ComDeX: a context-aware federated platform for IoT-enhanced communities

Nikolaos Papadakis, Georgios Bouloukakis, Kostas Magoutis

ComDeX: A Context-aware Federated Platform for IoT-enhanced Communities

Nikolaos Papadakis[1,2], Georgios Bouloukakis[2], Kostas Magoutis[1,3]
papadakni@ics.forth.gr,georgios.bouloukakis@telecom-sudparis.eu,magoutis@ics.forth.gr
[1]Institute of Computer Science (ICS), Foundation for Research and Technology - Hellas (FORTH), Greece
[2]Télécom SudParis, Institut Polytechnique de Paris, France
[3]Computer Science Department, University of Crete, Greece

ABSTRACT

This paper presents ComDeX, a context-aware federated architecture and IoT platform for enabling data exchange between IoT-enhanced communities. Today, such smart communities are highly heterogeneous and siloed as they can offer IoT applications and services only to their local community inhabitants. ComDeX uses property graphs to represent smart community entities and automatically maps them to context-aware publish/subscribe messages. Such messages can be discovered and exchanged between communities via a hierarchical federated topology and an advertisement-based mechanism. The ComDeX prototype is implemented using well-known IoT technologies such as MQTT and NGSI-LD. ComDeX is evaluated using a realistic smart port scenario and compared against different federation topologies. The experimental results demonstrate that our approach outperforms existing NGSI-LD solutions in realistic IoT scenarios with synthetically generated workloads, with low impact in larger deployments where the number of hops between brokers of the federation increases.

KEYWORDS

Smart Communities, Federation, Pub/Sub, IoT, Middleware

ACM Reference Format:
Nikolaos Papadakis[1,2], Georgios Bouloukakis[2], Kostas Magoutis[1,3]. 2023. ComDeX: A Context-aware Federated Platform for IoT-enhanced Communities. In *The 17th ACM International Conference on Distributed and Event-based Systems (DEBS '23), June 27–30, 2023, Neuchatel, Switzerland*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3583678.3596890

1 INTRODUCTION

The advent of the Internet of Things (IoT) has gradually transformed cities into intelligent communities (residential areas, universities, smart ports, smart hospitals, etc.) that have changed how people approach everyday activities. Smart communities offer important services such as smart governance and public safety, environmental monitoring, smart utilities, smart transportation, to name a few. To offer such services, IoT applications can be developed using open-source (Orion-LD, RabbitMQ), commercial (EvryThng, Google Cloud IoT), or visual programming tools (Node-RED, ThingsBoard, IFTTT). However, the above platforms expose APIs to provide access to IoT data of specific spaces (a city port, university building, etc) and IoT devices. Therefore, IoT applications created for specific smart spaces are not interoperable with IoT applications created for other smart spaces. This leads to vertical and siloed communities where information from IoT devices used by one smart community is not accessible to another smart community.

Smart communities often cover wide-scale areas (e.g., a university campus, an urban port). IoT applications in such communities may receive information from *many and diverse* IoT sources (e.g., motion sensors, cameras, climate sensors, smart meters, etc) spread in smart communities employing *different IoT platforms*. We call such applications ***widespread IoT applications***. To enable their development, various individuals/organizations find the need to access IoT data of different smart communities. Widespread applications can be spreading in scope over time, because of the continuous need to leverage ever more information from ubiquitous IoT devices. In one example, a city port may collect WiFi connectivity data in a management system to infer occupancy levels and provides them to its port-authority personnel via a "room finder" application. In case of an emergency, emergency responders (ERs) would like to obtain situational-awareness information (e.g., floor plans, occupancy levels, etc.) to their dashboards. However, to receive such data, ERs dashboards must exchange data with the port's community and automatically discover the port situation.

Exchanging data between smart communities requires the use of common data models, IoT protocols and APIs. This is unrealistic in today's IoT systems due to the lack of portable and extensible IoT applications, standard mechanisms for data dissemination and exchange between communities, and high-level data discovery (e.g., building-based static or dynamic properties) mechanisms. Additionally, community administrators must be supported with *selective-sharing* and *policy-driven* approaches when providing access to their community IoT data. Data sovereignty preservation approaches for advertising specific community information must be designed, along with high-level data dissemination, discovery and exchange mechanisms.

Existing state-of-the-art approaches have adopted **federated designs** for smart-city platforms such as MARGOT [26], Fogflow [12], Trustyfeer [21], ALMANAC [10] and CPaaS.io [14]. However, these mainly provide cross-domain interoperability and IoT resource discovery for *specific* communities/cities. An efficient federation solution should include scalable information dissemination through *distributed event-based routing* [8], which aims to minimize network and computational overhead when disseminating an event to a group of interested recipients. Such approaches can be leveraged as the software overlay of widespread IoT applications. The Space

DEBS '23, June 27–30, 2023, Neuchatel, Switzerland

ACM ISBN 979-8-4007-0122-1/23/06.
https://doi.org/10.1145/3583678.3596890

Broker [5] and SemIoTic [35] frameworks define smart space characteristics and contextual data such as spatial representation of spaces. However, both works have focused on a single smart space and do not offer a solution in a federation of smart communities/spaces.

This paper introduces ComDeX, a context-aware federated architecture and IoT platform for enabling data exchange between smart communities. ComDeX solves an *information integration problem* for enabling widespread IoT applications in smart cities by providing a novel underpinning architecture with context-aware data modeling and dissemination mechanism for cross-community data exchange. It uses property graphs to model data by including the communities' context (dynamic information coming from IoT devices), while scalable information dissemination mechanisms ensure *selective-sharing of IoT data*. Unlike previous research [5, 35], ComDeX addresses interoperability and data exchange across diverse smart communities, not just within a single smart space. With its scalable publish/subscribe architecture, ComDeX goes beyond traditional methods of creating context-unaware messages. It treats entity attributes as "first-class citizens" and allows for property/relationship-level querying/subscribing while preserving the entity as a whole. Like previous systems [11, 30, 31],ComDeX uses advertisements to disseminate data, but additionally showcases the location/broker where relevant data is published so that interested subscribers can find it directly without creating subscription tables across the broker nodes. This minimizes the number of messages required and preserves the privacy of community members by allowing data filtering before dissemination.

The main contributions of this paper are:

(1) A context-aware topic/type-based subscription and advertisement scheme for enabling data exchange in widespread IoT applications §4.
(2) A hybrid and hierarchical federation topology and algorithm for message routing between brokers of deployed widespread IoT applications, while also showcasing ways to support deployments with existing 3rd party IoT platforms §5.
(3) A prototype implementation of the ComDeX federated architecture using state-of-the-art technologies such as NGSI-LD, MQTT and existing broker implementations §6.
(4) An evaluation of the ComDeX prototype and topology using synthetic data for realistic communities in a smart port scenario validating the benefits of our overall approach §7.

We conclude this paper and describe future work in §8.

2 RELATED WORK

As smart communities mature, so do their complexity and overall requirements. Maintaining important aspects such as interoperability, context data discovery, high availability, and data privacy across diverse smart communities is a challenging endeavor. In the remainder of this section we compare ComDeX to previous research in a number of different research directions:

Federated and Interoperable Architectures. To enable cross-community collaboration, many initiatives have proposed *federated designs* for creating smart-city platforms. For example, MARGOT [26],Fogflow [12],Trustyfeer [21],ALMANAC [10],CPaaS.io [14], and WiseIoT [17] are smart-city platforms that provide methods for cross-domain communication, interoperability and IoT resource discovery, to support optimal, sustainable infrastructures for citizens. Interoperable wide-scale deployments often rely on cloud federation, context-broker federations, and service orchestration technologies. FogFlow introduces a fog computing based framework for designing and implementing city-scale IoT applications. Tricomi et al. [32], introduce a software-defined platform and use it to develop applications over a city infrastructure while dealing with numerous administrative domains through federated architectures. MARGOT [26] emphasizes on high availability disaster recovery (HADR) by exploiting caching capabilities in federated nodes. HADR scenarios are also supported in city-scale deployments by the Pradhan federated platform [23], which leverages the lightweight MQTT protocol aspect of supporting different QoS message-delivery modes. Trustyfeer [21] relies on federated cloud environments to improve the number of services exchanged by cloud providers that conform to SLAs. Most of the above smart community platforms are *limited in their openness and interoperability*, as they either lack service programming models, or specify a model solely based on their own private data model and interfaces.

The NGSI-LD standard [1] facilitates the open exchange and sharing of structured data among various parties and has been utilized in several federated smart city domains, such as the City-DataHub [18] project, an online platform that is under trial in several South Korean cities, providing a communication window for sharing various policies promoted by the public, and the Demeter [22] project, a large-scale deployment of farmer-driven, interoperable smart farming-IoT based platforms. While a promising standard, NGSI-LD is *limited in its support for federation and forwarding*. ComDeX leverages NGSI-LD to achieve openness, using property graphs combined with Semantic Web domain models and offering ways for stakeholders to choose what kind of information they wish to expose. ComDeX improves over other NGSI-LD solutions which suffer from limited federation and forwarding support, by proposing a novel context-aware data dissemination scheme.

Scalable Pub/Sub Architectures. Publish/subscribe middleware architectures [19] are the core technology for asynchronous and cross-platform communication of IoT applications and platforms, used in the distributed deployment of many real-time applications and high-speed data dissemination schemes. A key technology for *scalable information dissemination* is **distributed event-based routing** [8], whose goal is to minimize network and computational overhead when disseminating an event to a group of interested recipients, which could be a sizable group. There are various techniques for event routing, such as selective algorithms (rendezvous-based and filter-based), event gossiping algorithms, and flooding algorithms (event flooding and subscription flooding). In some cases some of these are enhanced with the use of **advertisements**, as in the cases of REBECA [31], SIENA [11] and Salehi et al. [30], where an advertisement is issued by a publisher to announce the series of events it will produce and is taken into account when building routing paths. These systems create paths based on the advertisements and subscriptions, and then use those paths to propagate event notifications appropriately. Advertisements are spread throughout the entire network and are primarily used in content-based pub/sub models. While such approaches can be leveraged as the software overlay of widespread applications sharing IoT data (even data captured from sensors including personalized information) to

other communities, this could endanger the privacy of community individuals if done without any filtering or control.

ComDeX incorporates a scalable pub/sub architecture at its core to achieve cross-community communication and data exchange. As in previous systems [11, 30, 31], it uses advertisements to disseminate data. ComDeX showcases the location/broker where the relevant data is published so that interested subscribers can find the node directly (as long as they receive the advertisement). This approach does not create subscription tables across the broker nodes. Contrasting this with Named Data Networking, NDN [6] employs a mechanism where interest packets replace this conventional advertisement process. These packets, signifying interest in particular data, traverse the network until they find a node capable of satisfying the interest with a corresponding Data packet. This mechanism creates a sort of 'subscription table' within the network, albeit in a different form: the path of interest packets creates a reverse route for the Data packet to follow back to the requester, establishing stateful information in each router along the way. Despite the utility of NDN in certain contexts, its approach diverges from the fundamental philosophy of ComDeX. Our platform, specifically avoids the creation of such stateful subscription tables, opting for a more direct data dissemination strategy. Another part of ComDeX's novelty lies in leveraging context semantics in its data models to reduce the number of advertisements that may have been needed in a general-purpose semantics-agnostic system.

Non-federated smart-space solutions. Previous research on enabling the development of interoperable and portable IoT applications within a homogeneous smart space includes Space Broker [5] and SemIoTic [35]. Space Broker defines a smart space based on contextual data, spatial attributes, and user-specified application requirements. Interoperability capabilities, reusability (smart apps reused in many settings), user privacy and data sovereignty are all crucial characteristics of IoT smart spaces that SemIoTic addresses. However, both of these works are *focused on a single smart space* and, contrary to ComDeX, they do not offer a solution to challenges faced in a federation of smart communities/smart spaces.

A scalable context-aware IoT platform that federates several (possibly heterogeneous) cross-smart-space brokers and addresses all requirements of IoT-enhanced communities, while being as effective as a single-smart-space solution, is still elusive. ComDeX is a distributed context-aware federation architecture platform that enables widespread IoT applications through novel context-aware information dissemination techniques and an efficient broker-federation topology, facilitating data sovereignty while enabling open collaboration between smart communities.

3 OVERVIEW

3.1 Motivating scenario

Large city ports such as shown in Fig. 1 comprise a variety of different organizations (e.g., maritime authority, port authority, city authorities, etc.) and communities that own or manage data, sensors, and applications. Such modern ports are already evolving into smart communities [25, 29] that stand to benefit from IoT applications, such as monitoring the occupancy of passenger stations, monitoring air quality using environmental sensors, marine and urban traffic management applications, etc. In such a heterogeneous environment, diverse sensors and actuators transmit data relevant to discrete data recipients. An example of a relevant application to a smart port is *smart urban transportation*, which we will see in detail below. ComDeX, is aimed at fostering data sharing among diverse stakeholders, with the objective to enhance operations and services, and advance the evolution of smart communities. In what follows, we will highlight the benefits of achieving cross-smart-community communication in this context, as a motivation for ComDeX.

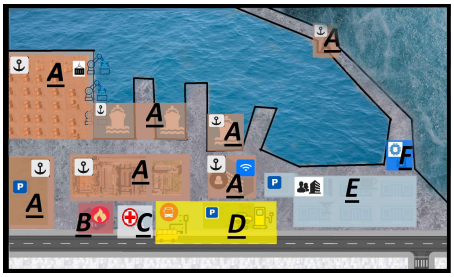

Figure 1: Smart communities in a modern city port

For simplicity, we assume the participation of just 3 different stakeholders in a smart port (Fig. 1), out of the 6 present in the figure Ⓐ-Ⓕ: the port authority (zone Ⓐ), city firefighting department (zone Ⓑ), and Smart InterCity Bus organization (zone Ⓓ), each storing and managing the IoT data they own. The smart transportation application could benefit from information-sharing between these different stakeholders. However, each of them would only want to share the part of their data that it considers relevant. The Intercity Bus Organization and Port Authority could benefit from exchanging information, such as real-time vehicle and ship positions. At times, the Firefighting Department might want to request information, like the current occupancy of the passenger station in the event of a fire emergency, but they may not necessarily want to provide any information to others.

Our main motivation is to design a federated IoT platform architecture for smart communities and build a prototype system using state-of-the-art technologies. Considering the wealth of diverse static and dynamic contextual information available across smart communities, ComDeX would enable the stakeholders to gain insights from all data exchanged in their ecosystems, and use them to improve their operations and services, ultimately enabling the development of the next generation of smart communities.

Section 6.4 provides details on how our proposed architecture, ComDeX, achieves these objectives in a smart port deployment.

3.2 The ComDeX Architecture

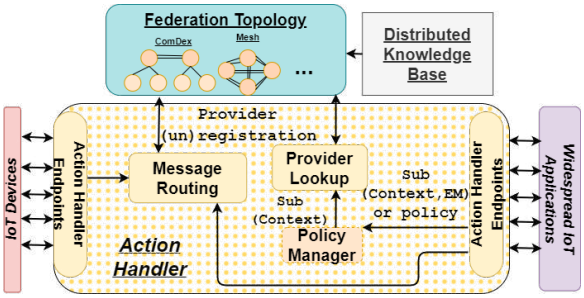

Figure 2: High level view of the ComDeX Architecture

Here we provide a high-level overview of the architecture of ComDeX, our context-aware federated platform for IoT-enhanced Communities. The high-level architecture of ComDeX (see Fig. 2) facilitates the data exchange between IoT devices and IoT applications in smart communities based on three major components: i) The Federation Topology component, which handles the connection between various brokers; ii) the knowledge base component, which corresponds to the information model of ComDeX; and iii) the Action Handler component, along with its sub-components, which interacts with clients for various data exchange operations. ComDeX works with a **federation of brokers**, a group of autonomous brokers that collaborate to conduct data discovery operations. To connect and cooperate, brokers must follow a predefined **topology**. ComDeX proposes **a hierarchical based hybrid topology** for the system architecture. This topology is comprised of a collection of hierarchical/tree sub-topologies belonging to different data stakeholders, connected to each other at appropriate hierarchical levels based on either smart domain separation, geographical areas, or both. This topology is presented in detail in section 5.

The **Distributed knowledge base** is made up of schema files that specify the concepts of ComDeX's information model, which is in turn based on property graphs. As a starting point, new data models can be developed by extending existing ones, although certain values may be redundant while other needed elements may be lacking. Another thing the knowledge base offers is information about the current broker topology that can be used by new brokers that wish to join the system and for any potential modification.

The Action Handler provides an API for the various clients (producers/consumers) to conduct diverse "Actions". "Action" can be defined as any operation inside the architecture that is necessary for the exchange of information between clients and the brokers. It offers high-level functions like data context discovery, both synchronous and asynchronous. It is responsible for executing commands and managing data flows using various sub-components, such as the **message routing component** (handles the different data flows), the **provider lookup component** (enables the discovery of remote data providers when requesting data), and the **policy manager**, which could affect the provider lookup and message routing components by taking into account certain policies, such as a change in the network or an emergency scenario. Although defined in the ComDeX architecture, determining how to approach the handling of different QoS requirements of applications in different scenarios (e.g. emergencies) in the general case is out of context for this paper and part of future work. Other actions include the **publish-data action**, which is performed using the **message routing** component. Each time a content producer creates data at an edge broker, the data is stored locally at the edge, and in the hierarchical network of brokers, a **provider-registration operation** is also performed to showcase what data are available at which broker. Finally the **request/subscribe-to-data action** is performed using **the provider lookup and message routing components**. The provider lookup action is used to find where in the topology is the information required by the client, in order to route the client command to the appropriate broker for data exchange.

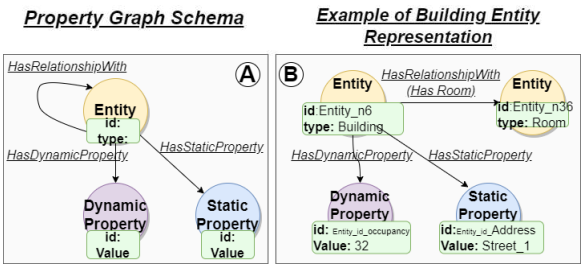

Figure 3: Property graph schema for smart communities

4 THE COMDEX FORMAL MODEL

In this section we provide the formal model of ComDeX, which consists of a data schema that is mapped to a publish/subscribe subscription scheme for the creation of context-aware messages. Then, we present the formal definitions of a federated system for the data exchange between smart communities. Finally, we formally present the actions that can be performed in such a system. In smart communities, information can be represented via separation into entities. To model entities in smart communities, ComDeX relies on property graphs. Property graphs [28] are a type of graph data model that focuses on a graph structure. In such data models, each graph consists of a collection of elements: vertices connected by a set of directed, labeled edges. Every element has a unique identifier and can have any number of key-value pairs called properties annotated on it [7]. A data schema is a strong data modeling feature that enables describing data structures and enforcing consistency. In this way, a graph schema allows establishing the graph structure by identifying the types of nodes, edges, and their properties.

Using property graph schema, we model the information of an entity using three aspects: entity details (static and dynamic attributes), entity type, and entity relationships (see Fig. 3A). An entity's "entity type" is a word or phrase in the information of the relevant knowledge base that indicates the entity's category information. Relationships between entities are known as "entity connections". Let $E = \{\epsilon_j : j \in [1..|E|]\}$ be the set of entities where each ϵ_j has features $\{id, type, attr\}$. We refer to each ϵ_j's feature (e.g., type) as $\epsilon_j.type$. Let $\epsilon_j.attr = \{\epsilon_j.attr.ea_i : i \in [1..|\epsilon_j.attr|]\}$ be a set of attributes with features $\{type, value\}$. We denote $\epsilon_j.attr.ea_i.type = \{stprop, dnprop, rel\}$ as the specific types of the i_{th} attribute that an entity ϵ_j can have static properties ($stprop$), dynamic ($dnprop$) properties and/or relationships with other entities (rel). We simplify the notation of attributes for an entity ϵ_j and refer to $\epsilon_j.attr.ea_i$ as ea_i. For example, suppose that we have the building entity ϵ_{n6} shown in Fig. 3B with information about its occupancy (dynamic property, $ea_i.type = dnprop$), its address (static property, $ea_i.type = stprop$) and that it is related to a room (HasRoom, $ea_i.type = rel$).

In order to continue working on ComDeX, a specific pub/sub data detection technique had to be chosen. Different approaches of defining the data of interest have resulted in the discovery of distinct pub/sub variations [8]. The expressive power of the subscription models that have surfaced in the literature is distinguished by their ability to precisely match subscribers' interests, i.e. getting just the messages that they are interested in. In topic-based pub/sub systems, notifications are grouped in topics i.e., a context subscriber declares its interest for a particular topic and will receive messages related to that topic. Let $T = \{t_j : j \in [1..|T|]\}$ be the set of topics available

in the system. By relying on pub/sub, ComDeX represents entities as stored messages that have been published to certain topics. Let $M = \{m_i : i \in [1..|T|]\}$ be the set of ComDeX messages where each m_i has features $\{topic, payload\}$. To enable clients performing context-aware actions (data requests and subscriptions), we map entities representing the context of smart communities to ComDeX messages. For this, it is essential to define a general rule as follows:

Algorithm 1 Algorithm to split of data in property graph into messages on specific topics

```
1:                          ▷ Input: Property Graph        ▷ Output: ComDeX Messages
2: procedure Splitting P.Graph:
3:     for each node x where node_type equals "Entity" do
4:         ε_j ← x
5:         ea_i ← edges.of.x
6:         for each ea_i do
7:             t_j ← ε_j.type + ε_j.id + ea_i.type + ea_i.id
8:             m_i.topic ← t_j
9:             m_i.payload ← ea_x.value
10:            send(m_i)
11:        end for
12:    end for
13: end procedure
```

For every entity ϵ_j with static and dynamic properties, as well as with its relationships, a message is created and stored using Algorithm 1. In this way, ComDeX messages are now *context-aware*. For instance, given as input the building entity of Fig. 3B in Algorithm 1, messages are created as shown in Listing 1. Note that topics mapped from dynamic properties aim to be updated by IoT devices attached to the corresponding entity.

```
"message1" : {
    "topic" : "Building/Entity_n6/HasDynamicProperty/Occupancy"
    "payload" : "value:32" }
"message2" : {
    "topic" : "Building/Entity_n6/HasStaticProperty/address"
    "payload" : "value:Street_1" |
"message3" : {
    "topic" : "Building/Entity_n6/HasRelationshipWIth/HasRoom"
    "payload" : "id:Entity_n36"
```

Listing 1: Entities mapped to pub/sub messages

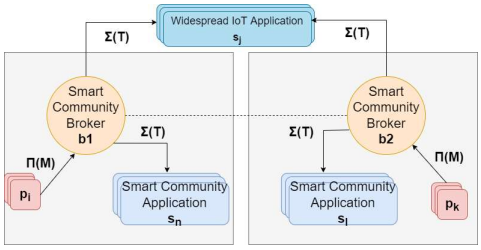

Figure 4: A federation broker-based model

To enable the exchange of context-aware topic-based messages among smart communities, ComDeX is built based on a distributed pub/sub system where context-related components serve as context brokers, publishers and subscribers (see Fig. 4). Let $P = \{p_i : i \in [1..|P|]\}$ be the set of publishers that correspond to IoT devices placed in smart communities publishing entities to a set of topics. We denote as $M_{p_i} \subseteq M$ the set of context-aware messages (i.e., entity values) that p_i publishes to a set of topics $T_{p_i} \subseteq T$ (i.e., characterized by entity types). Similarly. we denote as $S = \{s_j : j \in [1..|S|]\}$ the set of subscribers that correspond to community occupants interested in receiving messages. They subscribe to a set of topics, denoted by $T_{s_j} \subseteq T$ (including entity types, attributes and values) or perform direct on-demand queries.

A context broker can be deployed in a smart community providing context-aware messages. For the cooperation between brokers (i.e., communities), such brokers can form a federation offering routing, message management, query resolution and service discovery. Context brokers may be geographically dispersed in a real-world deployment. Multiple brokers split into administrative, network, geographic, contextual, or load-based domains are ideal to reduce administration and communication overheads. ComDeX necessitates inter-broker federation so that data providers and clients affiliated with various brokers can communicate with one another. This requirement can be met using a basic message system for data relaying implemented by an overlay network of distributed brokers.

We denote as $B = \{b_k : k \in [1..|B|]\}$ the set of federation brokers. A broker forwards messages from publishers to interested subscribers or to other brokers (in our case advertisement messages, more details below). We assume that each publisher/subscriber connects with a single broker that we refer to as its *home broker*: b_{p_i} in case of publisher p_i, b_{s_j} in case of subscriber s_j. Furthermore, we define the set of publishers and subscribers connected with b_k as $P_{b_k} = \{p_i \in P : b_k == b_{p_i}\}$ and $S_{b_k} = \{s_j \in S : b_k == b_{s_j}\}$. We also define the set of Brokers connected in the federation to a b_k as $B_{b_l} = \{b_l \in B : b_k == b_{b_l}\}$

In ComDeX, message routing between federated brokers is accomplished by relying on advertisements. Let $A = \{a_j : j \in [1..|A|]\}$ be the set of advertisements that are disseminated and stored in the broker federation. Advertisements are used by subscribers to find a broker that offers context-aware messages. An a_j is basically information about a broker that has messages published (entity types, attributes and values). Advertisements are composed from the following features: (i) $a_j.addr$ is the broker connection information-address where the advertisement was originally generated and (ii) $a_j.type$ is the entity type it advertises. An advertisement is created for each distinct entity type published in a broker. We denote the set of topics matching an advertisement a_j as $T_{a_j} \subseteq T$ and the entities matching a_j as $E_{a_j} \subseteq E$. To make clear how many advertisements a singular broker has at most, in the case that it receives advertisements from every other broker (e.g., mesh topology), we define as $|A_{max}| = \sum_{k=1}^{B} b_k * (distinct.\epsilon_j.type : j \in [1..|E|])$. To evaluate the efficacy of ComDeX, we define the following performance metrics: Let $\Delta_{gen}(a_j, p_i, b_e)$ be the generation time of an advertisement a_j by a p_i at an edge broker, b_e. Let $\Delta_{rec}(a_j, s_j, b_t)$ be the reception time of an advertisement a_j by a s_j a a top broker b_t. The time taking for an advertisement to be installed in the entire federation of ComDeX is $\Delta_{ins} = \Delta_{rec}(a_j, s_j, b_t) - \Delta_{gen}(a_j, p_i, b_e)$. We define the subscription notification latency metric as the time, from the creation of a Publication π_j at a broker b_k until its reception by an interested subscriber s_j

When ComDeX federation is setup (e.g., brokers in smart communities), community inhabitants can leverage the provided IoT devices and applications. A set of diverse ***actions*** is used to push/pull or subscribe to receive context-aware messages in the smart community (see Fig. 5). To enable the registration of context information (buildings, vehicles, etc.), community inhabitants can advertise the entity types that they provide to their home broker. Let $REG(a_j)$

be the *provider registration* action which creates an advertisement to a context broker. Then, publication actions including the advertised information may be performed. Let $\Pi = \{\pi_j : j \in [1..|\Pi|]\}$ be the set of publications with context-aware messages related to the smart community (e.g., Buildings of the port community). We assume that publications are generated from real-world activity or traces of existing IoT devices deployed in smart communities.

We now define actions used from data subscribers willing to discover entities in a smart community. Let $LUP(a_j)$ be the *provider lookup* action with which a context subscriber can discover what kind of data is available and where. A subscriber performs this using the stored advertisements in its local broker. Having discovered the community entities, a subscriber can setup a subscription action which defines the required instances of entities and the validity period of the subscription. Let $\Sigma = \{\sigma_i : i \in [1..|\Sigma|]\}$ be the set of subscriptions. To satisfy a subscription, each broker filters the entity type generated by the publishers based on the topics locally, and forwards the subscriptions to other community brokers that have relevant information according to the provider lookup action. Subscriptions are usually made to entities with dynamic properties (e.g., the temperature of a room). To receive the messages stored in brokers and especially entities with static properties, we define the data request action. Let $D = \{d_i : i \in [1..|D|]\}$ be the set of data requests. This is the action where a subscriber requests data synchronously from its home broker. Similarly, to satisfy the request the broker will also forward the inquiry to community brokers that have relevant information according to provider lookup. Having defined the ComDeX system model, we match the basic design principles and IoT platform requirements presented in section 3 and thus we can proceed to create a working Context-aware Federated Platform for IoT-enhanced Communities.

5 THE COMDEX FEDERATION TOPOLOGY

To design ComDeX's federation topology, our solution is based on advertising context-aware messages in the federation, as aforementioned in §2. In particular, instead of subscribers conveying the topics they are interested in across the system, each broker advertises the context-aware messages it can provide. When a new message is pushed, the ComDeX action handler creates a *provider advertisement message*. Brokers forward these advertisements subject to topic restrictions to allow broker owners to limit what advertisements are propagated and where.

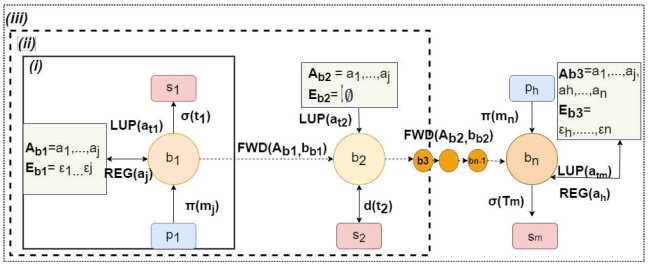

Figure 5: Actions in a number n of connected brokers

Using Fig. 5, we now describe how our federation model operates for the following sets of brokers: (i) one broker; (ii) two brokers and; (iii) n brokers. We define as $E_{b_k} \subseteq E$ the entities stored at a broker b_k; $A_{b_k} \subseteq A$ the advertisements stored at a broker b_k; and $FWD(a_j, b_k, b_l)$ the action with which a broker b_k forwards an advertisement a_j to one of its connected brokers $b_l \in b_{b_k}$.

For the trivial case of a single broker, consider that the local subscriber s_1 of broker b_1 requests messages via a subscription on topic t_1. A publisher p_1 publishes j messages of various entity types in different topics, and creates appropriate advertisements (if they do not exist) using the provider registration action. When a message is published to a topic and it matches the subscription, it notifies the subscriber and sends it the match context-aware messages.

For the case of 2 brokers, consider that there is one more broker (b_2) connected to the previous broker and a context subscriber s_2 which requests messages using d_i on a specific topic (i.e., entity type) T_{s_2}. Advertisements are now propagated along connected brokers. When the data request action is performed in b_2, the broker first checks for available messages stored locally that match the request. Then, using the provider lookup, b_2 searches for advertisements matching the messages requested. If a matching advertisement is found, then it forwards the data request to the relevant broker. If matching messages are found in b_1, these are sent to s_2.

Finally, for the case of n brokers, consider an arbitrary number of brokers connected along with the previous ones. When a message is published to b_1, its matching advertisement is propagated along the federation of brokers. Consider now a subscriber s_m and a publisher p_h connected to b_n. Since the advertisements are forwarded in one-way, from "left to right", clients connected to broker b_1 are not aware of the available messages in b_n. Similarly, a subscriber s_m becomes aware of the messages stored at b_n and the provider of messages that matches their subscribed topic, through the stored advertisements at broker b_n.

In large cross-community deployments, distributed solutions are favored over centralized ones due to their flexibility, modularity, and privacy benefits. The flexibility of distributed solutions allows for dynamic changes in the system by adding or removing brokers on-the-fly. Modifications made by a stakeholder to their broker set do not affect the entire system, which enhances modularity. Additionally, data is maintained in the owner's machines, leading to improved privacy. Federated systems are commonly structured as a tree, star, mesh, or ring topology [15], with the number of hops affecting the time to send and receive messages. For optimal performance, the number of brokers between the message origin and final destination should be minimized. ComDeX's topology is a *hierarchical-based hybrid topology* that benefits from hierarchical data flows, which resemble the reality of smart communities. Hierarchical flows enable the separation of context information in a *selective-sharing* manner, where each community chooses what data to advertise to the upper levels of smart communities. However, a tree/hierarchical topology has inherent disadvantages, such as an increase in the number of hierarchical levels leading to a higher number of hops for information traversal. A mesh topology would require every data provider to share their information with every other community, which is not practical. Adding new brokers to a mesh topology also becomes a complex task as the complexity and number of brokers increase. Our hybrid topology strikes a balance between the advantages and disadvantages of different topologies.

Fig. 6 illustrates ComDeX's topology for the port scenario (§3.1). Community A is the Smart Port Authority Community, with a central broker b_1 and two sub-brokers b_2 and b_3 that cover different

geographical areas in the smart port. Community B is the InterCityBus Transit Community, and here the division between brokers is by smart domain, where b_5 serves as the smart buildings broker, b_6 as the smart transportation broker. Lastly, community C is the Firefighting Community, with broker b_7 as its central broker. Advertisements are forwarded according to connections, where b_2 forwards to b_1, and b_1 forwards to b_4, and so on.

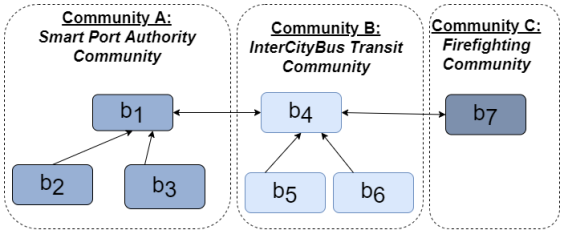

Figure 6: The ComDeX Topology

The algorithm for advertisement propagation in the federation (Alg. 2) is pretty straightforward: Suppose there is a broker b_k that has a set of directly connected brokers b_{b_k} in the federation (e.g, $b_1 \rightarrow b_2$ in Fig. 6). Broker b_k has a process that runs in parallel with all its other operations, which simply waits for an advertisement a_j to arrive. When an advertisement arrives at the broker, it checks all its connections with other brokers. These connections may have restrictions: For example, b_7 could only want to exchange information about specific entity types (e.g., smart_building), not all its information. It checks if the advertisement, in each connection, passes the connection's restriction, and if it does, the advertisement a_j is propagated to the appropriate connected brokers.

Algorithm 2 Algorithm for advertisement management by broker

```
1: procedure Advertisement Propagation at b_k:
2:     while 1 do
3:         a_j ← wait.for.advertisement.arrival()
4:         A_{b_k} ← A_{b_k} + a_j
5:         for each b_l ∈ b_{b_k} do
6:             pass ← Restriction.Check(a_j, b_l)
7:             if (pass == TRUE) then
8:                 FWD(a_j, b_k, b_l)
9:             end if
10:        end for
11:    end while
12: end procedure
```

Our approach offers a significant advantage over a purely hierarchical one: it enables high-level brokers from different smart communities to connect directly, without the need to create new higher-level nodes in the hierarchy. Specifically, our approach treats all high-level brokers as "first-class citizens", meaning that the existing hierarchy of the highest broker in each community is preserved. For instance, brokers b_1, b_4 and b_7 (the highest-level brokers in their respective communities) can be connected without requiring the creation of a new broker at a higher level of the hierarchy (such as a new b_8). This means that applications within each community can discover new entities without needing to connect to a new central broker. It is important to note that the system designer is ultimately responsible for choosing the topology. Depending on the use-case requirements, connecting brokers using a higher-level topology could reduce overall system complexity, especially if there would otherwise be many high-level brokers. Furthermore, if the topology has no bidirectional connections throughout (e.g., if community A's sub-topology constitutes the entire topology), it is functionally equivalent to a standard hierarchical topology, with a root node.

6 PROTOTYPE IMPLEMENTATION

In section 4, ComDeX's model was defined as a property graph of entities that are mapped and stored using a topic-based messaging scheme. We implemented this model taking as a basis the NGSI-LD specification [1], which combines linked-data entities with property graphs and provides an API that covers most actions required by ComDeX. To facilitate communication in ComDeX, MQTT was chosen as a lightweight topic-based pub-sub message protocol due to its wide deployment and acceptance in IoT applications.

Our ComDeX implementation provides a lightweight NGSI-LD federated broker that uses open-source MQTT brokers at its core. The ComDeX prototype implementation improves upon existing heavyweight NGSI-LD brokers. The NGSI-LD specification currently favors HTTP as a communication protocol, but this is not ideal for end-to-end MQTT capabilities, such as QoS delivery guarantees and necessitates the use of IoT-Agents for MQTT compatibility.

ComDeX implements a federated architecture to enable context-aware data exchange among smart communities. The NGSI-LD specification aims to support federated and distributed broker topologies through context sources and context-source registrations. Context-source registrations include details on the types of context information a context-source can provide, but not actual values. While context-information API operations are supported by many existing NGSI-LD brokers, context-source registrations and discovery operations are not quite there yet. Current NGSI-LD broker implementations have different solutions that are still under heavy development and may be dismissed by the NGSI-LD community when an official forwarding solution is defined. ComDeX demonstrates that federation in an NGSI-LD service naturally fits and can be embedded into a native MQTT topic-based architecture.

The ComDeX prototype implementation is provided as an open-source platform at `https://github.com/SAMSGBLab/ComDeX`.

6.1 NGSI-LD as ComDeX's information model

Recall that the ComDeX information model consists of modeling entities, their static and dynamic properties, and relationships using property graphs. The NGSI-LD information model is a good fit for it as it also derives from property graphs [27]: The core element is the *entity* (known as a *node* in property-graph language), which corresponds to a real-world concept. Every entity must have a unique identifier, which must be a URI (typically a URN), as well as a type, likewise a URI. This URI should point to a Web-based data model. Entities are associated with *properties* and *relationships*. To enable context awareness, each property's name should ideally be a well-defined URI that corresponds to a widely used notion on the Web. This knowledge graph is well-defined and infinitely expandable. Property graphs are versatile, scalable representations that have been widely used in the IT industry.

6.2 NGSI-LD to MQTT mapping

Moving from theory to practice in mapping from a property-graph representation to MQTT topic messages posed design challenges.

NGSI-LD entities are managed hierarchically via the namespace /NGSI-LD/v1/entities/<entity-id>/attrs/<attr-id>. This structure offers four endpoints for applying fine-grained or coarser-grained CRUD actions (e.g., modifying an individual entity or attribute, or querying all entities of a certain type). NGSI-LD context information API operations are divided into two categories: *provision* (creating entities and modifying attributes) and *consumption* (querying and subscribing to entities). All NGSI-LD entities must have an id, a type, attributes that describe the entity, and a "context" derived from JSON-LD.

We represent an NGSI-LD entity as multiple MQTT messages, one for each attribute. Each message is published on a topic constructed as (area+'/entities/'+context+type+id+attribute). The optional "area" parameter defines the geographical scope of the broker. The term "entities" in the topic indicates that the message describes an entity, with context, type, id, and attribute forming parts of the entity. Combining messages with the same entity-id in the topic allows us to reconstruct the fragmented entity when requesting it. This approach also enables attribute-level operations on each entity. Referring to "entities" aids in self-reference to NGSI-LD endpoints during development and maintenance. It helps distinguish when non-entity information is present in the broker. In the future, additional context information, such as active subscriptions, might be available.

To **create an entity**, the "POST" entity command is used with the format: `'POST' 'http://broker-address/:port/NGSI-LD/v1/entities/' 'entity-data.file'`. The action handler validates the command and the entity file. If the entity already exists, the publisher is notified. If not, an advertisement is created for the new context information available in the federated ComDeX service. The advertisement topic is "provider/'+broker_address+broker_port+/area+entitycontext+entity_type+(entity_id)" with the specified granularity (as mentioned in §6.3). For each attribute of the entity, a new MQTT message is created with the topic area+'/entities/'+entity_context+entity_type+entityid+attribute, and the attribute content serves as the payload. Messages for the *createdAt* and *modifiedAt* attributes are also generated to facilitate temporal queries.

The **creation of a subscription** involves a 'POST' request of a 'subscription.file' on the NGSI-LD broker at 'http://broker-address/:port/NGSI-LD/v1/Subscriptions/'. This process is similar to entity creation but has additional complexity. When creating a subscription, the action handler validates the command and the 'subscription.file', which must contain a valid JSON with at least one of the following: id, type, watched_attributes. The handler establishes a connection with the specified broker and subscribes to provider messages (advertisements) that align with the subscription. For each newly received provider message, a parallel connection/subscription is created with the broker that published the relevant information, provided it has not been received before. If an advertisement is deleted while a subscriber is still connected, the corresponding subscription is terminated. The fragmented entity data received is reconstructed accordingly. Our ComDeX prototype implementation is in Python but can be easily adapted to other programming languages. Although ComDeX does not introduce novel fault tolerance strategies, developers can leverage MQTT backup bridges [23] offered by MQTT brokers to establish redundancy using backup advertisement links and data brokers/servers.

6.3 ComDeX advertisements and bridging

Our broker federation implementation relies on advertisements to propagate exported context, which are MQTT messages propagated similar to the entities messages. An advertisement message includes the provider's id, broker address and port, region covered, and NGSI-LD entity context and type, as described in §6.2. Clients can utilize the advertisements available at the initially connected broker to establish new connections and retrieve the desired information, even if it is located on a remote broker. Our prototype allows the selection of advertisement granularity (entity type or id), with per-type being the current default. Provider messages are propagated among brokers through MQTT bridges. To prevent redundant message transmissions and network overload, proper filtering of data between bridges is crucial. Bidirectional linkages in MQTT bridges should be avoided. Alternatively, manual NGSI-LD-like context source registrations could replace the automated advertisement system. However, the automated advertisement system offers usability advantages and enables a functional NGSI-LD broker federation.

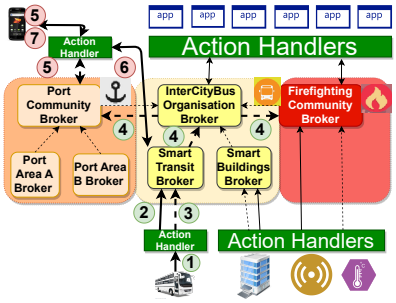

Figure 7: Smart Transportation Application using ComDeX

6.4 Smart transportation in the port scenario

The ComDeX prototype enables the implementation of the smart transportation motivating scenario detailed in §3, connecting stakeholders' brokers (such as firefighters, port community, and intercity buses, as shown in Fig. 7) hierarchically via MQTT bridges. High-level brokers receive information from all advertised sources through these bridges. For simplicity, we assume a single publisher and subscriber. A data publisher, such as a bus transmitting its GPS location, sends NGSI-LD vehicle-related data to the action-handler-broker (step ① in Fig. 7), which converts the data into MQTT messages and sends them to the MQTT broker (step ②). It verifies whether a vehicle entity has already been advertised to this broker and, if not, creates a data-provider-advertisement message and sends it to the broker (step ③) as described in §6.2. The broker receives and forwards the advertisement (step ④) to the IntercityBus, Firefighting Community, and Port Community Brokers through bridging. The advertisement paths are indicated by dotted lines in Fig. 7. A subscriber, such as a mobile application, seeking information on all available buses in the federation, sends a

relevant NGSI-LD subscription to the nearest broker (here the Port Community Broker) using the action handler (step ⑤), which verifies the subscription and searches for advertisements that match the requested entity type and, if specified, location or area of interest (e.g., Port Area A). The action handler determines that the InterCityBus Broker has bus data and connects to it to receive the relevant data (step ⑥), which is structured back as NGSI-LD entities. Finally, the requested information is sent back to the subscriber (step ⑦).

6.5 Federating with 3rd-party platforms

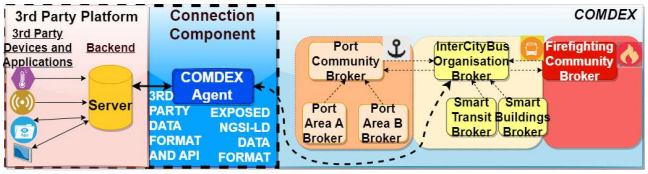

Figure 8: ComDeX federation with 3rd-party platforms

ComDeX is a federated system that can interoperate with established 3rd party architectures. In cases where the 3rd party platform has non-NGSI-LD data models and APIs, a ComDeX Agent component can be used to convert the data models to NGSI-LD and enable transparent data exchange between the legacy system and ComDeX. The ComDeX Agent is also responsible for exchanging data with a ComDeX federation and transforming a legacy system into a ComDeX broker node without any modification to the established system itself. The ComDeX Agent has a series of fundamental primitives that provide the essential functionality needed for integration with 3rd-party systems. In a rudimentary C-like syntax, these include operations such as: `adaptModel(sourceData, targetModel)`: This primitive is used to convert the data model of the source system to the target model (NGSI-LD in this case). It handles the conversion of different data formats and structures to a unified representation. `translateAPI(sourceAPI, targetAPI)`: This primitive translates the API calls from the source system to those used by the target system (to and from ComDeX in this case). This transformation allows seamless interaction between the legacy system and ComDeX. `subscribeLegacy(topic)`: The ComDeX Agent can express interest in specific data types or topics from the legacy system using this primitive. This helps in obtaining relevant data from the legacy system. `publishLegacy(topic, message)`: This primitive is used to send data to a specified topic in the legacy system. It assists in updating the state or data of the legacy system when required. `receiveLegacy()`: This primitive enables the ComDeX Agent to receive incoming data from the legacy system, furthering the data exchange process. Beyond these primitives, since the agent makes the system act transparently as a ComDeX node it also includes all the functionalities of the ComDeX Action Handler (joining the federation, automated advertisements, etc.) For example, a smart transportation system using GTFS[1], GTFS-RT[2] standards and belonging to the Intercity Organization broker community can be connected to the IntercityBus broker using a ComDeX agent placed at the "second level" of the federation (see Fig. 8). Similar data transformations can be performed for any type of third-party system. While previous works [4, 13] have dealt with the conversion of data models to NGSI-LD and vice versa, a prototype implementation of a ComDeX agent and its converter subsystem for 3rd-party platforms is beyond the scope of this paper.

[1]https://gtfs.org/
[2]https://developers.google.com/transit/gtfs-realtime

7 EXPERIMENTAL EVALUATION

This section evaluates ComDeX and overall design approach using our prototype. We utilise multiple AWS EC2 VM instances within a single region[3], with each broker, publisher and subscriber hosted on a different VM with the specifications listed in Table 1. For consistent time measurements, the VMs are synchronized using AWS's Time Sync Service over the Network Time Protocol (NTP).

Functionallity	Instance Type	Instance Family	Instance Size	VCPUs	Memory (GIB)	Network Performace
Brokers	c5.large	c5	large	2	4	Up to 10 Gigabit
Subscribers/Publishers	t3.nano	t3	nano	2	0.5	Up to 5 Gigabit
Publisher (§7.2)	t3.large	t3	large	2	8	Up to 5 Gigabit

Table 1: Configuration of experimental testbed

We first evaluate the performance of the ComDeX prototype with different topology sizes and compare it against other NGSI-LD brokers, for a normal case and a worst-case scenario, using simple synthetic data models of a generic NGSI-LD entity. Then, we evaluate the impact of changing the federation topology and advertisement granularity using realistic generated entities from smart buildings based on existing NGSI-LD data models and real IoT device traces. Our results throughout all experiments validate the performance of the ComDeX prototype under a broad range of deployment configurations and generated workloads.This performance can be attributed to two main things: (i) our overall design due to the mapping between property graphs to pub/sub topic filters and the direct connections enabled by our advertisements; and (ii) the use of MQTT in comparison to HTTP.

7.1 Comparison against other NGSI-LD brokers

7.1.1 ***Normal-case scenario.*** We first evaluate the performance of the ComDeX prototype with various topology sizes against current state-of-the-art NGSI-LD brokers Orion-LD and Scorpio.While the NGSI-LD specification is still evolving, these two brokers incorporate most up-to-date functionalities of the NGSI-LD specification. For more information related to these NGSI-LD brokers, refer to [3] [2]. ComDeX adopts 1, 3, 6 broker-*wide* topologies, where width is the distance (network hops) from the broker where the data is published to the broker where interested subscribers connect. The setups evaluated can be seen in Fig. 9. We target a normal-case scenario where the load is created to be uniform in terms of both access pattern and load over system resources.

For this set of experiments, we created a data model with simple attributes of no semantic meaning (see Table 2), following an approach similar to [16]. We developed a data generator in Python for the above data model, and used it to generate 2000 entities of 10 different entity types and 3 different attributes.

The generated entities are entered into the system and stored at the edge brokers of each setup (Orion-LD, Scorpio, ComDeX (Mosquitto) 1-, 3- and 6-broker-wide) (Fig. 9). We then deploy a workload generating synthetic sensor requests on 10 subscriber VMs, each request selecting a random attribute of a random entity approximately every 100ms and patching its value to a random

[3]the rationale behind this is that it represents multiple communities in the same city.

Name	Type	Generation Strategy
id	URN	A urn string generated by concatenating "urn:ngsi-ld:entity" and a unique number between 0 to total number of entities requested.
Type	String	A type that represent various different entity types, it is generated by concatenating the string "Dummy_entity" and a random number between 0 and 10
Attribute1	String	A string value generated by contacting "value" and a randomly generated number between 0 to total number of entities requested.
Attribute2	Integer	Random Integer between 0 and total number of entities requested
Attribute3	Interger	Random Integer between 0 and total number of entities requested
Context	NGSI-LD context	Use the default NGSI-LD context.

Table 2: Virtual data model of generated entities (§7.1)

value (e.g., entity_500, Attribute3, 42). We dedicate one subscriber VM to each data type in all cases (e.g., Orion-LD has 10 different subscribers).

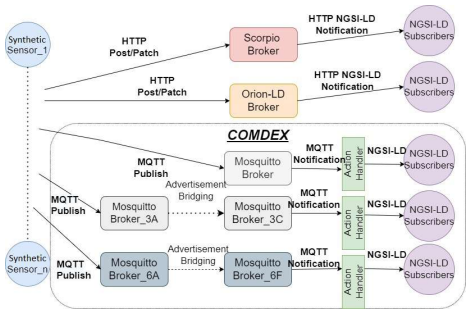

Figure 9: Normal-case experimental setup (§7.1.1)

We let the experiment run for a few hours to warm up the system. We then dynamically modify the behavior of sensors and subscribers to measure *subscription notification latency*[4]: instead of patching the value of the attribute of a random entity to a random value, the sensor patches it to the current time T_1 and immediately sends it. As soon as a subscriber receives a notification about an entity, it marks the current time T_2, computes subscription notification latency (T_2-T_1) and records it along with the entity. Fig. 10 reports a randomly selected sample of 50 observations of the subscription notification latency in milliseconds. We observe that ComDeX per-

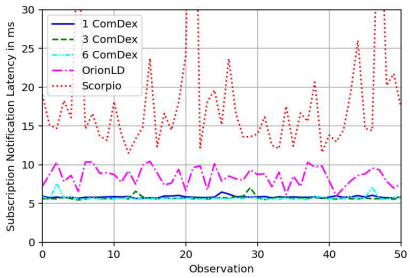

Figure 10: Subscription notification latencies (§7.1.1)

forms better in comparison with native HTTP NGSI-LD solutions since MQTT is used as the data exchange protocol. This result is in line with previous comparisons of HTTP and MQTT [33, 34] and highlights an advantage of the ComDeX design. Since our experiments include sufficient warm-up time, all advertisements are eventually installed, explaining the fact that there is no substantial difference between different-sized topologies of ComDeX, another one of its advantages. This can also be observed in Fig. 12 (five leftmost bars) which reports the mean subscription notification latency across multiple runs.

[4]this is delay between publication of an entity and notification of a subscriber.

7.1.2 ***Worst-case scenario for*** ComDeX. Next, we examine a worst-case scenario, performance-wise, for our prototype. This happens when a client needs to "re-discover" the data source and connect to it for every piece of information published to a broker. Recall from §6.2 that each time a subscriber connects to a broker searching for available data throughout the system, it must search through the existing advertisements to find where to connect. One way to emulate such a scenario is to have all data requested be composed of single entities of different types, which is feasible but not directly compatible with the single type subscription of NGSI-LD. Another way to emulate this is to have the data requested (subscribed to) be comprised of an entity-type for which its data and subsequently its advertisement are constantly being deleted and reinserted, which is the case we implement in this experiment (shown in Fig. 11). These worst-case emulated setups are luckily not expected in realistic systems. Again, we compare ComDeX to Orion-LD and Scorpio in similar conditions; since the latter two do not use advertisements, we expect them to not be affected as much.

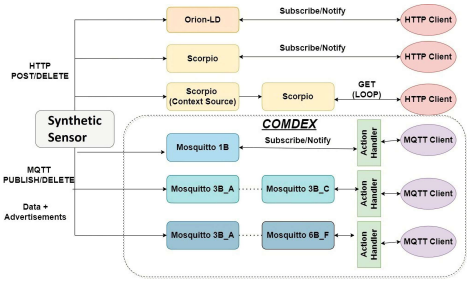

Figure 11: Worst-case scenario experimental setup (§7.1.2)

We use our data generator to create an entity of type "DummyEntity", which is then inserted into each broker along with the current timestamp. The subscribers receive this information and mark the time it arrives. The entity is then deleted from the broker it was inserted into, and consequently, the advertisement for that entity type is removed from the entire system. We repeat the process a large number of times and report our results in Fig. 12, where the metrics from this experiment (configurations with suffix *wc*) are displayed in comparison with those of the previous experiment (*nc*, §7.1.1). We observe that mean notification latency with our prototype in the worst-case scenario increases, as expected. This can only happen when a subscriber consistently wants an entity whose type does not exist and waits for its advertisement to arrive to discover it. We also use the setup to monitor the ComDeX *advertisement installation times*[5], depicted in Fig. 13. As expected, when the distance between the edge and top brokers increases, advertisement installation time increases as well, but not dramatically.

7.2 Evaluation with realistic IoT device traces

Next, we examine the impact of topology and advertisement granularity on the performance of our prototype. We consider *type-based advertisement granularity*, where an advertisement is created and propagated for every different entity type (e.g Building, Room, Device etc.), and *id-based granularity*, where an advertisement is created for each different entity (e.g urn:Device:devicen_0,urn:Door:doorn_427). Being able to discover exact entities with

[5]Advertisement installation time is the time it takes for an advertisement from its reception from the edge broker to arrive at the highest broker in the hierarchy

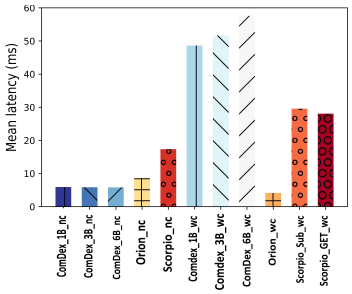

Figure 12: End-to-end subscription notification latency

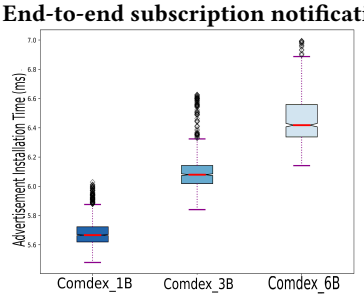

Figure 13: Advertisement-installation times for 1, 3, 6-broker-wide ComDeX

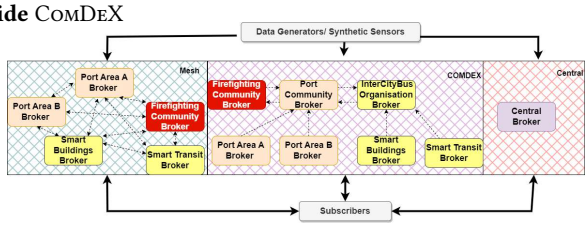

Figure 14: Emulating 3 smart communities under 3 topologies

their id allows for greater flexibility, such as remote actions on single entities. Having only advertisements on entity types reduces the number of advertisement messages, lowering traffic through system brokers. The downside of this is having to check if an advertisement already exists when creating an entity, which takes time. For this experiment, we implemented a generator of smart-building entities and IoT devices. The tool receives as input the number of federated brokers and generates communities, buildings, floors, rooms, and devices, capturing hierarchical (containment) relationships, using extended (by us) NGSI-LD based building and device data models[6]. Devices can exist within rooms, but they can also be embedded on windows and doors. A generation run for the entities of a single community sub-broker (Port Area B) gave 82 buildings, 212 floors, 1199 rooms, 1796 windows, 1781 doors, and 588 devices. with a rough total of 1700 synthetic devices across all communities. To create a realistic scenario for the experiment, the IoT devices and publishers are modeled using device characteristics reported by Kumar et al. [20], which process a dataset of 20 days of network traces generated by 20 IoT devices and extract their significant features. This dataset was chosen to generate synthetic traces because of its thorough analysis of real-world IoT device characteristics over a significant period, providing a comprehensive representation of IoT device behaviors and communication patterns.

Here we emulate the environment described in the motivating scenario using 3 different topologies (mesh, ComDeX topology, and a single central broker, Fig. 14). In the mesh topology, the brokers are connected to each other, thus every broker is aware of entities present in the other brokers through direct advertisements, unlike ComDeX where there is a hierarchy between brokers. Each of the different communities in each setup has a set of subscribers that are interested in changes in the values of devices across all smart communities. Such a subscriber could be, for example, a firefighting application that wants to know if a smoke detector detects a fire in a building regardless of location. The synthetic device sensors in this experiment work as follows: the devices are generated with a set of characteristics (message size, message frequency) described above [20]. Each device sends a message to its value in the appropriate broker according to these features continuously. To calculate the various metrics of this experiment, we used the tshark[7] network monitoring tool at each broker, similar to [9], to avoid modifying the content of the entities as we did in previous experiments.

The results of this experiment are depicted in Fig. 15. In Fig. 15(III), which depicts the number of MQTT messages needed for the creation of the entities of each community, we observe the impact of topology. In the mesh topology where "everyone knows everything", especially in the case of id granularity where the advertisements are equal to the number of entities inserted, the number of messages exchanged between brokers increases, since in this case every advertisement must be propagated to every broker.

Fig. 15(I-II) depicts advertisement-installation times for different

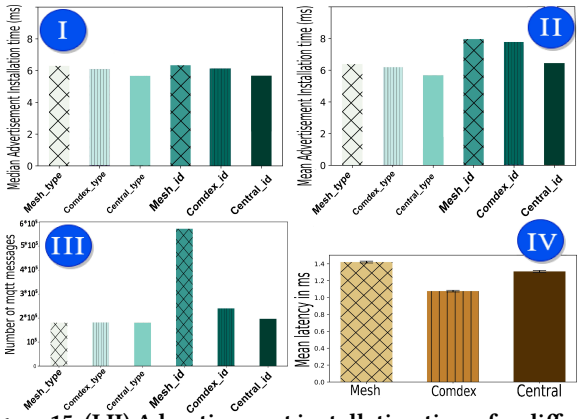

Figure 15: (I,II) Advertisement installation times for different topologies and advertisement granularities
(III) Number of messages in creation of 15,000 entities
(IV) Subscription notification latencies (§7.2)

topologies and advertisement granularities. As expected, all topologies perform similarly since the difference in the number of hops of each advertisement propagation does not exceed 2. The difference between the single broker and the two topologies (mesh, ComDeX) is similar to the difference between "broker widths" in Fig. 13. The variability between granularities is not significant and the slight difference between them can be attributed to the larger samples inherent to id granularity (1 advertisement for each entity).

Finally, we assessed subscription notification delay using the same setup as before, but instead of timestamping at endpoints, we inferred message departure and arrival times from network measurements. To simplify the matching process, we disabled Nagle's algorithm [24], which is enabled by default when using MQTT

[6]https://github.com/SAMSGBLab/iotspaces-DataModels/tree/main/Building

[7]https://tshark.dev/

libraries and brokers, to prevent message grouping at the TCP layer. This modification resulted in a significant reduction in latency compared to earlier experiments, highlighting the impact of network configuration on latency-sensitive experiments. The results in Fig. 15(IV) show that our proposed topology still outperforms the alternatives, although the difference is less pronounced. This improvement applies equally to all evaluated systems.

8 CONCLUSIONS AND FUTURE WORK

While smart communities can be established using existing platforms, a standard IoT platform architecture does not exist. This paper, introduces ComDeX, an approach for creating federated context-aware IoT platforms for smart communities. We propose a federation architecture that represents data as property graphs for smart community entities. We then create a context-aware messages using a topic/type-based pub/sub subscription scheme. Such messages are exchanged between smart communities via the ComDeX hybrid-hierarchical federation topology. We evaluate our architecture by creating and using a prototype that utilizes technologies such as NGSI-LD and MQTT, and we perform qualitative and quantitative comparisons with other existing solutions.

As part of our future work, we aim to enable the dynamic configuration of the federation based on the requirements of data recipients (e.g., emergency responders) or the deployed IoT applications of community. Cross-layer optimization mechanisms can be leveraged to tune such a federated system. ComDeX in its current state does not handle special security scenarios such as malicious agents/communities present in the federation, it could also be examined as future work. We strongly believe that ComDeX can serve as the basis for future research related to QoS-aware and self-adaptive data exchange between smart communities, fine-grain privacy-preserving policies for space occupants, as well as federated learning approaches to learn key properties of spaces that can be shared across multiple communities.

We thankfully acknowledge funding for this research by the Greek RTDI Action "RESEARCH-CREATE-INNOVATE" (ΕΠΑνΕΚ 2014-2020), Grant no. T2ΕΔΚ-02848 (SmartCityBus).

REFERENCES

[1] 2021. Context Information Management (CIM) NGSI-LD API V1.4.2. https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.04.02_60/gs_cim009v010402p.pdf

[2] Orion-LD Broker. https://github.com/FIWARE/context.Orion-LD.

[3] Scorpio Broker. https://github.com/ScorpioBroker/ScorpioBroker.

[4] Ahmed Abid, Jieun Lee, Franck Le Gall, and Jaeseung Song. 2022. Toward Mapping an NGSI-LD Context Model on RDF Graph Approaches: A Comparison Study. *Sensors (Basel, Switzerland)* 22 (2022).

[5] Hamim Md Adal, Colin Milhaupt, Jie Hua, Christine Julien, and Gruia-Catalin Roman. 2021. The space broker: a middleware for mediating interactions in smart IoT spaces. 101–110. https://doi.org/10.1145/3486611.3486664

[6] Alex Afanasyev, Jeff Burke, Tamer Refaei, Lan Wang, Beichuan Zhang, and Lixia Zhang. 2018. A Brief Introduction to Named Data Networking. In *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*.

[7] Renzo Angles. 2018. The Property Graph Database Model. In *AMW*.

[8] R Baldoni, Leonardo Querzoni, Sasu Tarkoma, and Antonino Virgillito. 2009. *Distributed Event Routing in Publish/Subscribe Communication Systems.*

[9] Eddas Bertrand-Martinez, Phelipe Feio, Vagner Nascimento, Fabio Kon, and Antônio Abelém. 2020. Classification and evaluation of IoT brokers: A methodology. *Int. J. of Network Management* 31 (06 2020). https://doi.org/10.1002/nem.2115

[10] José Carvajal Soto, Otilia Werner-Kytölä, Marco Jahn, Pullman J., Dario Bonino, Claudio Pastrone, and Maurizio Spirito. 2015. Towards a Federation of Smart City Services. https://doi.org/10.2991/racs-15.2016.28

[11] A. Carzaniga, D.S. Rosenblum, and A.L. Wolf. 2003. Design and evaluation of a wide-area event notification service. In *Foundations of Intrusion Tolerant Systems, 2003 [Organically Assured and Survivable Information Systems]*. 283–334. https://doi.org/10.1109/FITS.2003.1264940

[12] Bin Cheng, Gürkan Solmaz, Flavio Cirillo, Ernö Kovacs, Kazuyuki Terasawa, and Atsushi Kitazawa. 2018. FogFlow: Easy Programming of IoT Services Over Cloud and Edges for Smart Cities. *IEEE Internet of Things Journal* 5, 2 (2018), 696–707. https://doi.org/10.1109/JIOT.2017.2747214

[13] Flavio Cirillo, David Gómez, Luis Diez, Ignacio Elicegui Maestro, Thomas Barrie Juel Gilbert, and Reza Akhavan. 2020. Smart city IoT services creation through large-scale collaboration. *IEEE Internet of Things Journal* 7, 6 (2020), 5267–5275.

[14] Flavio Cirillo, Gurkan Solmaz, Everton Luis Berz, Martin Bauer, Bin Cheng, and Erno Kovacs. 2019. A Standard-Based Open Source IoT Platform: FIWARE. *IEEE Internet of Things Magazine* 2, 3 (Sep 2019), 12–18. https://doi.org/10.1109/iotm.0001.1800022

[15] The Apache Software Foundation. 2015. Apache Qpid, Messaging built on AMQ. https://qpid.apache.org/releases/qpid-cpp-1.39.0/cpp-broker/book/index.html

[16] Alireza Hassani, Alexey Medvedev, Arkady Zaslavsky, Pari Delir Haghighi, Prem Prakash Jayaraman, and Sea Ling. 2019. Efficient Execution of Complex Context Queries to Enable Near Real-Time Smart IoT Applications. *Sensors* 19, 24 (2019). https://doi.org/10.3390/s19245457

[17] Jaeyoung Hwang, JongGwan An, Hotaek Joo, ChanHyung Lee, and Jaeseung Song. 2017. Development and Application of Interoperability Techniques with Semantics for Global Internet of Things (GIoTs). *The Journal of Korean Institute of Communications and Information Sciences* 42 (11 2017), 2208–2216. https://doi.org/10.7840/kics.2017.42.11.2208

[18] Seungmyeong Jeong, Seongyun Kim, and Jaeho Kim. 2020. City Data Hub: Implementation of Standard-Based Smart City Data Platform for Interoperability. *Sensors* 20 (12 2020), 7000. https://doi.org/10.3390/s20237000

[19] Zhuangwei Kang, Robert Canady, Abhishek Dubey, Aniruddha Gokhale, Shashank Shekhar, and Matous Sedlacek. 2020. A Study of Publish/Subscribe Middleware Under Different IoT Traffic Conditions *(M4IoT'20)*. ACM, New York, NY, USA, 7–12. https://doi.org/10.1145/3429881.3430109

[20] Rakesh Kumar, Mayank Swarnkar, Gaurav Singal, and Neeraj Kumar. 2022. IoT Network Traffic Classification Using Machine Learning Algorithms: An Experimental Analysis. *IEEE Internet of Things Journal* 9, 2 (2022), 989–1008. https://doi.org/10.1109/JIOT.2021.3121517

[21] Heba Kurdi, Bushra Alshayban, Lina Altoaimy, and Shada Alsalamah. 2018. TrustyFeer: A Subjective Logic Trust Model for Smart City Peer-to-Peer Federated Clouds. *Wireless Communications and Mobile Computing* 2018 (02 2018), 1–13. https://doi.org/10.1155/2018/1073216

[22] Juan A. López-Morales, Juan A. Martínez, and Antonio F. Skarmeta. 2021. Improving Energy Efficiency of Irrigation Wells by Using an IoT-Based Platform. *Electronics* 10, 3 (2021). https://doi.org/10.3390/electronics10030250

[23] Pradhan Manas. 2021. Federation Based on MQTT for Urban Humanitarian Assistance and Disaster Recovery Operations. *IEEE Communications Magazine* 59, 2 (2021), 43–49. https://doi.org/10.1109/MCOM.001.2000937

[24] Jeffrey Mogul and Greg Minshall. 2001. Rethinking the TCP Nagle algorithm. *ACM SIGCOMM Computer Communication Review* 31 (01 2001), 6–20. https://doi.org/10.1145/382176.382177

[25] Anahita Molavi, Gino Lim, and Bruce Race. 2019. A Framework for Building a Smart Port and Smart Port Index. *International Journal of Sustainable Transportation* (04 2019). https://doi.org/10.1080/15568318.2019.1610919

[26] Alessandro Morelli, Lorenzo Campioni, Niccolò Fontana, Niranjan Suri, and Mauro Tortonesi. 2020. A Federated Platform to Support IoT Discovery in Smart Cities and HADR Scenarios. 511–519. https://doi.org/10.15439/2020F48

[27] Gilles Privat. 2021. Guidelines for Modelling with NGSI-LD (ETSI White Paper).

[28] Marko A. Rodriguez and Peter Neubauer. 2010. Constructions from Dots and Lines. *CoRR* abs/1006.2361 (2010). arXiv:1006.2361 http://arxiv.org/abs/1006.2361

[29] Bessid Sahbia, Alaeddine Zouari, Frikha Ahmed, and Benabdelhafid Abdellatif. 2020. Smart Ports Design Features Analysis: A Systematic Literature Review.

[30] Pooya Salehi, Kaiwen Zhang, and Hans-arno Jacobsen. 2020. On Delivery Guarantees in Distributed Content-Based Publish/Subscribe Systems. 61–73. https://doi.org/10.1145/3423211.3426400

[31] Wesley Terpstra, Stefan Behnel, Ludger Fiege, Andreas Zeidler, and Alejandro Buchmann. 2003. A Peer-to-Peer Approach to Content-Based Publish/Subscribe. *Proceedings of the 2nd International Workshop on Distributed Event-based Systems* (07 2003). https://doi.org/10.1145/966618.966627

[32] Giuseppe Tricomi, Giovanni Merlino, Francesco Longo, Distefano Salvatore, and Antonio Puliafito. 2019. Software-Defined City Infrastructure: A Control Plane for Rewireable Smart Cities. In *2019 IEEE Int. Conference on Smart Computing (SMARTCOMP)*. 180–185. https://doi.org/10.1109/SMARTCOMP.2019.00050

[33] Bharati Wukkadada, Kirti Wankhede, Ramith Nambiar, and Amala Nair. 2018. Comparison with HTTP and MQTT In Internet of Things (IoT). In *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*. 249–253. https://doi.org/10.1109/ICIRCA.2018.8597401

[34] Tetsuya Yokotani and Yuya Sasaki. 2016. Comparison with HTTP and MQTT on required network resources for IoT. 1–6. https://doi.org/10.1109/ICCEREC.2016.7814989

[35] Roberto Yus, Georgios Bouloukakis, Sharad Mehrotra, and Nalini Venkatasubramanian. 2022. The SemIoTic Ecosystem: A Semantic Bridge between IoT Devices and Smart Spaces. *ACM Transactions on Internet Technology – TOIT* (2022).