [retrieved: 2018-08-14].

A Reliable IoT-Based Embedded Health Care System for Diabetic Patients

Zeyad A. Al-Odat*, Sudarshan K. Srinivasan*, Eman M. Al-Qtiemat*, Sana Shuja†
*Electrical and Computer Engineering, North Dakota State University
Fargo, ND, USA
†Electrical Engineering, COMSATS Institute of Information Technology,
Islambad, Pakistan
Emails: *zeyad.alodat@ndsu.edu, *sudarshan.srinivasan@ndsu.edu, *eman.alqtiemat@ndsu.edu,
†SanaShuja@comsats.edu.pk

Abstract*—This paper introduces a reliable health care system for diabetic patients based on the Internet of Things technology. A diabetic health care system with a hardware implementation is presented. The proposed work employs Alaris 8100 infusion pump, Keil LPC-1768 board, and IoT-cloud to monitor the diabetic patients. The security of diabetic data over the cloud and the communication channel between health care system components are considered as part of the main contributions of this work. Moreover, an easy way to control and monitor the diabetic insulin pump is implemented. The patient's records are stored in the cloud using the Keil board that is connected to the infusion pump. The reliability of the proposed scheme is accomplished by testing the system for five performance characteristics (availability, confidentiality, integrity, authentication, and authorization). The Kiel board is embedded with Ethernet port and Cortex-M3 micro-controller that controls the insulin infusion pump. The secure hash algorithm and secure socket shell are employed to achieve the reliability components of the proposed scheme. The results show that the proposed design is reliable, secure and authentic according to different test experiments and a case study of the Markov model. Moreover, a 99.3% availability probability has been achieved after analyzing the case study.

***Index Terms*—IoT, security, embedded system, health care.**

I. Introduction

Cloud computing has been integrated with the Internet of Things (IoT) to enable the network devices to provide resilient services to all users and applications over the world. This integration helps to simplify the access of the IoT-enabled devices by all kind of users and applications, e.g., physical devices [1]. IoT is able to connect ubiquitous systems (including physical devices) using different network infrastructures to provide efficient services all the time [2].

The physical devices that are linked to the (IoT) are continuously increasing and emerging, which put a burden on the IoT service providers to provide secure and efficient services [3]. Physical devices are allowed to mimic human being's senses through various software and hardware that are connected together using the IoT. For example, the use of a smart home as an IoT-based application can turn on and off the air conditioning system when sensing the home residents leaving or coming their home [4]. Moreover, IoT-enabled devices can be controlled using a web page or smartphone applications, in the presence of Internet [5].

To utilize the IoT more efficiently, the industrial world has moved toward the use of IoT in small board and chips. For instance, manufacturers enable the internet connection on their small boards by adding the internet accessibility option to their products [6]. Moreover, different primitives can be connected together through IoT-based applications, and they can access a shared medium between them in the presence of IoT-cloud, e.g., the health care records that are shared between the patient, hospital, and eligible users can be accessed over the cloud through mobile applications [7].

The security and authenticity of the IoT-based applications become crucial, because many entities joined the world of IoT, and the possibilities of attacks and collisions have increased [8]. Therefore, the term of "Cyber-Physical System" (CPS) emerged to provide the integration between physical devices and cyber security [2]. Particularly, the integration of the IoT-base health care records where the health records are saved on the cloud and shared with different entities. Moreover, recent improvements in the IoT designs help with the support of health care systems, e.g., the tracking patient's records and bio-medical devices using the IoT applications [9][10].

Medical devices for diabetic care have also joined the world of IoT by supporting versatile design options [11]. However, security issues need to be addressed to ensure device security and the patient's privacy [12]. A system with an authentic security mechanism is required to guarantee the integrity and security of patient's records. One of the existing methods that can be easily implemented in hardware is the Secure Hash Algorithm (SHA) [13]. The SHA is an official hash algorithm standard that was standardized by the National Institute of Standards and Technology (NIST) [14].

SHA is compatible with hardware-level implementation, which makes it the most desirable methods for hardware designers to implement their reliable architectures [15]. The implementation of IoT technology in hardware has become crucial for high-performance applications [16]. The hardware allows a high-speed computation to manipulate and retrieve health records where health records are increasing day after the other. Therefore, medical-hardware designers have moved toward the use of IoT hardware-units in their designs to support high-speed computation power for IoT related functions [17].

This paper introduces an IoT-based embedded scheme for a diabetic insulin pump. The proposed design elaborates the mechanisms of data acquisition and monitoring between different parties (patient, cloud, hospital, and legitimate users). This design helps to share health data that are related to a patient's diabetes disease along with other health records on the cloud. All these data need to be secured and authenticated when they are retrieved from the cloud. We use the SHA algorithm to provide the security and authenticity terms for our proposal.

The rest of the paper is organized as follows. Section II provides preliminaries about the used components in this paper. Section III presents a literature review about the related work. The proposed methodology is presented in Section IV. Results and discussions are detailed in Section V. Section VI concludes the paper.

II. Preliminaries

Before going through the details of our proposal, brief descriptions about SHA-256, health care system components, and performance characteristics are presented in the subsequent text.

A. Brief Description of the SHA-256

SHA-256 is employed in our design to provide data integrity and authenticity. SHA-256 takes a message with an arbitrary size then, through message compression operations, produces a message hash of size 256-bit. Equation (1) shows how to get the hash (h) from a message (M) using compression function (H).

$$h = H(M), \tag{1}$$

where M is the input message and h is the digest generated using the hash algorithm H.

The secure hash algorithm is used to make sure that the data have not tampered during transmission. For instance, the message hash is computed at the sender side and appended with the transmitted message, then at the receiver side the received message hash is recomputed again and compared with the appended hash value. For the unchanged message, the hash values on both sides are equal, which means that the message has not tampered during the transmission.

Figure 1 depicts the general procedure that is used to compute the SHA-256 hash for any given message. The input message of size less than 2^{64} is padded first by adding 1 at the end of the message then add the least number of zeros to make it congruent to $448/512$. then the message size is appended to the end of the message as a 64-bit. At the end of the preprocessing phase, the final message size becomes multiple of 512-bit. Afterward, each message block is processed using the Initial Hash Value (IHV_0) and SHA-256 compression function (F). The output of each block is fed as IHV to the next block calculations.

At the end of the process, the hash value that is generated from the last block produces the final 256 bits hash. A detailed description of the secure hash algorithm can be found in [15].

Unlike the secure hash algorithm, the keyed-hash message authentication code (HMAC) involves a secure hash algorithm and a secret cryptography key. But, the *(*HMAC) algorithm is vulnerable against the length extension attack, which gives the attacker an opportunity to access the secret data [18]. Therefore, we avoid using the *HMAC* algorithm in our design. Though, data encryption functionality is the responsibility of the employed hardware and the encrypted *SSH* connection.

B. System Components

The proposed design consists of components that integrate together to form the overall architecture.

- Micro-controller unit. It is used to manage and control the medical devices according to a predefined procedure. This includes: delivers the control commands, daily patient's readings, and provide the secure connection layer. In our design, we use the Cortex-M LPC-1768 Keil board.
- Infusion Pump. It delivers the medical liquid (insulin) to the patient on a timely basis. In our design, Alaris-8100 infusion pump module is used.
- IoT-based cloud storage. In our proposal, we use the IoT-cloud as a medium between distributed medical institutions, patients and caregivers.
- Security components. They include a secure communication path using the secure socket layer (SSL/TLS), and cryptography mechanism to ensure the security of all system components.
- Legitimate users. The list of all authorized users to use the system according to predefined privileges.

C. Performance Characteristics

Today, some medical liquids are delivered programmatically without human intervention, e.g., insulin [19]. With medical devices that include embedded systems, a number of conditions need to be met to consider them as reliable and secure systems.

- Availability. The property that gives the probability of the system being in the normal state for a period of time.
- Confidentiality. The property that ensures the patient's information and system data are unavailable to unauthorized third parties.
- Integrity. All system data that can affect the treatment of the patient must not be altered without the patient's knowledge.
- Authentication. It means, only authorized parties or components should be able to act as a trusted user of the system.
- Authorization. The property of providing the verification of certain actions before execution.

These characteristics will be discussed in Section V.

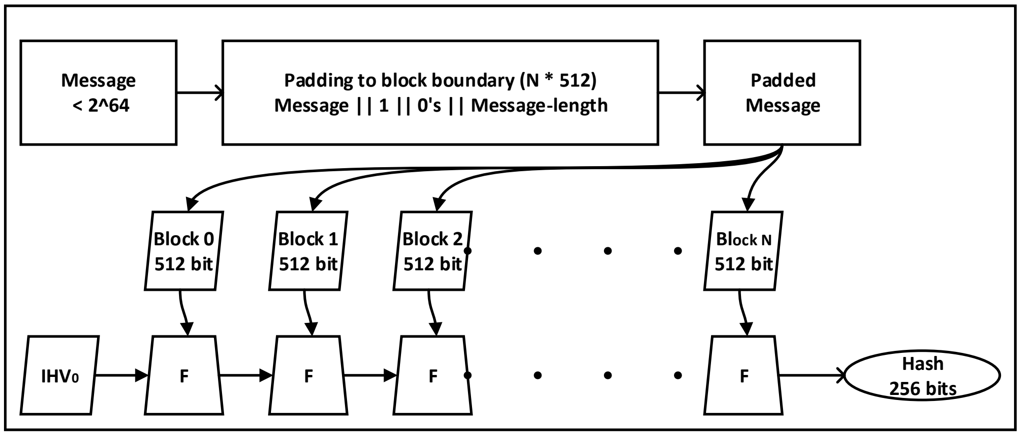


Fig. 1. General architecture to compute the SHA-256 hash function.

D. Contributions

The proposed design aims to provide the following contributions to the health care system, particularly diabetic patients. We use an external micro-controller (Kiel LPC1768) to program Alaris-8100 infusion pump. This design helps to solve current problems in the infusion pump.

- On-time medication, where a patient can get all his prescribed doses on time.
- Simplicity, affordability and the ease of use.
- Remote health record management through mobile applications or web browsers.
- Provide health service on the time of Off-Service physician.
- Provide secure and authentic health care service by employing cryptography and security approaches.

III. Related Work

Recently, the IoT-based applications have involved in all fields that influence Human life, especially, medical devices. The use of IoT in health monitoring and control is employed by different publications [9][13][20][21][22]. A novel IoT-aware smart architecture for automatic monitoring and tracking of the patient, personnel, and biomedical devices, was presented in [9]. The proposed work built a smart hospital system relying on three components: Radio Frequency Identification (RFID), Wireless Sensor Network (WSN), and smart mobile. The three hardware components were incorporated together through a local network to collect the surrounding environment and all related parameters to a patient's physiology. The collected data is sent to a control center in a real-time manner where all data are available for monitoring and management by the specialist through the Internet. The authors implemented a Graphical User Interface (GUI) to make the data access more flexible for the specialist.

To exploit the bridging point between the IoT and health care system, Rahmani *et al.* proposed a smart E-health care system for ubiquitous health monitoring [20]. The proposed work exploits ubiquitous health care gateways to provide a higher level of services. This work studied significant ever-growing demands that have an important influence on health care systems. The proposed work suggests an enhanced health care environment where control center burdens are transferred to the gateways by enabling these gateways to process part of the control center jobs. The security of this scheme was taken into consideration as the system deals with substantial health care data. The security scheme provides data authenticity and privacy characteristics.

A personalized health care scheme for the next generation wellness technology was proposed in [21]. The security of patient's records was addressed in case of data storage and retrieval over the cloud. The proposed work established a patient-based infrastructure allowing multiple service providers including the patient, service providers, specialists, and researchers to access the stored data. Their work was implemented on a cloud-based platform for testing and verification where a customized and timely messaging system for continuous feedback is tested. Moreover, multiple service providers are supported with an information infrastructure to provide unified views of patient's records and data. The use of special encryption schemes was also explored in [22], [23]. Liu *et al.* presented a scheme for secure sharing of personal health records in the cloud. The health records are ciphered before they are stored in the cloud. The proposed work uses Cipher-Text Attribute-Based Signcryption Scheme (CP-ABSC) as an access control mechanism. Using this scheme, they were able to get fine-grained data access over the cloud [22]. While Zhang *et al.* proposed a cloud storage scheme for electronic health records based on secret sharing. The proposed design consists of four phases, namely, the preprocessing phase, distribution phase, reconstruction outsourcing phase, and recovery and verification phase. In the preprocessing phase, each health record is uploaded to the cloud as a set of m blocks. Then in the distribution phase, the blocks are distributed over different storage locations in the cloud. In the reconstruction phase, the record's blocks are gathered from different storage locations. Lastly, in the verification phase, the gathered blocks are verified to determine whether if they belong to the accurate record or not [23].

With the emerge of IoT-enabled micro-chips, the researchers got benefited from this property by implementing embedded systems that provide IoT capabilities [24]. Different publications explored the use of embedded micro-controllers in medical devices. Particularly, the use of Keil LPC1768 micro-controller [13][17]. In [13], an online design for monitoring patient's data was presented. The proposed work employed an Advanced RISC Machine (ARM) architecture where Cortex M3 microprocessor is embedded in Keil LPC1768 board. In their work, the authors used pulse, temperature, and gas sensors to collect the patient's medical parameters. The LPC1768 board was used as a hardware layer between the Internet and the medical sensors. Each time the sensors' values change, the corresponding values on the Internet change immediately. However, their design was only used to monitor the surrounding environment without any interaction with the patient.

To have an embedded system with monitoring and control capabilities, Boppudi *et al.* proposed a data acquisition and control system using the ARM Cortex M3 microprocessor [17]. The proposed design send the monitored sensor data to the Internet using an Ethernet-controlled interface, which was built using Keil LPC1768 board. The proposed work employed two sensing devices temperature and accelerator-meter. Both sensors were used to collect data from the surrounding environment. The collected readings are sent to the Internet through the Ethernet interface. According to the uploaded readings, a specialist can change the behavior of the device through the Internet browser.

With the distributed components of the IoT-based health care systems, the need to verify and evaluate the integration of these components is crucial. The verification and evaluation of health care systems over the cloud is investigated by different researchers [25], [26]. Macedo *et al.* proposed a model to evaluate the IoT-based data redundancy. They employed a Markov model to test the probability of failure of one of the IoT components during the run time. They calculated the probability of failure of one of the cloud storage, then transfer the data store burden to less probability storage devices. The proposed design investigates the failure probability of the cloud storage components using the failure and recovery factor of each component. They were able to build a Markov model that describes the transition between the redundant storage locations at any given time [25]. However, Anastasiia *et al.* extended their work to build a model for IoT health care system [26]. The proposed work establishes a Markov model considering the failure of components for the IoT health care system. In their work, they gave the case study of Markov model to test the availability of health care components if any failure has happened at any time or location.

In the subsequent section, the integration between different components of the IoT health care system and the conjunction between the diabetic insulin pump (Alaris 8100) and Keil LPC-1768 board will be discussed in details.

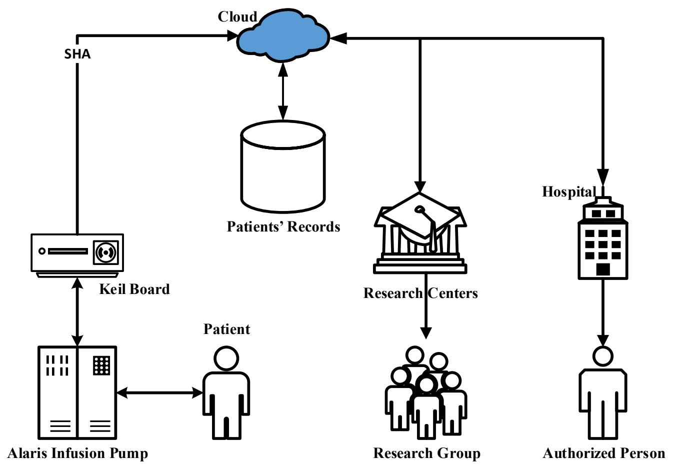


Fig. 2. General architecture of the proposed scheme.

IV. Proposed Methodology

In the proposed methodology, all system components that were mentioned in Section II will be integrated together to form the general architecture of the embedded IoT health care system. The proposed design comprises three main operations: monitoring, storing, and control, which are connected together to form the overall system. In this section, a case study of Markov model will be presented to test the availability of the proposed design.

For secure communication, the Secure Socket Shell protocol is employed. The SSH is the worldwide highest quality level for remote framework organization and secure document exchange. SSH is utilized in each datum focus and in each real endeavor. One of the highlights behind the enormous prevalence of the SSH is the solid verification utilizing SSH keys [27].

A. General Architecture of the Proposed Scheme

The proposed design employs the Alaris 8100 infusion pump to deliver insulin to the patient. The infusion pump is controlled using LPC-1768 board that contains the Cortex-M3 micro-processor. Figure 2 shows the general architecture of the proposed design.

The diabetic patient is attached to the infusion pump to get prescribed insulin doses. The Infusion pump is connected to the micro-controller unit (Keil LPC-1768 board) through a serial connection. A secure connection between the micro-controller and the cloud is established using the Secure Socket Shell (SSH) protocol and supported by the SHA-256 mechanism to authenticate the data exchange between cloud and micro-controller. Cloud computing provides the required infrastructure to handle all communications between the local and remote entities and reserves the desired amount of storage to store all health records and patient's data. The proposed architecture allows the authorized remote entities (e.g., medical and research institutions) to access the stored health records and monitor the patient's vital signs. Moreover, the proposed architecture provides the ability to control the infusion pump, remotely, through privileges that are given to an authorized physician.

Figure 3 shows the hardware setup of the proposed architecture. The Alaris-8100 infusion pump was disassembled to reach out the infusion components inside the pump. Then we built the interface between the Keil LPC-1768 board and the pump. Afterward, we used Keil μ-Vision Software Development Kit (SDK) to program the micro-controller.

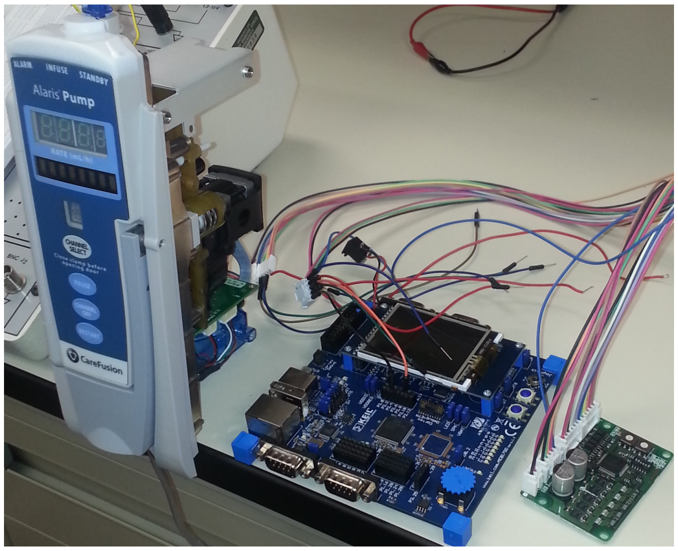


Fig. 3. Connection of Alaris Infusion Pump 8100 with Keil 1768 PCB board.

The hardware setup operations and system deployment were integrated together at the North Dakota State University (NDSU)-Electrical and Computer Engineering laboratories.

B. Monitoring, storing and controlling IoT health care system

The proposed design categorizes the IoT-health care system into three operations, which are the monitor, store, and control operations. The monitor operation involves the process of monitoring the status of the patient at any time and broadcasts the recorded data to the legitimate parties. The monitoring operation is accomplished by the micro-controller and insulin pump sensors. The store operation responsible for storing the collected data in local and remote databases, which is accomplished by the micro-controller. The control operation, which is accomplished by the micro-controller, changes the insulin pump schedule according to predefined or modified schedules. The schedule of the insulin pump is only generated by an authorized physician. Each operation is a complement to the other where the micro-controller operates as a common part between them.

1) Monitor health records: The process of health record monitor is accomplished according to Algorithm 1. The Secure Socket Shell (S) connection is initialized between the legitimate user and the cloud. Then the legitimate user receives the desired patient record appended with its SHA-256 hash value (H_p). The hash value (H_q) of the received record (P_q) is computed at the user side then, compared with the appended hash value (H_p). If both hash values are equal then the received health record is valid and contains the last updated health data.

Algorithm 1: Monitor patient's records

```
Input: Query ($Q$)
Output: $Q$ + Hash($c$)
1 for $q \leftarrow 0$ to $n$ do
2     $S = Init(SSH)$
3     $Receive(P_q + H_p)$
4     $H_q = Hash(P_q)$
5     $Compare(H_q, H_p)$
6         $Case(equal) \leftarrow$ Valid
```

2) Store health records: Each health record has a designated SHA-256 value that is appended to the health record at the time of generation. Algorithm 2 shows the general procedure that is carried out to store the newly generated or updated health record. The hash value (H_p) of health record (P) that is related to the patient (i) is computed using the SHA-256 hash function. The computed hash (H_p) is appended to the patient record (P_i). An SSH connection between the micro-controller and the cloud is initialized to send the combination of hash and record (A_p) to the cloud for storage. Moreover, the new health record is stored in a Local Storage (LS) unit for quick data access.

Algorithm 2: Store health records

```
Input: Health record ($P$)
Output: $P$+Hash($P$)
1 for $i \leftarrow 0$ to $n$ do
2     $H_p = Hash(P_i)$
3     $A_p = Append(P_i, H_p)$
4     $S = Init(SSH)$
5     $LS(A_p)$
6     $Send(A_p, S)$
```

As health records are sensitive information, the *SSH* uses a symmetric encryption mechanism to ensure the data privacy between different parties. This is accomplished after initialization of the *SSH* connection between client and server. The client initializes the connection by contacting the server, then the server responds to the client by sending the server's public key. Figure 4 shows the construction of data record (P_i). The data record is signed using the *SHA* algorithm, then the produced hash value (H_p) is appended to the end of the data record. Afterward, The *SSH* connection is used to transfer data record to the cloud.

3) Prescription control command: The prescription control command is generated by a remote caregiver. Algorithm 3

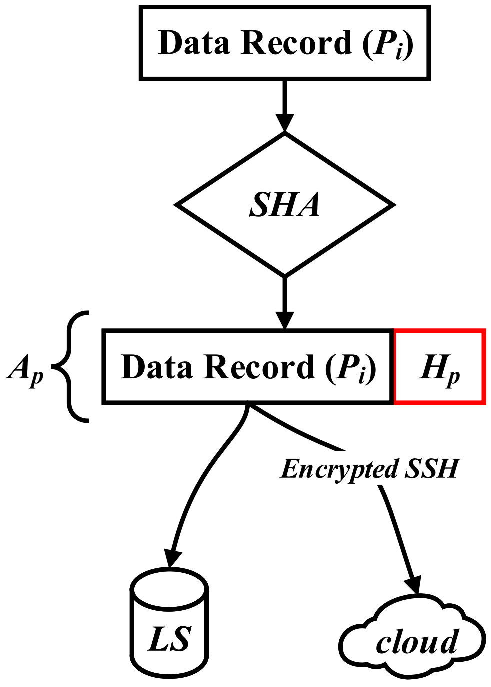


Fig. 4. Construction of data record.

shows the general procedure to send a new control command to the insulin pump. The prescription control command (C) is generated and appended with its corresponded SHA-256 hash value (H_c) to form the appended control command (A_c). A secure Socket Shell (S) is initialized between the remote caregiver and the micro-controller through the cloud. Then the new control command is sent through the SSH Chanel. At the receiving side, the micro-controller verify the received control command by following steps $3-6$ of Algorithm 1 where the received message is $C+H_c$. If the received prescription control command is valid, then the micro-controller will forward it to the insulin pump to start the new schedule.

Algorithm 3: Send prescription control command

```
Input: Prescription control command ($C$)
Output: $C$ + Hash($c$)
1 for $i \leftarrow 0$ to $n$ do
2     $H_c = Hash(C_i)$
3     $A_c = Append(C_i, H_c)$
4     $S = Init(SSH)$
5     $Send(A_c, S)$
```

Figure 5 shows the connection between different components of IoT health care system. The embedded micro-controller controls the insulin device and collects the required health information. This is accomplished using a serial connection (6.25Mbps) between the micro-controller and the infusion pump. The Cortex-M3 micro-controller, which is embedded in the LPC1768 board, uses a universal asynchronous receiver-transmitter (UART) that supports 8 bits communication without parity and is fixed at one stop bit per configuration. The Keil LPC1768 board is programmed using micro-vision-5 software development kit (SDK) under windows 10 and implemented under *C* software stack.

The micro-controller collects data and stores them on local storage (LS) and remote storage (Remote DB) through the SSH connection. The IoT-cloud takes the responsibly to provide a replica for the stored data, it is considered as one of the great benefits of using the IoT-cloud. The data between the IoT-cloud and local storage are synchronized all the time to provide quick local access for the patient's health records.

The insulin device receives the doses schedule and delivers insulin to the diabetic patient. A local caregiver (CG) is responsible for a group of patients in emergency situations. A patient using the Alaris 8100 infusion pump will take preset insulin doses regularly [28]. The Alaris infusion pump is controlled and monitored by the Keil Cortex M3 board through a serial connection. All dosages related records are sent to the cloud through the Keil board using the Ethernet connection. To ensure the security and authenticity, the recorded data are digitally signed using the *SHA-256* compression function and encrypted using a symmetric key encryption mechanism. Moreover, the The signature and patient's records are stored together in the cloud.

In the cloud, a Secure Socket Shell (SSH) is provided to authorized entities to access the health records. For instance, a physician can follow up with a patient's case using a mobile application or a web browser. Furthermore, research institutions are given the authorization to access health records upon agreements made between patient, medical centers, and research institutions.

The integrity of the health care records is verified using the SHA-256 signature. While the authenticity is ensured by the encryption mechanism and *SSH* connection. The SHA-256 value is computed after the health records or prescription commands are generated. Then the generated SHA-256 is appended to the corresponding data (health record or preset control command). The health record and its signature remain correlated in all places (cloud, hospital, and patient's side). For instance, the physician in the hospital confirms that the record is received without altering using the SHA-256 signature. When the health record is received at the hospital, SHA-256 computation will be carried out. The resultant SHA-256 value will be compared with the appended SHA-256 value. Once both values are equal, the record will be confirmed to their corresponding patient. Otherwise, the health record will be discarded as it does not belong to the patient. Bearing in mind that all connections and data transfer are carried out using an encrypted *SSH* connection.

In the case of the preset control command, this command is generated from the hospital and appended with its corresponding hash value. The preset control command and the SHA signature are sent through the cloud to the infusion pump. At the patient's side, the hardware takes the responsibility to check

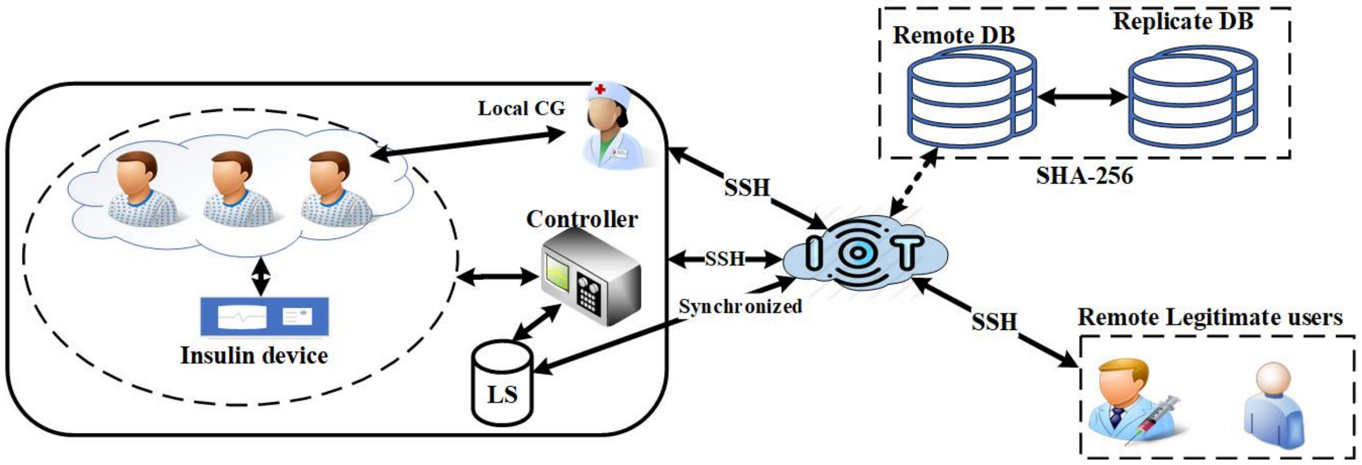


Fig. 5. General architecture of the proposed scheme.

the genuineness of the received control command by SHA-256 computation and comparison. The Keil micro-controller computes the SHA-256 value for the received preset control command and then compares the result with the appended SHA-256 value. Once authorized, the preset control command is passed to the infusion pump for a new schedule.

In the case of a fault exception, all Cortex-M processors (including Keil LPC-1768) have a fault exception mechanism embedded inside the processor. If any fault is detected, the corresponding exception handler will be executed [29].

C. Case Study: A Markov Model of proposed scheme

In IoT health care system, the failure of one or more components may lead to system failure. In our design, we have four main components: 1) Insulin Pump. It is represented by the Alaris 8100 infusion pump. 2) Micro-controller. It is represented by the LPC-1768 Keil board. 3) IoT-cloud. It provides infrastructure and medium. 4) Authority failure that represents the loss of security. Figure 6 shows the Markov model that connects the main components during system failure. The failure rate is represented by the symbol λ and the recovery rate is represented by the symbol μ.

The case study depicts 12 states that represent the transition from one state to another with the corresponding failure rate and recovery rate. However, some states are represented by the failure rate only because they are unable to recover. Thereby, the states are defined as follows: 1) Normal operation where all components work as required. 2) Insulin pump failure due to hardware defects. 3) IoT-cloud failure due to connection failure. 4) Failure due to data delivery between Insulin Pump and micro-controller. 5) Failure due to the power supply. 6) IoT-cloud software failure. 7) IoT-cloud hardware failure. 8) Insulin pump software failure. 9) insulin pump hardware failure. 10) IoT-cloud failure due to the failure of cloud components. 11) Insulin pump failure due to the failure of insulin pump components. 12) Failure of the system.

The Markov model depicted in Figure 6 can be represented as a system of Kolmogrov differential equations, as shown by equations (2)-(13). The probability ($P_i(t)$) represents the probability to find the system in state i. In our design, we chosen the initial conditions as follows: $P_1(t) = 1$, $P_i(t) = 0$ for i = 2, .., 12.

To collect the failure components and build our case study, we analyzed references [19][30][31][32][33][34][35]. All kind of failures are caused by software or hardware failures that might affect the main system components and cause the system failure. To further help other researchers, We list the values of failure and recovery rates in Table III.

$$\begin{aligned} dP_1/dt = -(\lambda_{1,2} + \lambda_{1,3} + \lambda_{1,4} + \lambda_{1,5})P_1(t) + \\ \mu_{2,1}P_2(t) + \mu_{3,1}P_3(t) + \mu_{4,1}P_4(t) + \mu_{5,1}P_5(t) \\ +\mu_{11,1}P_{11}(t) + \mu_{12,1}P_{12}(t) \end{aligned} \tag{2}$$

$$\begin{aligned} dP_2/dt = -(\mu_{2,1} + \lambda_{2,9} + \lambda_{2,8})P_2(t) + \\ \lambda_{1,2}P_1(t) + \mu_{9,2}P_9(t) + \mu_{8,2}P_8(t) \end{aligned} \tag{3}$$

$$\begin{aligned} dP_3/dt = -(\mu_{3,1} + \lambda_{3,6} + \lambda_{3,7})P_3(t) + \\ \lambda_{1,3}P_1(t) + \mu_{6,3}P_6(t) + \mu_{7,3}P_7(t) \end{aligned} \tag{4}$$

$$dP_4/dt = -\mu_{4,1}P_4(t) + \lambda_{1,4}P_1(t) \tag{5}$$

$$dP_5/dt = -(\mu_{5,1} + \lambda_{5,11})P_5(t) + \lambda_{1,5}P_1(t) \tag{6}$$

$$dP_6/dt = -(\mu_{6,3} + \lambda_{6,10})P_6(t) + \lambda_{3,6}P_3(t) \tag{7}$$

$$dP_7/dt = -(\mu_{7,3} + \lambda_{7,10})P_7(t) + \lambda_{3,7}P_3(t) \tag{8}$$

$$dP_8/dt = -(\lambda_{8,11} + \mu_{8,2})P_8(t) + \lambda_{2,8}P_2(t) \tag{9}$$

$$dP_9/dt = -(\mu_{9,2} + \lambda_{9,11})P_9(t) + \lambda_{2,9}P_2(t) \tag{10}$$

$$dP_{10}/dt = -\lambda_{10,12}P_{10}(t) + \lambda_{6,10}P_6(t) + \lambda_{7,10}P_7(t) \tag{11}$$

$$\begin{aligned} dP_{11}/dt = -(\mu 11, 1 + \lambda_{11,12})P_{11}(t) + \lambda_{9,11}P_9(t) + \\ \lambda_{8,11}P_8(t) + \lambda_{5,11}P_5(t) \end{aligned} \tag{12}$$

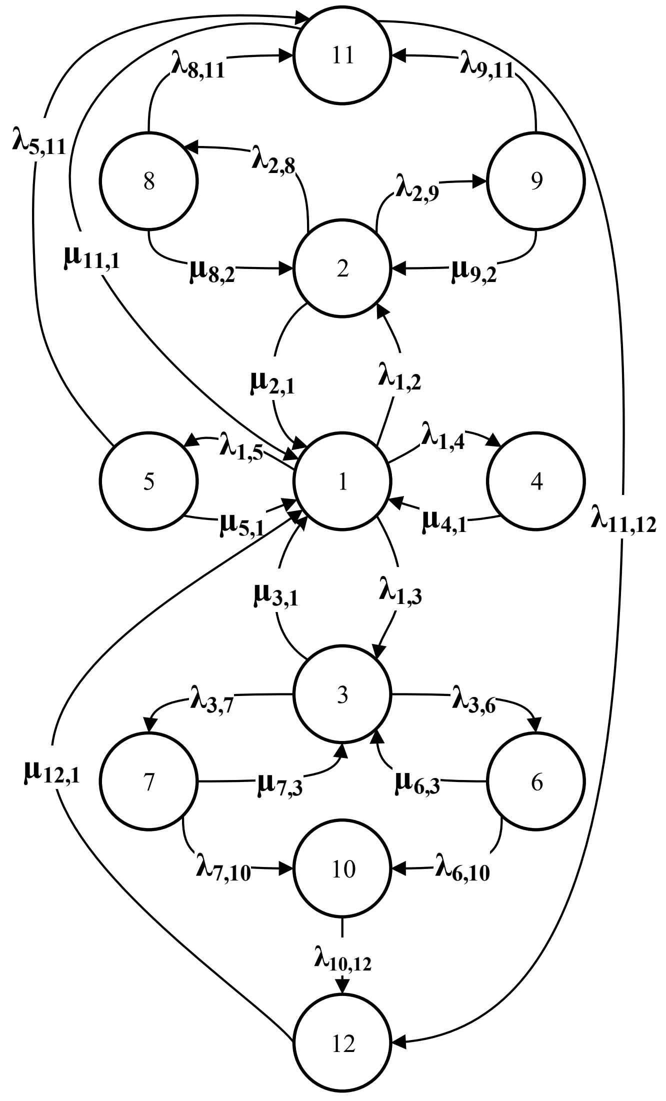


Fig. 6. Markov model graph for the IoT health care failure.

$$dP_{12}/dt = -\mu_{12,1}P_{12}(t) + \lambda_{10,12}P_{10}(t) + \lambda_{11,12}P_{11}(t) \tag{13}$$

In the subsequent section, we show the performance characteristics and their applicability to our proposal.

V. Results and Discussion

Our proposal has been tested toward the five performance characteristics that are mentioned in Section II.

A. Availability

As mentioned earlier, the availability property ensures that the system is available all the time. Our design is tested for availability by solving the system of Kolmogorov differential equations and compute the probabilities of system states. The values of system sates probabilities, after calculations, are as follows:

$$\begin{array}{ll} P_1 = 0.9925712 & P_2 = 0.0002091 \\ P_3 = 0.0005966 & P_4 = 0.002998966 \\ P_5 = 0.00009805 & P_6 = 1.09E{-}06 \\ P_7 = 2.99E{-}05 & P_8 = 0.0019989 \\ P_9 = 0.00049866 & P_{10} = 4.24E{-}07 \\ P_{11} = 0.0009958 & P_{12} = 3.00E{-}07 \end{array}$$

The availability function is represented by the probability value of $P_1(t)$, which means that the system has a probability of $\approx 99.26\%$ to stay at the normal state. The calculated probability proves the availability property of the IoT health care embedded scheme. Through this value, the proposed design ensures a high level of availability.

B. Confidentiality

To provide a confident system for data on transit, our design uses the *SSH* tunnel that is only given to the authorized entities. The *SSH* connection is initialized only by a legitimate user and supported by "private-public key pair authentication" scheme that ensures the connection is established between the designated two parties.

C. Integrity

The proposed design has been tested and verified for integrity using sample data from [36]. The sample data contains glucose levels in the patient's body during a 24 hour period, a patient's profile information, and the patient's medical information. A snipped portion of the sample data is shown in Figure 7, the figure shows the glucose levels in the patient's body after two meals (breakfast and dinner). To test the integrity property, the sample data is modified as shown in Figure 8. When both figures are compared, the only difference between them is the "AC breakfast Mean", it is equal to 142 in the original sample and 144 in the modified one.

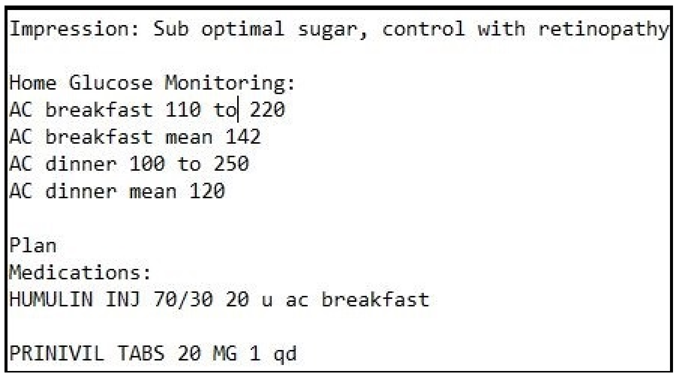
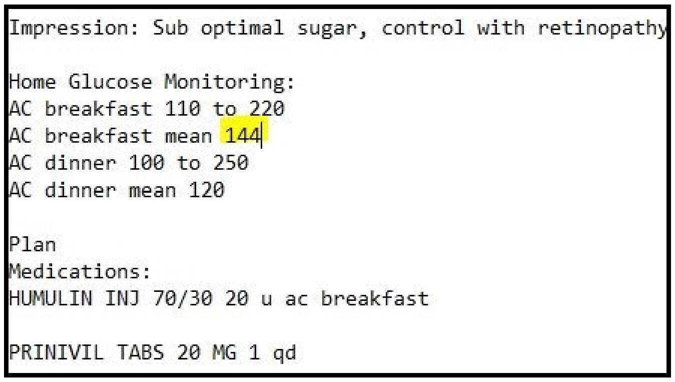

```
Impression: Sub optimal sugar, control with retinopathy

Home Glucose Monitoring:
AC breakfast 110 to 220
AC breakfast mean 142
AC dinner 100 to 250
AC dinner mean 120

Plan
Medications:
HUMULIN INJ 70/30 20 u ac breakfast

PRINIVIL TABS 20 MG 1 qd
```

Fig. 7. Snipped health record from the original sample.

The proposed design considers that the SHA-256 value is computed every time a health record is requested. The sample data is stored in the cloud and appended with the



```
Impression: Sub optimal sugar, control with retinopathy

Home Glucose Monitoring:
AC breakfast 110 to 220
AC breakfast mean 144
AC dinner 100 to 250
AC dinner mean 120

Plan
Medications:
HUMULIN INJ 70/30 20 u ac breakfast

PRINIVIL TABS 20 MG 1 qd
```

Fig. 8. Snipped health record from the modified sample.

corresponding SHA-256 value. If the patient's side requests the same health record, the micro-controller will compute the SHA-256 value of the record and compares it with the appended SHA-256 value. If both hash values (cloud and patient) are equal then the received record is valid and never been tampered during the transmission. Table I shows the SHA-256 value of the sample record on both sides where the sample record has not tampered.

However, any tiny modification to the health record will produce a totally different SHA-256 hash value. Table II shows two different hash values for the original sample that is requested from the cloud side and the modified sample at the patient's side. Both SHA-256 values are different because the received record on the patient's side has been altered during transmission. Then, the micro-controller at the receiver side will detect the alteration after comparing both hash values.

TABLE I. SHA-256 HASH VALUES OF THE SAMPLE DATA ON BOTH SIDES.

Cloud side:	14b93acf-ccdcbe40-ea3795be-c1073498-51a96c90-6cedfc9c-49d8e2cf-a141befb
Patient side:	14b93acf-ccdcbe40-ea3795be-c1073498-51a96c90-6cedfc9c-49d8e2cf-a141befb

TABLE II. SHA-256 HASH VALUES OF THE ORIGINAL AND MODIFIED SAMPLE DATA ON BOTH SIDES.

Cloud side:	14b93acf-ccdcbe40-ea3795be-c1073498-51a96c90-6cedfc9c-49d8e2cf-a141befb
Patient side:	358c4f29-f0e2bb60-8efa35d4-a88a6b3b-58939ffd-deebf824-8065c195-b834b8cd

On another hand, to ensure the integrity of prescription control command, the same procedure is carried out between the sender (corresponding physician) and receiver (micro-controller). At the patient's side, the micro-controller detects the alteration and discard the tampered control commands.

D. Authentication

To provide an authentic system, the SSH protocol is employed to ensure that only legitimate users are eligible to access the health records. Moreover, in the case of the prescription control command, special users are given a special SSH tunnel and a public-private key pair to ensure the security and authenticity of the communication medium between the Caregiver (CG) and micro-controller.

E. Authorization

The authorization and verification of certain actions before execution are accomplished by the encrypted *SSH* connection and the *SHA*, respectively. The encryption of health records ensures that only the authorized entities can decrypt and read the data contents. Moreover, if any certain action is tampered or modified before reaching the destination, then the corresponding hash value will determine whether the action is authorized. Moreover, the patient is given some privileges to change the schedule according to a predefined prescription from the corresponding physician.

F. Speed

The processing speed of the proposed design is tested using 70 samples of diabetic's records [37]. Figure 9 shows the time elapsed (in second), mean and standard deviation of the 70 samples. The elapsed time to process the samples depends on different factors, including, sample size, connection speed, and system utilization. The figure shows how the processing speed changes according to the aforesaid factors. The average time to process these samples is equal to $5.8e-04$-second, while the standard deviation value shows the amount of variation of the elapsed time for all samples.

G. Final Remarks

Our design provides a set of benefits to the health care systems, particularly, diabetic patients. We list these benefits as follows:

- Patients can access their health records easily and communicate with their caregiver instantly.
- Caregivers and physicians can control the insulin infusion pump remotely according to reliable information delivered through the proposed design.
- The security and integrity of patient's records are guaranteed by the encrypted *SSH* and the SHA.

However, the limitation of this approach can be seen in the case of a successful attack on the used security components. Until now, there is no successful collision attack for the *SHA-256* that is used in this design. The collision attack allows an adversary to tamper the data contents and produce the same hash (signature) of data before and after modification. Moreover, the length extension attack is a kind of attacks that targets the keyed hash algorithms (HMAC). Therefore, we avoid using the (HMAC) in our design and keep the authenticity requirements to the symmetric encryption mechanism of the *SSH* connection.

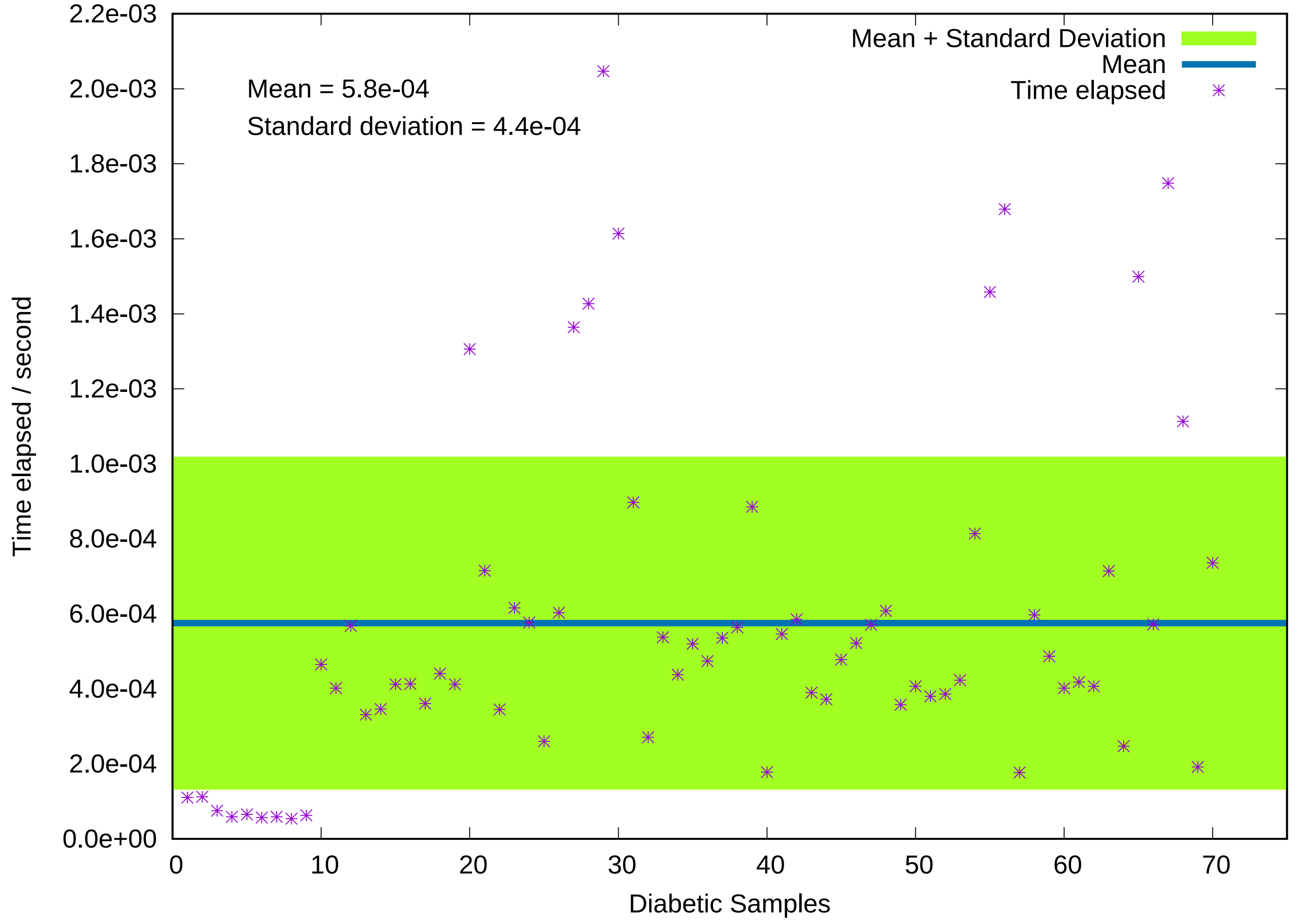


Fig. 9. Time elapsed to process 70 diabetic samples.

VI. Conclusion and Future work

In this paper, a reliable embedded health care system based on the Internet of Thing is presented. The proposed design employs secure hash algorithm SHA-256, Secure Socket Shell (SSH), Keil LPC-1768 board, Alaris 8100 infusion pump, and IoT-cloud to build the health care system. The proposed design showed that the reliability characteristics of availability, confidentiality, integrity, authentication, and authorization are accomplished. Moreover, the results showed that the proposed design has a 99.3% probability to stay in the normal operation stage and an average speed of 5.8×10^{-04} seconds to process the health records.

The scope of reliable IoT-based health care system is open. In the future, further analysis of the health care system to develop a generalized reliability model of the health care system including handheld medical devices.

Acknowledgments

This publication was funded by a grant from the United States Government and the generous support of the American people through the United States Department of State and the United States Agency for International Development (USAID) under the Pakistan - U.S. Science & Technology Cooperation Program. The contents do not necessarily reflect the views of the United States Government.

Computing services, financial and administrative support from the North Dakota State University Center for Computationally Assisted Science and Technology (CCAST) and the Department of Energy through Grant No. DE-SC0001717 are gratefully acknowledged.

Appendix

The values of failure and recovery rates, which were used in the case study, are listed in Table III.

References

[1] Z. A. Al-Odat, S. K. Srinivasan, E. Al-qtiemat, L. D. Mohana Asha, and S. Shuja, "Iot-based secure embedded scheme for insulin pump data acquisition and monitoring," in *The Third International Conference on Cyber-Technologies and Cyber-Systems*. IARIA, 2018, pp. 90–93.

[2] 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*, vol. 4, no. 5, pp. 1125–1142, 2017.

TABLE III. FAILURE AND RECOVERY RATES PARAMETERS

Failure (λ)	Value	Recovery (μ)	Value
$\lambda_{1,2}$	1.857E-09	$\mu_{2,1}$	99.57E-2
$\lambda_{1,3}$	2.499E-07	$\mu_{3,1}$	95.08E-2
$\lambda_{1,4}$	3.331E-07	$\mu_{4,1}$	98.76E-2
$\lambda_{1,5}$	4.985E-07	$\mu_{5,1}$	92.37E-2
$\lambda_{2,8}$	2.50E-07	$\mu_{6,3}$	2.12E-3
$\lambda_{2,9}$	2.50E-07	$\mu_{7,3}$	4.07E-3
$\lambda_{3,6}$	7.50E-3	$\mu_{8,2}$	4.20E-4
$\lambda_{3,7}$	3.56E-05	$\mu_{9,2}$	2.93E-4
$\lambda_{6,10}$	1.28E-2	$\mu_{11,1}$	1.23E-6
$\lambda_{7,10}$	1.63E-2	$\mu_{12,1}$	1.857E-8
$\lambda_{8,11}$	2.00E-4		
$\lambda_{9,11}$	3.11E-5		
$\lambda_{10,12}$	2.70E-3		
$\lambda_{11,12}$	25.87E-3		

[3] 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*, vol. 17, no. 4, pp. 2347–2376, 2015.

[4] L. Atzori, A. Iera, and G. Morabito, “The internet of things: A survey,” *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[5] R. Kazi and G. Tiwari, “Iot based interactive industrial home wireless system, energy management system and embedded data acquisition system to display on web page using gprs, sms & e-mail alert,” in *Energy Systems and Applications, 2015 International Conference on*. IEEE, 2015, pp. 290–295.

[6] I. Ungurean, N.-C. Gaitan, and V. G. Gaitan, “An iot architecture for things from industrial environment,” in *Communications (COMM), 2014 10th International Conference on*. IEEE, 2014, pp. 1–4.

[7] D. Hinge and S. Sawarkar, “Mobile to mobile data transfer through human area network,” *IJRCCT*, vol. 2, no. 11, pp. 1181–1184, 2013.

[8] M. Conti, A. Dehghantanha, K. Franke, and S. Watson, “Internet of things security and forensics: Challenges and opportunities,” *Future Generation Computer Systems*, vol. 78, pp. 544–546, 2018.

[9] L. Catarinucci *et al.*, “An iot-aware architecture for smart healthcare systems,” *IEEE Internet of Things Journal*, vol. 2, no. 6, pp. 515–526, 2015.

[10] J. Yu, K. Ren, and C. Wang, “Enabling cloud storage auditing with verifiable outsourcing of key updates,” *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 6, pp. 1362–1375, 2016.

[11] K. Gai, M. Qiu, L.-C. Chen, and M. Liu, “Electronic health record error prevention approach using ontology in big data,” in *High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS), 2015 IEEE 17th International Conference on*. IEEE, 2015, pp. 752–757.

[12] X. Chen, X. Huang, J. Li, J. Ma, W. Lou, and D. S. Wong, “New algorithms for secure outsourcing of large-scale systems of linear equations,” *IEEE transactions on information forensics and security*, vol. 10, no. 1, pp. 69–78, 2015.

[13] G. Harsha, “Design and implementation of online patient monitoring system,” *International Journal of Advances in Engineering & Technology*, vol. 7, no. 3, p. 1075, 2014.

[14] Q. Dang, “Changes in federal information processing standard (fips) 180-4, secure hash standard,” *Cryptologia*, vol. 37, no. 1, pp. 69–73, 2013.

[15] F. PUB, “Secure hash standard (shs),” *FIPS PUB 180*, vol. 4, pp. 1–27, 2012.

[16] S. Salinas, C. Luo, X. Chen, W. Liao, and P. Li, “Efficient secure outsourcing of large-scale sparse linear systems of equations,” *IEEE Transactions on Big Data*, vol. 4, no. 1, pp. 26–39, 2018.

[17] L. P. Boppudi and R. Krishnaiah, “Data acquisition and controlling system using cortex m3 core,” *International Journal of Innovative Research and Development*, vol. 3, no. 1, pp. 29–33, 2014.

[18] “HashPump - A Tool To Exploit The Hash Length Extension Attack In Various Hashing Algorithms,” Sep 2018, [accessed 04. May 2019]. [Online]. Available: https://www.prodefence.org/hashpump

[19] N. Paul, T. Kohno, and D. C. Klonoff, “A review of the security of insulin pump infusion systems,” *Journal of diabetes science and technology*, vol. 5, no. 6, pp. 1557–1562, 2011.

[20] A.-M. Rahmani *et al.*, “Smart e-health gateway: Bringing intelligence to internet-of-things based ubiquitous healthcare systems,” in *Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE*. IEEE, 2015, pp. 826–834.

[21] P.-Y. S. Hsueh, H. Chang, and S. Ramakrishnan, “Next generation wellness: A technology model for personalizing healthcare,” in *Healthcare Information Management Systems*. Springer, 2016, pp. 355–374.

[22] J. Liu, X. Huang, and J. K. Liu, “Secure sharing of personal health records in cloud computing: ciphertext-policy attribute-based signcryption,” *Future Generation Computer Systems*, vol. 52, pp. 67–76, 2015.

[23] H. Zhang, J. Yu, C. Tian, P. Zhao, G. Xu, and J. Lin, “Cloud storage for electronic health records based on secret sharing with verifiable reconstruction outsourcing,” *IEEE Access*, vol. 6, pp. 40 713–40 722, 2018.

[24] G. J. Joyia, R. M. Liaqat, A. Farooq, and S. Rehman, “Internet of medical things (iomt): applications, benefits and future challenges in healthcare domain,” *J Commun*, pp. 240–247, 2017.

[25] D. Macedo, L. A. Guedes, and I. Silva, “A dependability evaluation for internet of things incorporating redundancy aspects,” in *Networking, Sensing and Control (ICNSC), 2014 IEEE 11th International Conference on*. IEEE, 2014, pp. 417–422.

[26] S. Anastasiia, K. Vyacheslav, and U. Dmytro, “A markov model of healthcare internet of things system considering failures of components,” in *4th International Workshop on Theory of Reliability and Markov Modelling for Information Technologies*. CEUR-WS, 2018, pp. 530–543.

[27] S. C. Williams, “Analysis of the ssh key exchange protocol,” in *IMA International Conference on Cryptography and Coding*. Springer, 2011, pp. 356–374.

[28] K. L. Grant and B. D. Tracey, “Infusion pump assembly,” Sep. 16 2014, uS Patent 8,834,429.

[29] E. Alkim, P. Jakubeit, and P. Schwabe, “Newhope on arm cortex-m,” in *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer, 2016, pp. 332–349.

[30] M. U. Farooq, M. Waseem, A. Khairi, and S. Mazhar, “A critical analysis on the security concerns of internet of things (iot),” *International Journal of Computer Applications*, vol. 111, no. 7, 2015.

[31] P. A. Kodeswaran, R. Kokku, S. Sen, and M. Srivatsa, “Idea: A system for efficient failure management in smart iot environments,” in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2016, pp. 43–56.

[32] E. Solaiman, R. Ranjan, P. P. Jayaraman, and K. Mitra, “Monitoring internet of things application ecosystems for failure,” *IT Professional*, vol. 18, no. 5, pp. 8–11, 2016.

[33] M. Hassanalieragh *et al.*, “Health monitoring and management using internet-of-things (iot) sensing with cloud-based processing: Opportunities and challenges,” in *2015 IEEE International Conference on Services Computing*. IEEE, 2015, pp. 285–292.

[34] J. H. Abawajy and M. M. Hassan, “Federated internet of things and cloud computing pervasive patient health monitoring system,” *IEEE Communications Magazine*, vol. 55, no. 1, pp. 48–53, 2017.

[35] A. Guenego *et al.*, “Insulin pump failures: has there been an improvement? update of a prospective observational study,” *Diabetes technology & therapeutics*, vol. 18, no. 12, pp. 820–824, 2016.

[36] “Sample Medical Record: Monica Latte | Agency for Healthcare Research & Quality,” Oct 2018, [accessed 1. Oct. 2018]. [Online]. Available: https://www.ahrq.gov/professionals/prevention-chronic-care/improve/system/pfhandbook/mod8appbmonicalatte.html

[37] “UCI Machine Learning Repository: Diabetes Data Set,” Feb 2019, [accessed 3. Feb. 2019]. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/diabetes