DynPolAC: Dynamic Policy-based Access Control for IoT Systems

Mehdi Karimibiuki, Ekta Aggarwal, Karthik Pattabiraman, and André Ivanov
Electrical and Computer Engineering Department
University of British Columbia (UBC), Vancouver, Canada
{mkarimib, ektaa, karthikp, ivanov}@ece.ubc.ca

***Abstract*—In the near future, Internet-of-Things (IoT) systems will be comprised of autonomous, highly interactive and moving objects that require frequent handshakes to exchange information in time intervals of seconds. Examples of such systems are drones and self-driving cars. In these scenarios, data integrity, confidentiality, and privacy protection are of critical importance. Further, updates need to be processed quickly and with low overheads due to the systems' resource-constrained nature.**

This paper proposes *Dynamic Policy-based Access Control* (DynPolAC) as a model for protecting information in such systems. We construct a new access control policy language that satisfies the properties of highly dynamic IoT environments. Our access control engine is comprised of a rule parser and a checker to process policies and update them at run-time with minimum service disruption. *DynPolAC* achieves more than 7x performance improvements when compared to previously proposed methods for authorization on resource-constrained IoT platforms, and achieves more than 3x faster response times overall.

I. Introduction

Information security is an important requirement of the Internet of Things (IoT) systems where autonomous objects (nodes) are vulnerable to malicious attacks such as sniffing, snooping and impersonation [1]. Some future IoT systems will be increasingly dynamic, with mobility exacerbating the security challenges. For example, Unmanned Aircraft Systems (UAS) and Self Driving Cars (SDC) may move around and broadcast their sensitive information such as regional sensory data and mission planning to unattended zones and untrusted third parties [1]. Information leakage in mobile IoT are usually due to data sharing with unauthorized nodes or unintended parties from a network supposed to be closed or secure [2]. It is thus critical to develop models for data confidentiality, information disclosure (i.e., what should an entity be allowed to share), and authorization (i.e., what should an entity be allowed to access) for such emerging IoT systems.

One approach for ensuring the information security in IoT objects is to adopt *access control* techniques traditionally known as Role-Based-Access-Control (RBAC) and Attribute-Based-Access-Control (ABAC) [3], [4]. These techniques, while suitable for static environments, are not designed to work well in dynamic IoT systems. An alternate solution has been the use of the standard policy files in eXtensible Access Control Markup Language (XACML) [5], [6]. Unfortunately, *XACML*-based models are complicated and impose heavy performance overheads, which make them generally unsuitable for dynamic IoT environments [7], [8]. Furthermore, for highly interactive networks, a real-time responsive data permission control for communication between entities is required to mediate the data access [3].

In this paper, we introduce a data protection framework called "Dynamic Policy-based Access Control" (***DynPolAC***). The main idea behind *DynPolAC* is to satisfy the important IoT constraints *C1-C4* described below, thus making it uniquely suited for dynamic IoT systems.

- **C1**. *High levels of Dynamism*. Devices in emerging mobile IoT systems must interact either in one-on-one or one-to-many mode to gain information on the order of seconds [9], [10]. To cope with the natural behavior of such high-speed systems, agile data protection strategies are needed.
- **C2**. *Service Time*. Most IoT objects are timing- or mission-critical [2], and required to process communication requests with minimum service disruption. Thus, to serve the periodic requests adequately, a security enforcement mechanism must process authorization at low-latency with quick run-time update advancements.
- **C3**. *Resource Limitations*. Many IoT nodes, in addition to the power-consumption limits and the weight restrictions, are devices with limited memory and processing resources [11]. A viable security mechanism must be readily implementable in *resource-constrained* nodes.
- **C4**. *Protocol Expressiveness*. Many IoT systems are comprised of nodes collecting sensitive information spanning sensory data, action planning, control information, servoing, system monitoring, users' private and social data, passwords, connection properties, etc. A robust information security platform must maintain control of access to data with regard to a suite of different parameters, including access credentials (identity, whether human or machine), as well as temporal and other environmental conditions. We refer to this as *expressiveness*.

The key innovation of this paper is the proposal of a new policy construction that meets the characteristic properties *C1-C4*. We show that by careful syntax selection in *DynPolAC*, high performance results at system-level can be achieved, particularly between mobile IoT nodes (such as UAS or SDC). *To the best of our knowledge, we are the first to propose an access control framework for dynamic IoT systems that respects the constraints of these systems*. More specifically,

we make the following contributions:

- **Novelty**: We introduce *DynPolAC* as a new *policy-based* access control for IoT systems through which smaller size policy files can be created (Section IV). *DynPolAC* is a light-weight access control for data sharing particularly in dynamic and interactive mobile IoT nodes.
- **Comparison**: To compare with previous policy-based models dominated by XACML, we constructed *DynPolAC* in the XML language (Section IV-D). We posit that the expressiveness of our model is similar to that of the previous models. *DynPolAC* formulates comparator-based, permission-based, time- and attribute-based semantics, thereby allowing a wide variety of access control policies for many data types.
- **Simulation**: We demonstrate results by *discrete event simulation* that our model is a viable solution for dynamic IoT systems (Section VI). We find that (1) with *DynPolAC*, the information security check can be performed in the order of milliseconds rather than seconds, (2) *DynPolAC* achieves up to 3.4x faster response times than previous policy-based model (Section VI-B), and incurs a memory overhead of only 7.5% (Section VI-C).
- **Performance analysis**: We quantitatively measure and compare the processing time of *DynPolAC* on three IoT platforms, namely, Beagle Bone Black (BBB), Raspberry Pi 3 (Pi3), and Raspberry Pi Zero (Pi0) (Section VI-A). We show that *DynPolAC* achieves higher speed-ups on slower platforms. For example, BBB is our slowest platform, where, on average, we observed speedups of 7.28x. In the faster platforms, Pi0 and Pi3, the speedups achieved are 6.4x and 5.6x respectively, thus demonstrating *DynPolAC*'s suitability for resource-constrained nodes.

II. Background

The rapid emergence of the IoT is bringing an unprecedented expansion to global communications between objects. There now exists relatively well-established stationary IoT networks such as smart grids, smart buildings, and smart factories with at least 50 billion IoT devices predicted to be in use by 2020 [12]. However, the next generation of IoT systems will be characterized by *mobile wireless autonomous* vehicles. Two of the prominent examples of autonomous unmanned vehicles include drones (as part of UAS) and self-driving cars (SDCs). The Federal Aviation Administration (FAA) is forecasting the number of UAS will grow from 2.7 million in 2016 to 7 million by 2020 [13]. Similarly, it is estimated that there will be 10 million SDCs on the roads by 2020 with forecasts that one in four cars will be autonomous by 2030 [14].

The growing trends show that there is an emerging need for adaptable, real-time, low-latency communication frameworks with matching considerations for data privacy protection and information security. Towards this effort, two models have been proposed to govern such *dynamic* IoT networks.

The first model is an emerging plan to have trusted hubs manage the mobile IoT nodes. Hubs are central for data collection and traffic management. In this model, objects communicate with the hub to get their information. 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, contingency and congestion management, terrain avoidance, route and re-route planning, etc. (example in Section IV-C1) [15]. We will study this scheme in Section V-B. Similarly, as part of the intelligent transportation systems, Road Side Units (RSUs) provide cars with safety information, as well as the traffic and tolling management [16] (example in Section IV-C2).

The second model, which does not exist today to the best of our knowledge, is a more distributed and decentralized network with no hub holding the greater decision-making data and mechanisms. The autonomous objects build trust, and communicate with each other directly.

Our focus of study in this paper is the first model since it is an emerging technology yet with little prior work in its regime [1]. To provide further context, for example, in the case of drones, if we assume that 1% of the land surface area is occupied by drones typically near population centers such as cities, with 7-million flying by 2020, there could be between *one* and *ten* drones interacting with an UTM system in the WiFi range every second (Section V-B).

III. Related Work

We view earlier approaches to information security by access control for IoT networks as falling under two categories. The first category is one where traditional access control models have been adapted to the IoT environment. Mainstream models in this category include Role Based Access Control (RBAC), Capability Based Access Control (CBAC), and Trust Based Access Control (TBAC).

- *RBAC* is a standard mechanism based on the user roles [17]. It has been widely used, in particular, in operating systems [4]. RBAC assigns distinct roles to users and gives permission to access resources. Role assignment can be defined in an Access Control List and can be modified and re-applied with minimal service disruption. However, live-streaming IoT networks, require context control for assignments other than roles, such as time, location, state of the environment, etc. [3]. Hence, RBAC possesses limited expressiveness in that certain attributes of the network and its objects cannot be readily described.
- *CBAC* is a model that can include context assignments and generate certificates accordingly for accessing resources. Certificates are either *Attribute*-based (ABAC) properties or *Capability*-based (Cap-BAC) [18]. One challenge with CBAC, however, is that certificates need to be re-generated at every instance of the appearance or disappearance of an object. Thus, for IoT networks where mobile objects have frequent arrivals and exits, certificate generation can produce a bottleneck due to the significant processing time and service disruption that every single event incurs.

- *TBAC* is a newer access control paradigm. It is focused on the objects gaining *trust* with each other through information sharing. This is accomplished either by a certification authority [19], or by algorithmic methods using trust calculations between devices [20]. When a specified level of trust is established between two devices, there is no further need to certify the mutual relationship even if the devices cease their interaction prior to re-engaging again at a subsequent time. However, if the environment is a highly dynamic one, this approach tends to be computationally very intensive [4] and may be difficult or impossible to implement on resource-constrained devices.

The second category are *policy-based access control* schemes. In this category, protection of data relies on expressive predicates that comprise the policy files. To express policies, a structural XML-based syntax called *XACML* has been devised. *XACML* is the leading language for policy-based access control [21] , and is widely used for static environments such as large databases [22] and home automation services [23]. However, *XACML* processing and maintenance is complex, making it unfit for resource-constrained IoT devices [3], especially if policy sets are relatively large [8]. Two approaches have been proposed to mitigate the overheads of *XACML* processing.

1) A first approach toward speeding up the processing of *XACML* is through the use of servers to process policies and generate authenticated tokens. Tokens are then sent to the nodes giving permission to share corresponding information [6], [24], [25]. A drawback is that such solutions tend to be vulnerable when tokens are hijacked and used as attacks to yield denial-of-service. So, policy-based access control models that need back-end servers could impose high security risks with large service disruptions.
2) A second approach is to reduce *XACML* complexity by simplifying the language. Seitz et al. [7] provided a subset of *XACML* policy list defined in a compact JavaScript Object Notation (JSON), which reduced the number of assertions by one order of magnitude. However, the authors did not articulate the precise tradeoffs of the modified policy language, nor did they evaluate their solution in the context of a dynamic IoT environment. Additionally, with the use of JSON templates, it is not clear how much syntax readability is sacrificed to gain agility.

Table I presents a qualitative evaluation matrix that compares the fitness of the existing access control approaches for their application to dynamic IoT environments.

In summary, to the best of our knowledge, no prior access control technique is able to satisfy the requirements of highly dynamic IoT systems. We propose a new language for access control that overcomes the shortcomings of prior existing approaches without sacrificing their richness or expressiveness.

TABLE I: Fitness for high-speed dynamic IoT networks. A Checkmark (✓) denotes property satifiability while a Xmark (✗) denotes a shortcoming.

Access Control Category	(C1) Can cope with highly dynamic service environments	(C2) Minimum service disruption needed	(C3) Suitable for resource-constrained devices	(C4) Expressive attributes
Role-based (RBAC)	✗	✓	✓	✗
Capability-based (CBAC)	✗	✗	✗	✓
Trust-based (TBAC)	✗	✓	✗	✓
Policy-based models based on XACML language	✓	✗	✗	✓
Our method *DynPolAC*	✓	✓	✓	✓

IV. DynPolAC

We devise a new policy-based language, i.e., *DynPolAC*, to govern the security operation of distributed data in dynamic IoT networks. In particular, we come up with a new language model that satisfies the four main characteristics of IoT systems as described in Section I.

A. Meeting the constraints

In the following we describe how our language satisfies the constraints *C1 to C4*.

1) ***C1***: *High levels of Dynamism*. In mobile IoT environments there is a need to reconfigure the network settings including access control frequently. Our language construction must be capable of rearranging the rules in a modular form to modify access rules at any granular hierarchy. At the top layer, there must be a policy set (**PS**), which is a collection of policy blocks with the option to add or drop them (Table II). Policies themselves must be reconfigurable. There needs to be a Rule List (**RL**) to add a collection of rules or remove them arbitrarily. Rule Lists are a combination of rules that can be augmented to form a new set or partially reduced to remove unattended rules. The hierarchy shall make *DynPolAC* dynamic and in a portable form, which helps to commission new regulations at run-time without impacting current enforcements.
2) ***C2***: *Service Time*. Our language structure, while capable of reinforcing policies in real-time, is required to keep the system running by minimizing the update time. To meet this goal, we carefully selected only *six* primitives that are capable of describing the main IoT system properties, namely, environmental variables, time, and configuration settings. In Section VI-A we will show that it is indeed our deliberate anthology that helps maximize the service availability with optimized processing time.
3) ***C3***: *Resource-constrained Property*. Policy-based access control models are usually unsuitable for resource-constrained devices due to the language grammar size and the processing overheads. For example, page 25 of the XACML standard document gives a simple instance of a single rule that contains 39 lines with nested opening and closing element tags [26]. For our target environment where there could be thousands of rules applied to a single node, the complexity and lines of a policy rule

would matter and adversely affects the memory load and the processing performance (for results see Section VI-A). Rather than making complex constructions which usually requires nested data structures [26], we use fixed size rule blocks in plain language to describe the permission control of data. In Section IV-B, by using only *six* primitives, we show *DynPolAC* takes significantly less space to fit in resource-constrained devices' memory and less time to process in slower platforms (for results see Section VI).

4) ***C4***: *Expressiveness*. Many mobile IoT nodes collect spatio-temporal, geographical sensory data as well as the configuration and system maintenance information. They also collect sensitive data such as the mission planning, user names, addresses, origins and destinations. Therefore, their communication control requires two prime decision types; either check a range to give access for analog data, or, use a permit-deny binary-like access control. In the following section we show that *DynPolAC* constructs Rule Type (**RT**) primitives to cover for the necessary expressions in the IoT networks. Additionally, live-stream data in such systems are time-dependent attributes, for which we include Time (**T**) in our policy construction. Besides, we include the Attribute Types (**AT**) to filter queries based on their variable type.

B. Formulation

We choose a set of *six* primitives that are sufficient to express communication control between mobile IoT devices such as UAS and SDC. Depending on the platform requirements, the policy set can be described in any plain or markup languages such as TXT, INI, JSON, XML, or even Efficient XML Interchange (EXI). In the following, we present our language construction followed by some examples of its use. We also compare it with other approaches. The heuristic choices that we made in our language are to satisfy {C1-C4} that are the characteristics of the IoT networks postulated in Section IV-A.

We have a hierarchical construction order in our language as outlined in Table II. The top unit is called a Policy Set (***PS***), which is the union of Policies (***P***). For example, there could be *global* policies that form a ***PS*** for interaction between the nodes universally, or, there could be a set of *local* policies that form the regional ***PS***.

TABLE II: Hierarchical construction order of the *DynPolAC Language*.

Hierarchy	Construction
Policy Set (***PS***)	$PS_{Global} \mid PS_{Local} \mid P_1 \cup P_2 \cup ... \cup P_n$
Policy (***P***)	$P_1, P_2, ..., P_n \mid P_{node} \mid P_{utm} \mid$ [RL]
Rule List (***RL***)	R | [R] | RL $\circ$ [R]
Rule (***R***)	(RT, AT, V, T, U, G)
Rule Type (***RT***)	Accessor | Comparator

Policies (***P***) can be multiple object files or all can be aggregated into one file. For example, there can be policy files that set rules for a variety of drones in the network, one for each; or, there can be a monolithic file that sets access rules to a centric UTM gateway. Policies (***P***) in turn are comprised of Rule List (***RL***), which is the aggregation of Rules (***R***). There could be a ***RL*** that concatenates new Rules (***R***) as denoted by operator '$\circ$' in Table II row 3.

Rules (***R***) are the basic component of our *DynPolAC* language. Rules (***R***) are independent atomic decision units with a fixed set of elements as follows.

- **Rule Type (RT)**: Indicates the beginning of a new rule and defines its type. We have two types of rules; first, the "*comparator*" rule, where the visibility of analog data is given by a min/max range; second, the "*accessor*" rule, which gives a binary-like *permit* or *deny* access, for example, to sensitive data such as password, passenger information, and SSLKey.
- **Attribute Type (*AT*)**: Represents the type of data being monitored. Examples are temperature, heading, password, altitude, latitude, and longitude.

In addition, Rules (***R***) keep track of the origin of data by an ID or a vendor name. Data must also be tracked in time and by the users and groups wanting to access the data. As a result, in *DynPolAC* we add fields to represent Vendor (***V***), Time (***T***), User (***U***), and Group (***G***) as follows:

- **Vendor (V)**: A primitive that restricts access based on the origin of information. For example, if the vendor name is *Uber*, it means to monitor data in *Uber* devices only.
- **Time (T)**: A temporal element added to confine information access based on a particular date and time range. For example, we can specify to give access to data between Monday and Tuesday from 11 AM to 3 PM. If the keyword 'ALL' is used, it implies giving access to data at all times.
- **User (U)**: A wild card primitive providing a *comma spliced RBAC attribute* to give access to particular users. A user can be an application running in a node asking to have access to data, or, it can be a particular node name asking for access. If used 'ALL' as a wild card, the implication is to give access to all users.
- **Group (G)**: A wild card primitive providing a *comma spliced RBAC attribute* to give access to particular group of users. For example, a group 'hospital' can be assigned to let all hospitals have access to patients' smart wearables [27], [28].

For the sake of comparison with previous policy-based access control models and without the loss of generality, in this paper, we construct *DynPolAC* in XML languge[1]. To better illustrate *DynPolAC* construction we provide two examples as follow.

C. Examples

[1]Additionally, XML is interoperable — regardless of the underlying technology, there is no maintenance overhead for migration between different IoT platforms.

1) Drones near no-flying zones: Consider future UTMs that would develop dynamic surveillance and geo-fencing systems to inhibit drones from entering no-flying zones. There would be control hubs around airports, military bases, government buildings, power plants, etc., to enforce the no-flying zone by commanding autonomous flying objects to re-route. However, the UTMs would need to have the permission to check the drone information and to issue a command to change their direction. Otherwise, the access would be either unnecessary, or diagnosed as an adversarial communication which will need to be blocked.

```
<policyFile>
    <policy>
        <rule min=''33.941362''
              max=''33.941777''>comparator</rule>
        <attributes>
            <type>Latitude</type>
            <vendor>Rakuten</vendor>
            <time>ALL</time>
            <user>Zone1</user>
            <group>Airport</group>
        </attributes>
    </policy>
    <policy>
        <rule min=''−118.413695''
              max=''−118.414226''>comparator</rule>
        <attributes>
            <type>Longitude</type>
            <vendor>Rakuten</vendor>
            <time>ALL</time>
            <user>Zone1</user>
            <group>Airport</group>
        </attributes>
    </policy>
    <policy>
        <rule>accessor</rule>
        <attributes>
            <type>Heading</type>
            <vendor>Rakuten</vendor>
            <time>ALL</time>
            <user>ALL</user>
            <group>ALL</group>
        </attributes>
    </policy>
</policyFile>
```

Listing 1: $P_{Rakuten}$ in an UTM of the no-flying region.

Listing 1 is an example of a policy file ($P_{Rakuten}$) giving access to specific data in *Rakuten*[2] drones. This policy is assumed to be part of a ***PS*** that has been registered in the trusted UTM. We also assume that a *Rakuten* drone before sharing information, first, asks for its policy, $P_{Rakuten}$, from the UTM. Then, based on the policy, the drone will share information.

In Listing 1, there are three rules described. The first two are the *comparator* type rules. The first rule gives the UTM server permission to have the *latitude* data of Rakuten drones if the range is between "33.941362" and "33.941777" degrees. The second rule describes permission to have *longitude* information if it is in the range between "-118.413695" and "-118.414226" degrees. These are the coordinates of an airport. The last rule in Listing 1 is an *accessor* permission that lets the UTM change the *Rakuten* drone direction by accessing its *Heading* configuration parameter. All these three rules apply at all *times* to all *Rakuten* drones. However, the first two rules are made specifically for UTM nodes in Zone1 (User) of airport (Group). If a geo-fencing application is running in a UTM machine under User *Zone1* and group *airport*, then it can obtain access to *Rakuten* drones' data.

If we wanted to construct this example in XACML, our first two rules would be comparable to the predicates similar to page 34 of the expressions between line numbers 1200 and 1303 of the XACML document ($1303-1200 = 103\times2 =$ **206** lines) [26]. For the user and group, we must have used semantics as expressed on page 31 of the XACML document [26] between lines 1043 and 1119 ($1119 - 1043 = 76 \times 2 =$ **152**). Our last rule is comparable to Rule 4 of the XACML document [26], page 39, between lines 1537 and 1626 ($1626-1537 =$ **89**). Overall, in this example we see that with *DynPolAC* we can reduce over 400 (**206**+**152**+**89** = **447**) lines of XACML syntax to only 32 lines (447 - 32 = 415).

2) Tolling Cars in Highways: Consider the case of toll collection in highways. Electronic tolling requires either pre-registration of cars to debit drivers' account according to the type of car and the time of day, or, if a car is not registered, image processing of the plate is applied to identify the car type and send the bill at higher rate to the driver. In addition to the services overhead, in the past, there have been security and privacy breaches on the central databases of the tolling systems, which resulted in spoofing or hijacking the record of specific vehicles' debit accounts [29].

```
<policy>
    <rule>accessor</rule>
    <attributes>
        <type>Debit</type>
        <vendor>Tesla123XYZ</vendor>
        <time min=''06:30:00'' max=''18:30:00''>
            Mon−Tue−Wed−Thu−Fri
        </time>
        <user>RSU_Exit34</user>
        <group>HW−1</group>
    </attributes>
</policy>
```

Listing 2: A policy example for Toll collection.

DynPolAC can be used in an autonomous environment without the need for pre-authorization or surcharges. Listing 2[3] displays an example of a rule that can be passed to a tolling machine at the passing time of the car. With this rule no pre-registration is needed and an *accessor* block that gives permission to access the passenger's debit card information will suffice (via for example the driver's phone connected to the car's infotainment system). In the Listing 2, the tolling application (user) that is running in the RSU of Exit 34 and in group highway number 1, is given authorization to charge driver's debit account. When the Tesla car (vendor) with ID 123XYZ at specific times of the week (Monday through Friday, 6:30 AM to 18:30 PM) passes the tolling area, based on

[2]Rakuten is a Japanese company that manufactures consumer drones.

[3]This data structure format is a sample for demonstration.

the *accessor* rule, the tolling machine is permitted to query and charge the debit card that is registered with the car's database. Suspicious activities such as charges outside of the allotted hours will be blocked and reported for further inspection. Also the account will be charged if and only if the particular car passed the check point. If we wanted to construct the same policy in XACML, we would have had to use the request context as an example provided on page 29, lines 920 to 994 in the XACML document ($994 - 920 = 74$) [26]. *DynPolAC* thus reduces the language intricacy six-fold (i.e., from 84 lines to 12).

D. Comparison of Expressiveness

To cope with the characteristics of IoT environments and unlike nested decision sets used in the XACML standard, we only used a two-valued Rule Type (*RT*), i.e., *comparator* and *accessor* (Table II). The two type of rules that we chose are sufficient to satisfy the requirement of access control in our target platforms by permitting, denying or restricting access.

Moreover, *DynPolAC* excludes the *combining algorithms*[4]. Combining algorithms are complex and trigger heavy constructions, which defeats the initial purpose of our language, i.e., agility. Instead, our decision units (rules) are independent of each other and there is no prioritization between the rules; a rule can only be applied to one data property, at a given time. Nevertheless, we assert that *DynPolAC* has adequate elements, which makes it a comparable model to previous work. For example, Kim et al. [5] and Fysarakis et al. [6], specify characteristics such as *subject*, *resource*, *action*, and *condition*. Our model is comparable: ***AT*** in our language is analogous to their *subject*; Vendor (***V***) is analogous to their *resource* name, ***RT*** is similar to their *action*; and the time (***T***) as well as ***min/max*** conditionals will match with their *condition* parameter. In a later work, Vaidya and Sherr [30] build policies according to spatial properties of drones. They created primitives such as *time interval*, *capability*, *region*, *coordinate*, and *noise limit*. *Time interval* is similar to our primitive, *Time* (***T***). Their attributes are specific to drones, but for us ***AT*** is a generic element that can contain any attribute of the IoT systems.

V. Experimental Setup

In the previous section, we presented *DynPolAC* and its suitability with respect to the characteristics of the emerging IoT environments. In this section, we describe the experimental setup, which establishes the ground for Section VI to evaluate the performance of *DynPolAC*.

A. Research Questions

Our goal is to experimentally evaluate *DynPolAC* to answer the following three main research questions (*RQ*s):

- **RQ1**. At the micro level, how much processing overhead savings does *DynPolAC* offer compared to previous work?

  Emerging IoT environments are highly interactive (C1 in Section I). It is important to minimize the security overheads to achieve the optimum availability of the system (C2 in Section I). In *DynPolAC*, the overhead lies in parsing, processing, and enforcement of the policy objects. We will investigate how much improvement our framework offers to the processing time in a single IoT node.
- **RQ2**. At the macro scale, what significance *DynPolAC* makes when studied in an IoT environment?
  In this paper, our focus is on the implementation and plausibility of *DynPolAC*[5]. We resort to the use of a *discrete event simulator* that models an interactive IoT environment, where quick response times on the order of one second intervals are made. We evaluate the performance of *DynPolAC* in such systems. The system validation that we will report (in Section VI-B) is based on simulations and certain assumptions that will follow in Section V-B1.
- **RQ3**. How much memory overhead the construction of *DynPolAC* creates in IoT nodes?
  Most IoT nodes have strict memory limits. We measure and report the memory overhead of *DynPolAC* to investigate its suitability with the third characteristic explained in Section I.

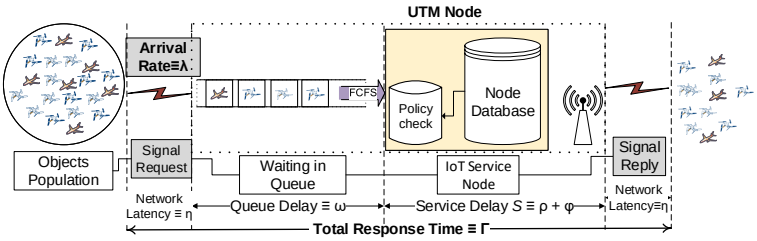


Fig. 1: Schematic view of 'M/M/1' queuing service for an UTM node. Notations described in Table III.

B. System Design

In Section II, we postulated that there are two kinds of emerging IoT schemes. Here we investigate the first scheme, i.e., central hubs known as UTM [15], [31] for drones, and RSU [32] for cars.

Consider an environment where drones share data frequently via a UTM system to guarantee collision-free navigation, congestion and traffic management, and to accomplish a mission as a swarm. The UTM in our construction is the only point of contact among the drones and between the applications running in the network. Every drone will need to first register its policy (P_{Drone}) with the UTM node to share information or ask for data. We model this scenario in Figure 1. Since such a model does not fully exist yet, we apply discrete event

[4]XACML policy standard provides *six combining algorithms* (*ca*) according to the order of decision sets and their priorities [26].

[5]DynPolAC is available in GitHub at www.github.com/DependableSystemsLab/DynPolAC.

TABLE III: Notations.

Parameters	Description	Unit	Value
λ	Arrival rate	1/s	1-10
ℓ_{Query}	Query size	Bytes	200-5K
ℓ_{Policy}	Policy size	no. of rules	1-2000
η	Network latency	ms	50
Variables	**Description**	**Unit**	
μ	Mean response rate	1/s	
Γ	Mean response time	ms	
N	Mean node delay	ms	
$\boldsymbol{S}$	Mean service time	ms	
ω	Wait time in the queue	ms	
ρ	Policy processing time	ms	
φ	Query time	ms	

simulation to implement Figure 1, i.e., a cluster of drones that contact the UTM server in one-second intervals want to query some data. The UTM node replies back with the data based on the registered P_{Drone} policy. For example, a drone may get access to the names of other drones in its neighborhood once registered a policy that authorizes access to such information in the UTM node.

1) Assumptions: We make *three* assumptions to simplify our simulation setup.

(a) Our UTM is a single-processor, single-threaded system based on the Queue Model type 'M/M/1' with service discipline First-Come-First-Serve [33].
(b) For calculating the total response time (Γ), we assume the wireless signal latency is a 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 [34].
(c) We heuristically chose the range for our parameters indicated in Table III. For the ***PS*** size (ℓ_{Policy}), we assume that drones will arrive and incrementally register their policies with the UTM. Therefore, if we assume there are 1000 distinct drones in the area and each register a two-rule list, on average, our accumulative ***PS*** size (ℓ_{Policy}) can reach up to 2000 rules. We apply the uniform distribution in our simulations to model the ***PS*** ranges between 1 and 2000 rules.
For the query size (ℓ_{Query}), we choose a range in accordance with the maximum transmission unit of the Ethernet packets [35], [36]. Thus, our query sizes follow a uniform distribution between 200 bytes and 5 KB.
For the arrival rate, λ, we choose the range 1 to 10 as discussed at the end of Section II. The arrival rate, λ (Table III), follows the Independent and Identically Distributed random variables based on the Poisson distribution stream [33].

2) Formulation: Based on the model presented in Figure 1, Table III shows the symbols in our system. The goal is to measure the performance of our system by calculating the response time (Γ) defined as: *the total time it takes for a drone to initiate a request until the reply is received*. We measure the latency at each stage in Figure 1.

We calculate the *UTM node delay*, *N*, by adding the queueing time (ω) for each drone to the service time *S*.

$$N_{(\lambda,\ell_{policy},\ell_{Query})} = \omega_{(\lambda)} + S_{(\ell_{Query},\ell_{Policy})} = \omega_{(\lambda)} + \rho_{(\ell_{Policy})} + \varphi_{(\ell_{Query})} \quad (1)$$

The overall response time (Γ) is the *UTM node delay* obtained from Eq. 1 plus the arrival and departure network latencies (50ms each) as follows:

$$\Gamma_{(\lambda,f_{cpu},\ell_{policy},\ell_{Query},\eta)} = \eta + N + \eta = 50ms + N + 50ms = 100ms + N \quad (2)$$

The response rate (μ) is the inverse of the response time:

$$\mu = \frac{1}{\Gamma} \quad (3)$$

We calculate the overall response time based on the Eq. 2 using the *Regeneration Method* [33]. We stop the simulation when the overall mean response time (Γ) is at steady-state of two decimal digits of a millisecond, and achieves the 95% two-sided confidence interval [33]. Then, we take the inverse of Γ to be the response rate (μ). The stability condition dictates that $\mu > \lambda$, which we will explore in our system study in Section VI-B1.

VI. Experimental Results

We answer the proposed research questions in Section V-A based on the experimental setup in Section V-B.

A. RQ1: Policy Processing Evaluation

To study the performance of *DynPolAC*, we first measure its processing time in IoT nodes.

A feature of *DynPolAC* is its *dynamic* capability to update the rules at run-time via a housekeeping thread. When a new query request arrives, the *DynPolAC* engine is activated; first, it parses the ***PS*** associated with the request; then, based on the rules, the server responds back to the query request. During this process, the housekeeping engine will check if it needs to augment and register new rules to the currently active ***PS***, or, remove the obsolete rules from the ***PS***. Figure 2 illustrates the performance cost of the service with the policy processing time results. Since we constructed our rules in XML, we employed an open-source parser called *Expat* [37] to parse our policy files. In Figure 2, we have varied our rules for parsing and registration between 1 and 2000. A one rule registration in DynPolAC takes 2.39 ms, on average[6], on a BBB platform. The maximum average processing time is for 2000 rules registration with 349.4 ms on the same platform. From Figure 2, the linear growth in the processing time is because of the one-on-one dependency between the parsing process and the size of the policy files (ℓ_{Policy}) located in our ***PS***. We have also assessed the generality of the results by measuring the processing time on 3 different platforms, namely, Pi3, Pi0, and BBB — our results show a homogeneous, monotonic increase.

[6]To achieve a steady-state average time with two significant decimal precision points, we ran DynPolAC 1000 times for every point.

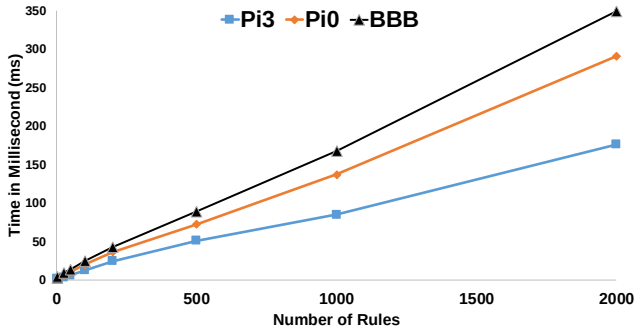


Fig. 2: Policy processing time in DynPolAC. Results are measured in milliseconds on three different platforms, Beagle Bone Black (BBB), Raspberry Pi Zero (Pi0), and Raspberry Pi 3 (Pi3).

Next, to study the performance overhead savings, we compare *DynPolAC* with the previous work that used XACML-based policy constructs for the IoT systems [6], [25]. We programatically produce XACML policy files up to 2000 rules [5], [6], [25]. For consistency of our comparison, we use the same parser (*Expat* [37]) used in *DynPolAC* for the XACML policy files.

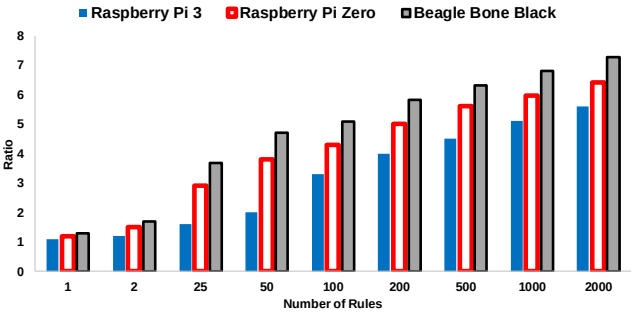


Fig. 3: Ratios of the policy processing speed-up for DynPolAC over the XACML-based access control. Results are measured as ratios on different platforms.

Figure 3 illustrates the results on the three different platforms. The results are the ratio of the processing speedup in *DynPolAC* over the XACML-based model. As the number of rules grow, the difference in the policy processing time between *DynPolAC* and XACML-based policies increases. The maximum difference in our study occurs during the 2000-rule policy comparison, where *DynPolAC* achieves up to 7.28x faster processing time. Even at smaller counts of 25, 50, and 100 rules, *DynPolAC* achieves a noticeable 4x processing time improvement. The speed-up improvements are mainly because *DynPolAC* uses a lighter policy language compared with previous XACML-based models (Section IV-D). Additionally, the speedup of *DynPolAC* is higher on slower platforms. The ratio is highest with BBB, followed by Pi0, and finally Pi3. This demonstrates the suitability of our scheme for low-capacity devices, i.e., the main target platforms for our work.

B. RQ2: DynPolAC Performance Results at System-level

In RQ1, we inspected the performance of our access control unit, *DynPolAC*, and compared it with the previous models to study the performance savings. It is important to evaluate our solution in the context of an end-to-end system. In RQ2, we study how much difference *DynPolAC* makes when used as an access control service in a dynamic IoT environment. Therefore, we ran two simulation tests based on the Figure 1.

1) Stability Condition: The first test is to check the stability condition by measuring the mean response rate (μ) and comparing it against the arrival rate (λ). In particular, if a number of drones arrive at our UTM node every second, does our service node have enough time to process the requests or will it experience instability? To answer the question, we need to measure μ, which is in turn dependent on three parameters: λ, ℓ_{Policy}, and ℓ_{Query}. We vary these parameters as described in Section V-B1 and Table III. For the arrival rate, λ, we used the *Inverse Transformation Method* to get the Poisson variates with *Mean 4*. For the number of policy rules, ℓ_{Policy}, we used the uniform distribution constructing rules between 1 and 2000. Finally, for the query size, ℓ_{Query}, we used the uniform distribution constructing queries between 200 B and 5 KB.

TABLE IV: Simulation results for Γ and μ. The *ratio* column tells how much overhead the policy-based simulations compare against the no-policy one.

Mean arrival rate	$\bar{\lambda}$ = 4/s		
Simulation	Γ (ms)	μ	Ratio
no-policy	178	5.6	1
DynPolAC	245	4.1	1.3
XACML-based	840	1.2	4.7

Table IV demonstrates our results. At steady state, the mean response time with a UTM node that includes *DynPolAC* is 245 ms, whereas, it is 840 ms when using an XACML-based *PS*. *DynPolAC* improves the overall response time by 70%. We also ran a test where there is no security enforcement, and observed a mean response time of 178 ms. *DynPolAC* ratio with the no-policy setting is 1.3, while this ratio is 4.7 for the XACML-based counterpart. Additionally, *DynPolAC* satisfies the system stability condition with the response rate being right above the threshold ($\mu > \lambda \equiv 4.1 > 4$), while if we use XACML-based models, the system becomes unstable.

2) Sensitivity Analysis: The second system test is the sensitivity analysis. By sensitivity analysis we seek at determining which of the varying parameters have the most impact on the total response time Γ. And, if *DynPolAC* is used, can better performance results be achieved?

TABLE V: Boundaries for sensitivity study.

Name	Min	Variable	Max
Arrival Rate	2/s	$\leq \lambda \leq$	8/s
Query Size	200 B	$\leq \ell_{Query} \leq$	2000 B
Policy Set (PS)	100 Rules	$\leq \ell_{Policy} \leq$	1000 Rules

We ran the sensitivity analysis while varying parameters λ, ℓ_{Query}, and ℓ_{Policy}. We used the boundaries as indicated in Table V to have two variables at their fixed extreme low and high boundaries while sweeping the third variable between its minimum and maximum ranges. Therefore, we have four combinations of extreme cases for each variable span, yielding a total of 12 scenarios. Figure 4 shows the results. For comparison, we performed these tests for both *DynPolAC* and the XACML-based counterpart. Our study reveals that *DynPolAC* achieves lower response time across the entire parametric space. In addition, the following trends can be observed from the figures:

- Figures 4a-4d show the arrival rate sweeps. *DynPolAC* has a maximum response time of 570.12 ms. However, for an XACML-based node, the Γ value at the extreme case of high query size and the large number of policy rules is 2.4 seconds. This can be considered as an extremely dynamic environment, where less than one-second decision making resolutions are needed. From these extreme case results of arrival rate, we conclude that XACML-based models cannot cope with highly interactive IoT nodes. The contrary holds for *DynPolAC*.
- Figures 4c-4d show a larger gap between *DynPolAC* and the XACML-based model. A similar behavior is shown in Figures 4g-4h. This means that a higher number of policy rules aggravates the overall response time of XACML more than it does for *DynPolAC*. These findings also highlight that processing policy rules can represent a large fraction of the total response time.
- In the last row in Figure 4, where the number of policy rules are varied, our XACML-based results display a sharp slope in the response time. However, with *DynPolAC* the response times show a relative steady-state with minimal fluctuations. This means that policy selection has the most pronounced effect in the overall response time compared to the other parameter.

C. RQ3: Size of DynPolAC

We measured the memory overhead of our implementation and compared it to the rest of our system [38]. *DynPolAC* is created as an integral component of a custom-made query service [38] with parsing and housekeeping modules. *DynPolAC* incurs only 7.5% memory overhead, which makes it possible to use it even in severely memory constrained nodes [11].

D. Implications

We view our design implications as twofold. First, the hierarchy of our language as explained in Table II keeps the policy blocks in modular form, which helps to commission new rules at run-time without impacting the dynamic state of the system. *DynpolAC*'s modularity also enables a Lego-like access control solution for bigger IoT hubs such as RSU and UTM, where a large cluster of heterogeneous IoT nodes connect and register their policy regularly [30].

Second, from the performance results we conclude that *DynPolAC* can be used as an access control service for information security between highly dynamic IoT nodes. The performance benefits of *DynPolAC* are mainly due to its deliberate language selection. At the micro level, *DynPolAC* outperforms other approaches at larger policy sizes, particularly, in our most constrained node under study, the BBB (Figure 3). This means that the benefits of using *DynPolAC* are more evident in resource-constrained devices. At the macro scale, *DynPolAC* outperformed the other systems consistently, particularly, when the system was under heavy loads (Figure 4).

VII. Conclusion

IoT environments are becoming highly interactive and require agile security techniques to fit their characteristics. We introduced *DynPolAC*, a dynamic policy-based access control that uses sufficient language to express rules in IoT systems. We evaluated *DynPolAC* by measuring its performance at micro- and macro-scales. At the micro level, *DynPolAC* outperforms previous XACML-based methods by a factor of up to 7.28x. At the macro scale, our results showed that *DynPolAC*-based solutions are significantly faster than previous methods across a wide range of system parameters.

Acknowledgment

This work was funded in part by a research grant from the NSERC (Natural Sciences and Engineering Research Council of Canada), and a four year fellowship from UBC.

References

[1] J. Won, S.-H. Seo, and E. Bertino, "A secure communication protocol for drones and smart objects," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, pp. 249–260, ACM, 2015.

[2] R. Pellizzoni, N. Paryab, M.-K. Yoon, S. Bak, S. Mohan, and R. B. Bobba, "A generalized model for preventing information leakage in hard real-time systems," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2015 IEEE*, pp. 271–282, IEEE, 2015.

[3] A. Ouaddah, H. Mousannif, A. A. Elkalam, and A. A. Ouahman, "Access control in the internet of things: Big challenges and new opportunities," *Computer Networks*, vol. 112, pp. 237 – 262, 2017.

[4] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497–1516, 2012.

[5] 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*, pp. 206–213, June 2012.

[6] 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)*, pp. 1–8, Sept 2014.

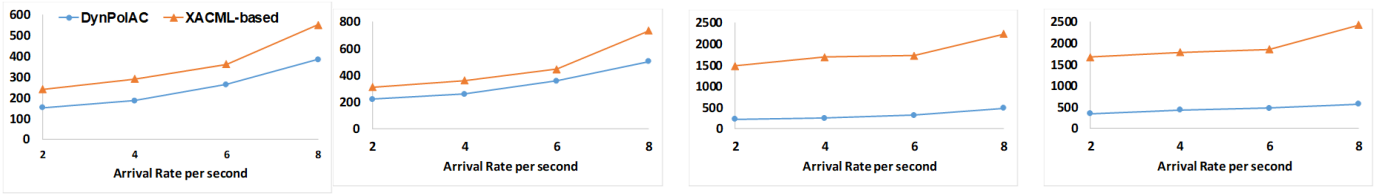
[7] L. Seitz, G. Selander, and C. Gehrmann, "Authorization framework for the internet-of-things," in *2013 IEEE 14th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, pp. 1–6, June 2013.

[8] F. Turkmen, J. den Hartog, S. Ranise, and N. Zannone, "Formal analysis of xacml policies using smt," *Computers & Security*, vol. 66, pp. 185–203, 2017.

[9] F. Schaub and P. Knierim, "Drone-based privacy interfaces: Opportunities and challenges," in *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, USENIX Association, 2016.

[10] D. Roca, D. Nemirovsky, M. Nemirovsky, R. Milito, and M. Valero, "Emergent behaviors in the internet of things: The ultimate ultra-large-scale system," *IEEE Micro*, vol. 36, no. 6, pp. 36–44, 2016.

[11] C. Bormann, M. Ersue, and A. Keranen, "Terminology for constrained-node networks," tech. rep., 2014.



(a) λ with fixed ℓ_{Query} = 200 B and ℓ_{Policy} = 100 Rules.

(b) λ with fixed ℓ_{Query} = 2000 B and ℓ_{Policy} = 100 Rules.

(c) λ with fixed ℓ_{Query} = 200 B and ℓ_{Policy} = 1000 Rules.

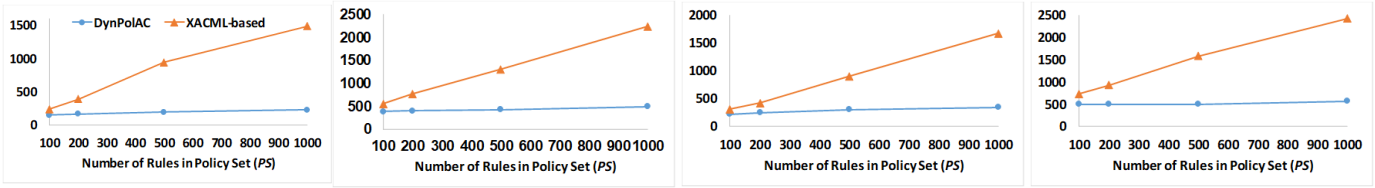
(d) λ with fixed ℓ_{Query} = 2000 B and ℓ_{Policy} = 1000 Rules.

(e) ℓ_{Query} with fixed λ = 2/s and ℓ_{Policy} = 100 Rules.

(f) ℓ_{Query} with fixed λ = 8/s and ℓ_{Policy} = 100 Rules.

(g) ℓ_{Query} with fixed λ = 2/s and ℓ_{Policy} = 1000 Rules.

(h) ℓ_{Query} with fixed λ = 8/s and ℓ_{Policy} = 1000 Rules.



(i) ℓ_{Policy} with fixed λ = 2/s and ℓ_{Query} = 200 Bytes.

(j) ℓ_{Policy} with fixed λ = 8/s and ℓ_{Query} = 200 Bytes.

(k) ℓ_{Policy} with fixed λ = 2/s and ℓ_{Query} = 2000 Bytes.

(l) ℓ_{Policy} with fixed λ = 8/s and ℓ_{Query} = 2000 Bytes.

Fig. 4: Sensitivity: Variables are λ, ℓ_{Query}, and ℓ_{Policy}. The vertical axes in all Figures is the *overall mean response time*, Γ, in ms. The triangular marker is for the XACML-based results while the circular marker is for the *DynPolAC* ones.

[12] D. Evans, "The internet of things: How the next evolution of the internet is changing everything," *CISCO white paper*, vol. 1, no. 2011, pp. 1–11, 2011.

[13] F. A. A. (FAA), "Faa releases 2016 to 2036 aerospace forecast.," March 2016.

[14] forbes.com, "10 million self-driving cars will hit the road by 2020 – here's how to profit," Mar. 2017.

[15] N. Aeronautics and S. Administration, "Unmanned aircraft system (uas) traffic management (utm)," Aug. 2017.

[16] M. M. Hasan and H. T. Mouftah, "Cyber-physical vulnerabilities of wireless sensor networks in smart cities," *Security and Privacy in Cyber-Physical Systems: Foundations, Principles, and Applications*, 2017.

[17] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[18] J. L. Hernández-Ramos, A. J. Jara, L. Marin, and A. F. Skarmeta, "Distributed capability-based access control for the internet of things," *Journal of Internet Services and Information Security (JISIS)*, vol. 3, no. 3/4, pp. 1–16, 2013.

[19] G. Rigazzi, A. Tassi, R. J. Piechocki, T. Tryfonas, and A. Nix, "Optimized certificate revocation list distribution for secure v2x communications," *arXiv preprint arXiv:1705.06903*, 2017.

[20] P. N. Mahalle, P. A. Thakre, N. R. Prasad, and R. Prasad, "A fuzzy approach to trust based access control in internet of things," in *Wireless Communications, Vehicular Technology, Information Theory and Aerospace & Electronic Systems (VITAE), 2013 3rd International Conference on*, pp. 1–5, IEEE, 2013.

[21] O. Standard, "extensible access control markup language (xacml) version 2.0," 2005.

[22] S. Jahid, I. Hoque, H. Okhravi, and C. A. Gunter, "Enhancing database access control with xacml policy," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pp. 130–133, 2009.

[23] M. Jung, G. Kienesberger, W. Granzer, M. Unger, and W. Kastner, "Privacy enabled web service access control using saml and xacml for home automation gateways," in *Internet Technology and Secured Transactions (ICITST), 2011 International Conference for*, pp. 584–591, IEEE, 2011.

[24] J. B. Bernabe, J. L. H. Ramos, and A. F. S. Gomez, "Taciot: multidimensional trust-aware access control system for the internet of things," *Soft Computing*, vol. 20, no. 5, pp. 1763–1779, 2016.

[25] M. Drozdowicz, M. Ganzha, and M. Paprzycki, "Semantically enriched data access policies in ehealth," *Journal of medical systems*, vol. 40, no. 11, p. 238, 2016.

[26] "Xacml document." http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf. Accessed: December 10, 2017.

[27] W. Sun, J. Liu, and H. Zhang, "When smart wearables meet intelligent vehicles: challenges and future directions," *IEEE wireless communications*, vol. 24, no. 3, pp. 58–65, 2017.

[28] F. Schaub, R. Balebako, A. L. Durity, and L. F. Cranor, "A design space for effective privacy notices," in *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*, pp. 1–17, USENIX Association, 2015.

[29] J. Balasch, A. Rial, C. Troncoso, B. Preneel, I. Verbauwhede, and C. Geuens, "Pretp: Privacy-preserving electronic toll pricing.," in *USENIX Security Symposium*, vol. 10, pp. 63–78, 2010.

[30] T. Vaidya and M. Sherr, "Mind your (r,\varphi) s: Location-based privacy controls for consumer drones," in *Cambridge International Workshop on Security Protocols*, pp. 80–90, Springer, 2015.

[31] P. H. Kopardekar, "Unmanned aerial system (uas) traffic management (utm): Enabling low-altitude airspace and uas operations," 2014.

[32] T.-J. Wu, W. Liao, and C.-J. Chang, "A cost-effective strategy for road-side unit placement in vehicular networks," *IEEE Transactions on Communications*, vol. 60, no. 8, pp. 2295–2303, 2012.

[33] R. Jain, "The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling," 1991.

[34] C. Sarr and I. Guérin-Lassous, *Estimating average end-to-end delays in IEEE 802.11 multihop wireless networks*. PhD thesis, INRIA, 2007.

[35] J. Postel and J. K. Reynolds, "Standard for the transmission of ip datagrams over ieee 802 networks," 1988.

[36] J. Postel, "The tcp maximum segment size and related topics," 1983.

[37] "Expat parser for xml language files." https://www.xml.com/. Accessed: Sept 05, 2017.

[38] M. Karimibiuki and A. Ivanov, "Minicloud: A mini storage and query service for local heterogeneous iot devices," in *Proceedings of the 8th International Conference on the Internet of Things*, ACM, 2018.