Security of Emerging IoT Systems

by

Mehdi Karimibiuki

M.A.Sc., University of British Columbia, 2012

B.A.Sc., University of British Columbia, 2009

Supervisors: André Ivanov and Karthik Pattabiraman

A THESIS PROPOSAL SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

(Electrical and Computer Engineering Department)

The University Of British Columbia

(Vancouver)

October 2017

Contents

List of Tables

List of Figures

Glossary

The following are abbreviations and acronyms used in this proposal, listed in alphabetical order:

ACL Access Control List

DynPolAC Dynamic Policy-based Access Control

IETF Internet Engineering Task Force

IoT Internet of Things

MQTT Message Queuing Telemetry Transport

RQ Research Questions

SDC Self Driving Cars

SDIoT Software Defined IoT

UAS Unmanned Aircraft Systems

WSN Wireless Sensor Networks

Chapter 1

Introduction and Overview

The state-of-the-art technology today is governed by the **fourth** industrial revolution, the **Internet of Things (IoT)** [2]. Internet connection is not a fancy feature anymore and can nowadays be found on almost all electronic devices [3]. Our social and networking services as well as science and engineering resources are unconditionally dependent on the Internet. An IoT roadmap reports that there will be 50 billion devices connected to the Internet by 2020 (Figure 1.1) [4].

The result of connecting every and many objects (things) to the Internet is dealing with the network complexity and data explosion. In particular, our current IoT networks inherit the following specifics:

- Network connected IoT devices generate live data and hold ubiquitous information, exchanged between objects in real-time. Consequently, connecting "smart devices"[1] implies dealing with dynamic data exchange intricacy continuously.
- Dealing with enormous amounts of data pushed and pulled every second necessitates large bandwidth and connection strategies. A dynamic and scalable management system is required to deal with transfer, sharing, storage, privacy and security challenges.
- IoT connected devices are heterogeneous — the data generated by the sensors or consumed by the actuators are different in structure and diverse in communication. Usually, information needs to pass through more than one communication medium to reach a target platform [5].

[1] A "smart device" is any electronic device that can communicate to other devices via a wired or wireless communication medium like LoRa, CoAP, WiFi, LAN, Modbus, CAN, NFC, BLE, MAVLINK, WAVE, etc. Several notable types of smart devices are smartphones, tablets, laptops, smart-watches, smart-bands, smart TV, thermostats, energy meters, and many other smart home and office appliances.

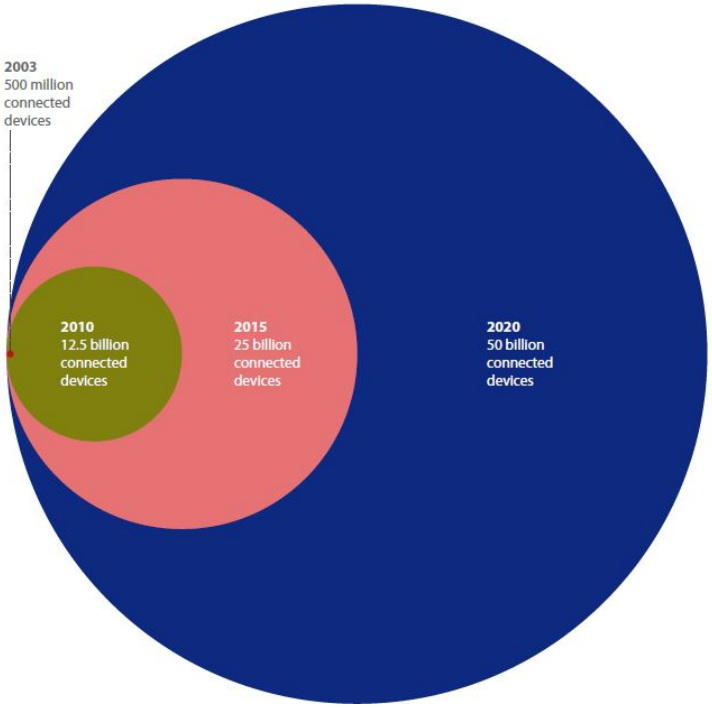


Figure 1.1: Bubble boom — *50 billion* devices shall connect to the Internet by 2020 [4].

As a result of the above particulars to IoT systems, and during the time that I started my PhD program in September 2015, I realized that main stream IoT research has generated different data models and information management solutions to mitigate the complexities involved with such networks. There had been proposals which described a universal architecture in IoT between heterogeneous devices [3, 6]. There were also IoT architectures that developed cloud connectivity in their network architecture [7–10]. These IoT architecture proposals are very numerous[2]; however, most of them lack implementation details, or did not provide any open-source access to their implementations to be able to evaluate their process.

Therefore, in the first year of my PhD work as I learned about the IoT network specifics and the fact that previous research has modeled IoT space with little effort in implementing envisioned architectures, I started designing my own IoT system that implements a gateway node.

In this PhD proposal, my design is used as a substrate for experimenting our IoT ideas and has the capacity to have several devices connected to it based on the available interfaces of the board. The IoT node that I developed is an abstract model for covering heterogeneous devices' connectiv-

[2] A Google Scholar search for "IoT + architecture + cloud" reveals more than 16,000 results by 2015.

ity and is called "MiniCloud". I called my design "MiniCloud" because it can be used as a cloud storage to a local network. It is capable of covering device connection and data management at the local network level, and at the same time, publish or subscribe, filter and distribute requests coming from the enterprise cloud services such as the Microsoft Azure. I will describe the characteristics of the *MiniCloud* in Section 1.1.

In addition to the communication and data exchange complexities in IoT networks, *security* in IoT devices is known to be the biggest challenge yet — many devices and communication channels that are being used in IoT space today, initially, were not meant to be universally applied in public network communications [3].

Additionally, in the MiniCloud, I overlooked the confidentiality of information and protection of data. This fact required me to study the *security in IoT systems* and learn about their common attacks. I took two graduate courses in this field (EECE 571R and EECE 512). By having the knowledge and understating that there are many security deficits in the IoT space, I have decided to conduct my PhD research in the security of the IoT systems, in particular, the emerging IoT networks. Throughout the course of conducing my research I am using the *MiniCloud* as the real embedded system to carry forward and implement my security ideas with it.

My main concentration in this PhD proposal is answering research questions (Section 1.3) pertaining to information security for dynamic IoT nodes. Information security, in general, can be looked at from two angles. A first is to keep the confidentiality of information from unknown, untrusted parties or adversarial attacks, especially in the emerging highly interactive environments such as Self Driving Cars (SDC) and Unmanned Aircraft Systems (UAS) where highly dynamic information exchanges occur in time windows of one second or less. In such compounds, adoptable data protection techniques needed to cope with such high level of dynamism. A second is to keep the integrity of information from the source (research questions related to tampering concerns) and from their exchange media (snooping and sniffing concerns). I will discuss the main findings of my research about the security of highly dynamic IoT nodes in detail from Section 1.2 onward.

1.1 MiniCloud: A Local Cloud for Heterogeneous IoT Nodes

For the pupose of evaluating my research ideas particularly in the *security* domain, I have constructed a middleware called "MiniCloud" that is capable of collecting and recording live-stream

[3] Some communication protocols such as Modbus [11] and Message Queuing Telemetry Transport (MQTT) [12] do not even have any security measures in their core stack design. This is while many of our home and enterprise equipment such as thermostats (like Nest [13]), video streaming cameras (like Dropcam [13]), energy meters, and sensors extensively use similar vulnerable communication solutions for streaming live private data.

data. *MiniCloud* is comparable in size and implementation to similar in-house embedded IoT storage systems such as TinyDB[4][5] [14]. It is also close to the smart gateways such as the one implemented by the Pervasive Computing Lab at ETH[6] [15].

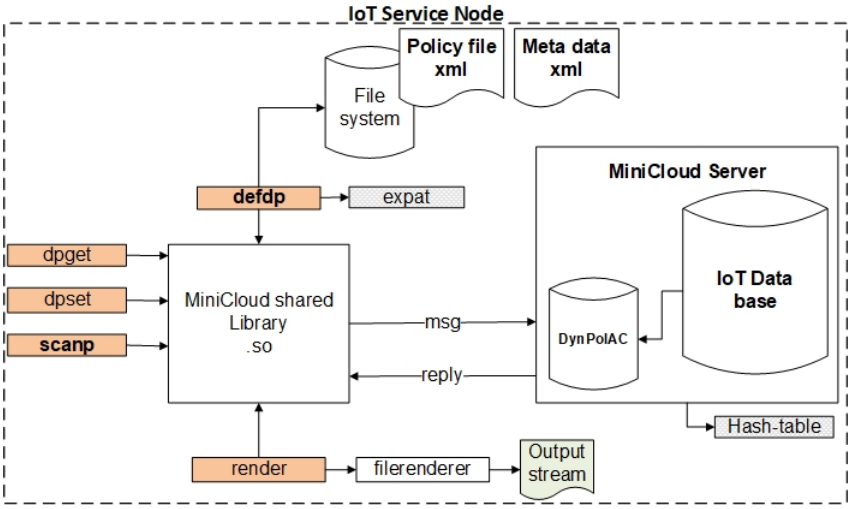


Figure 1.2: "MiniCloud" service node block diagram.

MiniCloud as shown in Figure 1.2, consists of two components, a server and a shared library. The server is accessible via only the shared library bundle. The shared library APIs are provided as C functions. The source code written in 3,091 lines of code (source and header) at image size 31,145B. The server implemented in C language at 4,854 lines (sources and headers), compiled with gcc v4.7[7] sized 34,796B. For connecting to the server, the client must first open and register connection with the MiniCloud to obtain access to the database. The client side provides a collection of Open, Close, Register, Set, Get, and Print APIs to communicate with the server. Any other method to contact with the MiniCloud server other than the provided APIs is inhibited by memory protection nature of the underlying operating system[8]. Figure 2.3 and Figure 2.4 show quantitative comparison between the MiniCloud and TinyDB. These Figures are provided as a proof that my

[4]TinyDB is comprised of 10,000 lines of C code excluding driver sections with compiled size 58KB [14].

[5]We picked TinyDB for out-of-the-box comparison because it is similar to our MiniCloud implementation design. Similar to TinyDB, MiniCloud provides a set of APIs for recording data that sample sensory events periodically.

[6]Details of the gateways designed by ETH Lab has not been provided nor their detail implementation method [15].

[7]GCC is the GNU C Compiler.

[8]Current implementation of the MiniCloud is developed in QNX real-time operating system (RTOS) supported by BlackBerry [16]. QNX is a reliable software service platform with exclusive Inter-Process Communication (IPC) that yields memory and stack protection. QNX is currently running in many mission-critical platforms and gives support to numerous embedded IoT hardware with excellent documentation and tool-chain diagnostic capabilities.

data storage service is a sizable implementation and a design close to TinyDB. One exception is with my design, since I have developed the code myself, I can easily modify the code as needed to suite the context of my PhD research. If I have picked TinyDB for instance as our node storage system to test our security ideas, there could have been risks involved in learning their system and to come up to speed with understanding their system deficits. Therefore, I chose not to pick third party IoT systems and, instead, had the knowledge and experience to come up with my own design.

In the following, I describe MiniCloud key characteristics (Subsection 1.1.1) with platform implementation (Subsection 1.1.2).

1.1.1 MiniCloud Database Server

The MiniCloud IoT hub is a real-time storage and sharing node implemented as an umbrella for collecting local heterogeneous devices' data. MiniCloud is "easy-to-connect-to" via a set of APIs. Below are the MiniCloud property highlights:

- MiniCloud presents an abstraction layer for heterogeneous data, which allows *loose coupling* between different IoT devices. Modules can be added or dropped at run-time without service interruption. Product behaviors can be extended or reduced at run-time depending on the detection of other modules within the system. For example, a module could query Ambient Temperature of the system, simply by requesting it by name. If the requested datapoint (variable) is not found, the requesting module knows that the Ambient Temperature is not available and can automatically adjust its behavior.

- Creation of data are made at single variable types. Table 1.3 enumerates possible data types available in the MiniCloud. Such granular data precision gives flexibility to monitor and manage singular information as needed for most IoT paradigms such as autonomous vehicles, Wireless Sensor Networks (WSN), miniaturized motes, etc. [17, 18].

- Additionally, single data point granularity in the MiniCloud provides a powerful templating mechanism. It can be used to provide dynamic output using static templates (XML or JSON template for example) by substituting run-time values for any name discovered in the template. The templating mechanism is also recursive in nature, so one template may reference a variable which refers to another template, and so on. Templates are often used for the web-page generation, email generation, faults and warning generation, and many other applications. Templates also support code-reuse since it is not necessary to hard-code reports. A sample is shown in Figure 1.3 where command '*scanp*' is querying for JSON and

XML printout of data variable names that contain ‘tesla’ (use command option ‘-k tesla’), and show them with their values (command option ‘-v’).

```
# scanp -v -j -k tesla
{
    "values": {
        "temperature.object.tesla": 25.50,
        "speed.object.tesla": 80.00,
        "Latitude.object.tesla": 90.00,
        "Longitude.object.tesla": 180.00,
        "Altitude.object.tesla": 50.00,
        "Drone.heading.tesla": "North",
        "Drone.SerialShell.password.tesla": "t@sL1",
        "Drone.Fuel.Level.tesla": "full",
        "count": "8"
    }
}
#
# scanp -v -x -k tesla
<?xml version="1.0" encoding="utf-8"?>
<defdp>
  <var name="temperature.object.tesla" value="25.50"/>
  <var name="speed.object.tesla" value="80.00"/>
  <var name="Latitude.object.tesla" value="90.00"/>
  <var name="Longitude.object.tesla" value="180.00"/>
  <var name="Altitude.object.tesla" value="50.00"/>
  <var name="Drone.heading.tesla" value="North"/>
  <var name="Drone.SerialShell.password.tesla" value="t@sL1"/>
  <var name="Drone.Fuel.Level.tesla" value="full"/>
</defdp>
#
```

Figure 1.3: Shell snapshot of the MiniCloud query in JSON and XML.

- MiniCloud simplifies the communication between heterogeneous devices because each device is connected to a common shared middleware which is capable of treating data at single data-point granular resolution. It is not necessary to create rigid data structures which suffer from version incompatibilities.

- IoT devices can present their data by running the “definition datapoint” application (*defdp* in Figure 1.2), which registers datasets with the server. Listing 1.1 shows a sample registration file, which is parsed by the *defdp* (see Figure 1.2) application). As can be followed from Listing 1.1, every data-point carries properties such as *name*, *format*, *initial value*, and *tags*. These fields are mandatory and it is to the discretion of the clients to supply a correct meta

data file. Tags provide the *data-type*, *location*, and the *user* and *group* belongings. As shown in Tables 1.1 and 1.2, users and groups are conventionally made. Analogous to the Linux filesystem[9], permissions in the MiniCloud has been architected to be intuitive such that every data-point is given a user and group owner based on the Access Control List (ACL). Users are not necessarily individuals, rather they can be IoT objects trying to query data from another node. Tables 1.1 and 1.2 are sample matrices, and the list can be edited by the authority convention.

Table 1.1: Example ACL for the MiniCloud assigned group numbers and conventional group names.

Group ID	Group name
1049	Manager
1048	Engineering
1047	Technician
1046	Customer

Table 1.2: Example ACL matrix between the user and group for data-points in the MiniCloud server.

User ID	User Name	Group Member
549	Gus	Manager
548	Doug	
539	Mike	Engineering
538	Tom	
429	Jackie	Technician
428	Lilli	
359	Bob	Customer
358	Madi	

- Similar to other IoT data storage systems such as TinyDB [14], MiniCloud can update data via polling mechanisms (for example, polling new temperature value from a sensor every 10 seconds), or by registered callback event handlers (for example, if the value of the temperature sensor changed, update the particular data-point).

- MiniCloud while time-stamping and keeping the latest values of the live-stream data, is capable of taking snapshots and logging data-points at particular time intervals. For example,

[9]Every file and folder in the Linux filesystem has an owner (user) and is member of a group.

it is possible to tell MiniCloud to take snapshots of the entire database every minute. The time interval can change at run-time without service interruption. For example, it is possible to change a one minute snapshot to a half-an-hour one routine, if no transient behavior experienced within the monitoring service.

```
<?xml version="1.0" encoding="utf-8" ?>
<defdp>
  <point>
    <name>Google.password</name>
    <description>Password for entering the Google
        device</description>
    <type>str</type>
    <format>%s</format>
    <value>g@o&le</value>
    <length>8</length>
    <meta>
      <tag>type:password</tag>
      <tag>location:Google</tag>
      <tag>user:Bob</tag>
      <tag>group:Customer</tag>
    </meta>
  </point>
  <point>
    <name>Rakuten.latitude</name>
    <description>Live latitude registration of Rakuten
        drone</description>
    <type>float</type>
    <format>%f</format>
    <value>0</value>
    <meta>
      <tag>type:latitude</tag>
      <tag>location:Rakuten</tag>
      <tag>user:Tom</tag>
      <tag>group:Engineering</tag>
    </meta>
  </point>
  <point>
    <name>Uber.temperature</name>
    <description>Cabin temperature in the Uber autonomous
        car</description>
    <type>sint32</type>
    <format>%d</format>
    <value>25</value>
    <meta>
      <tag>type:temperature</tag>
      <tag>location:Uber</tag>
      <tag>user:Doug</tag>
      <tag>group:Manager</tag>
    </meta>
  </point>
</defdp>
```

Listing 1.1: Sample data definition file. Larger data sets are available at www.ece.ubc.ca/~mkarimib/minicloud.

Table 1.3: Variable Types allowed in the MiniCloud server

Variable Type	Description
uint16	unsigned 16-bit integer (0-65535)
sint16	signed 16-bit integer (-32768-32767)
uint32	unsigned 32-bit integer (0-4294967295)
sint32	signed 32-bit integer (-2147483648-2147483647)
float 32	IEEE754 single-precision floating point
str (string)	NULL terminated C-style string (length specified by datapoint (variable) creator)
conjugate(binary or struct)	Blob of data that can be a structure type or array of bytes

1.1.2 Platform

MiniCloud is implemented in QNX RTOS [16] running on BeagleBone Black (BBB) [19] — BBB is an evaluation kit offered by TI, embedded with ARM Cortex-A8 processor with 720 MHz clock speed, 512 MB of RAM, on-chip Ethernet, a microSD slot, and two 46-pin expansion connectors for GPIO, SPI, I2C communication as well as CAN and UART compatibilities. The board is good enough to connect with several sensors and devices as needed. Figure 1.4 shows BBB hardware details and an instance connection with our available sensors in the lab.

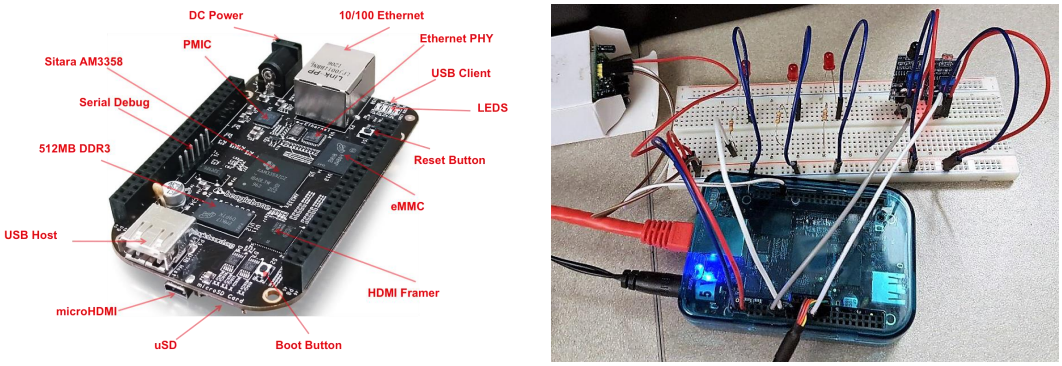


Figure 1.4: BeagleBone Black [19].

1.2 Future of IoT Networks

The rapid emergence of the Internet of Things (IoT) is bringing an unprecedented expansion of object-to-object or thing-to-thing wired and wireless communications. There now exists relatively well-established stationary IoT networks such as smart grids, smart buildings, smart factories, etc. However, the next generation IoT systems are prevalent by a key characteristic, *highly dynamic* wireless objects. Two candid emerging examples are UAS (also known as drones), and SDC (also known as autonomous vehicles). The Federal Aviation Administration (FAA) is forecasting the number of UAS will grow from 2.7 million in 2016 to 7 million by 2020 [20]. Similarly, there will be at least 10 million self-driving cars on the roads by 2020 with trajectory forecast that one in four cars will be autonomous by 2030 [21].

The growing trend proves there shall be a demanding need for adaptable real-time, low-latency communication frameworks with data privacy protection and information security considerations. Towards this observation, I realize that there is an emerging dichotomy of visionary communication schemes to govern in wireless IoT networks, as depicted in Figure 1.5.

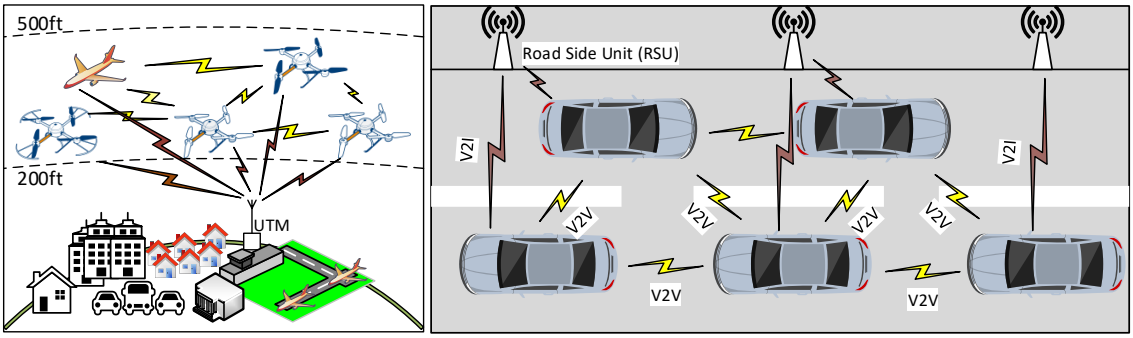


Figure 1.5: Dynamic IoT communication models.

The first scheme is to have objects query information from a central hub. One example is the UAS Traffic Management (UTM) system intended to mediate the safety of miniaturized flying objects by providing services such as dynamic geofencing, severe weather and wind avoidance, terrain avoidance, route and re-route planning, collision avoidance, etc. [22]. Similarly as part of the Intelligent Transportation System (ITS) there will be Road Side Units (RSU) to govern self-driving cars with traffic and safety information.

A second, longer span, visionary scheme, is to have the autonomous objects communicate with each other exchanging information directly. Table 1.4 shows some real-world examples of dynamic IoT networks with their anticipated interaction frequency and communication channel.

Table 1.4: Examples of emerging IoT networks.

Scenario	Examples	Anticipated Network Interaction	Communication Medium
Self-driving (autonomous) Cars	Tesla, Uber, Google, Delphi, Faraday Future, Waymo, etc.	6 objects per second	IEEE 802.11p WAVE, 5.9GHz band
Unmanned Aircraft Systems (UAS) Drones	Parcel Delivery: Amazon PrimeAir Search and Rescue: sardrones.org Wild-life Conservation: conservationdrones.org First Aid: TU Delft - Ambulance Drone Agriculture: senseFly Security: secom.co.jp	4 objects per second	MAVLINK
Transport Logistics	Warehouse Robots such as Kiva Knapp Open Shuttle Locus Robotics System	2 objects per second	Infrared, IEEE 802.11b and Bluetooth

Nevertheless, in many of these cases, there is little published work on the underlying data protection mechanisms. Nor is there a public document to outline security standards in the highly dynamic IoT environments. A main contribution is to find the right security methods that give the correct direction in protecting information of emerging IoT nodes.

The first secure preparation of keeping data confidential is controlling access. Examples that require a mandatory data access control are military drones in mission-critical assignments, or, autonomous cars sharing passengers' connected phone with the outside world. Some attempts for protection of data in current IoT networks have been given by adapting popular access control frameworks. One strategy has been to adopt access control techniques traditionally known as Role-Based-Access-Control (RBAC) and Attribute-Based-Access-Control (ABAC) [3, 23–27]. These techniques, though suitable for static IoT systems, are not fit to work well with the dynamic IoT systems under consideration here. Another solution has been given as the employment of the policy files using eXtensible Access Control Markup Language (XACML) [28–33]. Using policy files for access control is good for articulating relations among object to dynamically express the state properties of the network. But XACML models are complicated and impose heavy overheads, which after all, make them unsuitable for our target networks [24, 34].

1.3 Research Questions

In light of what was discussed earlier, an overarching goal of the proposed thesis is to first characterize the nature of dynamic IoT environments, and to establish this understanding for developing the right security features in target systems. I therefore aim at answering the following five Research Questions (RQ).

- **RQ1. What are the distinguishing characteristics of the Highly Dynamic IoT Networks?** Most research in the IoT domain has focused on outlining appropriate architectures for data connection and sharing in stationary networks such as smart homes and smart factory. There is little work done to articulate the characteristic properties of the highly dynamic IoT networks. The first goal of this proposal is to characterize such networks.

- **RQ2. How can we design and develop the security of highly dynamic IoT environments?** I have explained the initiative steps that undertook so far in Chapter 1 for the purpose to focus on the main work. The main question is how can we design and develop the security models for emerging IoT systems, in particular, for the highly dynamic IoT environments. In essence, the main focus of this PhD research is to find and discuss the right solutions for keeping information confidential from untrusted and unknown objects. The feasibility of this work is discussed in detail under a unique "Access Control" proposal in Chapter 2.

- **RQ3. How can we validate the correctness of security solutions?** One of the compelling research questions in the security of IoT systems has been to validate the security quality and correctness. I plan to find mechanisms to autonomously detect errors (perhaps by using formal models or anomaly detection techniques) in our proposed security paradigm.

- **RQ4. What are the attack models in the highly dynamic IoT environments?** RQ4 is paired with **RQ2**, that is, with the introduction of new security methods, their attack models need to also be investigated. I will answer this research question by using the general S.T.R.I.D.E. [10] taxonomy of six main security threats.

- **RQ5. How can we scale our security model to Software Defined IoT (SDIoT) networks?** In this research question, I aim to expand the security model proposed for RQ2 into "ultra-large-scale" IoT systems, in particular, Software Defined IoT (SDIoT) systems. Complex IoT environments benefit from the Software-defined Networking (SDN) to enhance the configuration and control of their nodes. SDN allows to decouple the control plane from the data plane. An SDN controller is a centralized software operation that controls the entire IoT network for performance agility and intelligence. However, employment of SDNs to

[10] *S.T.R.I.D.E.* is a security mnemonic which categorizes threats into six main groups: (1) **S**poofing, which is illegally accessing data and exploiting confidential information. (2) **T**ampering, that is malicious modification of data. (3) **R**epudiation is about performing an action without other parties having any way to prove. (4) **I**nformation disclosure means exposure of information to objects who are not supposed to have access to it. (5) **D**enial of service (DoS): deny service to valid users. (6) **E**levation of privilege: A damaging threat which concerns about an unprivileged user gaining privileged access to compromise or destroy the entire system.

simplify the IoT networks posses some risks. In this research question, I investigate how we can have an scalable security model for SDN-based IoT systems.

Chapter 2

Work Accomplished

In this Chapter, I describe the work accomplished with respect to answering the research questions listed in Chapter 1.3. *MiniCloud* service that engineered during the first year of my PhD. Specifically, I use MiniCloud for metric measurements and simulations conducted in answering the research questions in this chapter.

2.1 RQ1: What are the Characteristics of Future IoT Nodes

Emerging IoT systems contain wireless moving smart objects with the following characteristics:

1. ***Dynamism***: IoT nodes are highly *dynamic*. Many devices, either in one-on-one or one-to-many (in the form of platoons and swarms [35]) need to interact with each other to gain information in matters of a second [36]. Hence, to cope with the natural behavior of such high-speed systems, agile security preparations needed to protect the distributed data in storage and sharing.

2. ***Availability***: Many moving objects are mission-critical and required to stay at real-time availability operation with minimum service disruption. Security of highly dynamic IoT nodes must closely follow the principle of Economy of Mechanism that suggests security mechanisms should be as simple as possible, which in our scenario means should add minimal service disruption [37].

3. ***Resource-constrained Property***: Many IoT nodes are small miniaturized devices with limited memory and processing resources besides power-consumption and weight restrictions, that is, they are re-source-constrained. Internet Engineering Task Force (IETF) classifies con-

strained devices into three main categories as shown in Table 2.1 [38]. It is expected that the

Table 2.1: Classes of IoT Constrained Nodes.

Name	Data Size (RAM)	Code Size (Flash)
Class 0	$\ll$ 10KB	$\ll$ 100KB
Class 1	10KB	100KB
Class 2	50KB	250KB

security implementation should still fit into at least Class 1 IoT devices.

4. ***Expressiveness***: IoT nodes collect sensitive information spanning spatio-temporal sensory data, users' private and social data, passwords, connection properties, configuration settings, etc. The security mechanism for the protection of information and data sharing in dynamic IoT environments must carry sufficient *expressiveness* capabilities to cover all the possible attribute combinations of an evolving IoT network.

The conclusion here is if we want to have "*reliable and secure IoT systems of the future*", there needs to be proper designs and implementations that cover all four aforementioned characteristics. For this purpose, in my PhD proposal as the first major step, in response to RQ2, I develop a security mechanism that is an ***Acess Control*** model, and is the solution to data protection in IoT nodes. In particular, I *design a new Access Control framework for protection of data in highly dynamic IoT nodes*. Later in this Chapter, with analytical results, I show the plausibility of my method. Then I evaluate the performance of my access control model.

2.2 RQ2: Access Control to Protect Information in Highly Dynamic IoT Environments

Access Control is traditionally known to be the center of gravity in security engineering [39]. It outlines permission relationships between the stakeholders and the resources [39]. Stakeholders are users who want to access the resource or in the case of the IoT systems, could be the objects, web services, and applications too. Resources in a network are usually a significant collection of distributed files and data that are being shared between the nodes.

The two main sub-questions to answer here are:

1. There are several access control models implemented for IoT systems. Can we use them to protect information in highly dynamic IoT systems? The answer to this question is the result

of the literature review that is covered in Subsection 2.2.1.

2. How to design an access control mechanism that can protect data amongst dynamic IoT stakeholders? The answer to this questions with analytical results is covered under the Dynamic Policy-based Access Control (DynPolAC) framework that I describe through the rest of this Chapter.

2.2.1 Related Work

There are two main models describing access control for the IoT environments.

1. The first category describes the work that endeavors adapting traditional and static access control models to the IoT environments. Mainstream models are Role Base Access Control (RBAC), Capability Based Access Control (CBAC), and Trust Base Access Control (TBAC).

    - *Role Based Access Control* (RBAC) is a standard mechanism based on the user roles. It has been a native access control solution for resource protection mainly in operating systems [40, 41]. Although RBAC can be a dynamic model with minimum service disruption, it has been an integral component of many operating systems which cannot embed on the resource-constrained devices.
    - *Capability Based Access Control* (CBAC), on the other hand, is a stand-alone access control model that generates certificates for accessing resources. Certificates are either dependent to Attribute properties (ABAC) of a node or they can be Capability (CapBAC) dependent [42]. One challenge with CBAC however is that certificates need to be regenerated every time if the nature of the environment is dynamic, where frequent presence and absence of devices shall be experienced. Another unanswered question with CBAC is its unclear granularity level of attributes or capabilities.
    - *Trust Based Access Control* (TBAC) although known to be a newer access control paradigm, still poses similar CBAC challenges when it is about gaining trust and their frequency of generation in dynamic environments [41]. Additionally token generation for CBAC and TBAC is known to be a computationally intensive task [40, 41].

2. Existing deficiencies with traditional access control schemes prompted research to seek models that are at less complex, human readable and easily editable with potential capacities to become the standard [39]. This has lead to the birth of the second category, the *policy-based access control* scheme.

In policy-based access control, the protection of data relies on expressive attribute rules known as policy files. XACML has been the leading scheme for policy-based access control [28–33]. Although XACML is known to be a standard access control solution mainly for large databases, its parsing, processing, and maintenance is proved to be complex for resource-constrained IoT devices [24]. There has been research conducted using XACML for distributed systems that concluded XACML verbosity and complexity overpowers its expressiveness and flexibility [43, 44].

In addition, in some paradigms, XACML has required extra tri-components, namely, Policy Administration Point (PAP), Policy Decision Point (PDP), and Policy Enforcement Point (PEP) [45, 46]. Such solutions have aggravated the suitability of policy-based access control using XACML for dynamic IoT nodes — resource-constrained and availability properties have been two defeated factors in the tri-component systems.

Overall, what has been shown in the past work for policy-based access control, is its general unsuitability for mainstream resource-constrained devices. This has led research to take additional steps by proposing the employment of mediators to store and resolve policies. Authenticated tokens are then sent back to the target nodes giving permission to share corresponding information [29–31, 46]. But such solutions have increased the surface attacks and Denial of Service (DoS) possibilities. Additionally, server-based access control models does not seem to be responsive to agile IoT environments where the requests arrive from different objects and need to be serviced in counts of a second or less.

Table 2.2 outlines available access control schemes evaluating against the requirements of the high-speed IoT networks.

Table 2.2: Access control model fitness for high-speed dynamic IoT networks. A Checkmark (✓) describes a particular property is satisfied. An Xmark (✗) denotes the lack of the property.

Access Control Category	Can cope with highly dynamic service environments	Minimum service disruption needed	Suitable for resource-constrained devices	Expressiveness
Traditional models RBAC, CBAC, and TBAC	✗	✓	✗	✗
Policy-based models	✓	✗	✗	✓

In light of the above, I conclude that previous models cannot comprehensively fulfill the re-

quirements of the highly dynamic IoT environments. For this reason, to fill this gap and answer **RQ2**, I have proposed a solution that is a policy-based access control and integrates very well with the intrinsic characteristics of the highly dynamic IoT systems. I refer to such framework as Dynamic Policy-based Access Control (DynPolAC).

2.2.2 Dynamic Policy-based Access Control (DynPolAC)

I reviewed challenges involved with previous access control mechanisms. I devised a policy structure that fits well to the elements of emerging IoT networks. I have benefited from the expressiveness of the policy files and inspired by the previous research in defining privacy rules with markup languages. However, instead of XACML, I employ eXtensible Markup Language (XML).

XML is a light-weight, easy to understand, parse and maintain descriptive language. We constructed an intuitive policy file that inherits similar expressiveness as policies used in the related work. Additionally, I take into account, the spatio-temporal attributes of the environment where almost all IoT nodes gather their data. Accordingly, I present a rule block with the following primitive elements. A sample shown in Listing 2.1.

- **rule** — Indicates the beginning of a policy rule block. We have two types of rules. First is the "comparator", where min/max attributes articulate the permitted visibility range, and the second type is the "accessor" rule, which gives access to the sensitive data such as password, names, SSLKey, etc. (see Listing 2.1).
- **desc** — Plain text description of what the rule block is about. It is to help users construct right policy blocks.
- **type** — An attribute that represents the type of data being monitored. Samples are temperature, heading, password, altitude, Latitude and Longitude, etc.
- **vendor** — An attribute that presents the keeper of the information.

  For example, $< vendor > Uber < /vendor >$ means show me data that are in Uber devices only.
- **time** — ISO 8601-complaint temporal attribute devised to show information generated or updated since a particular date and time.
- **user** — A comma spliced RBAC attribute devised to give access to particular users.
- **group** — A comma spliced RBAC attribute devised to give access to particular groups.

```xml
<?xml version="1.0" encoding="utf-8"?>
<policyFile>
    <policy>
        <rule min="-10" max="110">comparator</rule>
        <desc>in degrees Celsius</desc>
        <attributes>
            <type>temperature</type>
            <vendor>Google</vendor>
            <time>2016-07-16T23:20:30</time>
            <user>User1, User2</user>
            <group>GroupXYZ</group>
        </attributes>
    </policy>
    <policy>
        <rule min="200" max="500">comparator</rule>
        <desc>Rakuten objects between 200ft and 500ft</desc>
        <attributes>
            <type>altitude</type>
            <vendor>Rakuten</vendor>
            <time>2017-08-22T09:20:30</time>
            <user>Zone1</user>
            <group>CompanyXYZ, HomeOwners</group>
        </attributes>
    </policy>
    <policy>
        <rule>access</rule>
        <desc>Access to ssh password</desc>
        <attributes>
            <type>password</type>
            <vendor>Latas</vendor>
            <time>2016-07-16T23:20:30</time>
            <user>Admin</user>
            <group>CorporateXYZ</group>
        </attributes>
    </policy>
    <policy>
        <rule>access</rule>
        <desc>Acess to car Fuel Level readings</desc>
        <attributes>
            <type>fuelLevel</type>
            <vendor>Uber</vendor>
            <time>2016-07-16T23:20:30</time>
            <user>Level1</user>
            <group>Team_XYZ</group>
        </attributes>
</policyFile>
```

Listing 2.1: Sample policy file.

2.2.3 Comparing Policy Rules Expressiveness

My rule block is the union of previous ABAC, RBAC, and policy-based access control models. With XML, lighter than usual policies with shorter file sizes can be constructed. These files are found at www.ece.ubc.ca/~mkarimib/minicloud/.

I have compared some of the previously introduced XACML files with the XML rule blocks that are constructed in this proposal. For example, in [28] and [31], Kim *et al* and Fysarakis et

al. specify attributes such as subject, resource, action, and condition. Their attributes are analogous to *type*, *vendor*, *access* and *rule* respectively in our construction.

In another comparison, Vaidya and Sherr in [32] build policies according to spacial properties for drones. They create primitives such as: time interval, day, capability, region, coordinate, and noise limit. Time interval and day attributes by Vaidya and Sherr are similar to the *time* element in our rule block. Capability, region, and noise limit are specific to drones and can be presented as the *type* in our generic access control model to reflect the environmental properties of a particular IoT system. Lastly, the 'coordinate' attribute is similar to the *type* primitive, which can take a variety of expressions (coordinates being one of them) in our policy.

Similarly, from a recent work, Drozdowicz et al. [46] presented primitives such as target, match and description, which can be matched in our case to *type*, *rule*, and *desc*.

With respect to comparing RBAC attributes in related work, Das et al. in [33] described their rules by attributes such as user, subject, and object. Their primitives are comparable with *user*, *group*, and *type* respectively.

2.3 Performance Analyses

In this section, I report the performance results that illustrate why *DynPolAC* is a good solution for protection of information in the target environment under discussion.

2.3.1 Parsing Time Measurements

So far, I have shown the construction of a policy file in XML that is comparable with the past models in terms of ease-of-use and expressiveness. However, our main claim for using XML against previous constructs such as XACML is to show that processing XML is significantly faster hence makes it the right candidate to employ as the access control language in highly interactive nodes.

I inspect the parsing times between DynPolAC, which is XML-based, versus previous policy models, which are XACML-based. For our results, to be comparable, we programatically produce policy files up to 2000 intuitive rules in both XML and XACML languages. Another remark in the comparison is resorting to the *Expat*[1] parser [47], which is the native XML parser. To keep the tool homogeneity, I use the Expat parser for both XML and XACML parsing[2].

Figure 2.1 shows our parsing comparison results. The results are captured at the steady-state

[1]Since XACML is a derivative of XML, it can be processed by the same parser.

[2]To my knowledge, there is no standalone parser for XACML.

#of rules in the policy file	Pi 3 XML (ms)	Pi Zero XML (ms)	BBB XML (ms)	Pi 3 XACML(ms)	Pi zero XACML (ms)	BBB XACML(ms)
1	1.05	*1.34*	**2.39**	1.2	*1.9*	**5.17**
2	0.96	*2.15*	**4.34**	1.89	*3.17*	**6.15**
25	3.22	*6.03*	**9.42**	21.98	*27.54*	**34.72**
50	6.02	*9.93*	**13.34**	43.46	*52.98*	**62.58**
100	12.22	*19.52*	**24.21**	88.92	*96.15*	**122.89**
200	24.06	*36.18*	**42.3**	120.94	*174.55*	**246.4**
500	56.82	*72.31*	**88.57**	240.55	*432.65*	**614.05**
1000	84.61	*136.89*	**167.26**	458.43	*838.55*	**1222.65**
2000	176.01	*290.55*	**349.4**	910.45	*1692.56*	**2543.28**

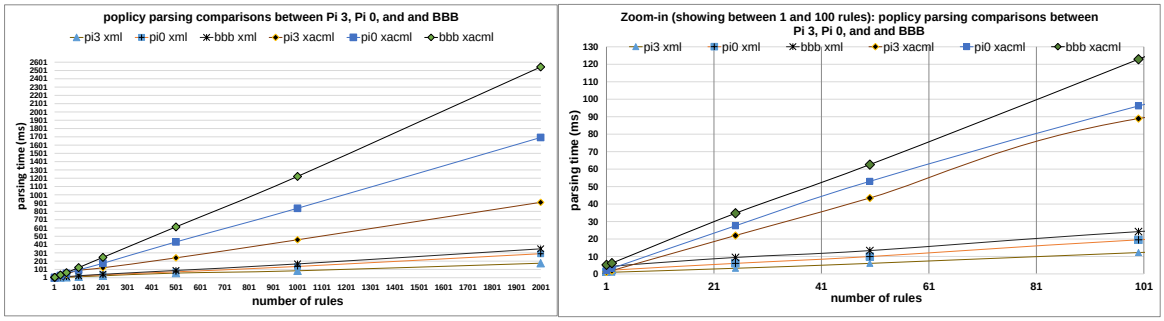


Figure 2.1: Parsing XML vs XACML files comparison on different platforms (Mean over 1000 time parsing for each point).

mean after 1000 trials. To show uniformity in parsing results, I ran our parsing engine with several platforms, namely Raspberry Pi 3 (1.2GHz, 64-bit quad-core), Raspberry Pi 0 (1GHz, 32-bit single-core), and Beagle Bone Black (720MHz, 32-bit single-core).

As can be followed from the Figure 2.1, XACML parsing takes longer time for all sizes — the maximum parsing time is with the file that contains 2000 rules, which is 2.5 seconds. These results show that for highly interactive environments, where periodic query of dynamic data occur at one second intervals, policy enforcement could surely be a bottleneck. In addition, from Figure 2.1 is the linear latency relation with the size of the files. Lastly, these results show that DynPolAC on average outperforms their XACML-based counterparts with ***five*** to ***seven*** times performance improvements. These results are particularly important when dealing with high-traffic IoT environments where an agile protection practice is needed with the sharing of sensitive information between dynamic objects.

2.3.2 System Design

In previous sections I discussed the parsing component of DynPolAC and showed its policy parsing time achieves significant speed advantages over other previous methods. However, to show the suitability importance of DynPolAC, in macro scale, we ran system-level simulations in the *MiniCloud* to analytically evaluate different access control mechanisms. We also performed the sensi-

tivity analyses and report which parameters are the most influential in the latency of response time between the IoT nodes.

Model Assumptions

In Section 1.2, I suggested that there were two main schemes envisioned for emerging IoT networks. Here, I model the first scheme only, that is, the existence of central hubs known as UAS Traffic Management Systems (UTM) for drones or Road Side Units (RSU) for autonomous cars. If we can show that DynPolAC is a suitable mechanism that protects confidentiality of information in highly dynamic IoT hubs, we would have shown that it is also a suitable methodology for the second scheme which is the object-to-object communication. In Figure 2.2, we see a dynamic service model framework, where objects contact a center to query information from it. The model is comprised of the queue theory, besides DynPolAC, and the MiniCloud. I assume the following in modeling:

- Our simulation results are based on the Queue Model type 'M/M/1':

  - The IoT service node is a single queue, single-processor system.

  - The arrival rate, λ, follows the Independent and Identically Distributed (IID) random variables based on the Poisson distribution stream.

  - There are no buffer or population size limitations and the service discipline is First-Come-First-Serve (FCFS). See Figure 2.2.

  - The queue is modeled as the birth-death process. That is, the number of jobs in the queue when each object lines up causes the state of the queue to change by +1 (birth) and when departed from the queue, causes the state of the queue to change by -1 (death). The birth-death process helps to keep track of the time that each object stays latent in the queue waiting for service.

- For calculating the total response time (Γ), I assume the wireless network signal latency is at the fixed *50 ms* delay time. This is assumed in reference to the analytical model employed M/M/1/K queue for 802.11 wireless networks [48].

- I employ the method of *Regeneration* for computing the overall mean response time [49]. In Section 2.3.3, I describe how I use the method of *Regeneration* to calculate the steady-state mean response time.

- The simulations were run on a real embedded system that is the Beagle Bone Black (BBB) with CPU speed 720MHz and 512MB RAM [19].

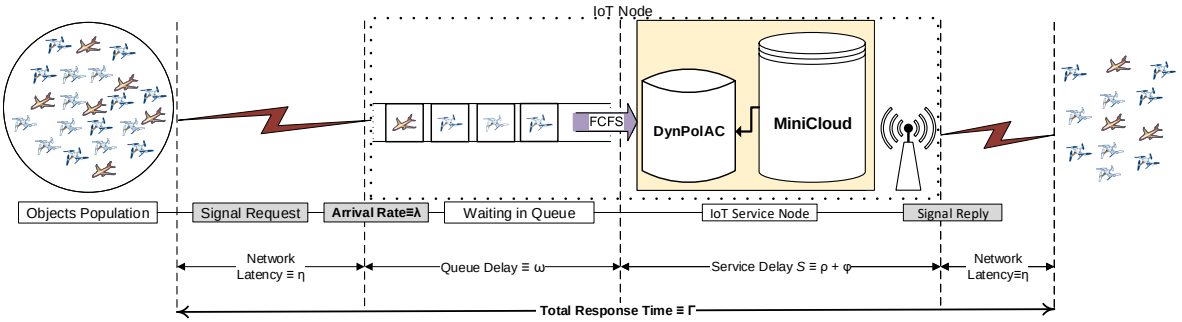


Figure 2.2: Schematic view of M/M/1 queuing service for an IoT node. Notations described in Table 2.3.

Formulation

We evaluate the stability condition ($\lambda < \mu$) of *DynPolAC* in highly dynamic environments by measuring the overall mean response time delay. Table 2.3 describes our notations. The response time (Γ) defines the time between a node request a query from another node and the time it receives the queried data. We formulate the response time duration by Equation 2.2. Our formulation is based on the assumptions made in Subsection 2.3.2, and the key variables shown in Figure 2.2, used by the queueing analysis.

$$\mu = \frac{1}{\Gamma} \tag{2.1}$$

$$\Gamma_{(\lambda, f_{cpu}, \ell_{policy}, \ell_{Query}, \eta)} = \eta + N + \eta = 50ms + N + 50ms = 100ms + N \tag{2.2}$$

Equation 2.3 formulates the node delay time. We characterize the node delay (N) by the time a query awaits in the queue (ω) and the node's mean service time (S).

$$N_{(\lambda,f_{cpu},\ell_{(policy},\ell_{Query)})} = \omega_{\lambda,f_{cpu}} + S \tag{2.3}$$

The mean service time (S) characterizes as the time it takes to parse and process the particular policy related to the query and the query time itself.

$$S_{(f_{cpu},\ell_{policy},\ell_{Query})} = \rho_{(}\ell_{Policy}, f_{cpu}) + \varphi_{(}\ell_{Query}, f_{cpu}) \tag{2.4}$$

Table 2.3: Queue theory notations.

Mnemonic	Description	Unit value
Γ	Mean response time	ms
μ	Mean response rate	1/s
λ	Mean arrival Rate	1/s
N	Mean node delay	ms
η	Network latency	ms
S	Mean service time	ms
ω	Wait time in the queue	ms
ρ	Policy processing time	ms
φ	Query time	ms
ℓ_{Query}	Query size	Bytes
ℓ_{Policy}	Policy size	no. of rules
f_{cpu}	Node clock speed	720MHz

Since I am simulating in a real embedded system, in the following I briefly describe the system integration.

Platform Integration

The simulations and latency measurements occur in the MiniCloud. I instantiated the *DynPolAC* engine inside the MiniCloud (Figure 1.2).

The original code size of the *MiniCloud* had been 66KB (Figure 2.4). We added *DynPolAC* as an integral component of the *MiniCloud*, which monitors the queried data and enforces access control. With the addition of the *DynPolAC* policy engine, our IoT node image size is 71KB (Figure 2.4). This means that with only 7.5% image overhead, creation of a policy engine in Class 1 (refer to Table 2.1) resource-constrained nodes is possible.

Another featured implementation of *DynPolAC* is its dynamic capability to update the rules at

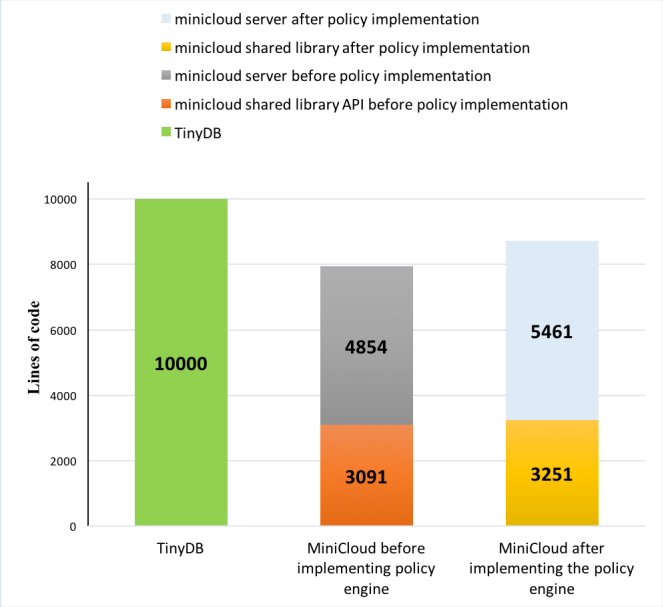


Figure 2.3: Code comparison between the MiniCloud and TinyDB.

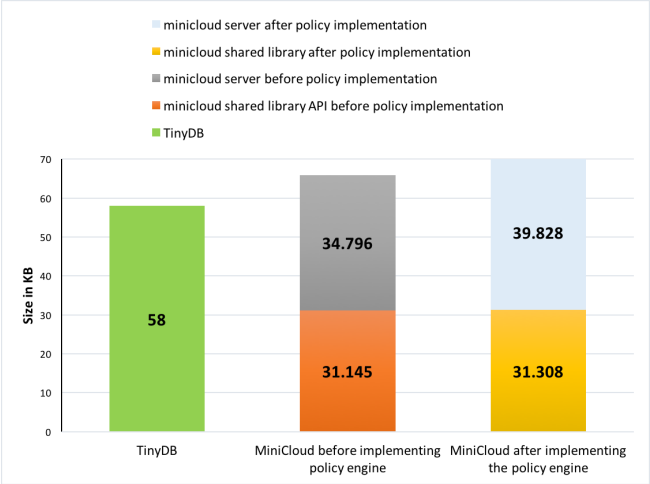


Figure 2.4: Size of image comparison between the MiniCloud and TinyDB in KB.

run-time via a housekeeping process. When a new request arrives, *DynPolAC* first parses the file and registers the rules that associate with the query request, and then, based on the rules, the server can respond to the query request. A major overhead during the policy check is the parsing time for creating new rules or updating the currently existing ones, discussed in Section 2.3.1.

2.3.3 Simulation Results

To evaluate the performance of the system, two sets of simulations were run.

In the fist set, I measure the overall mean at steady state condition amongst a machine that does not have any access control, a one that carries *DynPolAC*, and a one that is using XACML policy for its access control. All simulations were run on BBB platform as described in Subsection 1.1.2. The performance of the system were evaluated based on the stochastic analysis of three parameters, namely, the arrival rate (λ), the query length (ℓ_{Query}), and the policy size (ℓ_{Policy}). For generating random arrival rates (λ), we used the *Inverse Transformation Method* to get Poisson variates with *Mean 4* [49]. The arrival rate is per second and the number of objects arriving per second queue up in the system awaiting service (Figure 2.2). The other two parameters, follow a uniform distribution between 200 Byte and 5 KB for the query length (ℓ_{Query}), and between 1 and 2000 rules for the policy size (ℓ_{Policy}).

Stopping criteria. For recording the steady-state response times, their mean needs to be calculated. Since the waiting times are correlated[3], I cannot use arithmetic mean. Instead, I use the method of *Regeneration* [49]. In this method, I first record the mean at every epoch[4]: $\bar{\Gamma}_i = \frac{1}{n_i}\sum_{j=1}^{n_i}\Gamma_{ij}$, where Γ_{ij} is the single object response time, n_i is the number of objects that arrive at a time, and Γ_i is the mean response time at every epoch. Next, I incrementally compute the epoch sums: $Y_i = \sum_{j=1}^{n_i}\Gamma_{ij}$, and the overall mean accordingly: $\bar{\bar{x}} = \frac{\sum_{i=1}^{m} Y_i}{\sum_{i=1}^{m} n_i}$. I stop the simulation when the overall mean response time is at the steady-state with two decimal digit of a millisecond precision, and achieves the 95% two-sided confidence interval [49].

Figure 2.5 demonstrates our simulation results for the overall mean response time. The steady-state measurement is achieved after 3000 epochs (≈50 minutes of simulation time for each point).

In the second simulation set, the sensitivity analysis for the three influencing parameters was performed: the arrival rate (λ), the query length (ℓ_{Query}), and the policy size (ℓ_{Policy}). Figure 2.6

[3] In a queueing simulation the n^{th} job should be processed first before turning to $n+1^{th}$ job.

[4] Every epoch occurs every second which based on the arrival rate a number of objects queue up in the system awaiting the service.

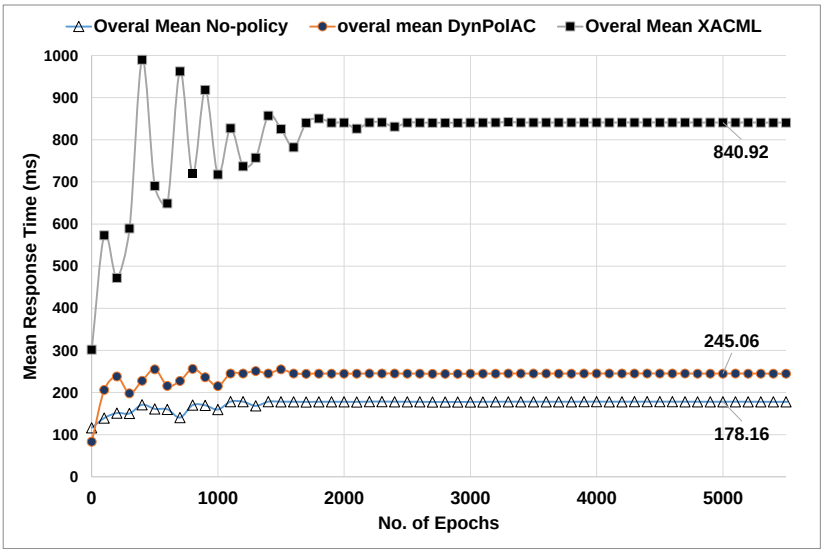


Figure 2.5: Simulation results for gaining the steady-state gain of the mean response time, Γ. Data are shown for when no policy engine enforced and the time that XML and XACML are active. Platform is Beagle Bone Black running at 720MHz.

shows the results. In Figure 2.6-a, the arrival rate sweep between 1 object per second and 10 objects per second is performed. Figure 2.6-b shows the sensitivity analysis for query length. The query in Bytes are swept between 200 and 5K. In our last evaluation, Figure 2.6-c, the number of policy rules between 1 and 2000 are scanned.

2.3.4 Discussion

RQ2 is about finding a suitable security mechanism for highly dynamic IoT systems. One way for dynamic environments, since there are frequent requests occur in counts of one second, is to show the stability condition holds with the addition of security measures.

The implementation of the *DynPolAC* engine meets the *stability condition* in dynamic IoT nodes. If the arrival rate (λ) is bigger than the service rate (μ), the system falls behind in query responses and said to be unstable [49]. The stability condition requires arrival rates smaller than the service rates ($\lambda < \mu$). For a "no policy" IoT node, I observe the mean service rate in our platform is: $\mu = \frac{1}{178.16ms} = \frac{1000}{178.16} = 5.61$/s (see Figure 2.5).

When *DynPolAC* is enforced, $\mu = \frac{1}{248.16ms} = 4.1$/s, which satisfies the stability condition ($\lambda = 4$, $\mu = 4.1 \Rightarrow \lambda < \mu$). In contrast, μ=1.19 for XACML-based policy systems. This is smaller

than the arrival rate ($\lambda = 4$, $\mu = 1.19 \Rightarrow \lambda \not< \mu$) and therefore, a node with an XACML-based policy access control may experience system instability.

Additionally, by the Little's law (Mean object requests in the system = mean arrival rate $\lambda \times$ mean response time Γ), it can be concluded that no jobs are left in *DynPolAC*; whereas the mean object request in XACML-based policy systems = $4 \times 1/1.19 = 3.36$.

From sensitivity analyses, among the three factors, namely policy, query, and arrival rate, policy is the most influencing factor due to sharper slope changes between the different sizes of policy rules (Figure 2.6-c). Arrival rate is second and query size is third. *DynPolAC* mitigates the heavy impact on the response time, making it an attractive answer to RQ2.

2.4 Summary

We know that NextGen IoT environments are highly interactive and require adaptive (if not new) security measure to protect their data. *Dynamic policy-based access control* (DynPolAC) is a framework that accommodates expressivenesses similar to previous data protection schemes, and is runtime editable. DynPolAC also imposes minimum service disruptions as oppose to previous models, which makes it the fit stand-alone framework for Class-1 resource-constrained devices with only 7.8% of code image overhead (Figure 2.4).

We evaluated DynPolAC by measuring performance metrics at both the micro and the macro scales. At the micro level, DynPolAC outperforms previous XACML-based methods up to 7.28x in parsing speed(Figure 2.1).

At the macro scale, we used queueing theory to study the feasibility of the *DynPolAC* in highly dynamic environments. *DynPolAC* satisfied the stability condition (in contrast to previous XACML-based models) with 245.06ms overall mean response time. Therefore DynPolAC is a viable solution and the answer to RQ2 for data protection in highly dynamic IoT environments.

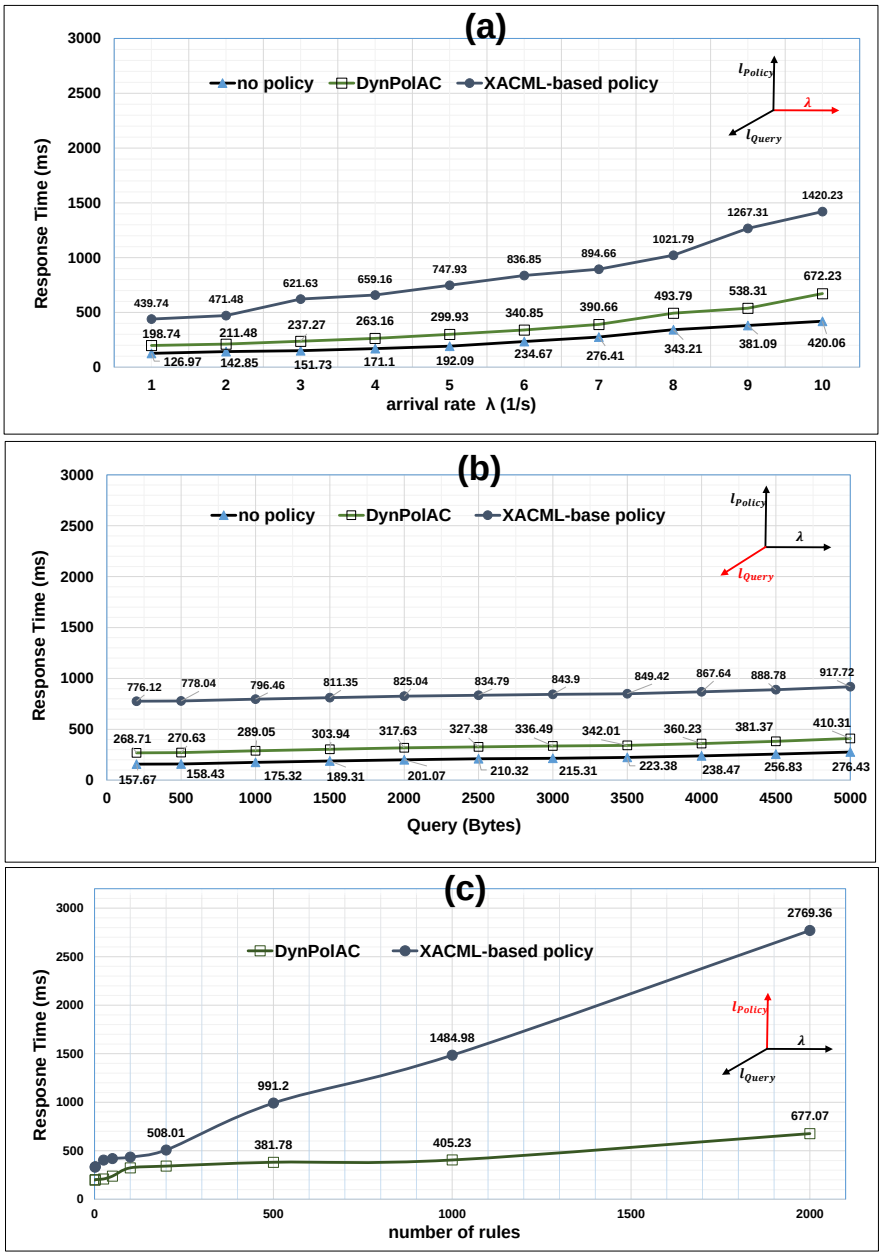


Figure 2.6: Sensitivity Analyses. Each data point recorded after steady-state mean response time arrived (∼3000 epochs) (a) Arrival rate λ sweep (simulating between 1 and 10 objects per second).(b) Data Query Sweep in Bytes (between 200B and 5KB). (C) Policy sweep in number of rules (between 1 and 2000 rules).

Chapter 3

Work Planned

In this Chapter, I describe the plan for future work according to the remaining research questions, RQ3, RQ4, and RQ5.

3.1 Security Validation — Answer to RQ3

One of the main concerns in the "security" of IoT systems is their quality of service assurance. It means how to ascertain the security models are correct at all scenarios. *DynPolAC* needs to have an assurance unit to check the policy rules are correct and unique at real-time. We want to ensure that the set of primitives in the rule blocks are conflict-free and there is no supersede or override occurring.

Conflicts in policies occur in two ways; (1) When fragmentary constraints are found in the policies or (2) When the constraints overlap with each other. For detecting fragmentary policies, we need to check if there are no shared common state variables. For detecting overlapping policies, we need to check the conditional part and the attributes part of policies do not superimpose.

I propose to add a checker to *DynPolAC* to parse the policy files and convert them to trees, then compare them with each other. If a discrepancy found between two or more rules defining the same behavior, *DynPolAC* should raise a flag and stop the operation from applying conflicted rules to the system.

Figure 3.1 shows an example of generated trees from a file that contains two rules, both of them are giving access to the temperature values of Google devices. However one is allowing access to values between -10 and 10, while the other rule is overriding the previous rule with values between 20 and 110. The timestamps are not matching either. The top rule is permitting to query the

temperature values that are generated since “2016-07-16”, while the bottom rule is asking for the same set but generated after “2017-01-05”. Therefore, in such cases, it is not clear which one of the two rules should be applied to the system. We stop enforcing these policies to our system and flag their parent nodes for inspection.

Additionally, *DynPolAC* requires to apply policy validation “frequently” because of the state of emerging IoT systems that policy updates must occur dynamically. This could affect the overall response time, Γ (Table 2.3), quadratically, dependent to the size of policy files. For example, to check if there is a conflict among n policies, a naive algorithm takes each policy and compare it against the other ones. This outputs the result in maximum of $O(\frac{n(n-1)}{2})$ comparisons. In the second part of RQ3, I look for an optimization method to reduce the latency in validation time employing the computer science search and comparison algorithms.

In the last part of RQ3, I propose three methods to resolve conflicts. One way can be assigning priorities to the policy files. For example, is a global policy has priority to a node policy? Second way to resolve issues is taking the intersection of the conflicts. A third approach could be to repair conflicts. Finally, I will investigate these three approaches and will describe the advantages and disadvantages of each method.

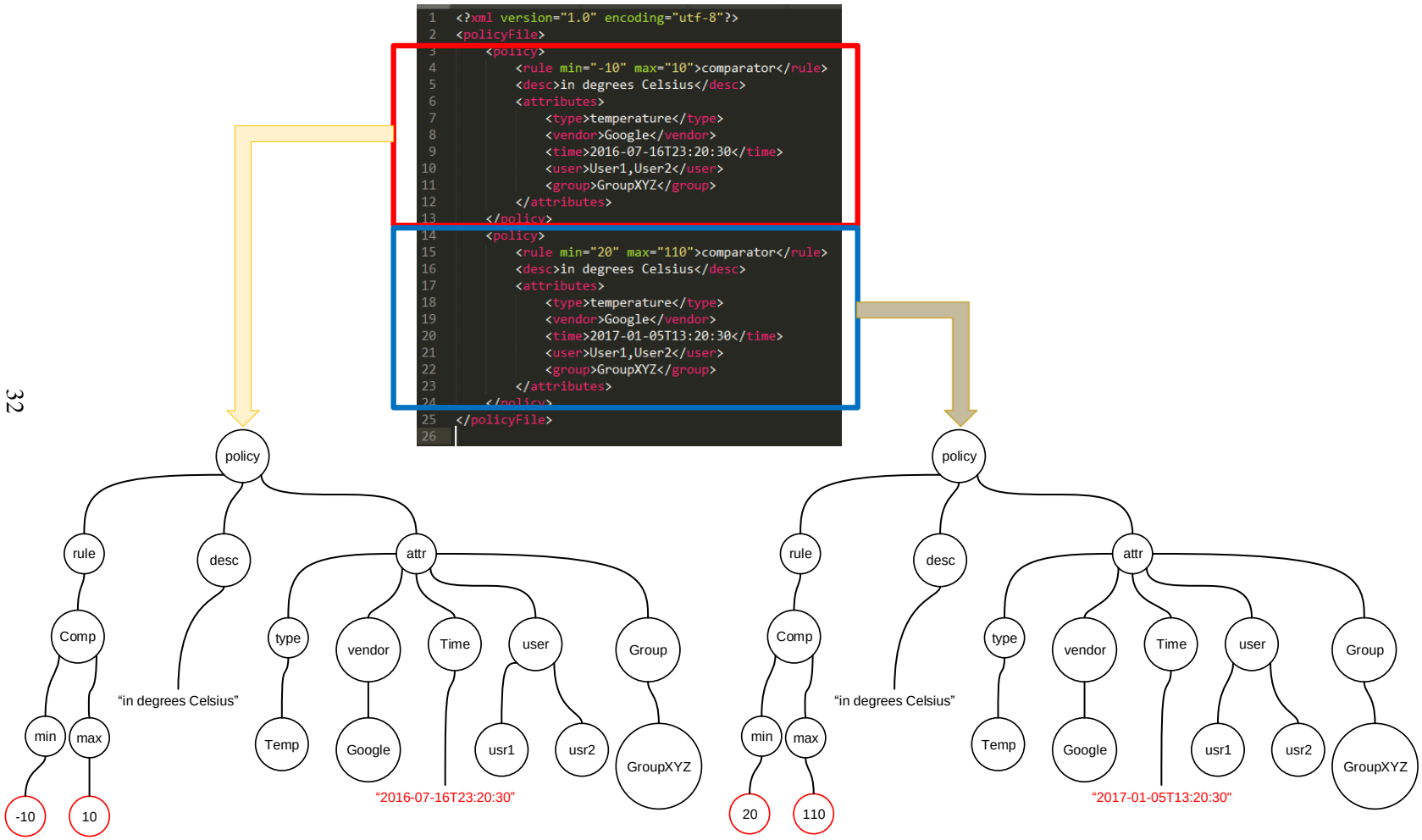


Figure 3.1: Example of checking two rule blocks in a policy file. Red nodes are showing the conflicts.

3.2 Attack Models — Answer to RQ4

Introducing *DynPolAC* may widen the attack surface risks. In this research question, I propose to discover vulnerabilities in *DynPolAC* subject to a set of predefined threat models known as the *S.T.R.I.D.E.* classification [50]. I plan to study three main threats that are significant to *DynPolAC* security framework. The assumption is that an attacker has found its way to gain the *root* access in at least one of the IoT nodes in the network.

1. **Spoofing**: Policy files are stored in the file system of the IoT nodes. If an attacker gains the *root* privilege, it can easily modify the content of the policy files. The current state of *DynPolAC* lacks protection for policy files and so the integrity of policy files can easily be compromised. In this research question, I plan to devise a technique to stop the risk of spoofing by *DynPolAC*. One possible solution is to encrypt the policy files with the Advanced Encryption Standard (AES) or Rivest Shamir Adleman (RSA) algorithms [39]. With encryption, an attacker cannot modify the policies unless he or she obtains the key. How to obtain the key is another compelling question that I will answer as the future work.

2. **Denial of Service (DoS)**: DoS is a very common adversarial attack that targets the service disruption. In our environment, DoS means block information exchange between the nodes. *DynPolAC* is very susceptible for this attack. Figure 3.2 shows the simplest DoS vulnerability that could happen in *DynPolAC*. An attacker can share a void policy file to the system. Therefore, any node that has been infected by the void policy will experience denial of service — essentially by receiving no information upon querying data.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <policyFile>
3
4  </policyFile>
5
```

Figure 3.2: A trivial DoS attack in *DynPolAC*.

   To mitigate this attack, one possible solution is to prohibit void policies. As a future work, I will put conditions for the *DynPolAC* parser to check whether the file has any content or not. Empty files must be discarded from the *DynPolAC*.

3. **Information disclosure**: This attack is about deliberately constructing the policy files such that sensitive information such as the system configuration settings (e.g., brake settings in

autonomous cars) or password data (e.g., ssh access to nearby drones) are shared between untrusted nodes. For this type of attack we assume that the adversary has write access to modify the policy files. We also assume that the adversary has the knowledge advantage of the system settings and knows about the type of information exchanged between the nodes. The adversary can subtly construct the policy files to deceive *DynPolAC* about sharing or blockage of queried data.

One solution to mitigate this vulnerability is to have a formal equivalence checker to compare the specification against the policy files. Specifications can be constructed as a form of invariants and compared against their counterparts in the policy file. Full coverage of design and implementation will be covered in my future work.

3.3 Security of Software Defined IoT (SDIoT) Networks — Answer to RQ5

One of the potential long-term goals of *DynPolAC* is to have it suitable for "ultra-large scale", highly dynamic IoT networks. In particular, in this research question, we will look at Software Defined IoT (SDIoT) networks.

3.3.1 Software Defined IoT (SDIoT) Networks

Future IoT networks will comprise billions of objects that need to communicate with each other in the network. It will be impractical to manually configure and control such a large number of nodes in a system. As a result, Software Defined Networking (SDN) has been used as a scalable paradigm to address the IoT network issues [51–53]. The main advantages of using SDNs are (1) The nodes in the entire network can be controlled by a centralized software operation therefore there is no need to configure them individually, and (2) The control plane is decoupled from the data plane and thus communication permissions between the nodes are handled by the central SDN application.

Figure 3.3 shows different layers of the Software Defined-based IoT networks, where at the lowest level there are IoT devices. Their data are being shared with each other via a network (switch) infrastructure. Permissions of sharing information is handled at the third layer, the SDN controller, by using the policy mechanisms. A permission mechanism be employed at the controller layer to mediate the data flow between the IoT nodes and the data going upstream to the application and services layer.

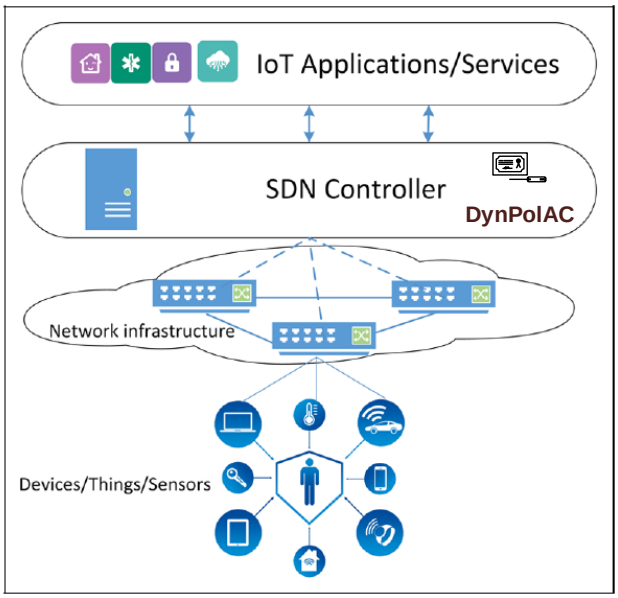


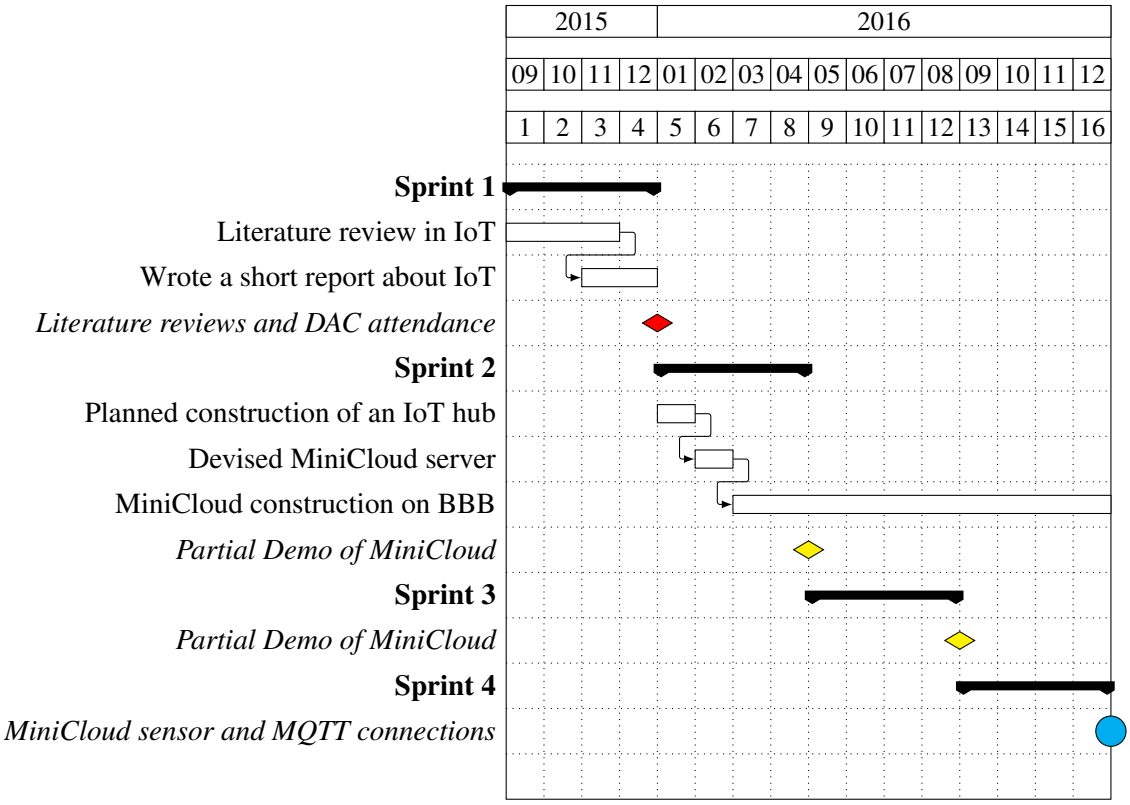
Figure 3.3: "SDIoT environment" [53]. DynPolAC is shown as an integral component of the controller layer.

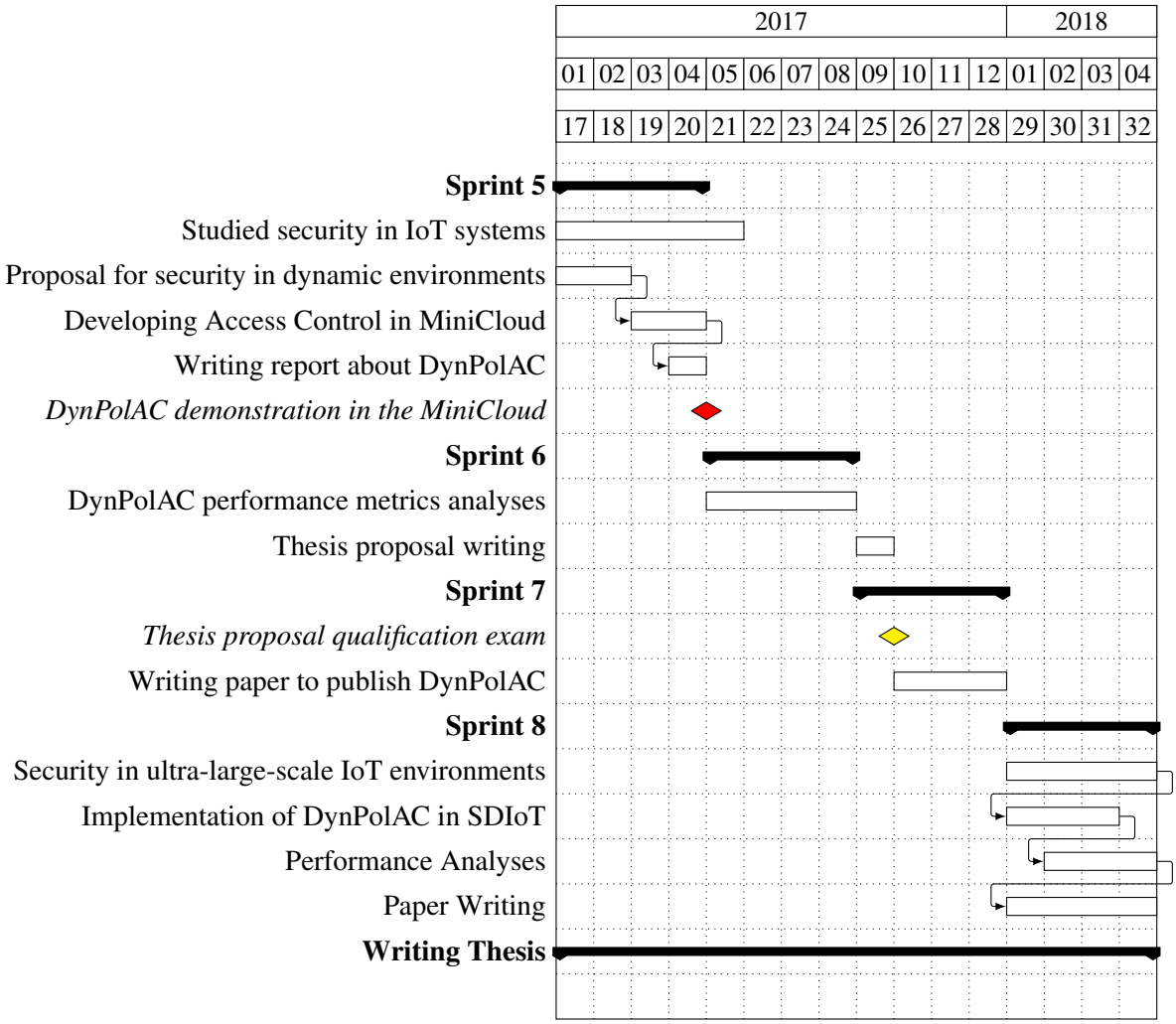
I plan to answer RQ5 by deploying *DynPolAC* in SDIoT control layer. I will investigate how *DynPolAC* helps in managing the permissions in the controller layer while giving service to both the network layer and the application layer.

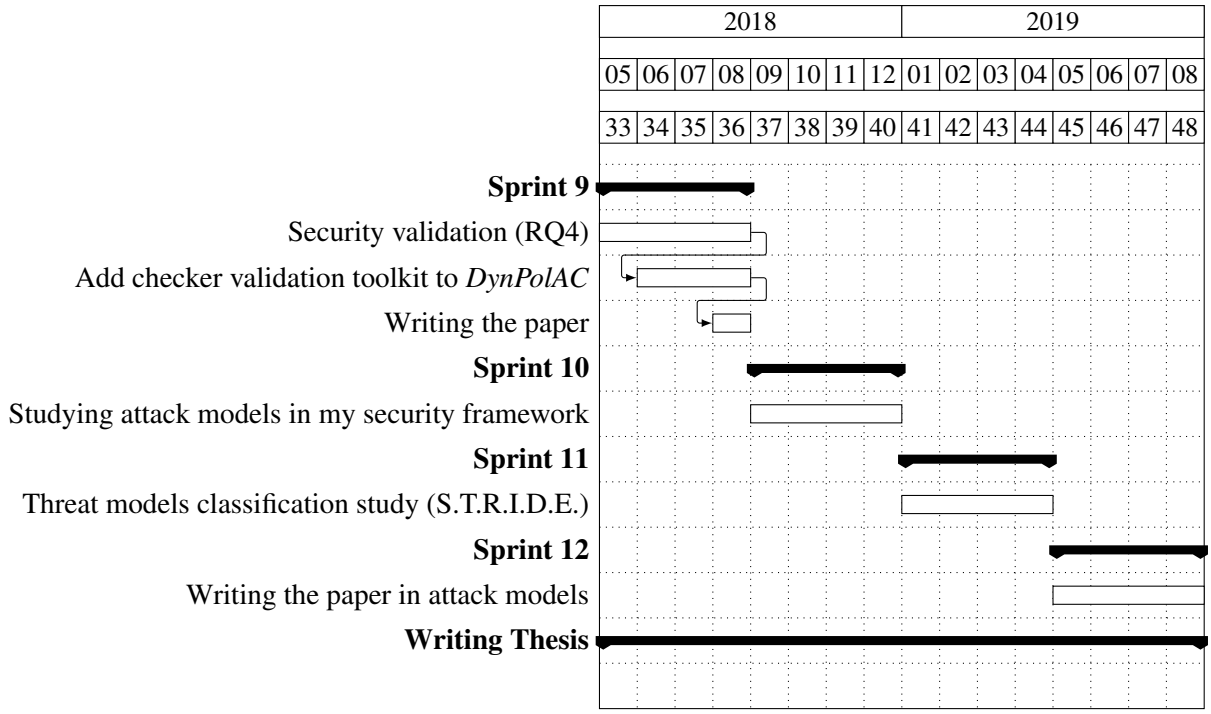
3.4 Milestones

Researching the security of highly dynamic IoT systems requires a well-scheduled plan. To this end, in 48 months with quadrimestral Sprints[1], I have three Sprints a year and 12 in total. The Gantt chart below elaborates on my timed plan to finish PhD describing milestones at the end of each Sprint:

[1]In Agile software development, Sprint means duration of a project that is expected to finish and deliver. At the end of each Sprint there is a demo of finished work and plan for moving to the next Sprint based on the last completed Sprint.


2015
2016
09 10 11 12 01 02 03 04 05 06 07 08 09 10 11 12
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Sprint 1
Literature review in IoT
Wrote a short report about IoT
Literature reviews and DAC attendance
Sprint 2
Planned construction of an IoT hub
Devised MiniCloud server
MiniCloud construction on BBB
Partial Demo of MiniCloud
Sprint 3
Partial Demo of MiniCloud
Sprint 4
MiniCloud sensor and MQTT connections


2017
2018
01 02 03 04 05 06 07 08 09 10 11 12 01 02 03 04
17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
Sprint 5
Studied security in IoT systems
Proposal for security in dynamic environments
Developing Access Control in MiniCloud
Writing report about DynPolAC
DynPolAC demonstration in the MiniCloud
Sprint 6
DynPolAC performance metrics analyses
Thesis proposal writing
Sprint 7
Thesis proposal qualification exam
Writing paper to publish DynPolAC
Sprint 8
Security in ultra-large-scale IoT environments
Implementation of DynPolAC in SDIoT
Performance Analyses
Paper Writing
Writing Thesis


2018
2019
05 06 07 08 09 10 11 12 01 02 03 04 05 06 07 08
33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
Sprint 9
Security validation (RQ4)
Add checker validation toolkit to DynPolAC
Writing the paper
Sprint 10
Studying attack models in my security framework
Sprint 11
Threat models classification study (S.T.R.I.D.E.)
Sprint 12
Writing the paper in attack models
Writing Thesis

Bibliography

[1] Networks on demand: The promise of software-defined networking. http://hightechforum.org/networks-on-demand-the-promise-of-software-defined-networking/. Accessed: 2017-09-21. → pages

[2] The 4th industrial revolution. http://www.i2cat.net/images/industry_4.0.png. Accessed: 2015-08-21. → pages 1

[3] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Gener. Comput. Syst.*, 29(7):1645–1660, September 2013. → pages 1, 2, 11

[4] Mark Fell. Roadmap for the emerging "internet of things". *Wite-paper*, 2014. → pages 1, 2

[5] Rolf H. Weber. Internet of things: Privacy issues revisited. *Computer Law and Security Review*, 0267-3649, 2015. → pages 1

[6] Feng Chen, Pan Deng, Jiafu Wan, Daqiang Zhang, Athanasios V Vasilakos, and Xiaohui Rong. Data mining for the internet of things: literature review and challenges. *International Journal of Distributed Sensor Networks*, 501:431047, 2015. → pages 2

[7] H. Sahli, C. Bouanaka, and A. Taki Eddine Dib. Towards a formal model for cloud computing elasticity. In *WETICE Conference (WETICE), 2014 IEEE 23rd International*, pages 359–364, June 2014. → pages 2

[8] Ying R. Ma. Research this is an example of security model and cloud computing strategy. *Applied Mechanics and Materials*, 556-562:6196–6198, 2014. → pages

[9] Yuan Zhang and Lei Huang. The research of cloud computing service model. *Applied Mechanics and Materials*, 556-562:6262–6265, 2014. → pages

[10] Andre Duarte and Miguel M. d. Silva. Cloud maturity model. pages 606–613. IEEE, 2013. → pages 2

[11] Modbus communication protocol. http://www.modbus.org. Accessed: 2015-08-23. → pages 3

[12] Mqtt communication protocol. http://mqtt.org. Accessed: 2015-08-23. → pages 3

[13] Dropcam and nest website. https://nest.com/camera/meet-nest-cam/. Accessed: 2015-08-27. → pages 3

[14] Samuel R Madden, Michael J Franklin, Joseph M Hellerstein, and Wei Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Transactions on database systems (TODS)*, 30(1):122–173, 2005. → pages 4, 7

[15] Vlad Trifa, Samuel Wieland, Dominique Guinard, and Thomas Michael Bohnert. Design and implementation of a gateway for web-based interaction and management of embedded devices. *Submitted to DCOSS*, pages 1–14, 2009. → pages 4

[16] Qnx real-time operating system by blackberry. http://www.qnx.com/. Accessed: 2015-09-06. → pages 4, 9

[17] Vlad Trifa, Dominique Guinard, Vlatko Davidovski, Andreas Kamilaris, and Ivan Delchev. Web messaging for open and scalable distributed sensing applications. In *International Conference on Web Engineering*, pages 129–143. Springer, 2010. → pages 5

[18] Walter Colitti, Kris Steenhaut, and Niccolò De Caro. Integrating wireless sensor networks with the web. *Extending the Internet to Low power and Lossy Networks (IP+ SN 2011)*, 2011. → pages 5

[19] Beagle bone black website. http://elinux.org/Beagleboard:BeagleBoneBlack. Accessed: March 1, 2017. → pages 9, 23

[20] The Federal Aviation Administration (FAA). Faa releases 2016 to 2036 aerospace forecast, March 2016. → pages 10

[21] forbes.com. 10 million self-driving cars will hit the road by 2020 – here's how to profit, March 2017. → pages 10

[22] National Aeronautics and Space Administration. Unmanned aircraft system (uas) traffic management (utm), August 2017. → pages 10

[23] J.A. Stankovic. Research directions for the internet of things. *Internet of Things Journal, IEEE*, 1(1):3–9, Feb 2014. → pages 11

[24] Aafaf Ouaddah, Hajar Mousannif, Anas Abou Elkalam, and Abdellah Ait Ouahman. Access control in the internet of things: Big challenges and new opportunities. *Computer Networks*, 112:237 – 262, 2017. → pages 11, 17

[25] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497 – 1516, 2012. → pages

[26] Yunpeng Zhang and Xuqing Wu. Access control in internet of things: A survey. *CoRR*, abs/1610.01065, 2016. → pages

[27] S. Sicari, A. Rizzardi, L.A. Grieco, and A. Coen-Porisini. Security, privacy and trust in internet of things: The road ahead. *Computer Networks*, 76:146 – 164, 2015. → pages 11

[28] J. E. Kim, G. Boulos, J. Yackovich, T. Barth, C. Beckel, and D. Mosse. Seamless integration of heterogeneous devices and access control in smart homes. In *2012 Eighth International Conference on Intelligent Environments*, pages 206–213, June 2012. → pages 11, 17, 19

[29] Jorge Bernal Bernabe, Jose Luis Hernandez Ramos, and Antonio F Skarmeta Gomez. Taciot: multidimensional trust-aware access control system for the internet of things. *Soft Computing*, 20(5):1763–1779, 2016. → pages 17

[30] L. Seitz, G. Selander, and C. Gehrmann. Authorization framework for the internet-of-things. In *2013 IEEE 14th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, pages 1–6, June 2013. → pages

[31] K. Fysarakis, I. Papaefstathiou, C. Manifavas, K. Rantos, and O. Sultatos. Policy-based access control for dpws-enabled ubiquitous devices. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pages 1–8, Sept 2014. → pages 17, 19

[32] Tavish Vaidya and Micah Sherr. Mind your (r,\varphi) s: Location-based privacy controls for consumer drones. In *Cambridge International Workshop on Security Protocols*, pages 80–90. Springer, 2015. → pages 20

[33] Prajit Kumar Das, Sandeep Narayanan, Nitin Kumar Sharma, Anupam Joshi, Karuna Joshi, and Tim Finin. Context-sensitive policy based security in internet of things. In *Smart Computing (SMARTCOMP), 2016 IEEE International Conference on*, pages 1–6. IEEE, 2016. → pages 11, 17, 20

[34] Jia Jindou, Qiu Xiaofeng, and Cheng Cheng. Access control method for web of things based on role and sns. In *Computer and Information Technology (CIT), 2012 IEEE 12th International Conference on*, pages 316–321. IEEE, 2012. → pages 11

[35] Damian Roca, Daniel Nemirovsky, Mario Nemirovsky, Rodolfo Milito, and Mateo Valero. Emergent behaviors in the internet of things: The ultimate ultra-large-scale system. *IEEE Micro*, 36(6):36–44, 2016. → pages 14

[36] Florian Schaub and Pascal Knierim. Drone-based privacy interfaces: Opportunities and challenges. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*. USENIX Association, 2016. → pages 14

[37] Matthew A Bishop. Introduction to computer security. 2005. → pages 14

[38] Carsten Bormann, Mehmet Ersue, and A Keranen. Terminology for constrained-node networks. Technical report, 2014. → pages 15

[39] Ross J Anderson. *Security engineering: a guide to building dependable distributed systems*. John Wiley & Sons, 2010. → pages 15, 16, 33

[40] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497–1516, 2012. → pages 16

[41] Yunpeng Zhang and Xuqing Wu. Access control in internet of things: A survey. *arXiv preprint arXiv:1610.01065*, 2016. → pages 16

[42] José L Hernández-Ramos, Antonio J Jara, Leandro Marin, and Antonio F Skarmeta. Distributed capability-based access control for the internet of things. *Journal of Internet Services and Information Security (JISIS)*, 3(3/4):1–16, 2013. → pages 16

[43] Sonia Jahid, Imranul Hoque, Hamed Okhravi, and Carl A Gunter. Enhancing database access control with xacml policy. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 130–133, 2009. → pages 17

[44] Markus Lorch, Seth Proctor, Rebekah Lepro, Dennis Kafura, and Sumit Shah. First experiences using xacml for access control in distributed systems. In *Proceedings of the 2003 ACM workshop on XML security*, pages 25–37. ACM, 2003. → pages 17

[45] Sun Kaiwen and Yin Lihua. *Attribute-Role-Based Hybrid Access Control in the Internet of Things*, pages 333–343. Springer International Publishing, Cham, 2014. → pages 17

[46] Michał Drozdowicz, Maria Ganzha, and Marcin Paprzycki. Semantically enriched data access policies in ehealth. *Journal of medical systems*, 40(11):238, 2016. → pages 17, 20

[47] Expat parser for xml language files. https://www.xml.com/. Accessed: Sept 05, 2017. → pages 20

[48] Cheikh Sarr and Isabelle Guérin-Lassous. *Estimating average end-to-end delays in IEEE 802.11 multihop wireless networks*. PhD thesis, INRIA, 2007. → pages 22

[49] Raj Jain. The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling. 1991. → pages 22, 26, 27

[50] Min Chen, Yongfeng Qian, Shiwen Mao, Wan Tang, and Ximin Yang. Software-defined mobile networks security. *Mobile Networks and Applications*, 21(5):729–743, 2016. → pages 33

[51] Zhijing Qin, Grit Denker, Carlo Giannelli, Paolo Bellavista, and Nalini Venkatasubramanian. A software defined networking architecture for the internet-of-things. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–9. IEEE, 2014. → pages 34

[52] Yaser Jararweh, Mahmoud Al-Ayyoub, Elhadj Benkhelifa, Mladen Vouk, Andy Rindos, et al. Sdiot: a software defined based internet of things framework. *Journal of Ambient Intelligence and Humanized Computing*, 6(4):453–461, 2015. → pages

[53] Tri-Hai Nguyen and Myungsik Yoo. Analysis of attacks on device manager in software-defined internet of things. *International Journal of Distributed Sensor Networks*, 13(8):1550147717728681, 2017. → pages 34, 35