# CAPE: Continuous Access Policy Enforcement for IoT Deployments

[1]Ashraf Alkhresheh, [2]Khalid Elgazzar, [1]Hossam S. Hassanein
[1]School of computing, Queen's University, Kingston, ON K7L 3N6, Canada
[2]Department of Electrical, Computer and Software Engineering
University of Ontario Institute of Technology, Oshawa, ON L1H 7K4 Canada
Email: khashraf@cs.queensu.ca, Khalid.Elgazzar@uoit.ca, hossam@cs.queensu.ca

***Abstract*—Advancements and convergence in IoT enabling technologies along with ubiquitous connectivity have led to the generation of new wave of smart services and applications based on real-time data access. The popularity of ubiquitous data access and accelerated adoption of these services pose significant challenges on user and data privacy. Thus, controlling access to such services in highly dynamic environments with continuously changing context becomes even more challenging. The wide adoption of IoT in our everyday life in many vital domains such as healthcare and military operations requires continuous and tight access control to prevent unauthorized and unintended access. A delay in making access decisions when context changes may result in consequences that cause harm and property damage. Therefore, continuity in access policy enforcement becomes a necessity in highly dynamic IoT environments for the entire access session not only at the time of request. This paper presents CAPE, a continuous access policy enforcement framework for IoT deployments. CAPE describes access control elements using predicates, and stores them as primitive facts in a K Dimensional tree data structure. Our algorithms automatically match access requests with primitive facts, generate access policies, make context-aware access decisions at run time and continuously monitor access control parameters based on which access decisions were made. Performance evaluation of CAPE demonstrates that this framework efficiently controls access in highly dynamic IoT environments.**

## I. INTRODUCTION

Traces of the IoT concept go back to the early work done by Kevin Ashton in 1999, which has received continuous attention from both academia and industry all over the world [1]. The idea behind the IoT is to extend everyday objects, not commonly considered computers, with computing capabilities to identify themselves, generate, and communicate information about their physical environments [2]. Typically, these everyday objects are physical, and virtual sensors, radio frequency identification (RFID) tags and smartphones [3]. IoT smart interaction has led to rapid developments of a plethora of smart applications and services that bring benefits to many aspects of our everyday life.

A recent study conducted by IoT analytics [4] to rank IoT applications based on their popularity shows that, in 2018 most of identified IoT projects are associated with Smart City (367 projects), Industrial Settings (265), and Connected Building IoT projects (193). However, the proliferation of smart devices in individual's surroundings has enabled pervasive collection, processing, and dissemination of personal information, raising a considerable privacy concerns among IoT users that, even if not realized, could undermine the user's confidence of the new technology and impede its widespread adoption [5]. In order to unleash the full potential of IoT, smart devices should make their data accessible to interested parties (e.g., smartphones, web services) in a controlled manner; otherwise, potential IoT privacy breaches will outweigh its benefits.

Access control schemes have been extensively studied over the past few decades and remain an area of intense research interest due to rapidly changing technology, growing privacy concerns, and rapid adoption of the IoT. However, the extremely large number of IoT devices with low power requirements communicating over dynamic, distributed and ad-hoc networks, creates a unique set of authorization and access control challenges, rendering traditional access control models such as Role Based Access Control (RBAC) [6] and Attribute Based Access Control (ABAC) [7] unfit for IoT scenarios. In general, these approaches lack the ability to incorporate the operational context (e.g., time and location) into the authorization process; a capability that is inevitably required to cater for frequent changes in security and privacy requirements. These contexts greatly affect the access control decisions and subsequently the performance of an access control system. Hereinafter, we refer to this information collectively as context information.

### A. Motivating Scenario

In the following, we provide an everyday life scenario through which we point out unique access control challenges in dynamic IoT environments. We begin by considering an outpatient Bob is going on a tourist trip. While in public spaces, Bob may share his location information with family members, close friends and third-parties (e.g., `hospitals, hotels and restaurants`) businesses that provide smart services based on proximity and personal preferences. However, Bob may prefer to restrict access to this information while visiting places that could indicate health care, financial or shopping preferences. In addition, Bob has a long term continuous health monitoring device (e.g., `mobile cardiac telemetry`) attached to his chest that can detect early health abnormalities and send alerts to relevant personnel. While in normal health situations, Bob may prefer that only his primary physician can have access to his health data. However, if Bob defines that when his vital signs go far above the norm and his health situation requires urgent care, the closest health care provider can gain access to this information and act accordingly.

### B. Dynamic access control requirements

By analyzing the above scenario, we identify two dynamic access control requirements:

978-1-5386-7747-6/19/$31.00 ©2019 IEEE

1) The interactions between IoT devices are generally characterized as spontaneous and opportunistic, and can result in frequent changes in operational context and consequently in security and privacy requirements. For instance, when Bob frequently changes his location between private and public areas. In this case, the policy management process (i.e., adding/removing users, resources, and rules) becomes a more complex and error-prone task, particularly for an entry level IoT device owner. Therefore, an automation of the access policies specification is necessary to minimize human intervention in policy specification and to improve adaptability of the access control policy to the high dynamic IoT deployments.
2) The sensitivity of the data that is continuously streamed from Bob's IoT devices to interested users will be frequently changing during the lifetime of an access session. A delay in capturing changes in context can result in an unauthorized access and could even endanger Bob's life in an emergency. Therefore, continuous access control enforcement mechanism is required to assign and revoke access privileges in real time.

In this paper, we propose CAPE a Continuous Access Policy Enforcement framework. CAPE continuously enforces access control policies over the lifetime of granted access sessions. In response to changes in access context, CAPE re-evaluates all ongoing access sessions and dynamically updates access privileges assignments associated to them in real time.

## II. BACKGROUND AND RELATED WORKS

Access control (AC) is the process of enforcing the system security requirements on protected services and resources [8]. AC determines the level of authorization an entity can be assigned by evaluating its associated properties (e.g., `group membership, roles and proximity`) against access control rules.

AC systems consist of three abstraction: policies, models, and mechanisms [9]. While AC policies determine the high-level security rules according to which AC must be regulated, AC models provide a formal description of the AC security policy and procedures. AC mechanisms provide the low-level functions that implement and enforce the security rules defined in the AC policies and formalized by the AC model. AC core elements represent the input upon which functions of an AC mechanism enforces the access rules defined in the AC policy.

Access control elements are: Subject (e.g., `person, device, application or process`) that actively causes information to flow between objects; Object (e.g., `mobile cardiac outpatient telemetry, GPS`), which is a passive entity that stores, streams or receives information; and Operation, which refers to a certain action (e.g., `read, write`) invoked by a subject and applied to an object.

The literature of access control models that use context information in making access decisions divide into two implementation approaches [10]. The first (and most simply implemented) is discontinuous enforcement, this approach verifies the context information, assigns the access privileges and makes the access decision only at the time access is requested. It does not consider changes in context after the access decision is made such as the work by authors in [11, 12]. This approach can result in security and privacy breaches by disclosing system resources to users whose access context has changed since the time access decision was enforced, and therefore they become non-authorized users.

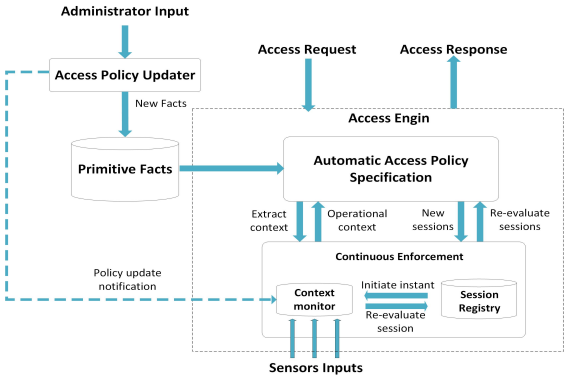


Fig. 1: Continuous access policy enforcement

The second is, continuous enforcement, this approach constantly monitors the operational context of an access session and continuously updates the assigned access privileges according to changes in access context and discloses system resources only to those users who are authorized under the current context. While discussion on the first approach has dominated the research area in recent years, there is little, if any, research investigating the practicality of the second approach in high dynamic IoT environments.

Nehme et al.[13] propose a continuous access control enforcement framework for dynamic data stream environments. In their work, they use predicates to describe the current access authorization at the query specifier side (i.e., `subject`) and the security policy on data stream at data provider side (i.e., `object owner`). Both types of predicates are streamed with data to a security aware query processing component, which continuously enforces access control policy on the data stream (i.e., `object`). Despite the promising features that this solution supports, it does not fit IoT environments due to the assumption that both the data provider and query specifier sides have sufficient computation and communication capabilities to specify, encode, and stream access policies to the query processing component. As well, such capabilities are not supported by most IoT devices.

## III. CONTINUOUS ACCESS POLICY ENFORCEMENT FRAMEWORK

At the core of our design and implementation of CAPE, we aim to develop a policy enforcement mechanism that can capture changes in access context at fine granularity and continuously enforce the appropriate access permissions in highly dynamic IoT scenarios while not posing extra computation or communication overhead. `Fig 1` depicts the main components of CAPE framework and the interactions between these components.

### A. Automatic Access Policy Specification (APS)

In our previous work [14], we proposed a context aware automatic access policy specification mechanism in which we defined two types of security con-

texts: The *guard context*, which is a set of a predefined application dependent values and thresholds represented as key-value pairs (e.g., `location: "Public", time: "08:00-16:00"`) that regulate access to protected resources. These key-value pairs describe the conditions defined by the object owner under which a requesting subject of certain attributes (e.g., `identity: "name", relationship: "family member"`) can perform the operation of certain attributes (e.g., `type: "read", granularity: "per minute"`) on a protected object with certain attributes (e.g., `CPU utilization: "<70 %", energy level: ">80 %"`), and the *operational context*, which is a set of real-time measurements, also represented in key-value pairs, that reflect the real-world conditions of the requesting entity, device owner, and IoT device at the time of access. Thus, if operational context measurements satisfy the guard context conditions, then access is permitted. Otherwise, access is by default denied.

### B. Primitive Facts (PF): Attributes and Guard Context

In this section, we describe the core access control elements (subject, operation, and object) using abstractions, in a key-value pair representation, which contains both the attributes that characterize elements and the guard context that determines the qualification context (or constraints) relevant to each element according to which access to protected resources is regulated. We build these basic abstractions and represent them in predicates as follows:

***element***$(X)$ is a descriptor represented by a tuple of the form: `{type:value, key1:value1, key2:value2, ...}`. The tuple contains a key *type* whose value is `{subject|operation|object}` to indicate that this tuple is pertaining to one of the access core elements. For example, the following basic abstraction represents a factual tuple for a subject $x$:

**element x=** `{type: "subject", name: "Any", relationship: "Close Friend",location:"Public", time:"8:00-16:00",coexistence: "True"}`.

This descriptor contains a type key, two subject attributes that basically identify the subject, and two access constraints that determine the guard context required for a `close friend` to gain access to the location information streamed by the Bob's smart phone. In this example, the key `coexistence` refers to collocation of both the requesting subject and object owner.

The ideal **request** abstraction consists of a request template that defines the request elements and their operational contexts. The request descriptor contains patterns of the form:

**p=**`{type:value, key1:value1, key2:value2, ...}`.

For example, the following tuple represents a localization service request (pertaining only to operation):

**p=**`{type: "operation", name: "read",location: "Public", time: "Any",Coexistence: \True"}`.

A request descriptor matches a resource if there is an operation in the fact base (i.e., `primitive facts`) that satisfies every pattern in the request template. Our access policy specification algorithm takes the primitive facts and the access request as inputs and automatically produces the access control decision as an output [14].

### C. Continuous Access Policy Enforcement (CAPE)

As we mentioned earlier, both the guard and operational context will be changing frequently over the lifetime of an access session. CAPE keeps track of these changes and maintains an authorized access over the lifetime of an ongoing access session. CAPE ensures that, at any point of time, the operational context of an access session satisfies its guard context. CAPE has two sub components:

*1) Session Registry (SR)*

This component stores a status record for each active session. Each status record maintains the session id, the initial operational and guard contexts upon which the request was granted, and a status flag that indicates the current status of an access session (e.g., `active, inactive or suspended`).

*2) Context Monitor (CM)*

*CM* continuously reads the inputs from available sensors and potential notifications from the Access Policy Updater (APU) and keeps track of changes in the contexts associated to the access elements involved in an active session. The APU provides a user-friendly interface through which the object owners can define and manage access control policies on their IoT objects.

The *CM* distinguishes two types of context changes: The *operational context changes*, which are more frequent and happen as a result of changes in real world conditions of the access elements (e.g., `proximity`). The *administrative context changes*, which are less frequent and happen as a result of either updating the access policies (i.e., `changing guard contexts`), or inferring new facts that could affect the access decision of an ongoing access session. For example, when the operational context of Bob's heart monitor device indicates an emergency health situation, our access engine can infer the new fact that Bob's privacy becomes less of a priority and that the closest hospital or ER department should have access to Bob's medical information and his current location.

**Listing 1:** Automatic access control policy specification (APS)

**input :** Primitive Facts *PF*, Access Request *AR*
**output:** Access Decision: *Permit/Deny*
**begin**
  **for** *AR* **do**
    **Extract** all attributes and operational context values of the subject $x$ = $\{type : \text{“}subject\text{”}, key_1 : value_1, key_2 : value_2 \dots\}$
    **Extract** all attributes and operational context values of the operation $y$ = $\{type : \text{“}operation\text{”}, key_1 : value_1, key_2 : value_2 \dots\}$
    **Extract** all attributes and operational context values of the object ,(if exist), $z$ = $\{type : \text{“}object\text{”}, key_1 : value_1, key_2 : value_2 \dots\}$
  **end**
  **Generate** *PF-query*$\{x\}$,*PF-query*$\{y\}$, *PF-query*$\{z\}$.
  Result := Deny
  **for** *all element* $\in$ *PF* **do**
    **if** $\exists$ *element(x)* ***and*** *element(y)* ***and*** *element(z) in PF:* **then**
      Result:= Permit
      CGC:= $x_{gaurdcx} \wedge y_{gaurdcx} \wedge z_{gaurdcx}$
      **Call CAPE** (*AR,CGC*)
      **Break**
    **end**
  **end**
**end**

**Listing 2:** Continuous Access Policy Enforcement (CAPE)

```
   input : Granted Access Request GAR , Common Guard Context
           CGC, policy_update, sensor_inputs, Session Registry SR
   output: Session Registry SR
1  begin
2      if GAR ∉ SR then
3          Create new status record R for GAR
4          R.session_Id := GAR
5          R.session_status := Active
6          R.guard_context := CGC
7          R.operational_context := sensor inputs
8          add R to SR
9          Initiate instant of context_monitor (GAR.contex_monitor)
10         while True do
11             if policy_update = False then
12                 if GAR.context_ monitor(operational_context) ⊈
                    R.Guard_Context then
13                     R.session_status =: inactive
14                     SR.Remove(R)
15                     GAR.context_monitor.destroy()
16                 end
17             end
18             if policy update = True then
19                 for all R ∈ SR do
20                     Re-evaluate := Call APS(GAR)
21                     if Re-evaluate = False
22                     then
23                         R.session_status =:inactive
24                         SR.Remove(R)
25                         GAR.context_monitor.destroy()
26                     end
27                 end
28             end
29         end
30     end
31 end
```

When APS grants access to new *AR*, it calculates the *Common Guard Context (CGC)* in `Listing 1,line 12`, which is the intersection of the subject, object and operation's guard contexts that matches the *AR*'s operational context. Subsequently, the APS passes the attributes and the *CGC* of the permitted *AR* to CAPE. Based on these two parameters and the current operational context of the Granted Access Request *GAR*, CAPE continuously enforces the *CGC* constraints over the lifetime of an access session. In `Listing 2, lines 14-19` ensure that the initial *CGC* of an ongoing access session is always satisfied by the current sensor inputs. Otherwise, the access session is suspended.

When the APU notifies the CAPE of administrative context changes, our algorithm, `lines 21-28`, does not interrupt the ongoing access sessions, rather it calls the APS to re-evaluate all ongoing access sessions against the changes in the primitive facts. If an access session is denied upon re-evaluation, our algorithm, `lines 25-27`, sets the session status to inactive and removes the session status record from the *SR*. Otherwise, the access session resumes, however, with new CGC that may restrict, relax or keep the access permissions previously associated to the access session .

At any point in time, the *SR* will only contain the status records of the active sessions. Yet we can choose to keep the status records of inactive and suspended sessions in the *SR* such that the *APS* can use these records to keep track of *AR* access history. As an example, *APS* can prioritize access decision making based on the history of the *AR(s)*, it could even deny an *AR* due to instability of its historical operational context.

## IV. USE CASE: HEALTH CARE DOMAIN

Returning to our motivation scenario we provided earlier, according to Bob the security requirements are: ($R_1$) all family members and close friends and third-party companies can have access to Bob's location only while he is in public places; Also, ($R_2$) only Bob's primary physician can have access to the information streamed by the medical device attached to him; and ($R_3$) access to Bob's health and location information becomes open for the closest hospital ER in an emergency situation.

In this scenario, Bob is sharing access to the data collected by the (GPS) in his smartphone and the medical device (MD) attached to his chest, and as policy administrator, Bob uses the APU to define the attributes and guard contexts for the two objects he owns based on the regulations and assumptions detailed in the following Tables:

- `TableI`: A subject is described by two attributes and three contextual conditions.
- `Table II`: An operation is described by two attributes and two contextual conditions.
- `Table III`: An object is described by two attributes and three contextual conditions.

TABLE I: Subject attributes and guard context.

| Attribute/Guard-Context | Range |
|---|---|
| Type | `subject` |
| Relationship | {`family member, close friend, primary physician, third-party`} |
| Location | {`Public,Private`} |
| Time | `24 hours` |
| Coexistence | {`True,False `} |

TABLE II: Operation attributes and guard context.

| Attribute/Guard-Context | Range |
|---|---|
| Type | `operation` |
| Operation-name | { `localize, read MD` } |
| Location | {`Public,Private`} |
| Time | `24 hours` |

TABLE III: Object attributes and guard context.

| Attribute/Guard-Context | Range |
|---|---|
| Type | `object` |
| Object-name | {`GPS, BLE, MD`} |
| Location | {`Public,Private`} |
| Time | `24 hours` |
| Emergency-case | {`True,False `} |

According to the resource profiles in `Tables I` to `III` and Bob's access regulations $R_1$-$R_3$, the system can generate the set of required primitive facts and present them to Bob for final approval. For example, `Table IV` lists the primitive facts Bob needs to define to fulfill $R_1$.

Alice, a friend of Bob's, is interested in visiting new tourist attractions during her upcoming vacation. Knowing that Bob is on a trip and both are interested in same places, Alice can subscribe to receive Bob's location information and submit an access request to Bob's GPS streaming data. Upon receiving Alice's access request, for instance at 12:00, our algorithm in `Listing 1` extracts all attributes and operational contexts associated with Alice's access request (i.e.,

TABLE IV: Primitive facts.

| | |
|---|---|
| $PF_1$ | {type: "subject", Relationship: "Family member", location: "Any", Time:"Any", Coexistence:"False"} |
| $PF_2$ | {type:"subject", Relationship: "close friend", location: "Any", Time:"Any", Coexistence:"False"}. |
| $PF_3$ | {type:"subject", Relationship: "third-party companies", location: "Any", Time:"Any", Coexistence:"True"}. |
| $PF_4$ | {type: "operation", Operation-name: "localize", Location: "Any", Time:"Any"} |
| $PF_5$ | {type: "object", Object-name: "GPS", Location: "Public", Time:"08:00-16:00", Emergency-case:"False"} |

through CM), and registers a continuous queries to the fact base (i.e., PFs) as follows:

***p1***={Type: "subject", Relationship: "close friend", Location: "Alice's location", Time: "12:00", Coexistence:"False"}.

***p2***={Type: "operation", "operation-name: "localize", Location: "Alice's location", Time: "12:00"}.

***p3***={Type: "object", object-name: "Any", Location: "Bob's location", Time: "12:00"}.

These request patterns p1-p3 collectively match the primitive facts $PF_1$, $PF_4$ and $PF_5$, see Table IV. Therefore, access to Bob's GPS information is granted. Afterward, our algorithm in Listing 2 creates a status record for the GAR and begins continuous monitoring of the granted session.

Now, suppose that the context changes when Bob's health declines and he suddenly requires immediate medical attention, our context manager will capture changes in Bob's context (e.g., Emergency-case:"True"), update the PFs and inform CAPE to re-evaluate all ongoing access sessions and update the status records of the access sessions affected. Based on the new context, CAPE grants access to Bob's health and location information, which is not allowed in normal cases. In addition, the status of Alice's access session becomes invalid when Bob enters a private place (i.e., hospital) and therefore, her current access permission to Bob's location will be immediately revoked.

## V. FRAMEWORK EVALUATION

In this section, we evaluate the performance of CAPE framework in two steps as follows:

### A. Analysis of time complexity

By analyzing our algorithms, we found that the performance of CAPE is limited only by the processing time required for APS component to re-evaluate all ongoing access sessions upon an administrative context change, see Listing 2, lines 21 -28. To achieve fast access policy specification, effective access policy management and incur less overhead to access policy enforcement, we structured the primitive facts in a K-Dimensional tree [15]. We chose KD tree data structure because of its usefulness in applications that involve multidimensional search key over large scale datasets. We built three separate KD trees for the subjects, objects, and the operations offline. The time complexity of CAPE algorithm is simply bounded to the time required to search each of the subject, object, and operation trees. Assuming the three KD trees are of the same size $n$, and the fact that KD is a special case of binary trees, CAPE has logarithmic time complexity $T$ which is given as follows:

$$T(n) = 3 * N * O(\log(n)) \tag{1}$$

Where $N$ is the number of ongoing access sessions. When $n >> 2^k >> N$ , where $k$ is the number of attributes and context constraints that describe each type of the core access control elements, the time complexity can be further simplified to:

$$T(n) = O(\log(n)) \tag{2}$$

### B. Implementation and performance significance

To quantify the processing overhead of CAPE, we implemented a proof of concept prototype using a microcomputer Raspberry Pi 3 [16]. We developed Python modules for the APS and CAPE components that run on the Pi microcomputer to control access to two representative IoT devices: Sensor Tag [17], and WeMo Switch [18]. Also, We developed a mobile Java application using Android Studio [19]. The application provides a user friendly interface through which users can search for and connect to available IoT devices in their proximity including their designated authorization server. In addition, device owners can use the mobile application interface to define and manage access control policies on their IoT devices.

To show the feasibility of our approach in highly dynamic, large-scale, and resource-limited environments such as IoT, we conducted two performance evaluation experiments as follows.

*1) Access Response Time vs Primitive Facts Number*

In this experiment, we compute the *access response time*, which is the time that CAPE requires to search a certain number of primitive facts and make an access decision in response to an access request.

Figure 2, shows the results of the worst case access response time against a variable number of primitive facts at a constant number of attributes per access element. At each number of primitive facts, we calculate the access response time for 10 runs and take their average. We compare, our search algorithm to the Brute Force(BF) search. KD Tree search outperforms the BF search at primitive facts number of 1250 with a response time of 2.25 ms. These results show that our approach can efficiently scale to IoT deployments.

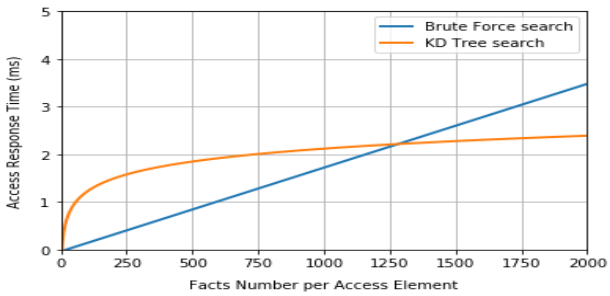


Fig. 2: Access response time Vs Number of Primitive Facts
Attributes Number = 13 per access element

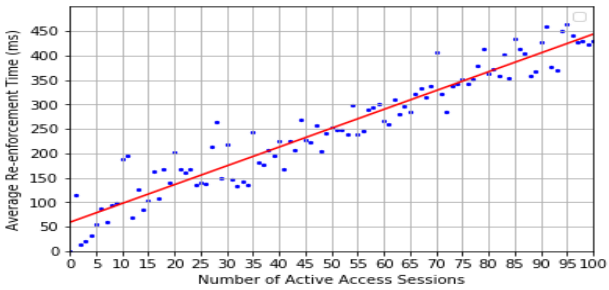


Fig. 3: Average Re-enforcement Time Vs Number of Access Sessions
Primitive Facts = 1000, Attributes Number = 13

*2) Average Re-enforcement Time vs Number of Access Sessions*

In this experiment, we compute the *access re-enforcement time*, which is the time that CAPE takes to reevaluate a certain number of active access session in response to administrative changes in guard contexts. For each number of access sessions, we calculate the average re-enforcement time over 50 runs. As shown in `Figure 3`, the re-enforcement time increases linearly as the number of active sessions increases. Therefore, our proposed CAPE can perform efficiently in time-sensitive IoT applications.

## VI. CONCLUSION

In this work, we propose a continuous access policy enforcement (CAPE) framework based on automatic policy specification. Our algorithm continuously enforces access policies over the lifetime of an access session, and re-evaluates every access session based on that session's guard and operational contexts. We conducted two performance evaluation experiments based the access response time and access re-enforcement time. The results show that our proposed approach can efficiently control access in highly dynamic, large-scale, and resource limited IoT environments. In the future, we plan to augment our access control engine with an offline fact matching mechanism that could further enhance the access response and re-enforcement times of CAPE. In addition, we plan to improve the access control granularity through incorporating more contextual information in access decision making (e.g., `access history`, `probability of misuse`) and conduct experiments to see the impacts of the added attributes and context constraints on the performance of CAPE.

## REFERENCES

[1] G. Santucci *et al.*, "From internet of data to internet of things," vol. 28, 2009.

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

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

[4] B. Janina, "The Top 10 IoT Segments in 2018 – based on 1,600 real IoT projects," "https://iot-analytics.com/top-10-iot-segments-2018-real-iot-projects", [Accessed : April, 2018].

[5] F. T. Commission *et al.*, "Internet of things: Privacy & security in a connected world," *Washington, DC: Federal Trade Commission*, 2015.

[6] 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.

[7] V. C. Hu, D. R. Kuhn, and D. F. Ferraiolo, "Attribute-based access control," *Computer*, vol. 48, no. 2, pp. 85–88, 2015.

[8] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence, "Aaa authorization application examples," Tech. Rep., 2000.

[9] V. C. Hu, D. Ferraiolo, and D. R. Kuhn, *Assessment of access control systems*. US Department of Commerce, National Institute of Standards and Technology, 2006.

[10] H. Martin *et al.*, "A generalized context-based access control model for pervasive environments," in *Proceedings of the 2nd SIGSPATIAL ACM GIS 2009 International Workshop on Security and Privacy in GIS and LBS*. ACM, 2009, pp. 12–21.

[11] G. Zhang and M. Parashar, "Context-aware dynamic access control for pervasive applications," in *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference*, 2004, pp. 21–30.

[12] D. Kulkarni and A. Tripathi, "Context-aware role-based access control in pervasive computing systems," in *Proceedings of the 13th ACM symposium on Access control models and technologies*. ACM, 2008, pp. 113–122.

[13] R. V. Nehme, H.-S. Lim, and E. Bertino, "Fence: Continuous access control enforcement in dynamic data stream environments," in *Proceedings of the third ACM conference on Data and application security and privacy*. ACM, 2013, pp. 243–254.

[14] A. Alkhresheh, K. Elgazzar, and H. S. Hassanein, "Context-aware automatic access policy specification for iot environments," in *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*. IEEE, 2018, pp. 793–799.

[15] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[16] C. Corporation", "Raspberry pi 3," http://www.canakit.com/raspberry-pi-3-starter-kit.html, [Accessed : May, 2017].

[17] SimpleLink™BluetoothSmart®/Multi-Standard, "Sensotag," http://www.ti.com, [Accessed : May, 2017].

[18] "Belkin's wemo switch," http://www.wemo.com/, [Accessed : May, 2016].

[19] "Android studio," https://developer.android.com/studio/, [Accessed : May, 2017].