Context-driven Policies Enforcement for Edge-based IoT Data Sharing-as-a-Service

Huu-Ha Nguyen, Phu H. Phung
Intelligent Systems Security Lab
Department of Computer Science, University of Dayton
Dayton, Ohio, USA
https://isseclab-udayton.github.io/

Phu H. Nguyen
SINTEF
Oslo, Norway
phu.nguyen@sintef.no

Hong-Linh Truong
Aalto University
Espoo, Finland
linh.truong@aalto.fi

***Abstract*—Sharing real-time data originating from connected devices is crucial to real-world Internet of Things (IoT) applications that especially use artificial intelligence/machine learning (AI/ML). Such IoT data are typically shared with multiple parties for different purposes based on data contracts. However, supporting these contracts under the dynamic change of IoT data variety and velocity faces many challenges when such parties (aka tenants) want to obtain data based on the data value for their specific contextual purposes.**

This work proposes a novel dynamic context-based policy enforcement framework to support IoT data sharing based on dynamic contracts. Our enforcement framework allows IoT Data Hub owners to define extensible rules and metrics to govern the tenants in accessing the shared data on the Edge based on policies defined in static and dynamic contexts. For example, given the change of situations, we can define and enforce a policy that allows pushing data to some tenants via a third-party means. In contrast, typically, these tenants must obtain and process the data based on a pre-defined means. We have developed a proof-of-concept prototype for sharing sensitive data such as surveillance camera videos to illustrate our proposed framework. Our experimental results demonstrated that our framework could soundly and timely enforce context-based policies at runtime with moderate overhead. Moreover, the context and policy changes are correctly reflected in the system in nearly real-time.

***Index Terms*—Context-driven, Data sharing-as-a-service, policy enforcement, IoT**

I. Introduction

One of the significant aspects of the Internet of Things (IoT) is to provide data sharing to connect organizations, customers, suppliers, or other stakeholders, bringing greater values for different stakeholders in the IoT's ecosystems [1], especially with AI/ML applications. Supporting stakeholders to securely and efficiently share the live stream and/or real-time or near real-time IoT data based on data context is becoming important. This is because each stakeholder in different sectors can use the data for their context-specific business and applications, e.g., in smart cities scenarios [2].

A. Background

Typically, IoT data is shared among multiple parties based on data contracts, implicitly or explicitly documented with-/without machine-readable policies. For many of today's scenarios, data sharing must happen at the Edge [3], [4] to facilitate edge analytics and to avoid complex issues of data security and privacy. Furthermore, the edge-computing level will allow tenant applications, especially cross-sector IoT AI services, to operate and enable trustful on-demand data acquisition effectively. A well-known challenge in this data sharing at the Edge is *real-time, context-sensitive* data access control for different stakeholders. Each stakeholder has a different contract, and the contract may be based on the context of the data that is dynamically changed at runtime. Moreover, the operations in these services are normally automated in real-time; therefore, the access control must also be adapted dynamically in real-time. Existing approaches for IoT data marketplaces, such as [5], [6] provide mechanisms to share data; however, they have not addressed the challenges mentioned above for access control based on dynamic IoT context. Indeed, IoT context sharing platforms have been surveyed in detail in [7]. However, no existing approaches provide edge-based enforcement solutions for controlling data sharing driven by dynamic contexts. To the best of our knowledge, no previous access control framework supports dynamic context IoT data sharing for cross-sector IoT smart services.

B. Our work and contributions

This work proposes a framework that allows dynamic context-driven IoT data sharing at the Edge based on contract agreements changed at runtime. We design and develop a set of Edge-based dynamic context policies representing real-world data sharing scenarios with access control changes at runtime. Furthermore, these dynamic policies can be enforced and updated in (near) real-time based on the context of the data. With this policy enforcement framework, we enable Edge-based IoT Data Hubs, which allow real-time data to be collected from various sources regardless of the protocols and data formats, to share data based on context-specific contracts.

Our main contributions for contract-based dynamic context IoT data sharing on the Edge are:

- Based on data contracts between the Hub provider and tenants, initialized and centrally stored on the Cloud, tenant-specific access control policies will be enforced on the edge level;
- The access control policies are context-driven to support data sharing in flexible ways according to application-

level IoT contexts, which are dynamically updated by context-sensing services on the Edge;
- Our Edge-based framework can enforce tenant-specific access control policies to the data being shared dynamically according to IoT context changes in the Edge and contract changes in the Cloud.

In the remainder of this paper: We provide a motivating example in Section II. Section III presents our approach, which is evaluated in Section IV. We discuss related work in Section V. Finally, we give our conclusions in Section VI.

II. A Motivating Example

Let us take the position of the IoT Data Hubs service provider X, which owns Edge-based IoT Data Hubs (edge servers) deployments distributed in a smart city. Note that Edge servers can be powerful[1], although, in the Edge, we cannot have elastic resources as much as we want. Each Hub deployment is close to IoT devices, the primary data sources considered in this work, and other data systems that are not necessarily owned by X, but instead owned by different data providers. Each data provider P has its own fleet of IoT devices and systems but offers part or all of its data to the IoT Data Hubs. Here, we assume that X has made agreements with any P that wants to sell their IoT data via the X's system and "onboard" P's IoT data streams to X's Data Hubs. P trusts X to find the best way to sell data value through monetary means and social incentives (e.g., support emergencies and health safety enforcement). Then, many IoT data consumers want to use P's IoT data for their businesses. Each consumer C subscribes to the data they want using e-contracts in the Cloud-based system of X where all the IoT data streams from different Hubs are listed for the subscription. Such a hub can be an essential part of the ecosystem of sensing as a service [2] and data marketplaces [8]. Currently, how X, P, and potential consumers establish their market relationships have been discussed in the literature in terms of marketplaces [1]. However, the issues of dynamic access control are still open and challenging.

Now, we take video camera data as an example. Let us assume that camera systems from a street have been "onboarded" to make live video data available to the Hub in that street. Such camera systems can be owned by shopping malls, parking houses, city administration, or other stakeholders. Consumers can sign contracts with X and any P to subscribe to their data but not necessarily get video images because of different constraints in the contracts. The city administration can subscribe to the X's system to receive the number of people gathering in a specific location, which is of value during the COVID lock-down period. The local police will be alerted if more than a certain number of people gather during the COVID lock-down period. For example, if more than twenty people were gathering simultaneously in one place, the local police might get access to the live video images on the Hub. If less than twenty people, only the number of people gathering, not live video images, is available to them. This means the context here is the number of people gathering that is changing dynamically. The privacy rule of the city only allows the police to watch live camera images on the Hub if the COVID lock-down enforcement (no more than twenty people can gather in one place) can override privacy rights. Note that the camera video images are not transferred outside of the Hub. The police department may also want to automatically access the camera system to recognize wanted criminals using AI. X supports such cases by allowing some tenant (AI) applications to be deployed in X's Hub to get close access to data.

The camera system is only one specific example of IoT data that can be shared. We can also name other IoT data resources from different sectors that can also be shared. This means that direct sharing one-to-many is not suitable for realizing the vision of cross-sector smart services where different data sources may even be combined for new services models. There are also security and privacy challenges that must be addressed. Data and data processing must stay as close to the data source as possible for privacy and performance. For an example of the camera system, each geographical area needs a local Edge server to receive all data from the cameras in that specific area and control how camera data can be shared for different stakeholders directly from the Edge.

Intelligent Data Hubs can infer the context of data and can even provide AI data filtering and pre-processing before sending data to the end-users. It means that multiple data sources can be merged to become new, high-quality data sources.

III. Edge-based Dynamic Context Policies

The key challenge we address in this work is how to define and enforce data-sharing policies based on dynamic contexts. To this end, we first present our proposed service model and its architecture in this section. We then introduce our data-sharing contract specification and its corresponding enforcement policies. Based on the policy specification, we will demonstrate how the system and application-level contexts can be used for dynamic policy enforcement.

A. Edge-based Data Sharing as a Service Model

In our main service model, the Hub owner has made agreements and integration in advance with data providers so that the Hubs have capabilities to extract and sense certain characteristics of the data from providers. We note that such a hub-based model exists and is common in practice [9], [10] Moreover, the Hub has already pre-calculated the max load it supports for IoT data providers. This means that IoT data providers, once onboarded to the IoT Hub, must have already signed a contract with a technical detail agreement with the Hub, e.g., the range of data and the frequency being sent. On the other hand, the Hub has the ability to monitor and detect the violations of data sent by IoT providers in a way that malicious attackers cannot cause the Hub any harm.

[1]e.g., see https://www.ibm.com/docs/en/cloud-private/3.2.0?topic=servers-preparing-install-edge-computing

Based on these assumptions, our service model mainly focuses on how the Hub can control data sharing as a service for different data consumers (called tenants) based on dynamic contexts. Context-based policies define contracts between the Hub and tenants. Edge-based IoT data sharing as-a-service carries out the enforcement of such policies. The conditions within contracts, and the contract itself, can be changed based on *contexts*. The Hub can *sense* data in the Hub to extract relevant *generic context and application-specific context* based on the predefined metadata and data from monitoring/sampling techniques for each data source. This is based on two reasons. First, modern IoT Data Hubs can leverage AI/ML techniques to infer specific data-specific contexts for different applications through plugin and sampling techniques. Second, intelligent IoT data sources can share metadata about provided data to the Hub. The critical business model is to allow tenants from cross-sectors to get data (driven by IoT contexts) from various data sources managed by the Hub.

```
{  "tenant":"tenant-1",
   "contracts":[
        {"Name":"Allow streaming camera based on people
             count threshold OR violence detected",
           "Action":[ "subscribe"],
           "Effect":"Allow",
           "Resource":["/smartcity/camera/stream/country_x
                /city_y/store_z/city_surveillance"],
           "Conditions": {
               "AnyOf":[
                  {"object":"people_count","location":"
                       store_z",
                   "max_5mins": {"gt": 30}},
                   {"object":"violence_detection","
                        location":"store_z",
                   "violence_last_1mins": {"gt": 0}}
               ],
               "All":[
                  {"object":"data_amount","protocol":"mqtt
                       ",
                   "lasthour_mb":{"lt": 3000}},
               ]...
}
```

Listing 1. Tenant1's contract example

B. System Architecture

Fig. 1 depicts the architectural design for an Edge-based IoT Data Hub. A single Edge IoT Data Hub has *Data Service* that allows data providers to publish their data and enables tenants to subscribe to the provided data. A tenant should use a portal or an API to subscribe to available data and agree to a set of conditions, establishing data contracts to access the data. In this work, to simplify the process, we assume that the contract has been predefined using a template described in §III-C, which includes dynamic parameters based on context information from the IoT data source. A set of such policy templates is stored in a *Contract DB*. Once a contract is in place, our framework will transform it into a low-level tenant-specific policy format and send the low-level policy to the *Policy Decision Point (PDP)* in the Edge. The Policy Decision Point will trigger the policy enforcement process to monitor the tenant's access to the subscribed data. Our framework contains the following vital components:

- *Data Service* gets data from multiple data sources and provides data access for tenants based on the authorization from the *PDP*.
- *PDP* receives the tenant's request from the Data Service and returns the corresponding permission based on tenant-policy policies and runtime contexts.
- *Context Sensing*: monitors the real-time data to extract the context value based on metadata from a data source. The sensed context data from a hub is synchronized with our central messaging system, deployed on the Cloud, to update the policy in real-time.
- *Messaging System*: synchronizes IoT context data and policies updates between the Edge-Data Hubs and the Cloud-based contracts management system. This scalable messaging system serves many synchronization tasks between multiple Hubs and the Cloud.
- *Context Interpreter*: receives context data from Edge and looks up the tenant contracts (from *Contract DB*) that depend upon the context data to generate the *tenant-specific context data*. Note that the *tenant-specific context data* is calculated on the context data from Edge (even sent from multiple Hubs) according to the tenant contract(s). For each tenant having contract(s) affected by the received context data, *tenant-specific context data* and the low-level tenant-specific policy (in Rego format [11]) generated by the policy generation component are synchronized back to the corresponding PDP(s) for authorization decisions.
- *Tenant-specific Policy Generation*: transforms high-level policies defined as contracts in the Contract DB to the low-level policy format (we use the Rego policy format) for enforcement.
- *Edge Synchronizer*: receives the low-level policy and the tenant-specific context data from the messaging service and synchronizes the updated values to the *PDP* for enforcement.

C. Data Contracts and Enforcement Policies

1) Contracts: There are many models for data contracts of IoT services in the literature. In our work, we leverage and extend the common Identify and Access Management (IAM) concept [12] to define the contract model with the following aspects.

- *Principal/Id:* represents the identification of the tenant.
- *Name:* describes the overview of the contract.
- *Effect:* specifies the permission of a given resource such as Allow and Deny.
- *Resource:* indicates the list of the IoT resource ID that the contract affects.
- *Actions:* specifies the tenant requests such as publish and subscribe.
- *Conditions:* specifies constraints based on some attributes that are dependent on runtime inputs such as system-wide or application-level contexts. We support two types of conditions: *AnyOf* and *All* that can be combined to include multiple specific conditions based on the attributes.

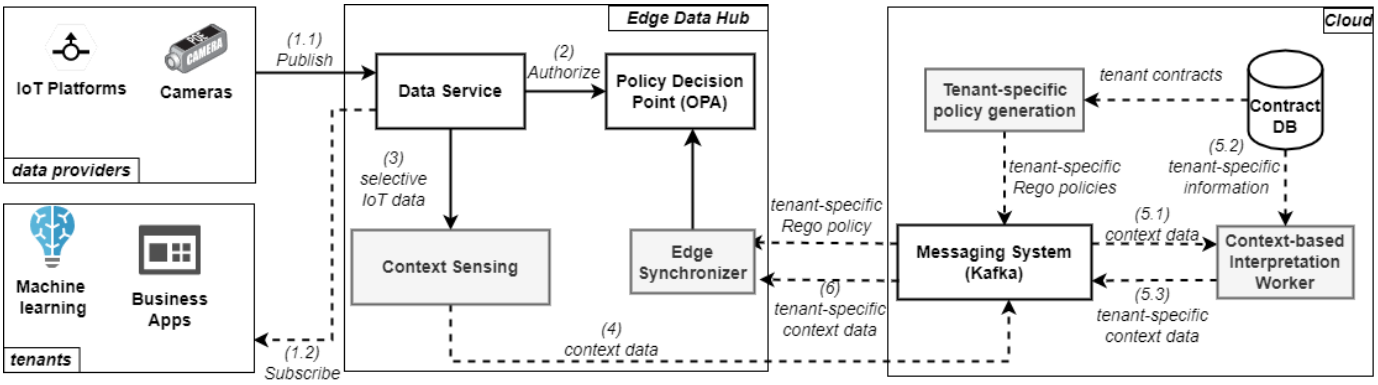

Fig. 1. Our proposed framework with an Edge IoT Data Hub.

```
{  "tenant":"tenant-2",
   "contracts":[
        {"Name":"Allow streaming camera when fire alarm
             triggered",
         "Action":[ "subscribe"],
         "Effect":"Allow",
         "Resource":["/smartcity/camera/stream/country_x
              /city_y/store_z/city_surveillance"],
         "Conditions": {
             "AnyOf":[ {"object":"fire_alarmA","location
                  ":"store_z",
                 "alarm_last_5mins": {"gt": 0}}
             ],
             "All":[
                {"object":"data_amount","protocol":"mqtt
                     ",
                 "lasthour_mb":{"lt": 2000}},
             ]...
}
```

Listing 2. Tenant2's contract example

We use JSON to define the contract model. Listing 1 illustrates a shortened example of a tenant-specific contract (Tenant 1's contract) in which city surveillance video data is only visible to Tenant 1's users if more than 30 people show up at the same time or any violent behaviors are detected at the last minute [13]. For another tenant (Tenant 2, e.g., a fire department), the contract for viewing city surveillance video data is defined for rescue missions only when fire alarm A triggered (Listing 2).

2) Tenant-specific enforcement policy: Once the Hub owner establishes a contract with a tenant, a specific instance of the contract for the tenant is generated and stored in our Contract DB. The contracts in JSON format will be transformed into a low-level policy format that can be used for the PDP in the Hub to make decisions per data access request. There are two main approaches of the logic code for checking the permission and making the decision at a PDP. The first approach is to have an all-in-one code to evaluate the permission based on the contracts of all tenants. However, this approach might require a complex implementation that can cover all cases that can happen in the contracts. The second approach is to generate logic code for each tenant from the contract information, making the code has smaller footprints. We have chosen the second approach because of the benefits of lightweight decision points, such as the easiness to change the logic code and the speed when evaluating the permission.

To demonstrate the second approach, we use OPA [11] for policy enforcement with runtime parameters and leverage its Rego policy language to define the policy. In OPA, the Rego policy is employed to evaluate the runtime context. While the runtime data changes frequently, the Rego policy is created during the initialized process and only re-created/updated when the contract is updated. Listing 3 shows the Rego policy example used in our framework. This Rego policy is transformed from and corresponds with the contract described in Listing 1.

The Rego policy must be updated to the PDP, e.g., implemented using OPA in the Hub when a tenant contract is established or changed. Therefore, we need an automated process to synchronize a tenant contract to the PDP as a Rego policy. To this end, we have developed a method to transform the contract defined in the management side (in JSON presented previously) on the Cloud to the logic code in the decision points (Rego file) on the Hubs. The generation process is triggered independently per tenant at the first deployment time or when the tenant's contract changes at any point in time. We describe this transformation process below.

3) Generation and synchronization of tenant-specific enforcement policy: The process involves four components in both Cloud and the Hub (Fig. 1), including the *Policy Generator* that contains the primary logic to transform the low-level policy to the Rego policy, the *Messaging System* transmitting the generated Rego policy from the Cloud to the Edge Hub, the *Edge Synchronizer* listening to the corresponding event from the Messaging System to update the Rego policy to the *PDP*, i.e., the *OPA* component.

The *Policy Generator* component is the most important and challenging in this flow. Typically, a tenant has multiple contracts that define their permissions to various IoT data sources in different context-based conditions. We implement a set of rules as a template engine that supports converting the meta-model and the context structure into a Rego policy. The template engine in the *Policy Generator* component is

```
package app.iot

default allow = false
default whitelist = false
allow{  whitelist }

whitelist {
 #Validate-Action
 some i
 actions := ["subscribe"]
 actions[i] == input.action

 #Validate Resource
 some j
 resources := ["/smartcity/camera/stream/country_x/city_y/
      store_z/city_surveillance"]
 regex.match(resources[j], input.topic)

 #Validate Condition AnyOf
 var_any_c_0 := data["context_data"]["people_count"]["
      store_z"]["max_5mins"]  > 30

  var_any_c_1 := data["context_data"]["violence_detection"]
       ["store_z"]["violence_last_1mins"]  > 0

 conditions_anyof := [ var_any_c_0,var_any_c_1]
 some k
 conditions_anyof[k] == true

 #Validate Condition AllOf
  var_all_c_0 := data["context_data"]["data_amount"]["mqtt"
       ]["lasthour_mb"]  < 3000
 conditions_allof := [ var_all_c_0]
 conditions_allof_negative := {value | value =
      conditions_allof[_]; value == false}
 count(conditions_allof_negative) == 0
}
```

Listing 3. A tenant-specific policy example in Rego language transformed from the contract in Listing 1.

extensible with new conditions and context information; thus, our system can handle a new type of contract without changing the entire framework. The *Policy Generator* transforms each tenant contract separately into a final Rego file (*tenant-specific policy*) using the white-listing and black-listing approach. This approach requires an operation, i.e., a resource access request must meet the action, the resource, and the conditions specified in the policy before checking the decision among allow, deny, and neutral. Beyond that, the final decision is made after going through all policies. Resource access from a tenant is only authorized if at least one allowed policy matches while no one is detected.

Because there are many Hubs connected to the Cloud system, we need a scalable mechanism to synchronize data contexts and enforcement policies between the Cloud and the Edge. As a demonstration, we use the Apache Kafka framework [14] to implement the *Messaging System* for synchronization. We note that the Kafka framework might not be sufficient in large-scale edges; however, we assume that the Hub with context sensing is a part of the network where a Kafka consumer/producer will be executed. Thus, it can reduce the technical challenges due to network configuration and consensus protocols between Edge and Cloud. The *Policy Generator* component will send the generated tenant-specific policy to the Messaging System as a Kafka topic with the payload of the Rego policy and the tenant's identification.

```
"All":
[{"object":"data_amount","protocol":"mqtt","lasthour_mb":{
     "lt": 3000}},
{"object":"data_amount","protocol":"mqtt","last24hour_mb":
     {"lt": 30000}}]
```

Listing 4. Volume-based policy configuration.

The *Messaging System* will send the message topic to the corresponding *Edge Synchronizer* in the Edge for updating the Rego policies in the *PDP* for that specific tenant.

D. Context Sensing

Context sensing is a component that generates context data based on metadata and real-time data of IoT data sources. Our framework supports two primary IoT contexts: system and application-level contexts.

1) System-wide contexts: This context category is about typical, well-understood attributes of data usage such as data amount, times, and duration. These contexts are real-time but independent of the data values. For example, the daily volume limit to access camera data is a system-wide context. We define these system-wide contexts in the high-level policy specification for tenant contracts. Listing 4 is an example of a system-wide context policy with a volume-based context that limits hourly data access to $3,000MB$ and daily access to $30,000MB$. The context sensing component will update the context data based on relevant system events to synchronize with the OPA enforcement component.

2) Application-level contexts: There are different types of real-time data from a data source that can generate contexts for the policies. These include raw data from sensors such as temperature or data generated by AI services from IoT devices such as the number of people or an accident captured in a camera. We also leverage these application-specific contexts for our policy specification, similar to the system-wide contexts. For example, Listing 5 illustrates such an application-specific context policy that involves a threshold policy with the number of people.

```
"AnyOf":
[{"object":"people_count","location":"store_z","max_5mins":
     {"gt": 35}},
{"object":"fire_alarm","location":"store_z","alarms_last_5
     mins":{"gt": 0}}],
```

Listing 5. An example of application-specific context: people count.

There are two main approaches to implementing context sensing. The first approach is to allow both raw and context data to go through a proxy or gatekeeper as a context sensing before forwarding it to end-users. The second approach is to fan out two data flows, one used for context-sensing, the other data flow going to be forwarded to the end-users via a Virtual Hub if the *PDP* approves it. In this work, we employ the second approach by developing a *plugin* API for each type of context data. By doing so, we can add new *plugin*s to generate new context data when a new context type from a new data source is added.

A common principle of a context-sensing *plugin* in the *Context Sensing* component is to listen to the fan-out data flow in the Data Gateway to capture the selective IoT data used as context data. Depending on the context, either system-wide or application-level context, the *plugin* can use an aggregation method (e.g., average, sum, max, min), or employ an AI component (e.g., for people count) to generate runtime context data as a context-based variable described above. The *Context Sensing* component will send these runtime generated values to *Messaging System* on the Cloud as Kafka topics. Upon receiving a context data message, the *Messaging System* will trigger *Context-based Integration Worker* component to generate the tenant-specific context data based on tenant contracts of relevance. The tenant-specific context data are synchronized back to the Edge for enforcing the corresponding context-based policies.

E. Tenant-specific Context Data Generation& Synchronization

Fig. 2 depicts the entire data flow and process of using context data from a data source to generate and update the corresponding tenant-specific context data for runtime policy enforcement.

In the first step, the *Context Sensing* component listens to the Data Service to capture the selective IoT data that is going to be used as context data. Then, we develop aggregation methods (i.e., average, sum, max, min) for some given windows such as 5 minutes, 15 minutes, last hour, and last 24 hours to create context-based variables described in Section III-D. Finally, these variables are transmitted to a communication channel via a Kafka topic for delivering the context data from the Edge Hubs to the Cloud. Table I shows the samples of the existing window-specific variables that are selected to be used as the tenant-specific context variables later on. This list can be extended when we introduce new contexts. We implement the method manually; however, it can be replaced or integrated with a standard other solution such as Apache Flink.

The vital component in this process is the *Context-based Interpretation Worker* that listens for context data from the *Messaging System* to generate tenant-specific context data based on tenant contracts. The *Context-based Interpretation Worker* listens on the corresponding Kafka topic to create a global context database. Based on each tenant contract of relevance, only selected context variables are used to generate an updated version of the tenant-specific context data, as described in the pseudo-code in Algorithm 1.

Finally, each generated tenant-specific context data, together with tenant identification, will be wrapped in a payload as a Kafka topic to be sent back to the Data Hub via the *Edge Synchronizer* component in the Hub. Similar to the policy synchronization process presented previously, this synchronizer also captures related Kafka topics for tenant-specific context data and updates them to the *PDP* for tenant-specific policy enforcement. Our demonstrated PDP implementation using OPA needs both Rego policy (static) and inputs from this context data (runtime) for each tenant to decide the access permission, e.g., allow or deny of a given access request.

```
Data: Context data, Contracts
Result: Tenant-specific context data are selected for all
        tenants
forall tenant's contracts in Contracts do
    read current contract;
    initialize tenant_context;
    forall policy rows in current contract do
        read current contract policy row;
        forall context variables in the policy row do
            read current context variable;
            if context variable does not exist in
             tenant_context then
                extract context variable value from
                 Context data;
                append the read current context variable
                 and it's value to tenant_context;
            end
        end
    end
end
```

Algorithm 1: Illustration of the Context-based Interpretation Worker component.

IV. Evaluation

In this section, we present our proof of concept implementation and the experimental results to demonstrate the soundness and effectiveness of our proposed framework.

A. Prototype Implementation

In the previous section, we have partly shown the implementation choices for our framework in Fig. 1. For the Data Service at the Hub, we support and implement the MQTT, RTMP, and HTTP Live Streaming protocols. We use MongoDB to store our contracts. We use *Node.js* to implement the *Tenant-specific Policy Generation* and *Context-based Interpretation Worker* components. In addition, we employ PugJS [15] as a template engine that uses static predefined template files to generate the final Rego files within this component. The static template files contain static strings and variables, including many functions, using their own defined markup language. The prototype is available on GitHub[2].

B. Experiment Settings

We use a real-world IoT sharing scenario mentioned in the motivating example in §II to evaluate the effectiveness of our proposed enforcement framework. In particular, we validate the following significant factors in our system: i) tenant contracts are correctly transformed to enforcement policy; ii) tenant-specific context data is captured and updated in the enforcement system at runtime; iii) dynamic policies with runtime contexts are soundly enforced for each tenant.

To this end, we consider a store with a surveillance camera illustrated in Fig. 3. In this system, the store camera video can only be viewed by the surveillance team within that store. The

[2]The full link is removed for double-blind review

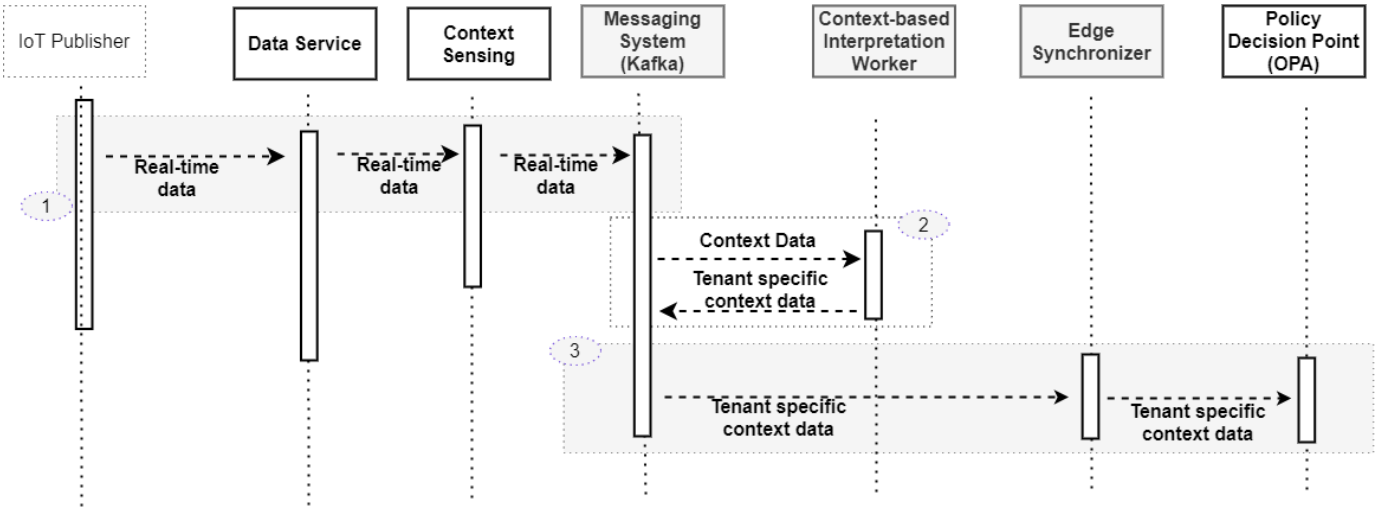

Fig. 2. Sequence diagram of tenant-specific context data generation

TABLE I
DATA HUB CONTEXTS VARIABLES

Attribute	Primary Index	Context variables	Description
data_amount	rtmp mqtt	lasthour_mb , last24hour_mb lasthour_mb , last24hour_mb	The data throughtput accumulated.
people_count	store_z	max_5mins , min_5mins, avg_5mins ,max_15mins min_15mins, avg_15mins	The aggregated people count.
fire_alarm	store_z	alarm_last_5mins alarm_last_10mins alarm_last_15mins	Alarms in the last X minutes
violence_detection	store_z	violence_last_1min violence_last_5mins violence_last_15mins	Violences in the last X minutes

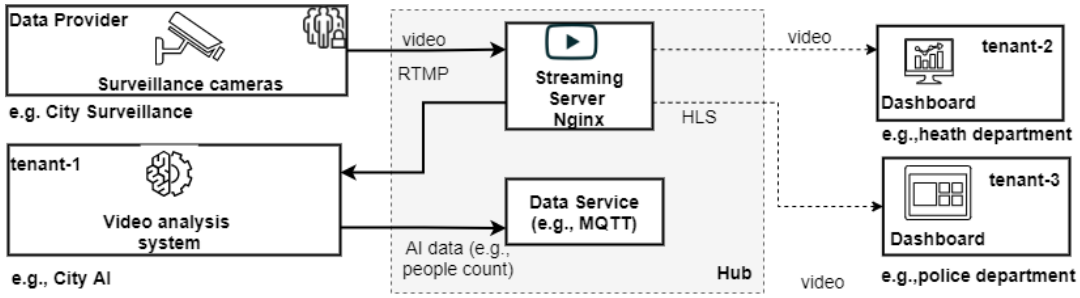

Fig. 3. Evaluation scenario of a camera system in a store department

store will be a *data provider* in our system with the data source of the video publishing to the Hub using RTMP protocol. This scenario has three tenants, including an AI company, a local police department, and a local health department. These tenants connect to the Data Hub to subscribe to the data from the store data provider. Each tenant establishes a different contract with specific runtime contexts. For example, the AI company (tenant-1) can only use video data to perform AI analysis during office hours. The health department (tenant-2) can only view the camera stream from the Data Hub if the number of people in the video is, e.g., more than or equals 30 at a time. Finally, the police department (tenant-3) can only view the camera stream from the Data Hub if the number of people in the video is, e.g., more than or equal to 15 in the last 5 minutes. In this example, tenant-1 has a dynamic contract with a system-wide context, i.e., office hours; tenant-2 has a dynamic contract with an application-level context, i.e., the number of people; and the contract of tenant-3 consists of both application-level context (people count) and system-wide context (last 5 minutes).

We have set up the Cloud and the Hub infrastructure on Amazon AWS with m5.xlarge instances using Docker. All cloud-related services, such as the Kafka cluster and database (MongoDB) cluster, are deployed in an EC2 instance to simulate the Cloud infrastructure. On the other hand, we leverage four `m5.large` instances to set up the MQTT clusters and hub-related services.

C. Experiments and Validation

To perform our experiments, we simulate the scenario described in §II by developing corresponding applications representing the data providers and tenants in the system. In addition to the key components developed and described in §III, we have developed context sensing plugins, including an AI-based plugin extracting people count (application-level context) from video and an aggregation service generating system contexts such as average, sum, max, and min. We deploy the applications and service components to an `m5.xlarge` instance on Amazon using Docker compose. In the Cloud, we use a Docker-compose cluster to deploy the *Messaging System* (Kafka), and another Docker-compose cluster for *Context-based Interpretation Worker* and *Tenant-specific Policy Generation*.

We performed the experiments on two Raspberry PIs with an attached camera to validate runtime performance and functionalities. To validate runtime contexts such as people count, we need real data; therefore, we use a video as a data source in our system. We have validated the following features of our framework.

1) Dynamic policy enforcement with runtime context data: We have tested our system in two levels of data amount: 1) a recorded video as a data source and 2) a simulation of 20,000 sensors that produce one message per second. We observed that runtime context data from our context sensing component is generated correctly and synchronized in real-time with the PDP through our cloud-based services to enforce the dynamic policies properly.

In particular, in our experiments, the health department can watch the video through the Data Hub only when the number of people detected in the video is 30 or greater. The video permission is denied/revoked if the threshold does not reach. Similarly, according to the contract, the police department (tenant-3) can only access the video if the number of people in the video in the last 5 minutes is equal to or greater than 15. Our experiment showed that the runtime context data of people count within the last 5 minutes are adequately generated and updated; as a result, this tenant has access permission when the context variables reach the threshold.

2) Performance and Overhead: To evaluate the performance and overhead of our framework, we set up two 4-instance MQTT clusters. The first cluster has only the authentication mechanism. The second one integrates with our policy enforcement mechanism and the OPA module for authorization per events of publishing a message, subscribing to a topic, and forwarding a message to a subscriber. We also set up an instance for benchmarking two clusters using an MQTT stress testing tool. To compare the overhead in two clusters, we use the tool to publish to both clusters with the frequency of 20,000 messages per second by simulating 20,000 MQTT clients publishing one message per second in a minute. On the other hand, 20,000 MQTT clients subscribe to fetch all messages. It is noticeable that the publishing rate is static, and both clusters can handle that rate. We measure the overhead by evaluating the receiving rate. The median receiving rate times without and with our policy enforcement are 82,969 messages per second and 70,801 messages per second, respectively, resulting in 17.2% in the overhead of the enforcement mechanism, as described in Table III. We also notice some outliers in the result produced by the cluster with OPA integrated. In our observation, the overhead is acceptable for the cost of authorizing the tenants on every single request.

We also performed experiments to evaluate the response times of our system when a policy or a context value is changed at runtime. For the policy change, we measured the times for the policy is updated to the database and when it is propagated to the OPA module. We also performed the test 10 times to get the average numbers. For rendering the Rego policies update in the database, it responds almost immediately without delay. However, the average time to deliver the policies over the network to update to the OPA takes around 18.5 ms.

In the third experiment, we evaluate how the system scale with a larger number of tenants when a context is changed. We tested with 100 tenants in propagating the context value from the data source to the OPA policy module. In this context-change scenario, we set a timer starting when the context changes until it is evaluated and published to the OPA. Since a context change may affect many tenants, we measured the propagated time of the first and the last tenants among 100 of them. Table II describes the times we measured. On average, for over 100 tenants, it takes about 1.64 ms to process and update the new context to an OPA instance.

D. Discussions on Edge-based services and policies evolution

1) Policies evolution: Our proposed framework can support the evolution of edge-based services and policies enforcement. Whenever a new IoT data provider is partnering with the IoT Hub, their data schema and protocols are in synchronization with the IoT Hub system. In other words, the onboarding process for new IoT data providers is done in the background in which the IoT data provider can work closely with the IoT Hub provider to make sure their data are adequately available to the Data Hub. Our focus is on the process of new tenants subscribing to new data sources and can access new data according to tenant-specific dynamic contexts. Let us consider an updated scenario when Tenant 1 is updating its contract for its system can access surveillance cameras' data in case of fire. For this existing tenant, the additional context is the status of the fire alarm triggered. This means that there will be an updated policy in the contract database that extends the existing policy with the newly added context based on the fire alarm status (Line 4, Listing 6).

TABLE II
CONTEXT CHANGE PROPAGATION TIME (IN MS)

	Earliest Receiving Time	Last Receiving Time	Extra propagation time for a new tenant
Maximum	103	275	2.3
Minimum	36	155	1.05
Average	46	210	1.64

TABLE III
MEDIAN RECEIVING RATE (MPS) REPORT IN TWO MQTT SYSTEMS

Test cases With OPA	Test cases Without OPA
64514	77923
93113	79480
89211	82766
88202	85230
71152	85353
92630	78365
67992	84292
63998	86298
73503	84390
36504	78594
63881	88390
44916	84552
Average: 70801	Average: 82969

```
"AnyOf":
[{"object":"people_count","location":"store_z","max_5mins":
    {"gt": 30}},
{"object":"violence_detection","location":"store_z","
    violence_last_1mins": {"gt": 0}}
{"object":"fire_alarmA","location":"store_z","alarms_last_5
    mins":{"gt": 0}}],
```

Listing 6. An example of an updated policy.

The rest of the enforcement is propagated from the new contract in the Cloud for generating the updated Rego policy, which is then synchronized into the OPA engine in the Edge.

2) Advanced contexts and trustable data sharing: Nowadays, there are many advanced and dynamic contexts available in practice that can be deployed and integrated into our proposed framework. For example, we can have many dynamic contexts in emergency situations such as fire alarms, floods, and traffic accidents. These contexts can also be imported from/provided by machine learning applications (in today's deployment, such applications can be executed within the IoT devices, e.g., smart cameras and drones).

Especially, our framework can lay a foundation for end-to-end dynamic industrial data sharing with traceability, trust, and security. In advanced scenarios where supply chain stakeholders call for an innovative way of trustable data sharing across companies, the dynamic contexts of data sharing (used in OPA for authorization) together with other key (business critical) transaction data can be stored in a traceability layer. Distributed ledger/blockchain and smart contract technologies will be used to ensure traceability and integrity of the data, enabling trustworthy and secure data exchanges. Our framework can be extended with such a traceability layer by implementing a secure proxy/connector integrated with the data service to allow all stakeholders to contribute to the same ledger/blockchain of the supply chain/ecosystem.

Our framework uses OPA for policy decisions, which is very popular in the industry, including OPA for the Cloud and network. Therefore, our proposed framework is compatible and can be integrated into existing industrial ecosystems.

V. RELATED WORK

The sensing as a service model presented in [2] shows the vision of utilizing (cross-sector) multi-parties IoT (data) resources to accommodate large numbers of consumers, which is making more sense in smart cities nowadays. However, it is still a long way to fulfill the vision, with few work that have barely made it any further. Our work on context-specific policy access controls can contribute to such sensing-as-a-service scenarios.

In [16], the authors present a context-aware security (conceptual) framework with the context management that makes use of context information and access control policies (e.g., XACML [17]) for secure data sharing decisions. They propose a secure data sharing mechanism for groups of smart objects according to contextual data. Their framework however does not address edge-based enforcement solutions for controlling cross-sector data sharing driven by dynamic contexts. In the same direction, the authors of [18] propose an edge-centred context sharing architecture. This study described an architecture that makes security decisions based on shared context information from multiple domains. However, this architecture is only a conceptual model because [18] is without any concrete enforcement solutions.

D. Preuveneers et al.[19] make use of the UMA OAuth 2.0 extension which extends OAuth 2.0 from only authorizing applications to access on a subject's behalf (person-to-self), to allowing person-to-party (person-to-person and person-to-organization) authorization by delegating access to third parties such as other individuals or organizations[20]. To define and enforce access policies in [19], the policy execution engine Open Policy Agent (OPA) [11] is used. OPA uses a JSON based policy language named *Rego* that is efficient in terms of parsing and policy size[19]. Even though the approach in [19] follows the same direction with our work, and also using OPA for the policy enforcement engine, it does not yet show the support for utilizing (cross-sector) multi-parties IoT (data) resources to accommodate large numbers of consumers. Nguyen et al.[4] uses the gatekeeper design pattern that deploys a policy enforcement instance for each tenant application in the Edge. The gatekeeper works as a gateway

by intercepting all incoming requests, decouples access control logic from the application's business logic, and enables real-time policy updates by redeploying the gatekeeper. However, no dynamic IoT context is supported in their approach.

There are quite some IoT Data Marketplace approaches such as [6], [21], [22] that focus on using blockchain technology and/or smart contracts to enable IoT data sharing. The main point in [6], [21] is about the application of blockchain/smart contracts to IoT data sharing for no trusted parties. However, these approaches do not touch upon IoT data sharing for trusted parties where data being shared are based on contracts via an intermediary and with dynamic contexts that can bring great values to the involved parties. The approach in [21] also employs an efficient proxy re-encryption mechanism, ensuring that the data is only visible to the data owner and the person present in the contract. In a similar direction, by employing the blockchain as an auditable and distributed access control layer to the data layer, the authors of [22] enable secure data sharing and resilient access control management.

VI. Conclusions and Future Work

This paper presents a framework that allows dynamic context IoT data sharing on the Edge based on contract agreements for multiple parties. This is a significant step forward to realizing the vision of utilizing (cross-sector) multi-parties IoT (data) resources to accommodate large numbers of consumers. We have tackled one of the most challenging problems in this direction by proposing an Edge-based security enforcement framework that enables multi-parties IoT (data) resources to be shared based on contracts, especially with dynamic IoT contexts. Our proof-of-concept prototype has shown how the proposed framework can be implemented and properly enforced dynamic context-based policies for accessing IoT data on the Edge for different tenants.

Future work includes doing more experiments with the scalability of our framework implementation for possible refinement. The framework can be developed further to even allow the deployment of tenant applications to be executed inside the Hubs. This means that the Hub controls IoT data being shared with tenants (externally) and tenant applications being deployed and executed right in the Edge for better real-time data usage. Furthermore, our proposed framework can also be combined with Data Marketplace approaches using blockchain and/or smart contracts technology to record the provenance of context-based IoT data sharing transactions. This way can fortify the trustworthiness of the framework for all the involved parties because the critical data of every transaction, such as application-level contexts, system-level contexts, and data quality, can be recorded securely by design.

Acknowledgments

The research leading to this publication has partially received funding from Novobi, LLC research contract M8J000, NSF EAGER award 2025234, and the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement 958363 (Dat4.ZERO).

References

[1] S. Jernigan, D. Kiron, and S. Ransbotham, "Data sharing and analytics are driving success with IoT," *MIT Sloan Management Review*, vol. 58, no. 1, 2016.

[2] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Sensing as a service model for smart cities supported by internet of things," *Transactions on Emerging Telecommunications Technologies*, vol. 25, no. 1, pp. 81–93, 2014.

[3] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the internet of things," *IEEE Access*, vol. 6, pp. 6900–6919, 2018.

[4] P. H. Nguyen, P. H. Phung, and H.-L. Truong, "A security policy enforcement framework for controlling iot tenant applications in the edge," in *Proceedings of the 8th International Conference on the Internet of Things*, ser. IOT '18. New York, NY, USA: Association for Computing Machinery, 2018.

[5] K. Mišura and M. Žagar, "Data marketplace for Internet of Things," in *2016 International Conference on Smart Systems and Technologies (SST)*, 2016, pp. 255–260.

[6] K. R. Özyilmaz, M. Doğan, and A. Yurdakul, "IDMoB: IoT Data Marketplace on Blockchain," in *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, 2018, pp. 11–19.

[7] E. de Matos, R. T. Tiburski, C. R. Moratelli, S. Johann Filho, L. A. Amaral, G. Ramachandran, B. Krishnamachari, and F. Hessel, "Context information sharing for the internet of things: A survey," *Computer Networks*, vol. 166, p. 106988, 2020.

[8] T.-D. Cao, T.-V. Pham, Q.-H. Vu, H.-L. Truong, D.-H. Le, and S. Dustdar, "Marsa: A marketplace for realtime human sensing data," *ACM Trans. Internet Technol.*, vol. 16, no. 3, may 2016.

[9] M. Blackstock and R. Lea, "Iot interoperability: A hub-based approach," in *2014 international conference on the internet of things (IOT)*. IEEE, 2014, pp. 79–84.

[10] M. Bajer, "Building an iot data hub with elasticsearch, logstash and kibana," in *2017 5th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*. IEEE, 2017, pp. 63–68.

[11] "Open policy agent," https://www.openpolicyagent.org/.

[12] I. Security, "Designing a modern IAM program for your business," https://www.ibm.com/downloads/cas/9YBEK41O, 2020, whitepaper.

[13] M. Ramzan, A. Abid, H. U. Khan, S. M. Awan, A. Ismail, M. Ahmed, M. Ilyas, and A. Mahmood, "A review on state-of-the-art violence detection techniques," *IEEE Access*, vol. 7, pp. 107 560–107 575, 2019.

[14] "Apache Kafka," https://kafka.apache.org/, (Accessed on 08/21/2021).

[15] "PugJS," https://pugjs.org/api/getting-started.html, (Accessed on 03/27/2021).

[16] J. L. Hernandez Ramos, J. B. Bernabe, and A. F. Skarmeta, "Managing context information for adaptive security in iot environments," in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops*, 2015, pp. 676–681.

[17] O. Standard, "extensible access control markup language (xacml) version 3.0," *A:(22 January 2013). URl: http://docs. oasis-open. org/xacml/3.0/xacml-3.0-core-spec-os-en. html*, 2013.

[18] E. de Matos, R. T. Tiburski, L. A. Amaral, and F. Hessel, "Providing context-aware security for iot environments through context sharing feature," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, 2018, pp. 1711–1715.

[19] D. Preuveneers and W. Joosen, "Towards multi-party policy-based access control in federations of cloud and edge microservices," in *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*, 2019, pp. 29–38.

[20] "User-Managed Access (UMA) 2.0 Grant for OAuth 2.0 Authorization," https://docs.kantarainitiative.org/uma/wg/rec-oauth-uma-grant-2.0.html.

[21] A. Manzoor, M. Liyanage, A. Braeke, S. S. Kanhere, and M. Ylianttila, "Blockchain based proxy re-encryption scheme for secure iot data sharing," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2019, pp. 99–103.

[22] H. Shafagh, L. Burkhalter, A. Hithnawi, and S. Duquennoy, "Towards blockchain-based auditable storage and sharing of iot data," in *Proceedings of the 2017 on Cloud Computing Security Workshop*, ser. CCSW '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 45–50.