2016 International Conference on Internet of Things and Applications (IOTA)
Maharashtra Institute of Technology, Pune, India 22 Jan - 24 Jan, 2016

A Model-Driven Approach to Automate the Development of Communication Channels for Internet of Things Applications

Roshan Rathod, Yu Sun
Department of Computer Science California
State Polytechnic University, Pomona Pomona,
CA 91768
Email: {rgrathod, yusun}@cpp.edu

Abstract— The paper postulates the feasibility and optimization of HVAC systems using programmable controllers. HVAC (heating, ventilating and air conditioning) systems are used for controlled maintenance of indoor ambient characteristics in optimal manner; with regards to outdoor ambient characteristics. The paper includes a simplified sequence of diagrams to represent the architecture of the system. Further, it also comprises of the systems and software's used for the same. Also, the paper gives an insight of the advantages of using programmable controllers, and the challenges which are overcome by it. For the ease of understanding a case study of a pharmaceutical company is given who is currently using the system. The main purpose of this project was to create a regulated monitoring system, integration of the utilities like electricity and water , creating a general report of the system for the given or needed time span, also to create a graphical visualization of the real time data and system.

Keywords— HVAC, PLC, SCADA, DDC

I. Introduction

Internet of Things (IoT) represents a network of physical objects containing sensors or electronic chips which are connected to the Internet [1]. With the advancement of mobile and cloud computing, IoT is gaining an increasing amount of attention due to its usefulness, flexibility, data-driven nature and cost-effectiveness. If statistics are to be believed, IoT devices have overtaken the human population by reaching 11 billion devices in 2011 and this number is expected to reach 24 billion devices by 2020 [2]. With the popularity and increasing demand of IoT applications, developers are getting attracted towards programming these devices which use the Internet for communication purposes.

A typical IoT system generally includes the hardware device, the embedded software running in the device, a web service that connects and supports all the devices, and optionally the mobile clients connecting the hardware devices with the web services [2]. The significance of IoT comes from the large amount of meaningful data brought by all the connected IoT devices. The upper part of Figure 1 shows the standard work flow for a data driven IoT system. The IoT device embedded with specific sensors consistently send the data to the data collector through the Internet or smart phones. The data collector persists the received data in database, and the data publisher fetches the data and generates analytical results and dashboards. The published data then becomes available for data consumers.

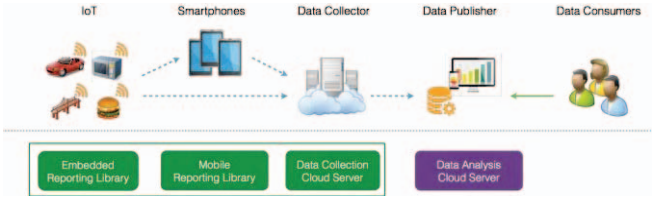

Fig. 1. The key components and work flow of a typical data-driven IoT system and the key software modules needed to implement each component

Building such a data-driven system for any type of IoT application requires developing and integrating a number of software modules. As shown in the lower part of Figure 1, the IoT devices need embedded or mobile data reporting libraries to send the data. Data collectors are implemented as HTTP web data collection servers. Data analysis tools, algorithms, and presentation views are applied in the data analysis servers to support data publishing. In order to facilitate the development of IoT systems, a number of development platforms and kits have been created to simplify the procedure of constructing the IoT systems. Development platforms and devices such as Raspberry Pi [3], iBeacons [4], and Particle [5] have been applied in practice to make these IoT devices network which helps in solving and improving a common problem in many use cases.

Open Problem: The connectivity and diversity of IoT platforms make the development of efficient, secure and reliable communication channels challenging and often complicated. The power of IoT comes from its connectivity to different components through the Internet. In order to connect these components, many efficient, secure and reliable

978-1-5090-0044-9/16/$31.00 ©2016 IEEE

communication channels have been built. However, due to the wide range of options and requirements in the field of IoT applications, developers often need to master a huge amount of technologies in order to implement the right system. Limited knowledge about IoT system components and functionality may cause the failure of the system or bring about an undesired performance. For instance, the development of IoT systems often depend on some key factors such as the platform dependency, protocols and security. Developers are not usually aware of all the available technologies in the market. Some would choose a device that is compatible with Android while the others prefer to work with Raspberry Pi. Some of these platforms may be compatible only with a messaging protocol such as MQTT [13] whereas other options are primarily based on the standard HTTP protocol such as RESTful HTTP [16]. Developers are also concerned about security, if the application involves sensitive information. However, integrating security features incorrectly may undermine the application performance which is an important factor for the end-users of the IoT system.

The implementation of the communication channels between the IoT devices and the related components is often the most challenging development task due to the variety of protocols and complexity of writing a communication framework for different platforms. There are various IoT solutions available in the market such as Amazon Web Service IoT [10] and IBM Bluemix [11], which provide a communication channel between these devices and the server. However, most of these platforms lack some or all of the following features:

- Protocols: There are various protocols in use for IoT systems, but most of the frameworks only support one type of protocol.
- Security: Lack of security can leave the system vulnerable to intruders resulting in loss/theft of sensitive information.
- Customization: Most of these services are not free and users are restricted to the service provider's server for all the data processing which leaves no room for customization. Users are provided with an interface to manage the IoT device which is developed by the provider. Adding new features for data processing such as graphical representation of data, is impossible unless the service providers offers the solution.

Solution Approach $\Rightarrow$ Model-based code generation framework for IoT communication channels. To address these challenges, this paper presents IoTCom, a model-based framework for communication between the IoT devices and the server. All IoT systems follow the same concept of sending data from devices to the server which processes the data and sends to other clients. Each device may differ based on its functionality, but the nature of communication using the Internet applies to all the IoT systems. The approach presented in this paper abstracts the common elements and entities used in the implementation of communication channels in order to generate code for various application scenarios. Let us take a couple examples to understand this:

- A fire alarm system which is triggered when the room temperature goes above a threshold or when it detects smoke.
- A doorbell which alerts the user when someone presses the bell at the door. In addition, a voice message is taken from the visitor at the door and sent to the user.

Although both the devices are unique with regards to their primary functionality, both perform data communication through the Internet. The smoke detector system is supposed to detect smoke, which is its primary functionality. However, being in an IoT system its additional responsibility would be to send the smoke detected event data to the server indicating that an event is triggered. Similarly, the doorbell sends the visitor information to the server. This implies that the data is the core component of any IoT system, which may vary in different application domains, but the implementation of the data communication channels can be abstracted and reused.

The solution presented in this paper focuses on data communication channel between the IoT device and the server. It abstracts the common code base to implement different communication channels, and ask users to input the domain-specific data fields and types, so that a complete implementation of the communication channel can be automatically generated. The main contribution of this paper is to provide the developers with a completely customizable framework which removes the overhead of writing the code for building the communication channels between the IoT devices and servers, and enabling them to use their own servers which leaves room for customizations; while incorporating key features of an IoT system such as security, diversity of platforms and protocols.

The remainder of this paper is organized as follows. In Section 2, we provide a motivating example for this paper. We list down the challenges faced while developing an IoT system in Section 3 and present our solution to address these challenges in Section 4. We analyze the related work in Section 5 and conclude this paper as well as provide some insight on future work and scope of IoTCom, in Section 6.

II. Motivating Example

The traditional smoke detector starts beeping when it detects smoke, but if the user is not around, these smoke detectors would be of no use as it is essentially just the alerting users in its vicinity by making alarming sounds. To make good use of these common and useful system that already exists in most houses and offices, smart smoke detectors are built that connect to the Internet and alert the user of any events that occur in their absence. The same technology can be applied to other common devices such as doorbell, refrigerator system or motion sensor system.

Figure 2 explains working of a smart smoke detector system connected to Raspberry Pi. The smoke detector system constantly keeps checking for any traces of smoke in its vicinity. Once it detects smoke and it goes above the threshold level, it sends the critical event information such as time of event, current room temperature and event description to raspberry pi. Raspberry Pi is a small computer [3] which is connected to

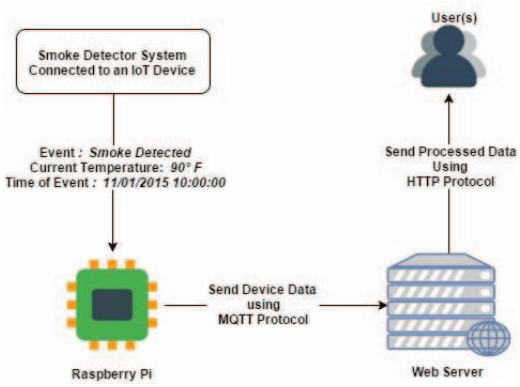

Fig. 2. A smart smoke detector device example

the Internet and sends the device data to the server using the configured protocol, in this example it publishes the data in the form of a message using MQTT protocol. The server acts as a subscriber to these messages and is configured accordingly to receive these messages. The server then processes and formats the data and forwards it to the user via HTTP Protocol. This can be a simple mobile notification or an email sent to the user over HTTP channel.

Many of these IoT systems may involve sending some private or sensitive information over the Internet, which if sent in plain text format can be intercepted and retrieved by any intruder, compromising the data and potentially the security of the system. Though there are many frameworks/services which offer an interface to connect with the IoT device and receive real time updates about events, it is not certain that the data has not been tampered while being transmitted from the device. Most of the frameworks provide features such as authentication and authorization to make sure data is accessible only to the owner which is very similar to logging into your account to check your emails; you cannot read them unless you authenticate yourself. For many individual programmers security might not be a big concern, however, for companies which cater to many clients; security could be very crucial. Knowledge about available libraries and frameworks as well as expertise on the crucial aspects of the system such as security and protocols is essential while working with IoT devices.

III. Development Challenges of IoT Communication Channels

A. The diversity of development platforms

IoT applications can be realized with different programming languages and platforms. Developers can choose between Java, C, C++, Python and others or between many other scripting languages such as Java script and Go. However, the environment of the IoT device might be restricted to certain programming language and some IoT devices may not work due to compatibility issues. Developers often need to enable different IoT devices to communicate with each other, which may be built on different platforms using different programming languages. Hence, they need to re-write the same code to work with different platforms. For instance, if the smart smoke detector system is implemented in java to work on Raspberry Pi and Android, it may not work with iOS devices due to programming language restrictions. It probably needs to be re-written in Swift or Objective-C.

B. The complexity of supporting different protocols

There is no definitive protocol to build the communication channel of an IoT system. Developers often need to use different options based on the system performance requirements and integration restrictions. Below we have listed a few commonly used protocols [6]:

- HTTP (Hypertext Transfer Protocol): This is standard protocol widely in use for many mobile and cloud applications, which is based on TCP/IP. [16]
- MQTT (Message Queue Telemetry Transport): A message-oriented protocol which allows devices to communicate asynchronously. [13]
- XMPP (Extensible Messaging and Presence Protocol): A message-oriented protocol which enables near-real-time communication for structured data such as XML. [15]
- CoAP (Constrained Application Protocol): Applied mostly in constrained networks and machine to machine applications. [14]

The smart smoke detector system connected to raspberry pi can use any of the above protocols to send the device data to the server. It could send data using MQTT for asynchronous communication and better performance or use CoAP protocol, if it is in a constrained network. Although several libraries exist to assist with implementation of these protocols, understanding all the low-level details and optimization strategies of every protocol is often a challenging task for the developers. With the diverse nature of platforms used by the IoT systems, the need for using diverse protocols arises; which forces developers to spend time and effort to master each type of protocols. Moreover, developers sometimes have to change these protocols due to performance requirements. For instance, MQTT protocol sends data much faster compared to HTTP [6]. If the IoT device is currently working on HTTP, modification has to be done for the entire piece of communication code i.e. on the device as well as the server, in order to switch to MQTT protocol.

C. The lack of a standard and reliable security control

Data security has become a top priority in all the software applications. In the context of IoT applications, one of the most critical requirements is to encrypt all the data that is being sent to the server over the Internet, particularly when it relates with users private information. Furthermore, authentication also plays a key role in an IoT system to restrict access to all the user data. Developers often realize the significance of the security but lack the expertise to offer the right implementation. Ensuring security is a complex process and requires knowledge on various factors such as performance and scalability. Ensuring the right security implementation is not trivial; Adding a security layer without knowing the effect

of its performance on the entire system could result in a system with potential security risks.

D. Obscure development libraries and coding practices

Programming the different IoT application components has become increasingly easier with the use of various libraries and frameworks available to developers. For instance, using libraries such Retrofit [18] in Android or SpringBoot [17] in Java has become a new standard in the community, but not all developers are aware of these trends. This increases the turnaround time for projects because something that could have been achieved easily through the use of libraries is written from scratch which doubles the time taken for the project completion as well as increases the required effort by the developer.

E. The difficulty of ensuring maintainability and scalability

Software systems are meant to evolve over time and same applies to the IoT systems. Adding more features to the IoT system often involves updating the data format or sending more event information to the server. For IoT applications, source code is usually built and deployed across different platforms which complicates the maintainability and extensibility of the system. For adding even a small feature, changes may be required to multiple files and the developer needs to incorporate these changes in each and every platform specific module. For example, if the developer wants to add a new variable in the smoke detector system, he will have to modify each and every module to incorporate the changes. Hence, it becomes essential to organize and manage code in an efficient way making the addition of new features hassle-free and eliminating the risk of breaking the existing code.

IV. Solution

To overcome the challenges and provide easier ways for a developer to work with the IoT devices, we present a model-driven framework - IoTCom that enables the automatic generation of the communication channels in IoT systems. As explained in the earlier sections, IoT devices communicate with the server and exchange data using specific protocols as configured by the developer. The server is configured to receive this data for further processing.

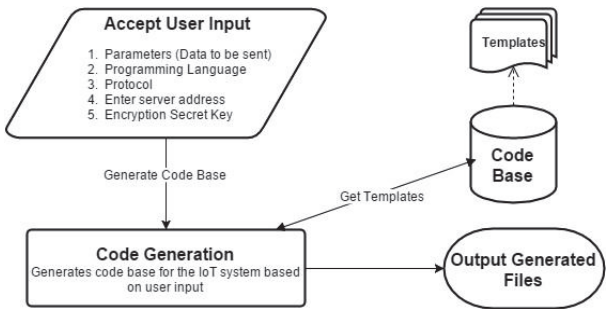

Fig. 3. An overview of IoTCom framework

Figure 3 shows an overview of the IoTCom framework. It starts by asking users for some basic metadata information used in the communication channels, including the data fields and parameters to be sent to the server, programming languages to be applied, the chosen communication protocol, the server endpoint as well as a private secret key for the device data encryption. Once the user inputs are finalized, the framework looks for the templates in the code base, in order to adapt to the protocol and the programming language selected. Once the right template is located, the customized parameters will be configured to send the domain-specific data to the server. Using the given server endpoint address, IotCom will configure the code base accordingly to send the data to the target server. Alternatively developers can manually enter the information directly into the code base. The generated code bases can be directly injected into their existing implementations by calling the functions responsible to send the data to server and supplying it with the correct input parameters. The generated code base handles the entire communication channel implementation.

A. Implementation

IoTCom provides a command line interface to the developers, as shown in the Figure 4. Running the IoTCom will trigger the user input on a series of questions needed for code generation.

Fig. 4. The command-line based interactive UI

IoTCom starts by taking the parameters that the IoT device will be sending to the server. Consider a temperature monitoring system which alerts the user that the temperature has gone above 90 degree Fahrenheit at 1PM. In this case, the parameters would be temperature and time of event. Then it gives a list of available programming languages for the user to select, which can be done using the arrow keys. Next, it provides with a list of available protocols to select from. Template also requires the server address to configure where the data has to be sent. For the encryption bit, IoTCom will ask the developers for a secret key to perform AES-128 encryption [21] on the data. Since it is a symmetric encryption, the engine would need just one key which the user can remember and can be used to retrieve encrypted data on the server.

Based on the selection of protocol and programming language, we fetch our pre-built templates and incorporate the user provided information into the template and generate all

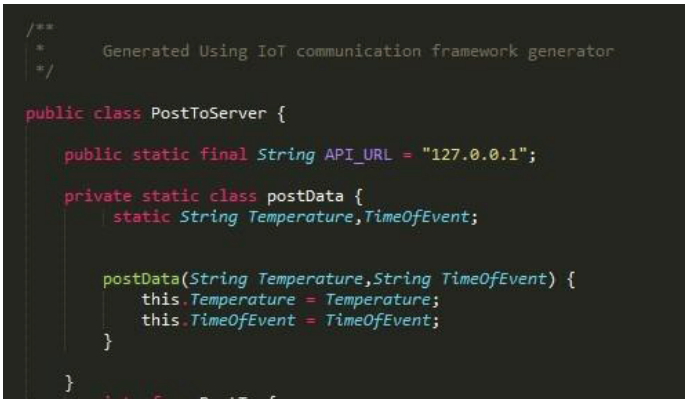

Fig. 5. Code Snippet from the Generated Files

the source code. The generated code base mainly consists of client side code for the IoT Device meant to send device data to the server(snippet shown in figure 5), file containing encryption and decryption logic which can be used on client side for encryption and server side for decryption using respective function calls, and the server side file configured to receive the device data; along with other helper files. These templates are written using different programming languages to handle communication in an IoT system using different protocols to send/receive data.

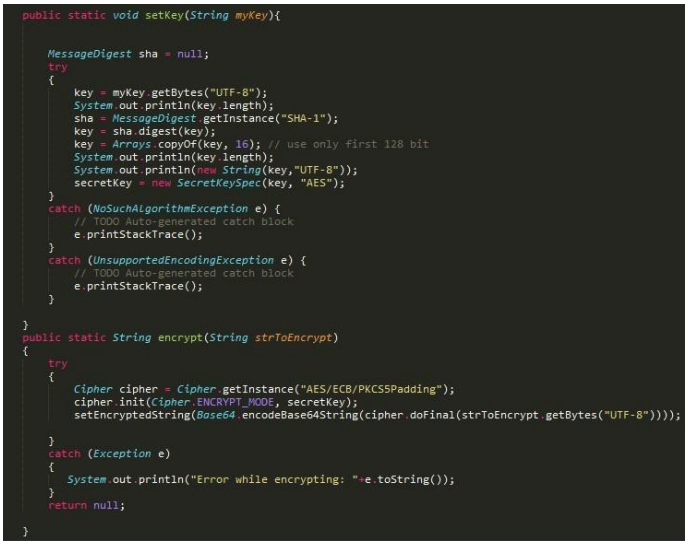

Fig. 6. The generated code implementing AES-128 encryption

The code base comes bundled with key security features which encrypt all the IoT device data using AES-128 before sending it over to the server. Figure 6 shows a code snippet used for encryption of the data. Developers do not need any changes to the encryption code, IoTCom handles all the configuration while generating the code base. All the data is encrypted by calling encryption functions before sending it to the server. The secret key for the server is provided by the developer and can be modified at any time in the generated code base.

Since IoT devices do not have enough processor speed to handle asymmetric encryption algorithms, symmetric algorithms are widely used in IoT applications [7]. We decided to use AES-128 for encryption due to its performance over other symmetric algorithms which makes it comparatively faster and secure than others [8]. These templates have been created by incorporating best coding practices with regards to an IoT system such as performance, security, flexibility, maintainability and scalability.

IoTCom is built using Yeoman Generator [9], which enables us to create a simple command line application to ask a set of questions to the user and generate the communication channel based on the input. Once the code base is generated, we output all the source code in a folder which can be used by the developers in their existing code base. Node.js [19], npm [20] and yeoman generator are some of the prerequisites to run IoTCom.

B. Addressing the Challenges

Below we briefly discuss how IoTCom helps to address the challenges described in Section 3.

1) Diversity of Development Platforms: We believe that flexibility is an important quality of any software system. IoTCom allows developers to select the programming language used to generate the code base, based on the specific development platform. The code base can be easily integrated with any existing IoT systems and are completely customizable. They can use the code base as it is generated or integrate system specific requirements easily on both the IoT device and server, whereas it is almost impossible to make any changes on the server side if using a third party service.

2) Diversity of Protocols: Protocols are an integral part of an IoT system which help in improving the overall performance of a system. IoTCom offers various protocols options to the user to select from, while generating these code bases. This allows them to use the same application in different environments based on system requirements. For example, if developers have an existing system which works on XMPP protocol and need the device to work with HTTP and MQTT protocols as well, they can use the generator to get files for the needed communication protocol instead of investing time in rewriting the entire code based to enable different protocols.

3) Secure Communication: While working with sensitive user data, it is important to secure the data to prevent any loss of the information. We have incorporated AES-128 encryption algorithm in IoTCom to encrypt all the data that is sent via the IoT device to the server to ensure a higher security level. The data is encrypted on the IoT device and sent to the server where it is decrypted back to plain text for further processing. Developers have complete control over the encryption and decryption process as well as setting the secret key to protect the data from any intruders while being transmitted over the Internet. In addition, it protects the receiving process on the server, since the data can never be read without the secret key.

4) Library Reuse and Adaption of Code Practices: As discussed in earlier sections, not all the developers are aware of the latest code standards and helpful libraries that could be used to increase performance of the IoT system. IoTCom

templates have been equipped with the latest libraries and ensured correctness with test cases as well as, based on our experiences, applied best programming practices in order to provide a superior performance. The code generation framework can also help new IoT developers to improve their knowledge on IoT programming and code standards.

5) Improved Maintenance and Scalability: All systems evolve based on user feedback and feature requests. It is very common to receive new updates to a system especially in IoT systems, with the advancement of technologies. IoTCom provides the source code to the developers which gives them complete control over the IoT system, and makes it really easy to add new modifications or features at any given time. The code is well organized by keeping maintainability as well as the performance standards in mind using single responsibility functions and incorporating various programming paradigms. This makes it easier to incorporate future modifications, because all the changes are needed in one file rather than spread across the entire project.

V. Related work

Being an emerging topic, not many IoT development platforms or services are available. Amazons AWS IoT [10] allows users to connect to their IoT devices using their admin portal. Its security standards involve authentication and authorization for all the connected IoT devices using their platform. Similarly, IBM's IoT solution BlueMix [11] allows user to connect any of the IoT devices to IBM servers and manage the devices via the management portal. BlueMix sets up device on the cloud and connects it using the light-weight MQTT protocol. In both the cases, the data, while it leaves from the device, may not be secured which leaves place for a man-in-the-middle attack [12]. Data while being sent to their servers is not encrypted which means any intruder can get the data while it is being transmitted through the communication channel. Users need to use the service providers server for managing the data and hence should be able to completely trust the provider for the safety of their data. If the server is compromised, so is the device data.

Furthermore, customizations are not supported with AWS IoT, since user have no control over the portal, although it does offer features like real-time device monitoring and data analysis tools, which removes some overhead from the developers. BlueMix, on the other hand, does provide some more flexibility to the developers with regards to the device data by providing real-time APIs to fetch the data received from their device onto IBM servers, which can be used for data analysis on the developers own server.

Our solution - IotCom, was developed based off features that are not being offered by these industry tools. There are many aspects to keep in mind while selecting a platform for an IoT device such as functionality, scalability, security, adaptability, protocol being used and performance. We have developed our system around these features to overcome the challenges faced by the developers and to provide them a completely customizable solution by offering the automatic generated source code.

VI. Conclusion and Future work

In this paper, we aim to provide an automated and customizable development solution for the IoT application developers while keeping the most important aspects of an IoT system intact, such as performance, security and flexibility. IoTCom has a very huge future scope due to the diversity of IoT platforms and protocols. We have currently implemented only two majorly used programming languages and protocols to test the working of the system. As the future work, we will expand it to include more programming languages and protocols by writing the templates and extending the code base. We also would like to build an open-source community for IoTCom, so that more developers can contribute to the template specification and code base construction.

References

[1] Atzori, Luigi, Antonio Iera, and Giacomo Morabito. The internet of things: A survey., Computer networks 54, no. 15 (2010): 2787-2805.

[2] Gubbi, Jayavardhana, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions Future Generation Computer Systems 29, no. 7 (2013): 1645-1660.

[3] Raspberry Pi, 'Raspberry Pi - Teach, Learn, and Make with Raspberry Pi', 2015. [Online]. Available: https://www.raspberrypi.org/. [Accessed: 06- Nov- 2015].

[4] Developer.apple.com, 'iBeacon for Developers - Apple Developer', 2015. [Online]. Available: https://developer.apple.com/ibeacon/. [Accessed: 06- Nov- 2015].

[5] Particle, 'Build your Internet of Things', 2015. [Online]. Available: https://www.particle.io. [Accessed: 11- Nov- 2015].

[6] Hersent, Olivier, David Boswarthick, and Omar Elloumi. The Internet of Things: Key Applications and Protocols John Wiley & Sons, 2011.

[7] Jing, Qi, Athanasios V. Vasilakos, Jiafu Wan, Jingwei Lu, and Dechao Qiu. Security of the Internet of Things: perspectives and challenges. Wireless Networks 20, no. 8 (2014): 2481-2501.

[8] Nadeem, Aamer, and M. Younus Javed. A performance comparison of data encryption algorithms. In Information and communication technologies, 2005. ICICT 2005. First international conference on, pp. 84-89. IEEE, 2005.

[9] Yeoman.io, The web's scaffolding tool for modern webapps — Yeoman', 2015. [Online]. Available: http://yeoman.io/. [Accessed: 06- Nov- 2015].

[10] Amazon Web Services, Inc., AWS IoT', 2015. [Online]. Available: https://aws.amazon.com/iot/. [Accessed: 06- Nov- 2015].

[11] Ibm.com, 'Internet of Things (IoT) Solutions - IBM Bluemix', 2015. [Online]. Available: http://www.ibm.com/cloud-computing/bluemix/solutions/iot/. [Accessed: 06- Nov- 2015].

[12] Desmedt, Yvo. Man-in-the-middle attack. In Encyclopedia of Cryptography and Security, pp. 759-759. Springer US, 2011.

[13] Mqtt.org, 'MQTT', 2015. [Online]. Available: http://mqtt.org/. [Accessed: 06- Nov- 2015].

[14] Coap.technology, 'CoAP Constrained Application Protocol — Overview', 2015. [Online]. Available: http://coap.technology/. [Accessed: 06- Nov- 2015].

[15] Xmpp.org, 'Protocols The XMPP Standards Foundation', 2015. [Online]. Available: http://xmpp.org/xmpp-protocols/. [Accessed: 06- Nov- 2015].

[16] D. Elkstein and V. profile, 'Learn REST: A Tutorial', Rest.elkstein.org, 2015. [Online]. Available: http://rest.elkstein.org/. [Accessed: 12- Nov- 2015].

[17] Projects.spring.io, 'Spring Boot', 2015. [Online]. Available: http://projects.spring.io/spring-boot/. [Accessed: 6 Nov- 2015].

[18] Square.github.io, 'Retrofit', 2015. [Online]. Available: http://square.github.io/retrofit/. [Accessed: 06- Nov- 2015].

[19] N. Foundation, 'Node.js', Nodejs.org, 2015. [Online]. Available: https://nodejs.org/. [Accessed: 06- Nov- 2015].

[20] Npmjs.com, 'npm', 2015. [Online]. Available: https://www.npmjs.com/. [Accessed: 06- Nov- 2015].

[21] Daemen, Joan, and Vincent Rijmen.The design of Rijndael: AES-the advanced encryption standard. Springer Science & Business Media, 2013.