SandBoxer: A Self-Contained Sensor Architecture for Sandboxing the Industrial Internet of Things

Galal Hassan, Abdulmonem M. Rashwan, Hossam S. Hassanein
School of Computing
Queen's University, Kingston, ON, Canada
Email: {ghassan, arashwan, hossam}@cs.queensu.ca

***Abstract*—The Industrial Internet-of-Things (IIoT) has gained significant interest from both the research and industry communities. Such interest came with a vision towards enabling automation and intelligence for futuristic versions of our day to day devices. However, such a vision demands the need for accelerated research and development of IIoT systems, in which sensor integration, due to their diversity, impose a significant roadblock. Such roadblocks are embodied in both the cost and time to develop an IIoT platform, imposing limits on the innovation of sensor manufacturers, as a result of the demand to maintain interface compatibility for seamless integration and low development costs. In this paper, we propose an IIoT system architecture (SandBoxer) tailored for sensor integration, that utilizes a collaborative set of efforts from various technologies and research fields. The paper introduces the concept of "development-sandboxing" as a viable choice towards building the foundation for enabling true-plug-and-play IIoT. We start by outlining the key characteristics desired to create an architecture that catalyzes IIoT research and development. We then present our vision of the architecture through the use of a sensor-hosted EEPROM and scripting to "sandbox" the sensors, which in turn accelerates sensor integration for developers and creates a broader innovation path for sensor manufacturers. We also discuss multiple design alternative, challenges, and use cases in both the research and industry.**

***Index Terms*—sandboxing; IIoT; PnP; virtual sensor; sensor integration; embedded scripting; cross-platform sensing; development-sandboxing**

I. Introduction

Towards realizing the vision of Industry 4.0 (I4.0), a need for accelerated research and development in each of its nine pillars is mandatory [1]. One such pillar is the Industrial Internet of Things (IIoT). One of the main challenges throttling the research and development of IIoT is the process of interfacing sensors and transducers with an IIoT system.

The current status quo requires the developer to write lengthy drivers for each transducer from each manufacturer. In order to simplify and modularize the development, system architects usually use a dedicated MicroController Unit (MCU) for each transducer that is specifically programmed to collect the transducer readings and communicate it to the main MCU on the central data collection device. Such method complicates the development of a new IoT system and radically reduces the flexibility of using different sensors post-production because the developer is required to reprogram the MCUs every time a different sensor is required [2]. Therefore, having the ability to plug and unplug a transducer seamlessly and without a dedicated MCU would reduce the development time and risk significantly, while allowing a simpler prototyping mechanism for researchers, which in turn will accelerate research and development of IoT.

Moreover, each sensor element is diverse by nature as it comes with a set of requirements and interfacing instructions set by the manufacturer. Such heterogeneity makes it increasingly challenging to integrate different sensors into an IIoT system. Even though plug-and-play (PnP) is no longer a novel concept, multiple efforts have been made towards achieving the complete vision of the IIoT paradigm [3], each focusing on a different layer rendering each with its limitations, ranging from development complexity to cost. SPROUTS, for instance, is a Wireless Sensor Network (WSN) platform architecture designed to achieve the vision of IIoT. Among the features of SPROUTS was an attempt at PnP functionality [4]. However, the implementation required a separate sensor board with a dedicated MCU, rendering Sprouts PnP impractical for the IIoT paradigm.

Our research introduces a scripting-based IIoT architecture (dubbed SandBoxer) that isolates sensors into a sandboxed environment using an Electrically Erasable Programmable Read-Only Memory (EEPROM), thus, facilitating PnP functionality and eliminating the need for an extra MCU for each transducer. SandBoxer acts as an enabler to key technological developments in the IoT paradigm, such as virtual sensors, PnP, sensor sandboxing, and platform independence. By enabling such technologies, the proposed research simplifies sensor and transducer integration, therefore, accelerating the progression of IoT research and development. Our architecture involves four main contributions, a sandboxed sensor IIoT system architecture, a light-weight kernel architecture optimized for resource-constrained embedded systems, a sensor/transducer system architecture for self-containment, and a low-footprint scripting language description.

The remainder of this paper starts with a discussion on some of the background and inspirational efforts in section II. Followed by an outline of the desired characteristics of the IIoT architecture in section III. We then propose our SandBoxer architecture in section IV and converse on the challenges and design alternatives in section V. We identify some use-cases for our architecture in section VI, then finally conclude the paper and identify the future directions in section VII.

978-1-7281-2373-8/19/$31.00 ©2019 IEEE

II. Background & Inspirations

Over the past decade, multiple efforts have been employed towards rapid IIoT research and development. PnP, and platform independence are only two of the enablers in the IIoT roadmap [3]. However, research on the use of embedded scripting to achieve such features has received minimal attention. In this section, we review some of the influential efforts that were the seeds of our proposed architecture.

A. Plug-and-Play (PnP)

The lack of customization is an integral aspect that is throttling the advancement of IIoT, due to the need to redesign a platform in order to re-use it in a different application. Such redesign leads to a longer time-to-market and costly development and testing procedures [4]. When combined with the IoT paradigm, the concept of PnP enables a customizable platform that can adapt to different applications without the need for redevelopment. Despite the added simplicity and flexibility, PnP presents some substantial challenges in order to be considered feasible for the IoT paradigm. One such challenge is the fact that sensor drivers are still required to be ported and tested by the IoT platform developers, as is the case with SPROUTS [4]. Moreover, the security risks presented with architectures such as UPnP (although mitigable) pose a significant threat when integrated with IIoT industry [5]. Therefore, an IIoT applicable PnP architecture is necessary for the advancement of I4.0 [3].

B. IEEE 21451

The IEEE registered a series of 21451 standards with the aim to add plug-and-play capabilities to sensors and transducers. The standard provides said sensors and transducers with capabilities such as self-identification, self-description, self-calibration, and standard data formats [6]. In order to implement such features, the standard specifies the use of an EEPROM loaded with a Transducer Electronic Data Sheet (TEDS). As with traditional datasheets, TEDS contains information about the manufacturer, the type of transducer, serial number, accuracy, measurement range, calibration data, and the supported data formats. By design, the standard is more inclined towards analogue sensors than digital ones, hence, lacking the support of critical features such as interfacing protocol and sensor register descriptions [7]. In order to use the standard with a digital sensor, system developers are still required to write an interfacing driver program, as well as use a dedicated MCU with the sensor [2]. Therefore, in order to render the standard more accessible for different types of sensors and transducers and enabling true PnP functionality, it is essential to reduce the burden on system developers and move the driver program from the system side to the sensor/transducer side, as well as remove the dedicated MCU on the sensor side.

C. Sandboxing

The term sandboxing is widely used to describe the concept of isolating the execution of software programs in separate restricted environments. Thus, safeguarding parallel and underlying software systems from unauthorized software access [8]. Research on different sandboxing mechanisms has been on the rise for the past decade [9]; for instance, fog computing uses virtualization in order to enable sandboxing [10]. Apple uses sandboxing in its iOS operating system in order to isolate each application and limit access to the file system, network, and the underlying hardware [11]. Googles Android, on the other hand, uses a sandboxing architecture based on the Linux kernel [12]. The sandboxing concept has been adopted in the WSN paradigm in order to create a live forensics framework [13] and enable software-based memory protection [14] for wireless sensor nodes. However, sandboxing adoption in the field of IIoT has been very limited. Moreover, there remains a massive gap in using the concept in other layers in the IIoT architecture aside from the application layer.

D. Embedded Scripting

Another source of inspiration for our research is the concept of script programming. Scripting is a programming language that is at a higher-level than system programming languages (often referred to as high-level languages) such as C, C++, and Java. Instead of compiling a program ahead of time to generate machine code (as is the case with system programming), an interpreter runs script commands during runtime [15]. Among the many advantages of scripting and most relevant to our research are platform independence, rapid development and runtime execution. Unfortunately, running a traditional scripting engine on an embedded system is not trivial, which motivated researchers to develop stripped-down versions of Lua and Python, eLua [16] and MicroPython [17] respectively. Another example of optimizing scripting engines for embedded systems is TinyScript, an open source scripting engine designed from scratch for memory-constrained microcontrollers [18]; and Tapper, a lightweight scripting engine designed for resource-constrained wireless sensor networks [19]. Alternatively, projects such as TinyOS [20] focused on enabling runtime system support on resource-constrained embedded systems through a low-footprint Operating System (OS), aiming to support programming abstractions and configurations. However, due to the extensive generic features included with TinyOS (and similar embedded OS), only a handful of platforms are capable of running the OS [20].

E. IoT Sensor Platforms

Other Efforts such as Arduino and its shields [21], Cloudbit and LittleBit [22], Particle Photon [22], SPROUTS [4], and Raspberry Pi [23] have created separate sensing units that provide a simple way to interface with the host device. However, such sensing units require an additional onboard MCU and remain dependent on the platform of the host device where the driver implementation remains. Additionally, only some such efforts provide the flexibility of hot-plugging a sensing unit.

Using runtime execution on the host for the sensor has received minimal attention in the efforts mentioned earlier as

well as many others in the literature. Having the ability to execute remote code during runtime for hot-pluggable sensors will enable PnP functionality. Another advantage would be the ability to relocate the sensor driver implementation from the host side to the sensing unit side, hence, reducing the storage used by the firmware on the host as well as sandboxing the sensing unit. While using an embedded scripting engine would enable shifting the driver implementation to the sensing unit, the current state-of-the-art languages lack the capabilities to make it possible. Finally, an IIoT system architecture that involves such capabilities is needed to act as a catalyst for the advancement of I4.0. In the next section, we discuss the desired characteristics of such architecture.

III. Desired Characteristics

A catalyst for IIoT research and development is necessary in order to achieve the vision of I4.0 [1]. However, in order to act as a catalyst, an IIoT system architecture must be able to simplify sensor integration by enabling development-sandboxing, platform-independence, and True-PnP sensors, as well as being optimized for IIoT operation.

A. Enable Development-Sandboxing

The most common adoption of the sandboxing concept is in isolating the software applications. Nevertheless, sandboxing need not only be implemented for execution isolation but also has a more substantial potential in the development of an entire IIoT system. Thus, we use the sandboxing analogy in order to present the concept of development-sandboxing. In order to differentiate between both sandboxing concepts, we use the term execution-sandboxing for the traditional sandboxing technique mentioned earlier and development-sandboxing for the latter.

Contrary to its counterpart, development-sandboxing is concerned with the isolation of the development of a particular aspect of a system, including hardware and software. Isolating the different development responsibilities in an IIoT system enables the separation of concerns (SoC) paradigm and creates a modular IIoT platform. Thus, reducing the development burden and complexity on IoT system developers. To achieve a high level of system modularity, the IIoT catalyst should be able to provide an encapsulated self-contained sensor sandbox architecture, that effectively shifts the driver development responsibility from the IoT developer to the sensor manufacturer. Such SoC ensures the most efficient driver implementation while reducing the coding efforts on IoT developers and enabling development-sandboxing.

B. Enable Platform-Independence

Given the diversity of IIoT platforms available in the market, as well as the heterogeneous IIoT infrastructure components, the complexity of developing inter-connected IIoT devices rises exponentially. Therefore, an IIoT architecture that enables platform independence will serve as one of the main ingredients in the catalyst for IIoT research and development.

C. Enable True-PnP Sensors

One of the key enablers in the IoT roadmap is PnP [3]. However, in order to achieve True-Plug-and-Play (T-PnP) sensors, the IIoT catalyst must be able to detect connection/disconnection, identify, and load and run the correct driver for the connected sensor. All with zero-configuration from the IoT developers end.

D. IIoT Optimized Operation

IIoT platforms usually use a resource-constrained architecture, and with simplicity and adaptability comes challenges for such platforms. Therefore, the catalyst should be compatible with a wide range of platforms by having extremely low minimum-operation-requirements. The lower the requirements, the highly adaptable the architecture becomes, increasing its effectivity as a catalyst.

Different concepts have been proposed to fulfill such characteristics individually. However, an architecture that combines all characteristics into a single system is missing from the literature. In the following section, we discuss our proposed IIoT system architecture, SandBoxer, that is capable of fulfilling each one of the characteristics discussed earlier.

IV. The SandBoxer Approach

SandBoxer is an IIoT system architecture that isolates sensor development from embedded system development, without the need for an additional MCU. Thus, creating a low-power hot-pluggable self-contained (sandboxed) sensor architecture optimized for the IIoT. Fig. 1.a illustrates a traditional system architecture of an IIoT sensor node, while Fig. 1.b illustrates the proposed SandBoxer architecture.

Traditional sensor drivers are either coded by an IoT system developer from scratch or ported from the manufacturers provided driver library (if available). Due to the locality of such drivers being part of an IIoT sensor node architecture, the IoT system developer is required to rewrite parts of the library in order to use it on a different platform. SandBoxer proposes a different approach, the use of an EEPROM integrated with the sensing unit, similar to the IEEE 21451 standard. Contrary to the standard, we suggest the manufacturer implements the driver on the EEPROM in a platform-independent scripting language, dubbed SandScript. Thus, instead of containing a TEDS, the EEPROM shall be populated with what we call a Transducer Electronic Instruction Set (TEIS). This shift in responsibility enables platform independence by allowing a system developer to load the platform-independent TEIS from a sensors EEPROM instead of coding a platform-dependent driver. Development-sandboxing is also enabled on the sensors by isolating all sensor-specific elements (sensing element and driver) into a single unit. Finally, the implementation of the sensor driver in the form of a TEIS includes an operation implementation, thus enabling T-PnP by allowing an IoT system developer to simply run the TEIS without the need for coding an operation procedure.

The SandBoxer architecture consists of an IIoT sensor node system architecture and a sandboxed sensor architecture,

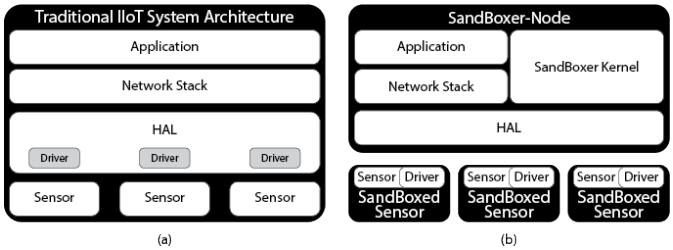

Fig. 1. Comparison between (a) a traditional IIoT system architecture and (b) a SandBoxer IIoT system Architecture

dubbed SandBoxer-Node, and SandBoxed-Sensor respectively. A SandBoxer system consists of multiple SandBoxed-Sensors that are connected to a SandBoxer-Node. Each SandBoxed-Sensor is loaded with a TEIS written in the SandScript language and built using the TEIS build toolchain.

A. SandBoxer-Node

The SandBoxer-Node must be able to load and execute a sensors TEIS while using the concept of execution-sandboxing in order to isolate and protect user processes from external threats. The architecture should also have the ability to coordinate and handle the use of multiple sensors concurrently. Finally, integrating such features into an IIoT platform must not disrupt the current architecture used. We recommend the layered system architecture approach in order to encapsulate each system and separate the concerns during development. Thus the proposed SandBoxer-Node consists of two parallel layers on top of the Hardware Abstraction Layer (HAL), namely, IoT and SandBoxer Kernel layers. The IoT layer consists of the traditional IoT sub-layers, such as a network layer, and an application layer.

The SandBoxer Kernel layer is the main scripting engine running on an IIoT sensor node, and the only component required to be used by the IIoT system developer. It is responsible for managing the connection and disconnection of supported sensors, scheduling the sensor program execution, and exposing the sensor information to the application layer. The kernel is to be provided as a light-weight platform-independent library with a transparent application programming interface (API) to simplify the integration of SandBoxer into existing IIoT systems. The integration process should involve importing the library, and ensuring that the system APIs are compatible with the kernel APIs. The proposed kernel consists of four main components, the exposer, the interpreter, the separator, and the scheduler. Fig. 2 illustrates the architecture of the kernel.

The exposer is responsible for communicating between the nodes application layer, and the SandBoxer kernel. It should also hold intermediate data such as a sensors reading for later retrieval by the application, while the interpreter is responsible for parsing a SandScript and invoking the corresponding system level functions in the HAL through the exposed API. The separator is responsible for receiving different parts of the sensors TEIS, then separating and forwarding the information to the correct component. The scheduler is responsible for initiating and coordinating the SandBoxer operation once the

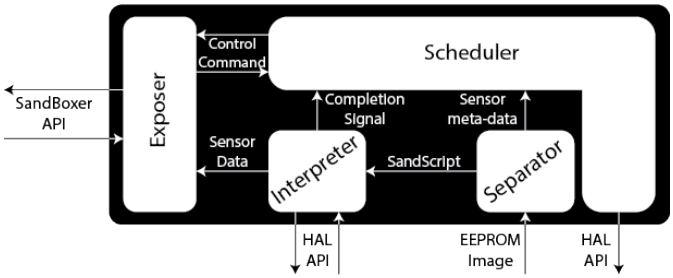

Fig. 2. SandBoxer Kernel Software Architecture

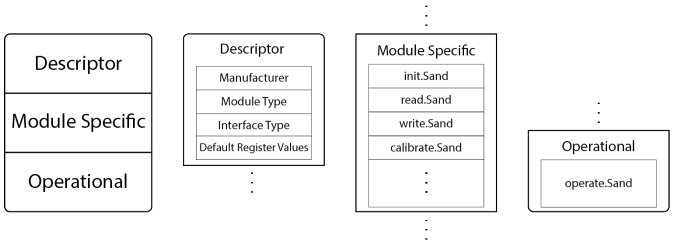

Fig. 3. The sensor SandBox memory map.

node layer initializes it. It should also schedule the execution of multiple connected SandBoxed-Sensors.

B. SandBoxed-Sensor

The SandBoxed-Sensor must have an integrated EEPROM to be consumed by a TEIS that is developed by the sensors manufacturer. The EEPROM image should be flashed by the manufacturer using a TEIS build toolchain that generates and signs the EEPROM image with the manufacturers digital signature. Thus simplifying the integration of the sensor as well as providing the IoT system developer with a method to authenticate the TEIS. The TEIS build toolchain can also be used by the IoT system developer to update the TEIS and alter the operation procedure of the sensor; however, the updated TEIS would no longer be signed by the manufacturer. Such concept enables a highly flexible sandboxed sensor architecture that allows system developers to customize the implementation of the sensor. Since the TEIS language must be platform-independent, it is essential that it be based on the concept of scripting. The use of a scripting language enables platform-independence and T-PnP by allowing an IoT system developer to run the TEIS SandScripts on any platform and with minimal involvement. A TEIS should consume the entirety of the EEPROM and consists of a Descriptor, module-specific, and an operational area. Fig. 3 Illustrates the Sensor SandBox memory map.

The descriptor area should contain key-information on the module, such as the manufacturer ID, sensor module type, communications interface, and so on. The module-specific area should contain multiple SandScripts that perform module specific operations, such as initialization, calibration, sensor reading, and so on. This section of the TEIS can be considered as the sensor driver implementation. The Operational area should contain a single SandScript that performs the sensor module operation loop. This area can be considered as the sensor application layer.

V. Design Challenges and Alternatives

The implementation of SandBoxer imposes multiple challenges; however, the SandBoxer design provides a flexible architecture that allows the developers to trade off speed, energy-efficiency, and memory-usage according to the desired application.

A. SandBoxer Kernel

Multiple challenges are inherent with interpreter implementations and usually embodied in energy consumption, processing overhead, and memory footprint. The implementation of the scheduler is possible using three main approaches, greedy, cached, and the read then delete.

1) The Read then Delete Approach: only stores the descriptor part of the TEIS for each sensor upon connection. The SandScripts are dynamically read from the sensors EEPROM to the nodes memory when needed, then immediately deleted from the nodes memory once the SandScript is processed. Using this approach compromises the kernel iteration time by increasing the execution time of each task; in return, it uses only the required minimum of the nodes memory. This approach is suitable for applications in which the time response is not constrained, but the IIoT nodes are resource-constrained.

2) The Greedy Approach: stores the full contents of each sensors EEPROM upon connection, therefore compromising the run-time memory of the node in favour of gaining an increased processing speed. Such improvement is due to the elimination of communicating with the sensors EEPROM on every iteration of the kernel. Using this approach would be suitable for applications in which memory resources are not constrained; however, the time response is critical.

3) The Cached Approach: uses a caching mechanism that only stores the frequently used SandScripts, therefore reducing the nodes memory usage while utilizing the communication with the sensors EEPROM. Such an approach would increase the complexity of the kernel implementation while optimizing energy and memory resource usage.

B. Build Toolchain

Multiple challenges are inherent with the design of Integrated Development Environment (IDE) implementations, usually embodied in ease-of-use, generated code size, and the level of debugging support. The implementer can choose to trade off ease-of-use for a higher level of debugging or choose to use a simpler compression algorithm but with a larger generated code size. The choice of generating bytecode will effectively reduce the generated code size while increasing the complexity of development. Finally, the choice of the code signing algorithm will directly affect the kernel implementation complexity and response-time.

C. SandScript Language

There are a few scripting language candidates that can be used as a foundation for the development of the SandScript language. However, the language flexibility and readability play a massive role in the adoption of the language. Flexible scripting languages such as uPython and eLua [16], [17] are bloated with features that are unnecessary in the IIoT paradigm. Such features increase the energy and memory footprint of the language interpreter. Languages such as TinyScript [18] are designed for resource-constrained embedded systems; however, lack the flexibility available in other languages.

VI. Use-Cases

SandBoxer is designed to act as a catalyst for the research and development of IIoT. However, to validate the value of our architecture, we outline several use-cases of SandBoxer in different sectors of the IIoT industry.

1) The Sensor Manufacturing Sector: Sensor Manufacturers are continually struggling to balance innovation with seamless integration, mainly due to the lack of standardization of interfaces. A manufacturer that creates a new type of accelerometer is forced to create different versions of the module for the different interfacing standards. Followed by the development of a driver library on a single platform that is to be ported by IoT system developers to different platforms. Finally, the manufacturer writes a datasheet that describes how the module is to be integrated, initialized, programmed, calibrated, and used. The porting process of the driver library by the IoT system developer might affect the performance of the sensor, resulting in possible inaccurate sensor readings and unpredicted behaviour. If the manufacturer uses SandBoxer to design the module as a SandBoxed-Sensor, the process would involve creating the modules with an EEPROM, developing the driver library using the platform-independent SandScript, and flashing the EEPROM with the generated TEIS. The datasheet, in this case, would be much simpler, as it would only describe how to use the driver. Thus, the manufacturer is confident that the driver implementation reflects the quality of the new accelerometer, achieves a broader adoption for the module, and is no longer hindered by balancing innovation with seamless integration.

2) The IIoT System Development Sector: Multiple companies offer IIoT system development services for clients that are application specific; however, only a handful of companies develop modular IIoT platforms due to its inherent development complexity. A company that is tasked with the development of an IIoT system for monitoring a manufacturing pipeline is forced to redesign its asset-tracking IIoT platform in order to incorporate the requirements of the new application. Such redesign involves the choice, development and testing of drivers, and the operation development of each of the new sensors. If the company uses SandBoxer, the redesign process would merely involve modifying the operation of the new sensors to link with the current platform. Using SandBoxer enables post-deployment flexibility that transforms the asset-tracking IIoT platform into a manufacturing pipeline monitoring IIoT platform, resulting in a cost-effective implementation and a faster time to market.

3) The Research Sector: An IoT researcher spends countless hours developing and testing on IoT platforms. The majority of those hours are usually spent integrating different sensors and modules. A researcher that is designing a new network protocol might need to test and compare the performance of different radio modules on a specific IoT platform. Such comparison involves the development of a single application and multiple drivers, one for each radio module. Using SandBoxer in such application, the researcher can diminish the number of hours spent on integrating the radio modules by simply using a SandBoxed-Sensor radio module. Thus, eliminating the hours spent on driver development and instantly test an unlimited number of radio modules.

4) The Hobbyist Sector: Hobbyists are an integral part of the research and development of IoT; however, they usually have limited experience in complex system development. Thus, the process of crafting a new idea for an IoT platform becomes very challenging. A hobbyist that wants to connect a CO2 sensor to their current smart home IoT platform is faced with multiple challenges including, finding a sensor that can be connected, finding or coding a driver for the sensor, finding or coding the application implementation for the sensor, and finally linking the operation of the new sensor with the current sensors. The difficulty of each of these challenges is inversely proportional to the development experience the hobbyist has. If the hobbyist uses SandBoxer, the challenges will be significantly reduced to finding a CO2 SandBoxed-Sensor and linking the operation with the current sensors. Therefore, creating a larger community of IoT hobbyists and subsequently accelerating the development of IIoT.

VII. Conclusion

In this paper, we introduced the concept of development-sandboxing as a viable catalyst for the development of IIoT. We outlined the desired features and proposed SandBoxer, an IIoT system architecture based on such a concept. We found it evident that SandBoxer reduces the development efforts of IoT system developers by shifting the driver development from their scope to the sensors manufacturer. We conclude that the implementation of SandBoxer presents multiple challenges and tradeoffs which we discussed in section V. We also believe that our concept is not only applicable to sensors but can be also be used for other modules, such as radio transceiver modules. We plan on implementing a prototype of SandBoxer to evaluate and compare it against bare-metal implementation. Afterwards, we plan on investigating a solution for the wide-range of interface protocols used to connect sensors. Finally, we plan on exploring the possibility of extending SandBoxer beyond the realm of IIoT and into personal IoT and mobile devices.

VIII. Acknowledgement

This research is supported by a grant from the Natural Sciences and Engineering Research Council of Canada (NSERC) under grant number: STPGP 479248.

References

[1] M. Rüßmann, M. Lorenz, P. Gerbert, M. Waldner, J. Justus, P. Engel, and M. Harnisch, "Industry 4.0: The future of productivity and growth in manufacturing industries," *Boston Consulting Group*, vol. 9, 2015.

[2] Q. Chi, H. Yan, C. Zhang, Z. Pang, and L. Da Xu, "A reconfigurable smart sensor interface for industrial wsn in iot environment," *IEEE transactions on industrial informatics*, vol. 10, no. 2, pp. 1417–1425, 2014.

[3] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[4] A. El Kouche, H. S. Hassanein, and K. Obaia, "Wsn platform plug-and-play (pnp) customization," in *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on*. IEEE, 2014, pp. 1–6.

[5] S. Agarwal, S. Majumdar, A. Maiti, and A. Nath, "Security and privacy issues of internet of things: Challenges and threats," *International Journal of Advanced Technology in Engineering and Science*, vol. 3, no. 11, pp. 89–98, 2015.

[6] T. R. Licht, "The ieee 1451.4 proposed standard," *IEEE Instrumentation & Measurement Magazine*, vol. 4, no. 1, pp. 12–18, 2001.

[7] M. Suárez-Albela, P. Fraga-Lamas, T. M. Fernández-Caramés, A. Dapena, and M. González-López, "Home automation system based on intelligent transducer enablers," *Sensors*, vol. 16, no. 10, p. 1595, 2016.

[8] A. RADOVICI, C. RUSU, and R. ERBAN, "A survey of iot security threats and solutions," in *2018 17th RoEduNet Conference: Networking in Education and Research (RoEduNet)*, Sep. 2018, pp. 1–5.

[9] D. S. Peterson, M. Bishop, and R. Pandey, "A flexible containment mechanism for executing untrusted code." in *Usenix Security Symposium*, 2002, pp. 207–225.

[10] A. Rayes and S. Salam, "Fog computing," in *Internet of Things From Hype to Reality*. Springer, 2019, pp. 155–180.

[11] C. Miller, D. Blazakis, D. DaiZovi, S. Esser, V. Iozzo, and R.-P. Weinmann, *iOS Hacker's Handbook*. John Wiley & Sons, 2012.

[12] M. S. Ahmad, N. E. Musa, R. Nadarajah, R. Hassan, and N. E. Othman, "Comparison between android and ios operating system in terms of security," in *Information Technology in Asia (CITA), 2013 8th International Conference on*. IEEE, 2013, pp. 1–4.

[13] A. Zaharis, A. I. Martini, L. Perlepes, G. Stamoulis, and P. Kikiras, "Live forensics framework for wireless sensor nodes using sandboxing," in *Proceedings of the 6th ACM workshop on QoS and security for wireless and mobile networks*. ACM, 2010, pp. 70–77.

[14] R. Kumar, E. Kohler, and M. Srivastava, "Harbor: software-based memory protection for sensor nodes," in *Proceedings of the 6th international conference on Information processing in sensor networks*. ACM, 2007, pp. 340–349.

[15] J. K. Ousterhout, "Scripting: Higher level programming for the 21st century," *Computer*, vol. 31, no. 3, pp. 23–30, 1998.

[16] elua project. [Online]. Available: http://www.eluaproject.net/

[17] C. Bell, "Introducing micropython," in *MicroPython for the Internet of Things*. Springer, 2017, pp. 27–57.

[18] P. A. Levis, D. E. Gay, and D. E. Culler, *Bridging the gap: Programming sensor networks with application specific virtual machines*. Computer Science Division, University of California, 2004.

[19] Q. Xie, J. Liu, and P. H. Chou, "Tapper: a lightweight scripting engine for highly constrained wireless sensor nodes," in *Proceedings of the 5th international conference on Information processing in sensor networks*. ACM, 2006, pp. 342–349.

[20] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer *et al.*, "Tinyos: An operating system for sensor networks," in *Ambient intelligence*. Springer, 2005, pp. 115–148.

[21] A. Nayyar and V. Puri, "A review of arduino board's, lilypad's & arduino shields," in *Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on*. IEEE, 2016, pp. 1485–1492.

[22] K. J. Singh and D. S. Kapoor, "Create your own internet of things: A survey of iot platforms." *IEEE Consumer Electronics Magazine*, vol. 6, no. 2, pp. 57–68, 2017.

[23] V. Vujović and M. Maksimović, "Raspberry pi as a sensor web node for home automation," *Computers & Electrical Engineering*, vol. 44, pp. 153–171, 2015.