IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received September 21, 2021, accepted October 8, 2021, date of publication October 14, 2021, date of current version October 26, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3119975

Complex Event Processing in Smart City Monitoring Applications

BEHNAM KHAZAEL[1], **HADI TABATABAEE MALAZI**[1,2], **AND SIOBHÁN CLARKE**[2]
[1]Faculty of Computer Science and Engineering, Shahid Beheshti University, Tehran 19839 69411, Iran
[2]School of Computer Science and Statistics, Trinity College Dublin, Dublin 2, D02 PN40 Ireland

Corresponding author: Hadi Tabatabaee Malazi (tabatabh@tcd.ie)

This work was supported by the Science Foundation Ireland (SFI) through the Enable Project under Grant 16/SP/3804.

ABSTRACT Managing multi-tenant edge devices with heterogeneous capabilities scattered across an urban area requires significant communication and computing power, which is challenging when devices are also resource-constrained. These devices play a crucial role in smart city monitoring systems by notifying various municipal organizations about a wide range of ongoing complex events. Some recent approaches in complex event processing use a publish-subscribe architectural pattern to decouple simple event producers from complex event consumers. However, they did not fully address communication efficiency or diverse quality of service (QoS) requirements in aggregating events. This paper proposes a new architecture that integrates the publish-subscribe architectural pattern with software-defined network technology for urban monitoring applications. The architecture enhances monitoring applications with capabilities of distributed processing and detection of complex events. It also enables application developers to define QoS requirements and supports the TESLA complex event specification language. The main focus of our work is on energy and network efficiency. The simulation results demonstrate significant improvements in energy consumption and data packet traffic compared to three close baselines.

INDEX TERMS Fog computing, Internet of Things, edge computing, smart city, software-defined networking, publish-subscribe architecture, complex event detection, event-based system.

I. INTRODUCTION

Smart cities are one of the main ecosystems of the Internet of Things (IoT) applications [1]–[6], where edge devices with a wide range of sensors and diverse capabilities are employed for monitoring the city environment and reporting ongoing events. Different organizations (e.g., fire department or emergency medical services) define their events of interest in this environment according to their missions. For instance, the fire, police, and medical emergency services look for specific traffic incidents from different perspectives. Simple events are the primary building blocks of event-based monitoring systems. They are usually aggregated to form a complex event that provides an in-depth overview of ongoing phenomena of interest (e.g., traffic congestion) detected based on a collaboration of several traffic sensors. The complex events are defined according to applications' objectives and have different QoS requirements.

The associate editor coordinating the review of this manuscript and approving it for publication was Chi-Yuan Chen.

The tight coupling of edge devices and urban applications makes the management of smart city applications a challenging problem. In this multi-tenant environment, edge devices report their detected events to different applications, and each application needs event notifications from various edge devices distributed across a city. In this paper, we address the problem of decoupling simple event producers (edge devices) from complex event subscribers (smart city applications) in a multi-tenant environment with diverse QoS requirements while considering the resource limitations of devices. The problem faces several complications. First is the diversity of applications and their various QoS requirements [7], while applications may dynamically subscribe to an event or unsubscribe. The next challenge is resource limitations of edge devices in energy, communication, and processing [8]–[10]. Finally, network dynamics (e.g., edge nodes' leaving and joining) introduce additional challenges.

Complex Event Processing (CEP) systems can be categorized into centralized and distributed approaches. In centralized ones, all events are processed in a central node.

The main drawback is high messaging overheads due to the submission of all events to a central processing node. However, this category of approaches benefits from central coordination mechanisms among edge devices that lead to the efficient management of nodes. On the other hand, distributed approaches take advantage of in-network processing for aggregating simple events that elevate their efficiency. However, they require a coordination mechanism, which can be centralized or distributed among edge and processing (fog) nodes to handle network dynamics and process tasks.

Researchers address the problem from different viewpoints. The challenge of decoupling event producers from consumers is addressed from the middleware perspective [11]–[14]. They show that information exchange among edge devices can benefit from a publish-subscribe architectural pattern. However, these works did not fully address event routing and QoS requirements challenges. From the networking viewpoint, several works integrated Software-Defined Networking (SDN) [15], [16] with a publisher-subscriber pattern [17]–[19]. Their main goal was to separate the control plane that handles network dynamics from the data plane in charge of disseminating the detected event notifications. The utilization of SDN technology helps reduce unnecessary message passing and provides efficient message routing. Although they addressed network dynamics and efficient data exchange challenges, they did not consider several aspects, including processing complex events, application diversity, and different QoS requirements. Finally, from the complex event detection perspective, some research [20]–[24] concentrated on detecting complex events using CEP features, such as event definition rules, aggregating event notifications, pattern matching, and stream processing. Additionally, researchers introduced complex event detection languages, such as TESLA [25], to facilitate the development of CEP systems [26].

This paper proposes a new SDN-based complex event detection architecture for urban monitoring applications that tackles the following challenges:

1) The tight coupling of edge devices and urban applications in a multi-tenant environment
2) The resource limitations of edge devices (i.e., energy and communication resources)
3) Heterogeneity of edge nodes in the sensing capability as well as sensing quality
4) Network dynamics (i.e., edge nodes may join or leave the network.)

The devised architecture comprises edge, fog, and application server layers. We adapt a broker-based publish-subscribe architecture to decouple simple event publishers (edge nodes) from complex event subscribers (urban monitoring applications) in a multi-tenant smart city environment. We also exploit SDN technology to handle network dynamics by managing the network topology and optimum routing of event notifications. The integration of publish-subscribe architectural pattern and SDN technology leverages the delegation of detecting complex events to specific processing fog nodes. We also enhance the TESLA complex event process language to support QoS attributes in the definition of complex event rules. Furthermore, we implement a prototype of the architecture for an urban application as a proof of concept. Finally, we evaluate the performance results with [21], [22], [27] regarding energy efficiency, network traffic, delays, and the percentage of detected events. The main contributions of this paper are summarized as follows:

1) A CEP rule definition grammar is extended to enhance urban monitoring applications in defining their required sensing quality.
2) A new architecture is proposed that applies the extended CEP grammar to the publish-subscribe architecture and integrates them with SDN technology.
3) A number of modules (i.e., CEP rule pre-processing, publisher head selection, and optimum route calculator) are devised in the proposed architecture to prevent unwanted traffic and reduce energy consumption.
4) A prototype of the proposed architecture is implemented, and its performance is extensively evaluated against the state of the art methods.

The remainder of this paper is organized as follows: In Section II, we review some of the well-known and recent related works. Then, we describe the system model in Section III. Our proposed architecture is described in Section IV. Section V introduces our prototype implementation, followed by the performance evaluation results in Section VI. Finally, Section VII concludes the paper.

II. RELATED WORKS

Researchers have devised various edge/fog-based architectures in recent years. Adeniran *et al.* [28] introduce an edge-enabled architecture. The edge layer is designed with the minimum number of edge devices and optimum connectivity structure through several optimization formulations to identify edge servers' placement and connectivity structure. Hasnat *et al.* [29] discuss the data analytics aspect of the edge computing platform and focus on situational awareness and its role in enhancing edge computing. Lan *et al.* [30] propose a framework to enable computation-intensive and delay-sensitive applications in smart cities, creating on-demand process engine data flow spanning multiple device layers with different resource constraints by leveraging the data-flow programming model. The researchers in [31] study different programming frameworks of fog systems, addressing challenges such as heterogeneity, scalability, and mobility with respect to the fog architecture and application types. Naranjo *et al.* [32] introduce Fog Computing Architecture Network (FOCAN), in which applications are running on things that jointly compute, route, and communicate with one another through the smart city environment.

The contributions of our work lie at the intersection of three research topics of publish-subscribe middleware, software-defined networking, and complex event processing. In the

TABLE 1. An overview of the related works.

Work	Middleware	Networking	CEP	Advantages	Limitations	Distributed(D) /Centralized(C) /Hybrid(H)
Gaamel et al. [11]	✓	✗	✗	1- Reduce energy consumption 2- Reduce end to end delay	1- Event quality of service did not consider 2- Event routing did not consider 3- Complex events did not consider	D
Davis et al. [12]	✓	✗	✗	Supports different quality of services	1- Event quality of service did not consider 2- Event routing did not consider 3- Complex events did not consider	C
Wang et al. [17]	✓	✓	✗	1- The decoupling of publishers and subscribers 2- Supports routing and network management	1- Event quality of service did not consider 2- Complex events did not consider	C
Shi et al. [18]	✓	✓	✗	1- The decoupling of publishers and subscribers 2- Supports routing and network management 3- Enabling Specific QoS for message delivery.	1- Event quality of service did not consider 2- Complex events did not consider	H
Park et al. [19]	✓	✓	✗	1- The decoupling of publishers and subscribers 2- Supports routing and network management 3- Reduces network congestion in MQTT	1- Event quality of service did not consider 2- Complex events did not consider	H
Kohler et al. [20]	✓	✓	✓	Application specific programming language	1- Only P4 language is supported 2- Not suitable for resource constrained devices	H
Esposito et al. [21]	✗	✗	✓	1- Supports complex event detection 2- The decoupling of publishers and subscribers	It has communication overhead	D
Khazael et al. [27]	✗	✗	✓	1- Communication overhead has been minimized 2- Supports complex event detection 3- The decoupling of publishers and subscribers	It has communication overhead	D
Meslin et al. [22]	✓	✗	✓	1- Supports complex event detection 2- The decoupling of publishers and subscribers	Event routing management did not consider	H
Guzel et al. [24]	✓	✓	✓	1- Supports complex event detection 2- The decoupling of publishers and subscribers	Application specific. Focused on air pollution.	H
Our work	✓	✓	✓	1- Supports Complex event detection 2- Supports various types of applications 3- The decoupling of publishers and subscribers 4- Routing and networking management	Centralized SDN controller	H

following, we review some of the most related works on each topic. Table 1 summarizes the related works.

A. MIDDLEWARE APPROACHES

Data Distribution Service (DDS) is an interoperable publish-subscribe solution to support real-time distributed systems. Gaamel *et al.* [11] analyze different DDS approaches for wireless sensor and actuator networks. The comparative study includes default tinyDDS (DefTDDS), broker-less tinyDDS (BLTDDS), and hybrid tinyDDS (HyTDDS). Their experiments reveal that BLTDDS demonstrate the best throughput. BLTDDS and HyTDDS protocols present a higher packet delivery ratio than others. BLTDDS outperforms in terms of end-to-end delay, and finally, HyTDDS was the most energy-efficient protocol. They further propose a broker-less protocol, called Enhanced Energy-Aware TinyDDS (E-EATDDS), to improve energy efficiency. However, their work does not provide any QoS support, and it delegated the responsibility of QoS support to application developers.

Davis et al. [12] introduce publish/subscribe-based mechanism for wireless sensor networks to enable four different QoS levels regarding packet delivery and timeliness. The first level is the best effort way of packet delivery. The next level, called reliable packet delivery, uses re-transmission timeout according to additional information from subscribers, such as packet delivery ratio. The third level aims for energy efficiency for data aggregation. Finally, the last level brings timeliness in packet delivery by using a deadline mechanism. The limitation of their mechanism is that it uses static routing and does not address network dynamics.

This group of works concentrates on middleware aspects by providing mechanisms for decoupling event publishers from subscribers. However, they do not address network-related issues such as routing, topology management, and network dynamics.

B. NETWORKING-BASED APPROACHES

The second group of works focuses on networking aspects.

Wang *et al.* [17] introduce an SDN-based Publish/Subscribe system (SDNPS). It is a communication platform combining SDN with topic-based publish-subscribe architecture in which any authorized IoT service can be a publisher of an event, or it can subscribe to/unsubscribe events. SDNPS also orchestrates event routing by forwarding events to subscribers. According to the platform, sensors and actuators are connected to local processing brokers equipped with an event broker network module. The module is responsible for mediating events among IoT services using topic-based matching under the SDN model. Nonetheless, SDNPS does not address the processing of complex events.

Shi *et al.* [18] present SDN-Like, a topic-based publish/subscribe middleware architecture, to facilitate users to define their differentiated QoS requirements. SDN-Like encodes events and their priorities in packet headers that facilitate prioritizing event packets. Additionally, it defines priority queues on OpenFlow switches to provide differentiated services. The authors also introduce eXtreme Gradient Boosting (XGBoost), a machine learning model, to predict the queuing delays. Then, they implement a two-layer queue management mechanism to guarantee the reliability of differentiated services. Although their work empowers users to define differentiated QoS, their middleware does not support the processing of complex events.

Message Queuing Telemetry Transport (MQTT) is a publish-subscribe network protocol that transports messages between devices. The standard MQTT may cause network congestion in the presence of a centralized broker. Park *et al.* [19] introduce Direct Multicast-MQTT (DM-MQTT) that uses a hierarchical structure, where edge brokers are slaves, and the central broker is a master. All edge nodes are connected to an MQTT slave broker. A slave broker is responsible for collecting edge information and delivering it to the master edge broker. The edge information includes IP address, topic, QoS level, and emergency data flag of an edge device. The master broker sends all the edge information to an SDN controller to analyze and create a group table to set paths between different edge networks. Additionally, the controller categorizes groups based on three levels of QoS, considering that a sender and a receiver must agree upon the same QoS. Moreover, multicast trees are established between the publisher and the subscribers to bypass the centralized broker whenever multiple nodes subscribe to a topic. A multicast tree is a core-based tree whose root is the node with the most negligible average delays and is selected by the SDN controller as a rendezvous point. Then, the SDN controller generates a bidirectional multicast path between edge devices and the rendezvous point and forwards it to the SDN switch. Finally, the information is disseminated according to the core-based tree.

Inspired by SDN technology, the researchers adopt the idea of separating a control plane from data planes to address challenges of network dynamics such as network congestion in routing packets and new policy enforcement using SDN technology. While they mainly consider simple events, the processing of complex events is not fully addressed.

C. CEP APPROACHES

The third group of work concentrates on complex event processing capabilities.

Kohler *et al.* [20] introduce a CEP solution that uses in-network computing capabilities of network nodes along the communication path, reducing communication latency in detecting events. They use Programming Protocol-independent Packet Processors (P4) language and introduce the P4CEP compiler. The compiler receives a CEP design configuration and creates the corresponding P4 source code. The CEP design configuration contains various information that defines header fields, parser instructions for primary events packets, declarations of window operators, and event definition rules. Although the solution supports complex event rules, QoS requirements for each complex event cannot be declared explicitly.

Esposito *et al.* [21] present a broker–less communication method for their publish/subscribe model. Their method uses beaconing to perform low-level multicasting for data dissemination and signaling data exchange. Initially, a node beacons *msg_join* message to announce its interest in a specific topic. Then, the node creates a data structure called routing topic table. The data structure contains various information about neighboring nodes, the topics of interest, and received notifications for the topics. When a node receives a join message, it updates its internal routing topic table based on the topic of the message. Then, the node propagates the change by issuing an *msg_update*. The neighboring nodes, receiving the update message, amend their routing topic table and send an update message to the neighbors. The introduced protocol uses a flooding mechanism to deliver the message to all interested subscribers from the publisher's reachable area. It also uses in-network data fusion, but the beaconing method of delivering a message to all subscribers reduces the network lifetime due to inefficiency in message passing. Moreover, the approach does not scale due to the flooding mechanism.

Khazael *et al.* [27] introduce a coordination protocol to overcome the unnecessary re-beaconing problem in broker–less communication methods. They introduce a structure for communication packets and use packet headers to store a list of destination nodes. Furthermore, each node maintains a routing topic table. A node announces its presence by beaconing a *join_message* to join the network, and by receiving a *join_message*, other nodes add the neighbor's address to their tables and exchange their table information. A node can create a subscribe message and use beaconing to deliver the message to their neighbors to subscribe to an event. By receiving a subscribe message, the neighbor's address will be added to the routing topic table. To propagate an event, a publish message can deliver to most adjacent nodes by one-time beaconing since the nodes can be within the coverage area of multiple neighboring subscribers. However, nodes receiving this packet do not become aware that their neighbors may already receive the same. Thus, they process the list of recipients (destination nodes) in the packet header to decide re-beaconing, which leads to reducing unnecessary communication. Although this approach reduces unnecessary communication, the scalability of the method cannot be guaranteed due to the flooding nature of the solution.

Mobile Urban Sensoring and Actuation Network (MusaNet) [22] is a scalable three-layer middleware that supports CEP for smart city applications. In the first layer, mobile objects provide an interface to sensors and actuators. The mobile objects also perform local data processing primitives such as filtering, capturing, and detection. This layer is developed based on Mobile-Hub to support connectivity. The second layer comprises processing nodes and gateways, which are implemented using ContexNet middleware. The processed information in the second layer is passed to the third layer, called the storage layer, using network infrastructure. The third layer is responsible for structured storage and support for information queries. It also provides an interface to applications that are out of the platform. However, it does not consider optimizing energy consumption and traffic reduction, which are essential in resource-constrained IoT environments. Our work addresses these aspects by devising modules, including network topology management and optimum route calculator using SDN technology.

Guzel *et al.* [24] introduce an air quality prediction framework. They divided the functionalities into three layers of IoT, fog, and cloud. The IoT layer is responsible for sensor activities, and it reports sensor readings (including a timestamp, node ID, pollutant ID, and momentary reading of the pollutant) to the fog layer. According to the time windows using CEP tools, the fog layer calculates pollutant concentrations and extracts continuous pollutant concentration patterns of individual IoT nodes. It is also responsible for pollution prediction and storing the perdition model for each sensor. Finally, the cloud layer performs pattern matching by analyzing the pollutant concentration patterns of IoT nodes and identifies the nodes with similar patterns. However, this work did not address any QoS features.

The surveyed works in Section II-C focus on the CEP features of an IoT environment. However, these methods leave the handling of QoS concerns to application developers. Additionally, they do not fully address the challenges of network dynamics.

III. SYSTEM MODEL

This section introduces the system model of our smart city monitoring environment. It describes the event model, including simple and complex events, introduces the network model and its comprising elements, and defines the energy model. Finally, it formulates the problem.

A. EVENT MODEL

Edge (sensor) nodes are the primary sources of events in smart city monitoring applications. These nodes are equipped with embedded sensing devices to measure environmental phenomena, such as air quality sensors and traffic sensors. The nodes also have heterogeneous QoS specifications [33], such as sensing accuracy and sampling rate. A simple event happens when a measured value of a sensing device lies within a predefined range. The detected simple event is reported in the form of a notification. In our event model, each type of event is associated with an event topic. We denote α_i as a simple event notification comprising of an event topic (α_i^e), measured value (α_i^v), detection timestamp (α_i^t), node_location (α_i^l), and the set of meta-information of the publisher such as QoS metrics.

Edge nodes advertise their sensing capabilities in event topics, which helps the interested entities subscribe. Then, the edge nodes publish their detected events by sending notifications to the subscribers of the event. We denote p_i as a sample publisher of a simple event and M_i as the set of meta-information associated with the publisher p_i.

Simple events provide a confined perception over a monitoring area, while complex events provide a more inclusive view. Complex events can be detected by aggregating simple events in a specific geographical area within a specific time window. They are defined in the form of application-specific rules using an Event Processing Language (EPL) such as TESLA. Figure 1 shows the fire complex event.

```
Define Fire(area:String,
    measuredTemp:Double)
From Smoke(area="StreetA") and Each
     Temp(tmp_value>50 and area="
    StreetA") within 30 from Smoke
where area=Smoke.area and
    measuredTemp=Temp.tmp_value;
```

FIGURE 1. Definition of Fire as a complex event. From TESLA language on T-Rex project [25].

Several preconditions have to be checked to detect a complex event. The following notations introduce these preconditions. We denote r_i as a sample complex event rule, A_i as its set of required simple events, T_i as an acceptable time window, L_i as geographical area, and Q_i as its set of predefined QoS requirements.

In our event model, we consider four preconditions to detect a complex event. The first precondition for detecting a complex event rule (r_i) is to check if the set of predefined simple events (A_i) has occurred. In Eq. 1, N_x represents the set of received simple event notifications.

$$Precondition1 : \forall \alpha_i \in A_i, \quad \exists \alpha_j \in N_x : \quad \alpha_i^e = \alpha_j^e \qquad (1)$$

QoS requirements ($\forall q_i \in Q_i$) is the second precondition. Let Γ be the set of simple event notifications that match one of the simple events in A_i. Each QoS requirement must be satisfied by at least one meta-information (such as m_j) of a simple event notification (such as $\alpha_j \in \Gamma$). Eq. 2 presents a formal definition of the precondition.

$$Precondition2 : \forall q_i \in Q_i, \quad \exists \alpha_j \in \Gamma : \quad q_i \triangleq m_j \qquad (2)$$

The next precondition considers the time correlation of the received simple event notifications. This precondition checks if the time difference between the simple event notifications is bound to the acceptable time window (T_i) predefined in the complex event rule (r_i). In Eq. 3, $|\alpha_j^t - \alpha_k^t|$ shows the time difference between two simple event notifications α_j and α_k.

$$Precondition3 : \forall \alpha_j, \alpha_k \in \Gamma : \quad |\alpha_j^t - \alpha_k^t| < T_i \qquad (3)$$

The last precondition is the spatial correlation of simple event notifications. Eq. 4 presents the precondition where $||\alpha_j^l - \alpha_k^l||$ shows the Euclidean distance between the publishers of α_j and α_k.

$$Precondition4 : \forall \alpha_j, \alpha_k \in \Gamma : \quad ||\alpha_j^l - \alpha_k^l|| < L_i \qquad (4)$$

Upon detecting a complex event r_i, a notification is generated and transmitted to its subscribers. We denote S_i as the set of subscribers for a complex event rule r_i. These subscribers can be a monitoring application or other CEP module(s).

B. NETWORK MODEL

We consider a network structure comprised of three layers of edge nodes, fog nodes, and application servers.

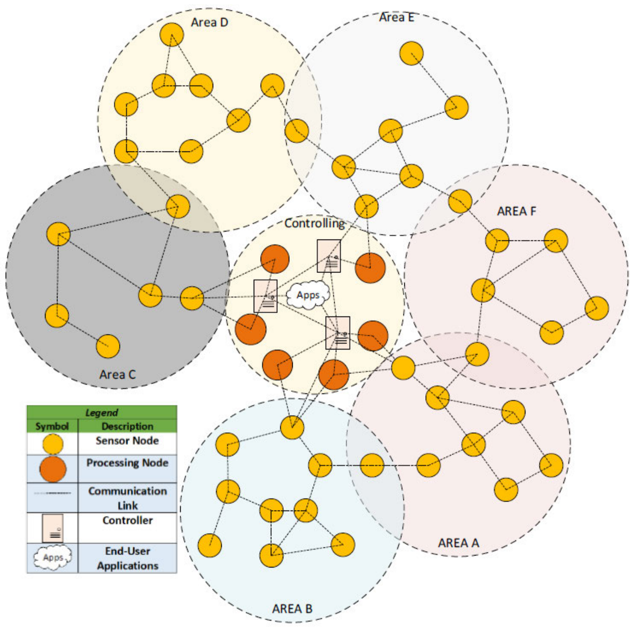


FIGURE 2. **Clustered network model.**

The edge layer includes edge nodes (sensors nodes) responsible for monitoring their surrounding environment and sending simple event notifications according to their sensing capabilities. The edge nodes have similar communication coverage and communicate with fog node(s) using IEEE 802.15.4. These nodes are resource-constrained and cannot take the burden of detecting a complex event. They are also clustered according to their deployed geographical region [34]. Figure 2 illustrates the clustered network model where each cluster is associated with at least one fog node. The edge nodes connect to a fog node to send event notifications. The associated fog node is selected according to the minimum hop count and energy efficiency considerations.

Fog nodes constitute the middle layer (fog layer) of the network structure. These nodes are resource-rich and have two primary responsibilities. The first is to monitor the status of edge nodes and collect simple event notifications. The second is to process the notifications, detect complex events, and send complex event notifications to the upper layer subscribers.

Application servers (upper layer) represent subscribers' roles from a publish-subscribe viewpoint. They interact with end-users and provide monitoring services to municipal organizations. They also communicate with fog nodes to subscribe to complex events or receive complex event notifications using application programming interfaces (APIs).

C. ENERGY MODEL

Edge devices are energy-constrained nodes and consume their energy for sensing, processing, and communication activities. The energy consumption of sensing activities is highly dependent on embedded sensor devices. Additionally, the sampling rate and a resolution accuracy of the sensor device is an influential factor. The processing activities spend energy in a processor (e.g., micro-controller) and memory units. Finally, communication activities consume a significant amount of energy for transmitting and receiving data between nodes. Among these activities, communication is the primary source of energy consumption. Therefore, we consider transceivers to consume the most significant portion of edge/fog nodes' energy.

The detection of a complex event (e_j) requires sending/receiving several simple event notifications. We denote E^j_{simple} as the sum of consumed energy to detect and transmit the simple event notifications by the edge nodes. Moreover, several message passings are required for multi-hop transmission of simple event notifications to fog nodes since all the edge nodes are not directly connected to a fog node. We denote the consumed energy for multi-hop communication as E^j_{multi}. Finally, exchanging the processed intermediate data between fog nodes needs energy E^j_{inter}. We present the consumed energy for send/receive of the intermediate packets in E^j_{multi}. The energy efficiency in detecting a complex event j is formulated in Eq.5.

$$\min \sum (E^j_{simple} + E^j_{multi} + E^j_{inter}) \tag{5}$$

IV. PROPOSED ARCHITECTURE

This section introduces the structural and behavioral view of our new SDN-based complex event processing architecture for smart city monitoring applications. We integrate the publish-subscribe architectural pattern with SDN technology in a three-layered architecture. The network nodes in each layer have different responsibilities fulfilled by interactions between the architectural components. We introduce these interactions by describing the workflows of various processes, including publisher registration, subscription, matching, publisher head node selection, and notification dissemination.

A. STRUCTURAL VIEW

The network nodes are organized into three layers: *edge layer*, *fog layer*, and *application server layer* aligned with the network model introduced in Section III-B. The edge layer nodes have similar responsibilities, while the nodes in the fog layer can be a processing node, a broker, or a controller node. The application server layer comprises monitoring applications that connect our proposed architecture to municipal organizations. Each network node includes multiple components that are to be described in the following. Figure 3 shows an overview of the proposed architecture.

1) EDGE LAYER

Edge nodes are the primary source of monitoring information by sensing the environment and reporting the detected simple events. These nodes have heterogeneous sensing capabilities and support different QoS requirements such as sensing resolution and accuracy. They use *pub/sub service* to communicate at the service level with the nodes in other

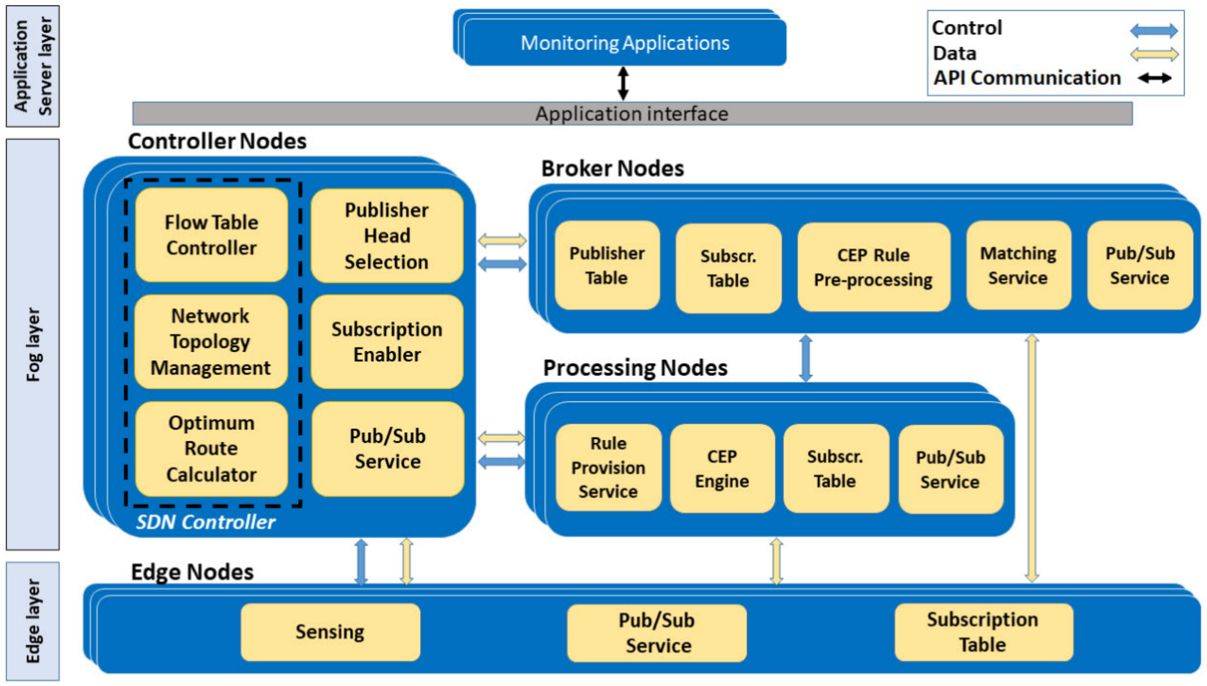


FIGURE 3. An overview of the proposed architecture.

layers. Additionally, edge nodes maintain a *subscription table* that keeps track of the subscribers interested in the event topics that can be published.

2) FOG LAYER

The middle layer (fog layer) handles decoupling complexities of publishers and subscribers by using event brokers and integrating them with SDN technology. This layer is responsible for handling network dynamics, decoupling publishers from subscribers, and providing computing resources to process complex events. Fog layer nodes interact with the upper and lower layers in various cases. These nodes receive the detected simple events from the edge layer and send the complex event notifications to the application server layer. They also accept new subscriptions for complex event rules from the application server layer. Finally, they send network management control packets to edge layer nodes. The Fog layer comprises three types of nodes: a *broker node*, a *controller node*, and *processing nodes*.

(i) A **broker node** addresses the decoupling issues of publishers and subscribers by utilizing five components. The first one is the *publisher table* component that registers the capabilities of edge nodes. The table contains the publisher's ID, the event topics that it can produce, the location of its monitoring area, and its supported QoS requirements. The *subscriptions table* is the second component that keeps track of all subscriptions. Each entry in the table contains the ID of the subscriber, the requested complex event rule, QoS requirements, and a list of eligible publishers that can satisfy the rule. The next component is called *rule pre-processing*, which is used to break down a complex event rule into a set of simple event subscriptions. *Matching service* is the fourth component that has two functionalities of *subscriber matching* and *publisher matching*. The first one receives the information of a new publisher and extracts all subscribers that can benefit from the new publisher. The second one receives a set of simple event subscriptions and returns all publishers that can potentially satisfy the set.

(ii) A **controller node** is the second type of fog layer node that has two main responsibilities.

- The first responsibility is the network management that addresses network dynamics by implementing the SDN controller duties defined in SDN technology [35], [36]. The controller node uses three components to accomplish this responsibility. First, the *network topology management* component enables the controller node to be aware of the network configuration and communication costs between network nodes. The *optimum route calculator* is the second component that finds the best path from an edge node to its closest processing node. It is an essential component for the edge nodes that are not directly connected to a processing node. The last component is the *flow table controller* aiming to manage the network nodes' flow table [35]. Each network node has a flow table that consists of *matching-rule*, *action*, and *statistics* sections. When a node receives a packet, it looks up the matching rules to find an entry/entries that suit the incoming packet. Then, it performs the defined action in the entry. The *flow table controller* component manages all the flow tables. For instance, it sets the paths from edge nodes to processing nodes based on the calculated optimum route. It also manages the paths from simple event publishers to simple event subscribers as well as complex event publishers to complex event subscribers.

- The second responsibility is to enhance the SDN controller capabilities to support publish-subscribe aspects. The *publisher head selection* component selects the appropriate processing node(s) to process a particular complex event rule. The selection is based on the communication costs between the processing node and the edge nodes that satisfy the rule. The head selection process is described in Section IV-B5. The *subscription enabler* component receives the subscribed complex event rule and the selected processing node(s). It invokes the *rule provision service* of the processing node to configure the complex event rule. Finally, the *flow table controller* component is in charge of updating nodes' flow tables.

(iii) **Processing nodes** provide the computation resources to process the received simple event notifications. These nodes communicate with edge nodes using *pub/pub service* that facilitates marshaling of pub/sub messages. The received notifications are processed in the *CEP engine*, and the detected complex events are reported to the interested nodes stored in the *subscription table*. The *CEP engine* component runs a separate thread and creates an automata model for each registered complex event rule. *Rule provisioning service* is a service enabler that provision a thread in the *CEP engine* for listening to the related incoming notifications.

3) APPLICATION SERVER LAYER

The application server layer is the uppermost layer of the architecture. It comprises monitoring applications that provide the required services to the municipal organizations while having diverse QoS requirements. They act as gateways between the devised architecture and other smart city applications. They use APIs to interact with other smart city applications and use *notification services* to communicate with the fog layer.

B. BEHAVIORAL VIEW

The behavioral view of the proposed architecture elaborates the way various components collaborate to perform main processes such as publisher registration, publisher-subscriber matching, subscription, head node selection, and dissemination of event notifications.

1) THE CONTROL PLANE AND DATA PLANE

Each node must have a flow table containing a rule to send controlling packets to the controller. The path to the controller is constructed by initializing a beacon packet, exchanging, and collaborating with the neighboring nodes. When a node identifies its neighbors using beacon packets, it updates its shortest path to the controller. When the path from a node to the controller is constructed, the data plane sends a network event to the SDN controller requesting a command of packet flow.

2) PUBLISHER REGISTRATION PROCESS

Edge nodes advertise their capabilities to participate in detecting complex events. The registration process starts by creating a registration message that includes the node's ID, the event topics that it can produce, the location of its monitoring area, and the supported QoS. Then, the node sends the message to its closest broker node. The broker node extracts the message and adds the publisher to the publisher table. It also invokes the subscriber matching service. The service returns a list of subscribers that can benefit from the newly added publisher. Finally, the broker asks the controller node to update the flow table of the new publisher by sending a message to update its flow table rules. In a multi-broker case, brokers synchronize publisher tables to maintain consistency. Figure 4 shows the flow of actions.

3) PUBLISHER-SUBSCRIBER MATCHING PROCESS

Considering a complex event rule (r_i), the objective of the *publisher-subscriber matching* process is to find a list of candidate publishers that can satisfy the set of required simple events (A_i) for detecting r_i. We refer to this set as a subscription set. Then, it checks if the publishers can satisfy the QoS requirements (Q_i) of the rule. That is, the defined attributes of the publishers have to match the QoS requirements of the subscription set. The output of the process is the list of publishers that can contribute to detect the complex event. Algorithm 1 shows the pseudocode of the process.

4) SUBSCRIPTION PROCESS

The process is initiated by an application server that requests a subscription for a complex event rule from the closest broker. The request contains the ID of the subscriber, the complex event rule, and the associated QoS requirements. Then, the broker adds the request information to its *subscribe table*. The next step is to decompose the subscribed complex event rule into its comprising simple events (subscription set) performed in the *CEP pre-processing* component (Figure 5).

Next, the subscription set is sent to the *matching service*. The output of the matching service is a list of candidate publishers that can satisfy simple events in the subscription set. Then, the list of candidate publishers, the subscriber ID, and the subscribed complex event rule are sent to the *publisher head selection* component of the controller

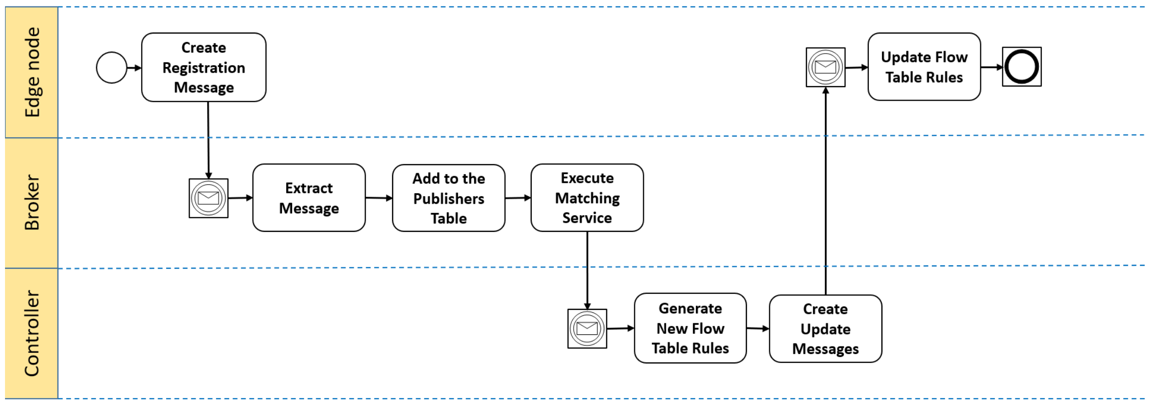


FIGURE 4. Publisher registration process.

Algorithm 1 Publisher Matching

```
**Trigger**: Receiving the subscription set from *rule pre-processing* component.
**Input:** Set of subscriptions $S$, Set of publishers $P$ (Currently stored).
**Result:** Set of $\mathcal{P}$ which can satisfy at least one rule in $S$
for each subscription $s \in S$ do
    for each $p \in P$ do
        if $s.topic = p.topic$ then
            if $s.topic.attributes \cong p.attributes$ then
                Append $p$ to the $\mathcal{P}$;
            end if
        end if
    end for
end for
return($\mathcal{P}$);
```

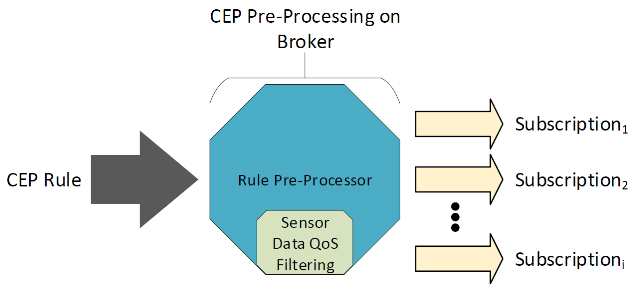


FIGURE 5. CEP rules pre-processing.

node. The publisher head is a processing node that is in charge of detecting the subscribed complex event. The details of the publisher head selection process are described in Section IV-B5. The output of the publisher head selection process is the ID of the selected processing nodes responsible for detecting the subscribed complex event rule. The output is used in the *subscription enabler* component that sends the complex event rule to the *rule provisioning service* of the publisher head to prepare the *CEP engine* component for detection and updating the subscriber table of the processing node. The *subscription enabler* also invokes the *flow table controller* to amend the flow table of the corresponding publisher nodes. Figure 6 shows the subscription process.

5) SELECTION OF PUBLISHER HEAD NODE

Processing nodes are in charge of collecting the event notifications from edge nodes. These notifications are aggregated to check whether a complex event has happened. It is crucial to select a processing node as close as possible to the group of edge nodes capable of detecting a complex event cooperatively to provide energy efficiency and fast event detection. We refer to this processing node as the *publisher head* and the group of edge nodes as *publisher nodes* since it acts as a head node for the local group of edge nodes (publisher nodes) responsible for detecting a complex event.

The system analyzes the shortest paths between the publisher nodes and potential publisher heads to select the *publisher head*. We demonstrate the process of selecting a publisher head by an example. Figure 7 illustrates a sample scenario with two fog layer nodes (labeled 1 and 2) along with 7 edge nodes (labeled 3 to 9). Let r_i be the subscribed complex event rule; the *matching service* of the broker node identifies nodes 3, 5, and 9 as the group of candidate simple event publishers for r_i. Then, the candidate list and r_i are sent to the *publisher head* component of the controller. The *publisher head* uses the *network topology management* and *optimum route calculator* components to find the proper processing nodes selected as a publisher head. The *optimum route calculator* component builds a shortest path table and introduces the nearest processing node as the publisher head. Figure 8 depicts the shortest path table where the number of hops between two nodes is presented. The table shows that node 1 is the closest processing node to the publishers of r_i. Thus node 1 is chosen because it has a smaller path length. If several processing nodes have the same path length, then the processing node with the highest energy level will be selected. In a more advanced case, the communication cost

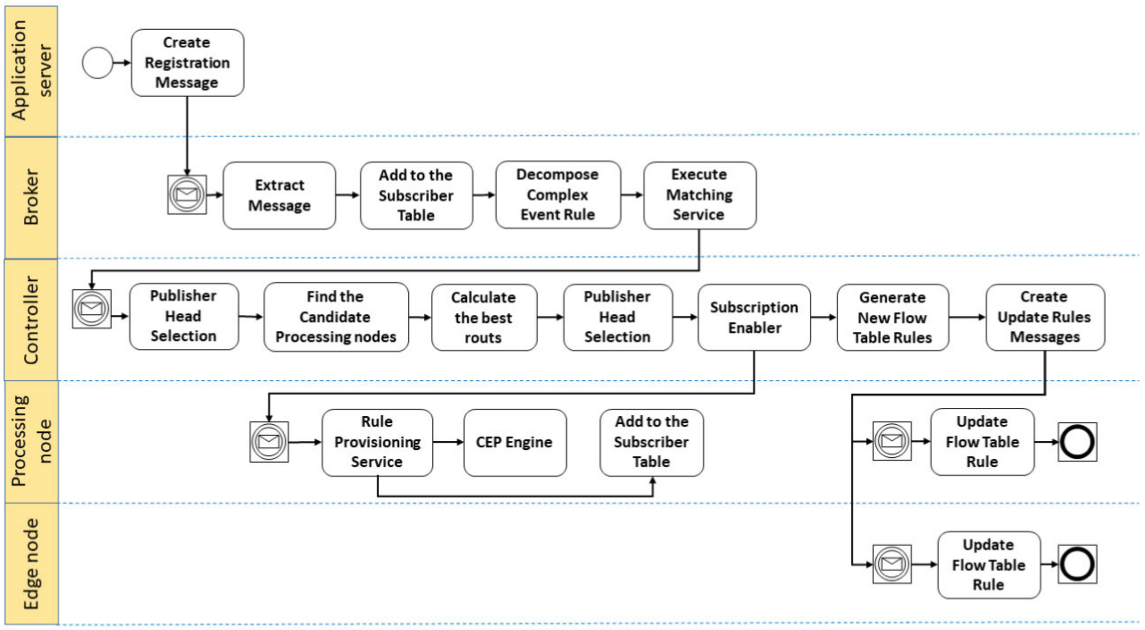


FIGURE 6. Subscription process.

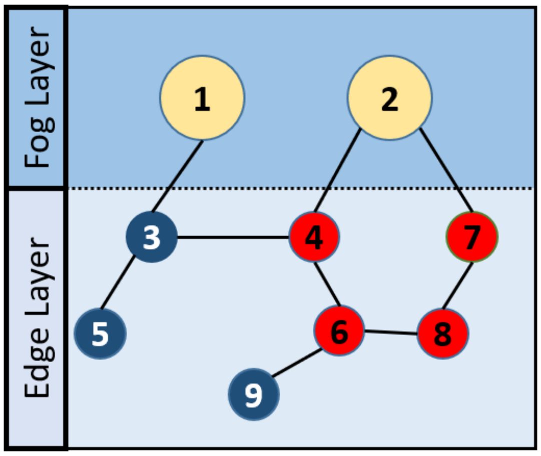
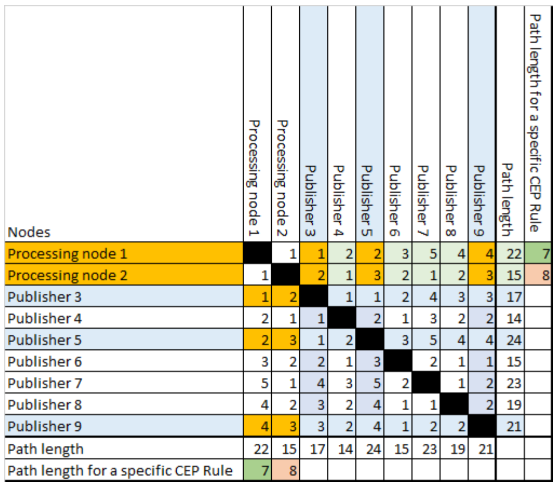


FIGURE 7. Sample publisher head selection scenario.



Nodes	Processing node 1	Processing node 2	Publisher 3	Publisher 4	Publisher 5	Publisher 6	Publisher 7	Publisher 8	Publisher 9	Path length	Path length for a specific CEP Rule
Processing node 1	■	1	1	2	2	3	5	4	4	22	7
Processing node 2	1	■	2	1	3	2	1	2	3	15	8
Publisher 3	1	2	■	1	1	2	4	3	3	17	
Publisher 4	2	1	1	■	2	1	3	2	2	14	
Publisher 5	2	3	1	2	■	3	5	4	4	24	
Publisher 6	3	2	2	1	3	■	2	1	1	15	
Publisher 7	5	1	4	3	5	2	■	1	2	23	
Publisher 8	4	2	3	2	4	1	1	■	2	19	
Publisher 9	4	3	3	2	4	1	2	2	■	21	
Path length	22	15	17	14	24	15	23	19	21		
Path length for a specific CEP Rule	7	8									

FIGURE 8. Shortest path calculation.

(e.g., link quality, energy consumption) can be used to select a suitable publisher head.

6) EVENT NOTIFICATION DISSEMINATION

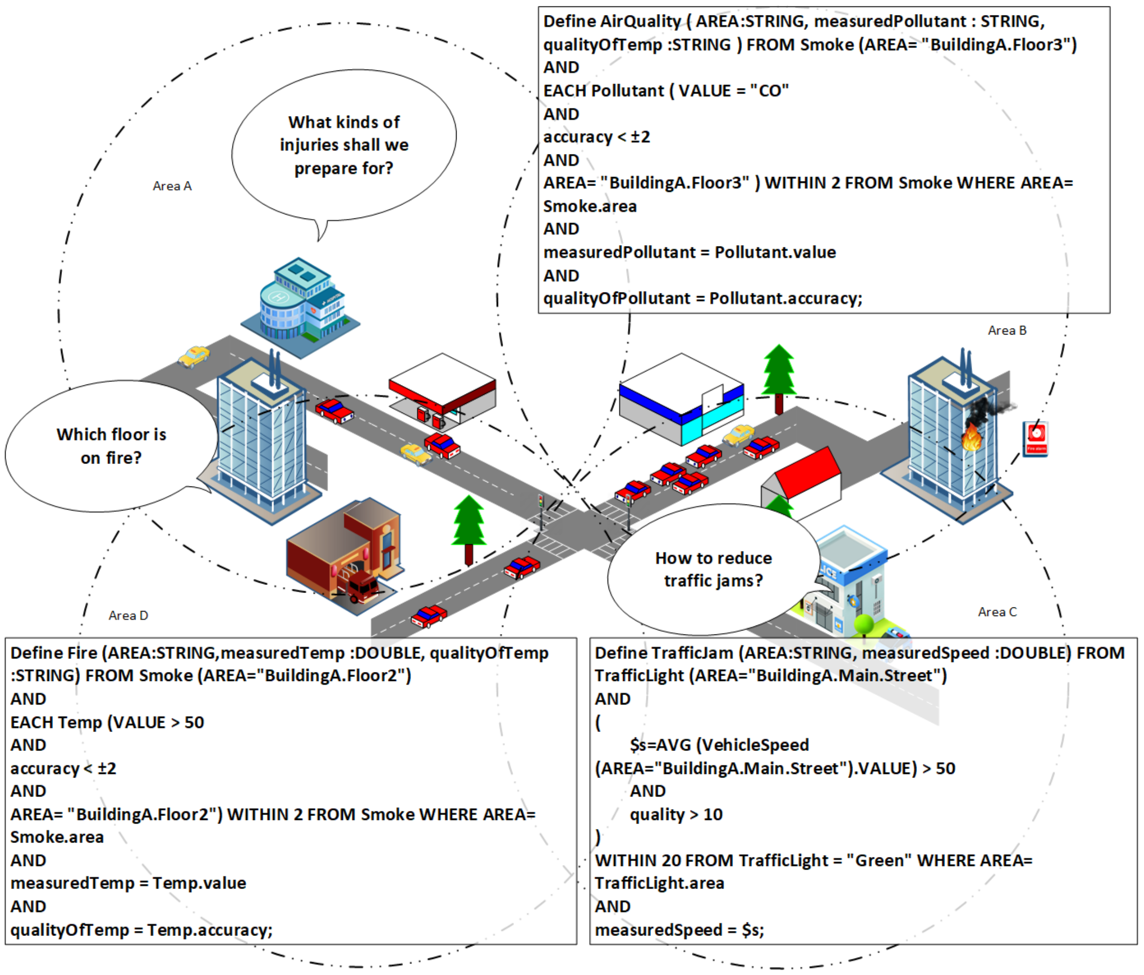
By the time an edge node p_i detects an event, it looks up the subscribers of the event in the *subscribe table* component and prepares a notification message using *pub/sub service*. Then, it looks up its flow table to send the notifications to the subscribers. When node x receives the notification, it checks its flow table to act. The action can be forwarding the notification to the nodes that have been previously configured in the flow table or extracting the message payload and process the notification to detect a complex event. Notifications of complex events follow a similar approach.

V. PROTOTYPE IMPLEMENTATION

This section describes the prototype implementation of our *SDN-based CEP* architecture and introduces the evaluation scenario. The prototype implementation is built on the *SDN-WISE* [37] networking solution integrated with the Cooja simulator[1] and *T-Rex* [38] engine for processing complex events.[2] SDN-WISE is designed for wireless sensor networks

[1] *https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja*

[2] The source code of the prototype implementation is accessible from: *https://github.com/BehnamKhazael/ComplexEventProcessingInSmartCity*


What kinds of injuries shall we prepare for?
Area A
Area B
Which floor is on fire?
How to reduce traffic jams?
Area D
Area C

```
Define AirQuality ( AREA:STRING, measuredPollutant : STRING,
qualityOfTemp :STRING ) FROM Smoke (AREA= "BuildingA.Floor3")
AND
EACH Pollutant ( VALUE = "CO"
AND
accuracy < ±2
AND
AREA= "BuildingA.Floor3" ) WITHIN 2 FROM Smoke WHERE AREA=
Smoke.area
AND
measuredPollutant = Pollutant.value
AND
qualityOfPollutant = Pollutant.accuracy;
```

```
Define Fire (AREA:STRING,measuredTemp :DOUBLE, qualityOfTemp
:STRING) FROM Smoke (AREA="BuildingA.Floor2")
AND
EACH Temp (VALUE > 50
AND
accuracy < ±2
AND
AREA= "BuildingA.Floor2") WITHIN 2 FROM Smoke WHERE AREA=
Smoke.area
AND
measuredTemp = Temp.value
AND
qualityOfTemp = Temp.accuracy;
```

```
Define TrafficJam (AREA:STRING, measuredSpeed :DOUBLE) FROM
TrafficLight (AREA="BuildingA.Main.Street")
AND
(
     $s=AVG (VehicleSpeed
(AREA="BuildingA.Main.Street").VALUE) > 50
     AND
     quality > 10
)
WITHIN 20 FROM TrafficLight = "Green" WHERE AREA=
TrafficLight.area
AND
measuredSpeed = $s;
```

FIGURE 9. The simulation environment.

and IoT applications, and it considers various edge node resource limitations. It is an open-source project [39] written in Java programming language with adequate documentation for developers to implement additional features. We extend SDN-WISE and add new functionalities according to the devised architecture. We also use the T-Rex CEP engine [38] and customize it to support QoS requirements of complex event rules. T-Rex engine is also an open-source solution written in C++. We re-write the engine in Java and deploy it in fog layer processing nodes. T-Rex engine supports the syntax of TESLA event processing language. We also implement a parser and use it in the broker node of the fog layer to pre-process the submitted complex event rules.

The evaluation scenario for the proof of concept is a smart city scenario. We consider a network area comprised of four overlapping geographical regions such as a road intersection, a park, or a shopping center, as depicted in Figure 9. The network nodes include several edge nodes, a broker, an SDN controller, and processing nodes. The edge nodes are randomly distributed over the regions. These edge nodes are heterogeneous in their sensing capabilities and organize a mesh network where each edge node is approximately connected to 8 other nodes.

We use the complex event rules used in [25] as a typical example and add QoS requirements. The monitoring applications that subscribe to the rule are interested in a *Fire* complex event defined as the case in which *smoke* is detected in the region of interest (*Street A*) followed by the sensing temperature of above 50°C. It must only take 30 seconds to capture temperature values of above 50 degrees, and the sensing accuracy of the temperature sensor is above *level 2* (as a sample accuracy level). The notification of the detected complex event is published in the form of:

$$\begin{aligned}&(\text{Area: } \textit{area},\\ &\text{Measured-temperature: } \textit{value},\\ &\text{Quality-of-temperature-sensor: } \textit{quality})\end{aligned} \tag{6}$$

VI. PERFORMANCE EVALUATION

This section provides the performance analysis of the devised approach in comparison with the baseline methods. We introduce the experimental setup and evaluation results from five viewpoints, followed by discussing the limitations.

A. EXPERIMENTAL SETUP

We use the smart city scenario described in Section V with a different number of edge nodes to assess the scalability of the proposed architecture ranging from 250 to 1000, equipped with two different sensors. We consider two levels of accuracy for the sensors to define different QoS.

Each experiment starts with an initialization phase to register the publisher nodes, capabilities, and supported QoS levels. We consider four different complex event rules, and the application servers subscribe to their complex events of interest. Once the setup phase has been completed, we generate four simultaneous complex events in random locations across the network area introduced in section V and iterate them 15 times. We repeated the experiments thirty times and averaged the results to reduce the effect of randomness. The average value and the 95% confidence interval are presented for each plot. In addition, reported results are based on the snapshots of the simulations captured at the end of the simulations with a predefined 15 minutes duration.

We select three approaches as the baseline methods for comparison. The first approach is *Event data exchange* [21] that uses the publish/subscribe paradigm and distributed coordination mechanisms for managing tasks. It is designed for IoT domain applications and uses complex event processing features compatible with our evaluation scenario. The approach is described clearly, which paves the way for accurate implementation. The second approach is *Distributed coordination protocol for event data exchange* [27] that introduces an efficient publish/subscribe communication protocol for IoT applications to eliminate unnecessary packet re-distributions. The third approach is *MusaNet* [22], one of the states of the art solution that uses a similar layering architecture to ours. It is well described and utilizes open-source solutions.

B. ENERGY CONSUMPTION

For the first evaluation metric, we measure the energy consumption of edge nodes during the experiments. Figure 10 shows the consumed energy of edge nodes for transmitting and receiving packets for *MusaNet*, *Distributed coordination protocol*, *Event data exchange*, and *SDN-based CEP*, where the horizontal axis is the network size. The results show that our devised SDN-based method consumes less energy compared to the baselines. The results also reveal that *Distributed coordination protocol* and *Event data exchange* have the highest energy consumption since they use a beaconing mechanism to disseminate data messages, resulting in high communication overhead and energy consumption compared to *MusaNet* and our proposed method (*SDN-based CEP*). Our proposed method performs 88% on average better than *Distributed coordination protocol*

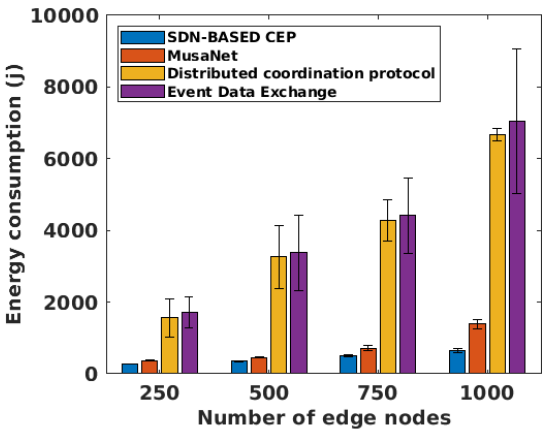


FIGURE 10. The energy consumption of edge nodes.

and *Event data exchange*. While the figure demonstrates *MusaNet* and *SDN-based CEP* have higher performances against distributed approaches (i.e., *Distributed coordination protocol* and *Event data exchange*), our proposed approach shows 24.9% and 53.5% improvement in the network sizes of 250 and 1000 compared to *MusaNet*. The first reason is the filtering of the publishers before they start to send the event notifications. Besides, using SDN controllers to manage the flow of events helps to reduce the number of packets transmitted for each flow of events, from the publishers to the subscribers. Thus edge nodes consume less energy compared to other solutions. Finally, selecting the closest processing node helps to reduce the number of intermediate nodes that a notification passes to reach a processing node.

C. NETWORK TRAFFIC

The second performance metric is the network traffic that reveals the system's efficiency in data packet transfers. Figure 11 presents the total number of disseminated data packets in the experiment for different network sizes. The figure shows our devised approach has 89.6% improvement on average compared to *Distributed coordination protocol* and *Event data exchange*. In addition, *SDN-based CEP* shows 39.7% better performance than *MusaNet*. Several factors help improve, such as using the optimum route calculator component and selecting an appropriate publisher head node. The central management of the publishers and subscribers reduces the communication overheads compared to *Distributed coordination protocol* and *Event data exchange* methods. Moreover, utilizing SDN to manage the edge layer and applying pre-processing to the rules helped reduce unwanted packets compared to *MusaNet*.

D. DELAY ANALYSIS

We measure three delay-related metrics, including publisher's registration delay, subscription delay, and propagation delay of simple event notification.

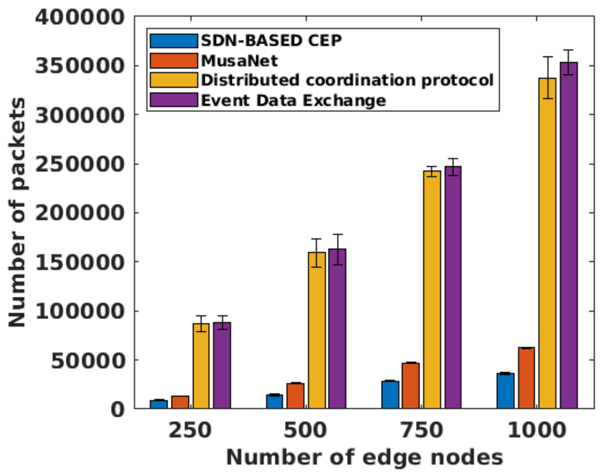


FIGURE 11. Total number of data packet transfers.

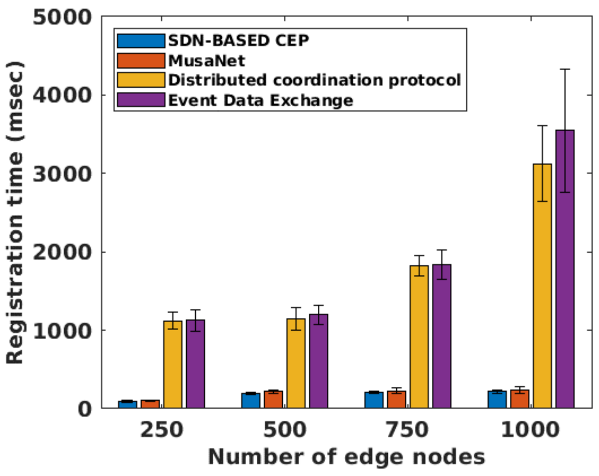


FIGURE 12. Publisher registration delay.

1) PUBLISHER REGISTRATION DELAY

A new edge node has to send a registration message that advertises its sensing capabilities and all the required information to be recognized by the system once it joins the smart city monitoring system. This registration process takes some time which we refer to it as a *publisher registration delay*. Figure 12 demonstrates publisher registration delays for *SDN-based CEP* and the baseline methods where the horizontal axis presents the network size, and the vertical one shows the delay time in milliseconds. The figure exhibits that the proposed approach and *MusaNet* significantly outperform *Distributed coordination protocol* and *Event data exchange*. The main reason is that in contrast with distributed approaches, all the nodes have to recognize the new edge node. The other two approaches use a centralized coordination mechanism that helps faster publisher registration process. In addition, It shows that *SDN-based CEP* performs a faster registration process rather than *MusaNet*. This is due to the nodes' flow table rules that lead the registration message to the shortest path towards a broker node.

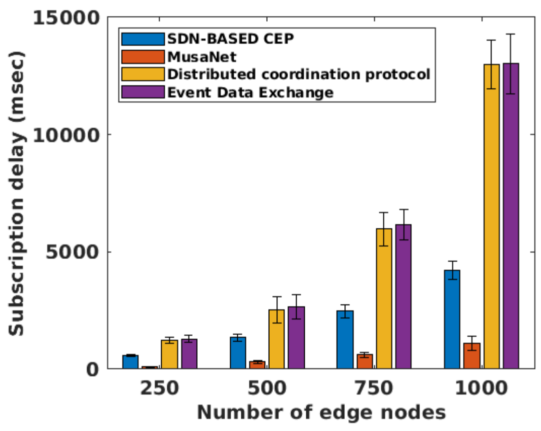


FIGURE 13. Subscription delay.

2) SUBSCRIPTION DELAY

Subscription delay is the time from subscribing to a complex event by a monitoring application until the system sets up the complex event rule in the CEP engine and subscribes to all the required simple events on edge nodes. Figure 13 shows the results where the horizontal axis is the network size, and the vertical one is the subscription delay in milliseconds. The figure shows that *Distributed coordination protocol* and *Event data exchange* have the highest subscription overhead due to the costly process of subscribing simple events on edge nodes. Our proposed approach ranks second, indicating a higher subscription overhead compared to *MusaNet*. This is the main trade-off of our devised architecture for achieving better energy efficiency and network performances. The selection process (Section IV-B5), responsible for choosing the best processing node, causes delays and is the main reason for the under-performance of our proposed method.

3) PROPAGATION DELAY OF SIMPLE EVENT NOTIFICATION

Edge nodes report their detected simple events in the form of simple event notifications. These notifications propagate in the network to reach a processing/aggregation node to be checked for detecting possible complex events. The propagation delay is the performance metric showing how efficiently and fast a simple event notification can reach the processing/aggregation node. The propagation delay of a simple event notification is the time from creating an event notification from an edge node until the processing/aggregation node receives the notification. Figure 14 shows the simple event propagation delay of *SDN-based CEP* and baseline methods where the X-axis presents the network size, and the Y-axis is the measured delay in milliseconds. The figure shows a significant overhead in *Event data exchange* and *Distributed coordination protocol* approaches that generate many replica messages, leading to a long queuing delay for incoming packets that have to be processed by each node, and consequently, extend the propagation delay compared to the central approaches. In addition,

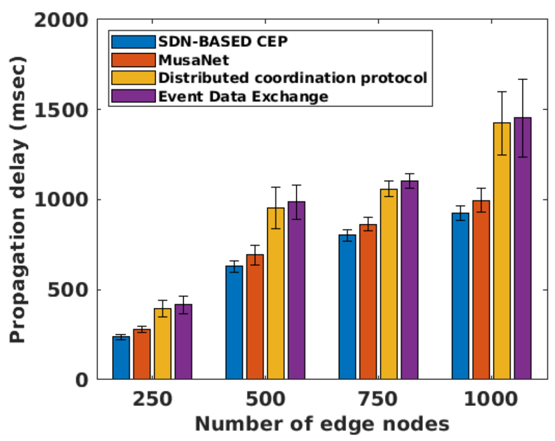


FIGURE 14. Simple event notification propagation delay.

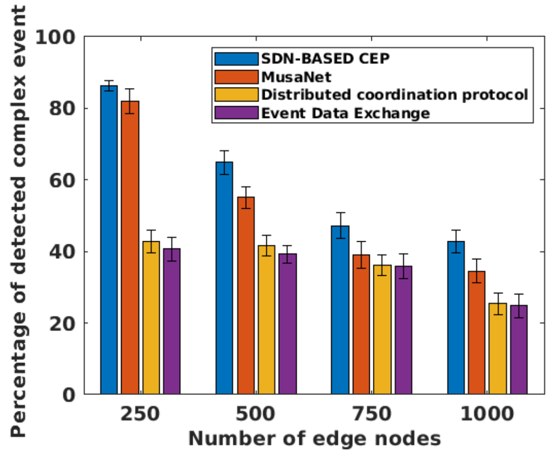


FIGURE 15. Percentage of detected complex events.

the results indicate marginal improvements for *SDN-based CEP* compared to *MusaNet* from 14.9% in 250 nodes to 7.2% in the largest network. The main reasons are the selection of the closest processing node to edge nodes and avoiding unnecessary packets dissemination which causes network traffic and, as a result, makes packets transmission delay.

E. PERCENTAGE OF DETECTED EVENTS

We study the percentage of generated complex events that have been successfully detected. Figure 15 shows the results in which *SDN-based CEP* and *MusaNet* perform better than distributed approaches to detect the generated complex events. In contrast, *Distributed coordination protocol* and *Event data exchange* miss more than 58% in the network size 250 and 74% in the network size 1000 due to the excessive number of networking overheads that produce long queues of incoming packets and network congestion. *SDN-based CEP* shows up to 7.6% better performance than *MusaNet*. Our approach performs better than the *MusaNet* in detecting complex event, since *SDN-based CEP* pre-processes the detection rules to avoid unnecessary events to reach CEP engines.

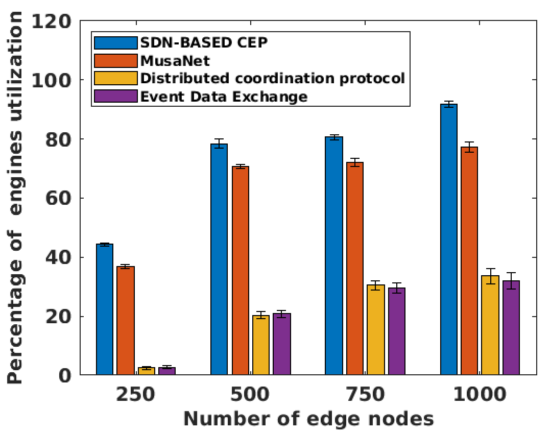


FIGURE 16. Percentage of engines utilization.

F. PERCENTAGE OF ENGINES UTILIZATION

The last evaluation metric investigates the average percentage of complex event engine utilization when engines are busy processing received published notifications. Figure 16 demonstrates the results of the four methods and shows that *SDN-based CEP* and *MusaNet* utilized engines more than *Distributed coordination protocol* and *Event data exchange*. By considering both the percentage of detected events and the engine utilization, the figures indicate that *SDN-based CEP* and *MusaNet* deliver event notifications to the subscribers more efficiently than the distributed methods (i.e., *Event data exchange* and *Distributed coordination protocol*). The reason is the inefficiency in delivering event packets that causes excessive network traffic and network congestion. The figure also shows *SDN-based CEP* utilizes engines up to 14.4% compared to *MusaNet* in the largest network.

G. LIMITATIONS

Despite all the improvements that our devised architecture has shown, it faces limitations as well. The main limitation is the centralized SDN controller that introduces a form of a single point of failure. The failure of an SDN controller prohibits processes such as introducing new publishers, subscribers, and new complex events, while the nodes can work with their previous topology and flow table settings as long as the publishers, subscribers, and the rules have not changed until the SDN controller recovers. Besides, *SDN-base CEP* architecture is evaluated based on static edge nodes. However, future work is to analyze the approach for the combination of mobile and static nodes and address challenges such as dynamic shortest path routing towards the closest fog node and handling the handovers with reasonable overheads to address the mobility of edge nodes.

VII. CONCLUSION

A growing number of urban monitoring applications are widely used in a smart city environment assisting municipal organizations to respond promptly to ongoing events of interest. The coupling of these applications with multi-tenant edge (sensor) nodes introduces new challenges where the edge nodes have limited resources and the applications require diverse QoS. This paper introduced a new architecture that addressed the challenges by integrating SDN technology, publish-subscribe architectural pattern, and complex event processing features. The devised approach benefits from distributed processing of event notifications while using centralized (semi-centralized) coordination mechanisms to perform its duty. For instance, the *optimum route calculator* and *publisher head selection* modules took advantage of SDN-based *network topology management* and boosted the overall performance. The proof of concept implementation was compared with three close baselines, and the results demonstrate improvements in terms of energy consumption, network traffic, and propagation delay of event notifications. However, subscribing to a complex event by a monitoring application that happens only once appears costly. Future research will extend the architecture to support mobile edge nodes and uncertainty in detecting an urban complex event.

REFERENCES

[1] A. Shahid, B. Khalid, S. Shaukat, H. Ali, and M. Y. Qadri, "Internet of Things shaping smart cities: A survey," in *Internet of Things and Big Data Analytics Toward Next-Generation Intelligence*. Cham, Switzerland: Springer, 2018, pp. 335–358.

[2] S. Dustdar and I. Murturi, "Towards distributed edge-based systems," in *Proc. IEEE 2nd Int. Conf. Cognit. Mach. Intell. (CogMI)*, Oct. 2020, pp. 1–9.

[3] A. Palade, C. Cabrera, F. Li, G. White, M. A. Razzaque, and S. Clarke, "Middleware for Internet of Things: An evaluation in a small-scale IoT environment," *J. Reliable Intell. Environ.*, vol. 4, no. 1, pp. 3–23, Apr. 2018.

[4] Y. Qin, Q. Z. Sheng, N. J. G. Falkner, S. Dustdar, H. Wang, and A. V. Vasilakos, "When things matter: A survey on data-centric Internet of Things," *J. Netw. Comput. Appl.*, vol. 64, pp. 137–153, Apr. 2016.

[5] Z. Pooranian, J. Abawajy, P. Vinod, and M. Conti, "Scheduling distributed energy resource operation and daily power consumption for a smart building to optimize economic and environmental parameters," *Energies*, vol. 11, no. 6, p. 1348, May 2018.

[6] J. M. Bailey, H. T. Malazi, and S. Clarke, "Smoothing speed variability in age-friendly urban traffic management," in *Proc. Int. Conf. Comput. Sci. (ICCS)*, 2021, pp. 3–16.

[7] Z. Pooranian, M. Shojafar, P. G. V. Naranjo, L. Chiaraviglio, and M. Conti, "A novel distributed fog-based networked architecture to preserve energy in fog data centers," in *Proc. IEEE 14th Int. Conf. Mobile Ad Hoc Sensor Syst. (MASS)*, Oct. 2017, pp. 604–609.

[8] J. Ren, Y. Pan, A. Goscinski, and A. R. Beyah, "Edge computing for the Internet of Things," *IEEE Netw.*, vol. 32, no. 1, pp. 6–7, Jan./Feb. 2018.

[9] F. Pisani, F. M. C. de Oliveira, E. S. Gama, R. Immich, L. F. Bittencourt, and E. Borin, "Fog computing on constrained devices: Paving the way for the future IoT," in *Advances in Edge Computing: Massive Parallel Processing and Applications*, vol. 35. Amsterdam, The Netherlands: IOS Press, 2020, pp. 22–60.

[10] B. L. R. Stojkoska and K. V. Trivodaliev, "A review of Internet of Things for smart home: Challenges and solutions," *J. Cleaner Prod.*, vol. 140, no. 3, pp. 1454–1464, 2017.

[11] A. Gaamel, T. Sheltami, A. Al-Roubaiey, and E. Shakshuki, "Broker-less middleware for WSAN performance evaluation," *Future Gener. Comput. Syst.* vol. 110, pp. 372–381, Sep. 2018.

[12] E. G. Davis and A. M. C. Augé, "Publish/subscribe protocol in wireless sensor networks: Improved reliability and timeliness," *KSII Trans. Internet Inf. Syst.*, vol. 12, no. 4, pp. 1527–1552, 2018.

[13] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for Internet of Things: A survey," *IEEE Internet Things J.*, vol. 3, no. 1, pp. 70–95, Feb. 2016.

[14] J. Dizdarević, F. Carpio, A. Jukan, and X. Masip-Bruin, "A survey of communication protocols for Internet of Things and related challenges of fog and cloud computing integration," *ACM Comput. Surv.*, vol. 51, no. 6, pp. 1–26, 2019.

[15] B. G. Assefa and Ö. Özkasap, "A survey of energy efficiency in SDN: Software-based methods and optimization models," *J. Netw. Comput. Appl.*, vol. 137, pp. 127–143, Jul. 2019.

[16] S. Saraswat, V. Agarwal, H. P. Gupta, R. Mishra, A. Gupta, and T. Dutta, "Challenges and solutions in software defined networking: A survey," *J. Netw. Comput. Appl.*, vol. 141, pp. 23–58, Sep. 2019.

[17] Y. Wang, Y. Zhang, and J. Chen, "An SDN-based publish/subscribe-enabled communication platform for IoT services," *China Commun.*, vol. 15, no. 1, pp. 95–106, Feb. 2018.

[18] Y. Shi, Y. Zhang, H.-A. Jacobsen, L. Tang, G. Elliott, G. Zhang, X. Chen, and J. Chen, "Using machine learning to provide reliable differentiated services for IoT in SDN-like publish/subscribe middleware," *Sensors*, vol. 19, no. 6, p. 1449, Mar. 2019.

[19] J.-H. Park, H.-S. Kim, and W.-T. Kim, "DM-MQTT: An efficient MQTT based on SDN multicast for massive IoT communications," *Sensors*, vol. 18, no. 9, p. 3071, Sep. 2018.

[20] T. Kohler, R. Mayer, F. Dürr, M. Maaß, S. Bhowmik, and K. Rothermel, "P4CEP: Towards in-network complex event processing," in *Proc. Morning Workshop-Netw. Comput.*, Aug. 2018, pp. 33–38.

[21] C. Esposito, A. Castiglione, F. Palmieri, M. Ficco, C. Dobre, G. V. Iordache, and F. Pop, "Event-based sensor data exchange and fusion in the Internet of Things environments," *J. Parallel Distrib. Comput.*, vol. 118, pp. 328–343, Aug. 2018.

[22] A. Meslin, N. Rodriguez, and M. Endler, "Scalable mobile sensing for smart cities: The MUSANet experience," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 5202–5209, Jun. 2020.

[23] H. T. Malazi and M. Davari, "Combining emerging patterns with random forest for complex activity recognition in smart homes," *Appl. Intell.*, vol. 48, no. 2, pp. 315–330, Feb. 2018.

[24] M. Guzel and S. Ozdemir, "A new CEP-based air quality prediction framework for fog based IoT," in *Proc. Int. Symp. Netw., Comput. Commun. (ISNCC)*, Jun. 2019, pp. 1–6.

[25] G. Cugola and A. Margara, "Complex event processing with T-REX," *J. Syst. Softw.*, vol. 85, no. 8, pp. 1709–1728, Aug. 2012.

[26] G. Cugola and A. Margara, "TESLA: A formally defined event specification language," in *Proc. 4th ACM Int. Conf. Distrib. Event-Based Syst.*, 2010, pp. 50–61.

[27] B. Khazael and H. T. Malazi, "Distributed coordination protocol for event data exchange in IoT monitoring applications," in *Proc. 11th Int. Conf. Inf. Knowl. Technol. (IKT)*, Dec. 2020, pp. 113–118.

[28] A. Adeniran, M. A. Hasnat, M. Hosseinzadeh, H. Khamfroush, and M. Rahnamay-Naeini, "Edge layer design and optimization for smart grids," in *Proc. IEEE Int. Conf. Commun., Control, Comput. Technol. Smart Grids (SmartGridComm)*, Nov. 2020, pp. 1–6.

[29] M. A. Hasnat, M. J. Hossain, A. Adeniran, M. Rahnamay-Naeini, and H. Khamfroush, "Situational awareness using edge-computing enabled Internet of Things for smart grids," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Dec. 2019, pp. 1–6.

[30] D. Lan, Y. Liu, A. Taherkordi, F. Eliassen, S. Delbruel, and L. Lei, *A Federated Fog-Cloud Framework for Data Processing and Orchestration: A Case Study in Smart Cities*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 729–736.

[31] D. Lan, A. Taherkordi, F. Eliassen, and G. Horn, "A survey on fog programming: Concepts, state-of-the-art, and research challenges," in *Proc. 2nd Int. Workshop Distrib. Fog Services Design (DFSD)*. New York, NY, USA: Association for Computing Machinery, 2019, pp. 1–6.

[32] P. G. V. Naranjo, Z. Pooranian, M. Shojafar, M. Conti, and R. Buyya, "FOCAN: A fog-supported smart city network architecture for management of applications in the internet of everything environments," *J. Parallel Distrib. Comput.*, vol. 132, pp. 274–283, Oct. 2019.

[33] F. Colace, M. D. Santo, V. Moscato, A. Picariello, F. A. Schreiber, and L. Tanca, *Data Management in Pervasive Systems*. Springer, 2015.

[34] A. Shahraki, A. Taherkordi, O. Haugen, and F. Eliassen, "A survey and future directions on clustering: From WSNs to IoT and modern networking paradigms," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 2, pp. 2242–2274, Jun. 2021.

[35] L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, "SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for wireless sensor networks," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 513–521.
[36] N. Abdolmaleki, M. Ahmadi, H. T. Malazi, and S. Milardo, "Fuzzy topology discovery protocol for SDN-based wireless sensor networks," *Simul. Model. Pract. Theory*, vol. 79, pp. 54–68, Dec. 2017.
[37] A.-C. Anadiotis, L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, "SD-WISE: A software-defined wireless sensor network," *Comput. Netw.*, vol. 159, pp. 84–95, Aug. 2019.
[38] Deib-Polimi. (2017). *Trex*. [Online]. Available: https://github.com/deib-polimi/TRex
[39] SDN-WISE Lab. (2019). *SDN-WISE-Java*. [Online]. Available: https://github.com/sdnwiselab

BEHNAM KHAZAEL received the B.Sc. degree in computer software engineering from the University of Science and Culture, Tehran, Iran, in 2013, and the M.Sc. degree in computer software engineering from Shahid Beheshti University, in 2014, where he is currently pursuing the Ph.D. degree. He has been working with Telecom companies in telecom network architecture and MNO/MVNO business support systems. His research interests include pervasive computing and the Internet of Things, emphasizing middleware architectures and design based on SDN platforms.

HADI TABATABAEE MALAZI received the Ph.D. degree in computer engineering from the University of Isfahan, in 2012. He was an Assistant Professor with the Computer Science and Engineering Faculty, Shahid Beheshti University, from 2013 to 2019, where he established the Pervasive Computing Laboratory (PerLab). He is currently a Research Fellow at the School of Computer Science and Statistics, Trinity College Dublin, and a member of the Connect Research Centre, Ireland. His main research interests include pervasive and edge computing, with an emphasis on smart city.

SIOBHÁN CLARKE received the B.Sc. and Ph.D. degrees in computer science from Dublin City University. She is currently a Professor and a fellow of Trinity College Dublin, where she leads the Distributed Systems Group and the Director of Future Cities, the Trinity Centre for Smart and Sustainable Cities. Her research interests include software engineering models for the dynamic provision of smart and dynamic software services to urban stakeholders in large-scale and mobile environments.