An IoT Architecture Leveraging Digital Twins: Compromised Node Detection Scenario

Khaled Alanezi and Shivakant Mishra

***Abstract*—Modern Internet of Things (IoT) environments with thousands of low-end and diverse IoT nodes with complex interactions among them and often deployed in remote and/or wild locations present some unique challenges that make traditional node compromise detection services less effective. This article presents the design, implementation, and evaluation of a fog-based architecture that utilizes the concept of a digital twin to detect compromised IoT nodes exhibiting malicious behaviors by either producing erroneous data and/or being used to launch network intrusion attacks to hijack other nodes eventually causing service disruption. By defining a digital twin of an IoT infrastructure at a fog server, the architecture is focused on monitoring relevant information to save energy and storage space. This article presents a prototype implementation for the architecture utilizing malicious behavior datasets to perform misbehaving node classification. An extensive accuracy and system performance evaluation was conducted based on this prototype. Results show good accuracy and negligible overhead especially when employing deep learning techniques, such as multilayer perceptron.**

***Index Terms*—Compromised node detection, digital twin, fog computing, Internet of Things (IoT).**

I. Introduction

THE Internet of Things (IoT) has brought increased convenience and productivity to homes, factories, malls, hospitals, sidewalks, city squares, and more. Typically, an IoT application is a distributed system with components deployed across the IoT-fog-cloud continuum. IoT components perform data collection from the environment and feed the data to smart decision-making systems running in the cloud with the fog layer playing a staging role for data filtering and aggregation.

Building applications on top of such complex infrastructure is inherently difficult due to several reasons. First, the components that form the IoT system typically come from different vendors. The lack of an agreed communication and integration standards often leads to interoperability issues [1]. To tackle this issue researchers proposed to use standard web technologies [2] and web gateways to enable sensor discovery and semantic communication [3]. Second, IoT systems face scalability issues since they require connecting large number of sensors producing massive amounts of data. Mining this data requires shipping it to cloud servers, which is impractical as it will overwhelm the backbone of the network. To tackle this issue, researchers proposed to push the computation to the edge of the network near the data sources [4]. Edge computing is also used as a means of minimizing the data reaching to the cloud so as to protect the security and privacy of IoT users.

Given the aforementioned challenges, digital twins (henceforth DTs) have been proposed as a tool for managing IoT systems, predicting their behavior under different scenarios and improving their performance. DT is defined as a virtual replica of a physical entity, such as people, assets, systems, or processes [5]. The replication with the physical world is often envisioned to be bidirectional and happening in near real time. It also incorporates multidimensional data views, such as quantitative, qualitative, historical, and environmental data, which can bring many benefits to IoT systems [6]. First, in the design phase, new concepts can be modeled and applied to the virtual environment for verification before application to the actual IoT system. Second, during the operation phase, thorough analysis can be done on virtual twins in order to discover and react to abnormalities resulting from malfunctioning hardware or software components or even cyberattacks [7]. Finally, the DT virtual environment can serve as a medium for performing predictive maintenance to discover failures before they occur. The concept of DT is not new, however, as it stems from existing mature concepts, such as IoT, AI, and Big Data [8]. Despite of the abovementioned benefits the concept of DT provides for IoT, a systems implementation showing how to utilize DTs in an IoT application and how the interplay of different components affects the performance is still lacking. Hence, the motivation of this work is to fill this gap by designing, implementing, and evaluating the performance of a DT-IoT architecture while focusing on compromised node detection as an example application. The key novelty of this work comes from utilizing the concept of DT for efficient misbehavior monitoring and combining data anomaly and network intrusion for enhanced misbehavior detection coverage. Our choice of application stems from the fact that privacy and security are major obstacles hindering wide adoption of IoT systems. While IoT applications have enhanced our lifestyles in many areas, there is an associated cost of potentially exposing user's privacy and security. Typically, IoT nodes embed wide array of sensors and are connected to the network, thereby making them prime targets for attackers. Once an IoT node is exploited, an attacker would typically couple the attack with payload codes that can achieve certain objectives. First, an attacker can carry a cyber-physical

Manuscript received 2 August 2023; revised 14 December 2023 and 21 March 2024; accepted 8 May 2024. Date of publication 12 June 2024; date of current version 20 June 2024. *(Corresponding author: Khaled Alanezi.)*

Khaled Alanezi is with the College of Computing and Systems, Abdullah Al Salem University, Khalidya, Kuwait (e-mail: khaled.alanezi@aasu.edu.kw).

Shivakant Mishra is with the Department of Computer Science, University of Colorado Boulder, Boulder, CO 80309 USA (e-mail: mishras@colorado.edu).

Digital Object Identifier 10.1109/JSYST.2024.3403500

attack [9] to introduce *data anomalies* causing control system failure thereby leading to physical damage [10], [11]. Second, once inside the network (i.e., *network intrusion*), an attacker can utilize the node to launch large scale botnet attacks, such as the Mirai IoT Botnet [12], which can lead to large scale service disruption.

While there has been a plethora of research done in the area of preventing and/or detecting node compromises over the last 30+ years, compromised node detection in an IoT environment presents some unique challenges. First, typically only weak security measures are taken to protect IoT nodes, which can be attributed to the focus on keeping the node cost low and ensuring a plug-and-play operation, so that they can be deployed in large numbers. Second, these nodes are often placed remotely in the wild with little or no monitoring with many IoT vendors failing to provide proper measures or automated-tools for timely security updates [13]. Third, the sheer number and variety of IoT devices in an IoT infrastructure typically running into hundreds and even thousands means an increased attack surface available to the attackers and for the system administrators to monitor and maintain. Finally, complex interplay among thousands of nodes makes it difficult to identify faulty or anomalous behavior often confined to a small number of nodes under a variety of different scenarios.

To address these challenges, we propose a fog-based architecture that utilizes the concept of a DT in which a virtual replica of the IoT infrastructure is created in the fog to momentarily replicate IoT node state changes where threat detection and mitigation can take place. Our proposed architecture uses DT to employ detection techniques for both data anomaly and network intrusion attacks for enhanced IoT network security while benefiting from two conceptual features of DTs [14]. First, by definition, the DT concept proposes to only mirror properties and characteristics that are of importance to the application context. Consequently, we only reflect status information represented by sensor values and network activity summary information in the DT. Clearly, by focusing only on important information our solution can save device energy, fog server space, and network resources. Second, creating DTs for all IoT nodes paves the way for understanding the aggregate behavior of those nodes thereby leading to better governance and control of the complex IoT system. In summary, adopting the DT model brings the following benefits when compared to traditional approaches.

1) Efficient monitoring by mirroring only needed data and sharing of sensor data across various machine learning (ML) models thereby avoiding possible redundant data collection, storage, and processing.
2) Design flexibility since new solutions can be easily integrated to the architecture by calling Ditto's standard web APIs.
3) Enhanced IoT environment control by gathering necessary data from various IoT nodes leading to smarter actions that consider the environment as a whole rather than focusing on individual nodes.

The methodology followed in designing and building the DT-IoT architecture is based on several factors. First, open-source components were utilized, such as Eclipse Ditto [15] and Docker [16]. This in turn puts other researchers in a better position to read and interpret our results. Second, although most similar research focused on industrial IoT (IIoT), we design the framework to be applicable to all IoT applications at large, such as smart cities, smart buildings, and smart healthcare. Third, we used compromised node detection scenario as a motivating application to demonstrate the effectiveness of the work. Last, we report the performance results of the architecture when different combinations of models and datasets are used to reflect on the impact of such design choices on the usability of the solution. In summary, the contributions of this work are three fold.

1) We provide a detailed design of a fog-based architecture utilizing the concept of DTs for compromised node detection in an IoT environment.
2) We show how this architecture can utilize various models for compromised IoT node detection based on data anomaly and network intrusion detection. We also report the accuracy and performance of those models.
3) We provide a prototype implementation and extensive performance evaluation of the architecture by utilizing open-source solutions, such as Ditto and Docker.

The rest of this article is organized as follows. Section II presents related works. In Section III, we present the design of the architecture and introduce the details of the datasets utilized to build the data anomaly and network intrusion models. Section IV provides technical details of the prototype implementation for the architecture and the models. In Section V, we present system performance measurements for the prototype. We discuss the benefits and possible improvements of the work in Section VI. Finally, Section VII concludes this article.

II. Related Work

This work investigates *integrating DTs in IoT systems*. The application served by the resultant architecture is *compromised node detection in IoT*. We summarize in this section the recent works in each of these areas. In addition, summarization and classification for the literature focusing on the use of *DTs for anomaly detection in IIoT* is presented to reflect on the novelty of our architecture.

A. DTs in IoT

DTs, a concept popular in manufacturing, have been proposed mainly for use in IIoT. With the aid of DT concept, an IIoT controller collects massive data from devices to take smart environment-wide decisions [17], [18]. To serve this purpose, a reference model of DTs in IIoT is proposed [19], which involves mirroring the internal structure, runtime environment, APIs of the physical objects as well as features, such as scalability, interoperability, security, and privacy. Also, the concept of DTs has been recently proposed in the IoT context for staff safety management in cold warehouses' hazardous environments [20]. Here, a DT is created to synchronize the time/space information of the staff with the controller. Consequently, the controller will run algorithms to detect staff motionless status so as to alert first aid workers in a timely manner. In addition to safety, the concept

of DTs is utilized in IoT context for elderly health [21] where DTs are used to mirror medical data obtained from wearable devices to monitor and diagnose health issues. Furthermore, the concept of DTs is utilized to enable smart buildings and smart cities. For the former, DTs replicate static and dynamic building data for enhanced building monitoring and management [22]. Whereas for the latter, DUET [8] is proposed as a DT framework built for smart city applications. DUET creates a cloud of integrated DT models that can be queried to perform smart city planning (i.e., traffic, air pollution, or noise pollution) at the city level. DTs are also proposed in IoT for smart agriculture systems [23] where the goal is to monitor farm information remotely thereby reducing manual efforts and to simulate the effect of intervention techniques on farm productivity. Also, battery energy storage systems (BESS) [24] are also shown to benefit from the DT concept so as to prevent possible failures and cyberattacks. When implemented with mobile edge computing, DTs can suffer from congestion and the lack of incentives. The work by Lin et al. [25] addresses these issues. Another solution utilizing DTs is proposed by Revetria et al. [26] in which DTs were integrated with the IIoT, augmented reality technology and compensated with simulations to build a solution for monitoring stress on metal shelves. Lastly, a survey paper by Minerva et al. [14] discussed DTs features and architectures in the context of IoT.

B. Compromised Node Detection in IoT

IoT applications involve data collection to provide controlled services to users. Consequently, an attacker can impact the accuracy of the controller by creating anomalies in sensory data [27]. There are several works aimed at detection of data anomaly in IoT each focusing on a specific methodology or goal. AnoML-IoT [28] presents a pipeline for data anomaly detection with the goal of masking heterogeneity in IoT systems. The presented pipelines simplify the process of deploying anomaly detection solutions by integrating various communication protocols and detection algorithms and providing the ability for deployment on the edge. Cauteruccio et al. [29] focused on detecting data anomalies for multiple IoT (MIoT) scenarios. In an MIoT scenario, scattered IoT deployments are linked together via bridging nodes, which is typical since IoT deployments does not work in isolation. The presented approach depends on the fact that anomaly detection must consider the size of the IoT network, the distances between nodes, and the centrality features of a node. Malicious node behavior other than producing data anomaly could manifest in performing anomalous network activity. Researchers created datasets (e.g., [30]) containing the network traffic of benign and malicious nodes in order to build models that can automatically detect malicious traffic. Based on such datasets various techniques and architectures to detect malicious network behavior are proposed. Early detection of IoT malware activity (EDIMA) [12] is a full architecture geared toward detection of network intrusion activity during the scanning phase instead of the attack phase. The presented architecture runs ML algorithms on gateway devices in large scale IoT networks of enterprises or internet service providers. Diro et al. [31] proposed the use of deep learning models over traditional ML in order to cope with newly surfacing small mutations of malware. Their work also favors the use of distributed attack detection scheme based on fog resources. The work by Raza et al. [32] deviates from other works by focusing on intrusion detection of IoT devices connected using IPv6. The presented solution demonstrated the capability of detecting intrusion attacks on networks of resource-constrained devices connected using lossy links. The work by Shafiq et al. [33] presents a novel feature selection method for building accurate models for IoT network traffic analysis. Their efforts are orthogonal to our work as such techniques can be employed in our architecture for building more accurate network intrusion classifiers.

C. DTs for Anomaly Detection in IIoT

In its most basic form, a DT can be thought of as a virtual replica for a physical object. However, there is a wide range of characteristics imposed by this software dematerialization on the DT that an IoT ecosystem can leverage [14]. First, *representativeness* where a DT must cover all features intended for analysis and *reflection*, which necessitates replicating the features in a timely manner. Second, more advanced characteristics include *augmentation* where a DT can be augmented by smart functionality from running ML algorithms on known datasets relevant to the physical object and *predictability*, which is the ability to simulate the behavior of the DT in the environment to predict its performance. Most of the works we list in Table I and compare our work against utilize the two latter more advanced characteristics of DTs. However, they vary in terms of the type of application and targeted behavior as discussed next. Salim et al. [34] utilized data from DTs to run deep learning models for detection of botnets. The implemented solution employs blockchain technology to protect DT data from tampering. Also, the paper by Gupta et al. [41] focused on securing patients' DT data by using federated learning. The ideas presented in these works can be employed by our framework to protect sensor and network data from tampering by malicious entities. Collaboration between DTs is utilized by Calvo et al. [35] and Sahal et al. [44] to accommodate for contextual data while performing anomaly detection. In the same manner, we propose to combine DT knowledge based on data anomaly and network intrusion to enhance the chances of detecting malicious nodes. Continuous feedback between DTs and the physical environment is proposed by Piltan et al. [36] and Xu et al. [37] to build an intelligent signal estimator and cater for outdated ML models, respectively. The sensor behavior and network behavior monitoring components described in Section III-A utilize the same concept so as to ensure continuous improvements for the data and network anomaly ML models. Other sources for compensating for DT data include simulations [38] and artificial intelligence [42].

Another focus of the DT literature is on building application-dependent models such as models for detecting anomalies in building automation systems as proposed by Lu et al. [40] or in nuclear reactors (Adebena et al. [43]). We proposed a generic DT architecture but used behavior anomaly as an application to reflect on the validity of our work. Finally, the work by

TABLE I
SUMMARY AND CLASSIFICATION OF RELATED WORKS

Study	Mechanism	Used application	Targeted behavior
Salim et al. [34]	Integrating DTs with deep learning models for early detection of botnets.	Smart factory	Botnets discovery
Calvo et al. [35]	Utilizing collaboration between DTs to detect anomalies that are difficult to detect using ML models.	Generator cooling System	Anomaly detection
Piltan et al. [36]	Creating an intelligent DT that combines raw signals and intelligent signal estimators for smart bearing fault detection.	Bearing fault detection	Anomaly detection
Xu et al. [37]	Fine-tuning of DT data with real-time data to tackle the issue of outdated ML models	Cyber-physical Systems	Anomaly detection
Yang et al. [38]	Compensating DT data coming from the physical device with simulations under various manufacturing strategies.	Car body-side production	Fault diagnosis
Huang et al. [39]	Deploying the DT and ML classification on the edge for timely action in case of discovered faulty data.	Absorption chiller	Fault diagnosis
Lu et al. [40]	Using the DT to build an integrated platform for collecting building data needed for the operation and maintenance phase.	Building HVAC	Anomaly detection
Gupta et al. [41]	Utilizing the DT concept to build a secure virtual replica for patient information for trying different treatments.	Patient Monitoring	Anomaly detection
Cha et al. [42]	Augmenting the DT model with data from generative AI to compensate for missing data.	IIoT	Malware detection
Adebena et al. [43]	Utilizing DT to replicate nuclear plant parts and be able to perform plant health management.	Nuclear plant	Anomaly detection
Sahal et al. [44]	Utilizing interactions between DTs to build semantic knowledge for enhanced data anomaly detection.	Smart factory	Anomaly detection

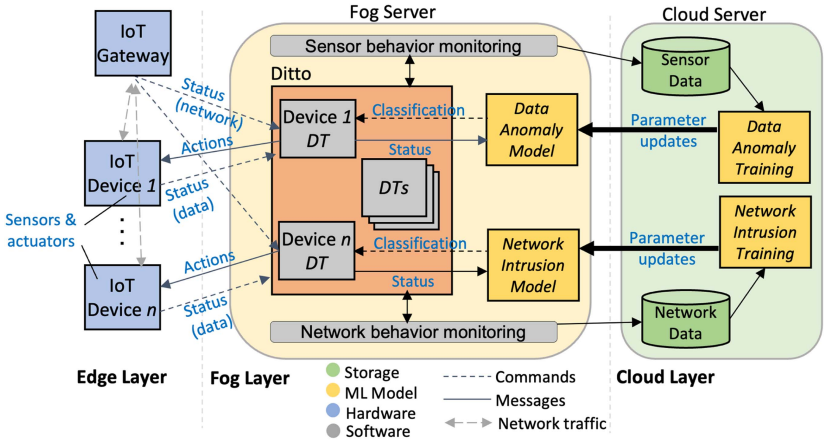


Fig. 1. Solution architecture.

Huang [39] proposed to host DTs on the edge of the network, which brings the benefits of reduced latency and faster detection of anomalies. We also propose to push the models trained on the cloud to the edge for similar reasons.

III. DESIGN

A. Architecture

As shown in Fig. 1, the proposed architecture spans the edge-fog-cloud continuum. The *edge layer* typically consists of battery-powered *IoT devices* with limited resources. Those devices are meant to perform sensing and actuation in the environment. At the other extreme end of the architecture sets the *cloud layer*, which typically has abundant storage and computing resources. The architecture utilizes a *cloud server* to perform compute-intensive training of malicious node detection models. In the middle of the architecture is the *fog layer* providing low-latency access for IoT devices to the *fog server*. We propose to utilize an edge server [4] installed in the fog to provide low-latency access to trained malicious node detection models. The edge server is a powerful compute box installed on the edge of the network to be utilized by edge nodes for computation offloading. Models deployed on the edge server will be harnessed to monitor nodes activity at the edge of the network to detect malicious behavior on spot. Each of the abovementioned components provide an important functionality in the architecture. First, the edge layer provides the perception needed to detect malicious behavior. Second, the fog layer provides low-latency access to malicious behavior detection models. Third, the cloud layer offers the required resources and power for costly model training. We now turn into describing the functionality of each component in the architecture. Starting from the edge layer, *IoT devices* have the sensing and actuation capabilities to aid various IoT applications. Those devices are connected to the network using an *IoT gateway*. IoT deployments typically follow the star network topology where all devices utilize this gateway to access the rest of the network and the Internet. Due to this central role, an IoT gateway can be used as the basis for detecting anomalous network behavior exhibited by IoT devices as we will describe later. Devices at the edge layer have one-network-hop access to a

fog server running Ditto [15]. *Ditto* is an open-source framework for managing DTs. It follows the device-as-a-service model to mask the complexity of IoT devices stemming from different device manufacturers and communication protocols. Instead of dealing with device specific features, Ditto exposes callable web APIs that can be used to interact with DTs and update their internal status whenever a change in the physical twin takes place. Through APIs exposed by Ditto, device DTs receive status updates from IoT devices reflecting corresponding device status change (i.e., sensor data changes). Also, DTs receive status update from the *IoT gateway* that are related to the network behavior of the device (details of DT design are discussed in Section III-C).

DT status updates received from the IoT environment are used in two ways. First, the sensory status and the network status are used as input for the *data anomaly model* and *network intrusion model*, respectively, to discover any anomalies in node behavior. When building these models, we follow one of the typical usage scenarios envisioned for a DT, which is to accompany it with ML models [37] to detect events of particular interest in the environment, such as data anomaly and network intrusion in our architecture. Selecting the proper training dataset and ML algorithms used to build the models is a design choice that must be made carefully by system designers as it can have impact on the performance of the architecture as will be shown in Section V-C. The classification result from the models is sent to the corresponding DT, and only if the result shows that there is a possible anomaly, a message is sent from the DT to the corresponding device to take the needed action. Note that the actions themselves are defined by the system administrators, e.g., quarantine the device or shut it down to avoid infecting other nodes in the environment. Notice that combining results from both classifiers will lead to decreased false negatives thereby increasing confidence in the overall system. The second usage of reported status updates is to forward them via the *sensor/network behavior monitoring* components to the *sensor/network databases* in the cloud. The latter will act as the ground truth for the *data anomaly* and the *network intrusion* training done periodically. Note that behavior monitoring should involve system administrators utilizing malware analysis techniques for detecting IoT malware [45]. Once a model is updated in the cloud by retraining, new *parameters* of the model are pushed to the fog server to be used. The periodic updates ensure that the architecture is adaptive to changing norms in the environment. Our approach follows the on-device inference and cloud training paradigm for EdgeAI [46]. The benefits of this paradigm are low latency access to ML models as well saving energy by performing compute intensive training on the cloud. We discuss the design of the data anomaly model and network intrusion models in details in Sections III-D and III-E, respectively.

B. Role of DTs in the Architecture

In this section, we describe the role of DTs in the architecture and the advantages they bring compared to traditional approaches. First, since a DT replicates the state of the physical world in near real time, it can be leveraged for multitude of important tasks concurrently. For example, as demonstrated in this article, a DT is useful for detecting compromised nodes in the physical world. At the same time, the same instance of DT can be leveraged to actuate appropriate actuators by analyzing the current state. Furthermore, the same instance can be leveraged to perform predictive maintenance. The key advantage here is that there is no need for separate, customized end-to-end designs for each of these tasks. Second, a DT controls the rate and nature of the data being sent from the physical world based on the current state. For example, if the communication network gets congested, a DT can direct the physical world to either slow down the rate at which sensor data is being transmitted, or perform some data preprocessing (e.g., averaging sensor values) before transmission. This enables various tasks being performed at the edge to continue despite network congestion. Third, a DT can be used for continuous feedback and retraining of ML models. As discussed in [37], an ML model can be trained on labeled historical data and unlabeled real-time data with the DT model's ground truth label. Fourth, as mentioned in [39], a DT at the edge allows for reducing latency by reducing delays while accessing the cloud. Finally, a DT can be used as a staging step for selective replication of data to the cloud to protect the privacy and security of the monitored entity [41] while detecting anomalies. In this work, the DT played a central role in monitoring the environment and in building and retraining of ML models. We also plan to extend the usage of the DT to cover more from the aforementioned roles as part of our future work.

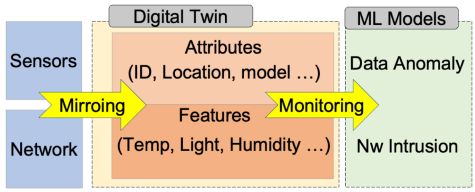


Fig. 2. DT deployment status.

C. DT Design

The concept of DTs has been around for almost two decades now. However, a standard design and implementation guidelines for the concept is still lacking [47]. Consequently, the design approach for the DT in this project will follow the 5-steps guided approach for DT evolution [5] of mirroring, monitoring, simulation, federation, and autonomous. We begin by covering the first two steps of mirroring and monitoring. The remaining three steps represent later design stages that build on the understanding of the first two stages, which we leave as a future work as described in Section VII. As can be seen in Fig. 2, there are two types of information mirrored about an *IoT device* in the DT. First, the *attributes* of the device cover static metadata such as the unique identifier of the device, the manufacturer and the location coordinates if any. Second, the *features* contain dynamic information that changes over time and are replicated to the DT momentarily. For data anomaly monitoring, we mirror changing sensors' values as captured by the IoT node microcontroller and

TABLE II
UTILIZED DATASETS FOR DATA ANOMALY AND NETWORK INTRUSION MODELS

Dataset name	Scenario	No. of records	No. of features	Sample features	Labels
AnoML-IoT [28]	Data anomaly	6,558	6	Timestamp, temperature reading in C, and humidity percentage.	Normal and Anomalous
DS2OS [48]	Data anomaly	357,941	13	Timestamp, src ID, src type, dest. type, operation, and value.	Normal and Anomalous (Probing, DoS, Mal. Control, Mal. Operation, Scan, Spying, and Wrong Setup.)
IoTID20 [49]	Network intrusion	625,783	86	Timestamp, flow duration, min/max fwd packet size, and flow bytes/s.	Benign and Malicious (Mirai, DoS, Scan, and MITM ARP Spoofing.)

the calls that the IoT node make to other nodes or services. On the other hand, for network monitoring, we mirror summary information related to the exchanged network packets by the device. The exact sensor values and network summary status that must be replicated to the DT depends on the input parameters of the utilized data anomaly and network intrusion models. Sections III-D and III-E describe three ML models that we will evaluate for use in the architecture. Whenever any of the models is actually deployed on the fog server, its corresponding features must be replicated to the DT in order to be used by the ML model for device behavior classification. We evaluate the performance of the models when integrated with the DT in Section IV.

D. Data Anomaly Models

Monitoring data anomaly involves inspecting data generated from sensors to be consumed by an application or a service. Table II lists three example datasets, we used for the compromised node detection along with summary information for each dataset. First, the *AnoML-IoT* dataset contains sensor readings captured over two days for temperature, humidity, light, loudness, and air quality sensors. The authors of this dataset created data anomalies by subjecting the sensors to an air dryer for some periods of time. Data coming during this time is marked as an anomaly. Otherwise, sensor readings are marked as benign. Second, *DS2OS* is a synthetic dataset created by the distributed smart space orchestration system. The data are generated from a simulated setup including four IoT sites each containing various services that can call each other. Example services include thermostat and door lock controllers. A service can read or write a value to or from another service depending on the required action. Beside the normal traffic, the authors simulated communications for various attacks including data probing, denial-of-service (DoS), malicious control, malicious operation, scan, spying, and wrong setup. In summary, we will use the first two datasets as an example for an IoT deployment generating malicious behavior at the perception layer (i.e., data anomalies).

E. Network Intrusion Models

In addition to false data injection attacks to confuse an application, a compromised node could be exploited by an attacker to launch network attacks, such as port scans, distributed denial of service (DDoS) attacks, or spoofing. These attacks often lead to data theft, data corruption, or system failures. In this part of the architecture, we focus on discovering and stopping network intrusion attacks. In order to achieve this goal, we utilized the IoTID20 dataset also described in Table II. This dataset is built using an actual IoT network consisting of two smart home devices (AI speaker and security camera) and multiple laptops and smartphones connected to an isolated local area network (LAN). Various attacks are then deployed or simulated using this setup, such as Mirai, DoS, scan, and man-in-the-middle address resolution protocol (MITM ARP) spoofing attacks. Network packet captures (*.pcaps) are then extracted and network packets containing summary features of communication sessions are labeled as benign or malicious accordingly.

IV. EXPERIMENTAL WORK AND IMPLEMENTATION

We propose a dynamic architecture to detect malicious IoT nodes. This requires real-time monitoring of node behaviors and flagging of observed anomalies as they occur. To achieve this objective, the architecture must support two requirements. First, the ability to capture relevant node behavior. Second, the availability of dynamically trained ML models for online behavior classification. We begin this section by describing the hardware and software components involved in building the architecture and how these components are integrated. These details are described in Section IV-A. After that, we describe the classification models and their accuracy based on various ML classifiers when applied to the chosen datasets. As stated in Section III, we propose that the architecture must train two types of models namely the data anomaly model and the network intrusion model. We report in this section the design and accuracy for the two models when trained with various classifiers. Note that the models along with the codes to build them are available on OSF.[1]

A. Architecture Implementation

We built a prototype for the architecture presented in Section III. The prototype is a distributed solution spanning the edge of the network and the fog layer as described earlier. The components utilized to build this prototype are listed in Table III. Note that the code for this prototype is also shared under the same OSF project below the *Online Experiment* component. We utilized a MacBook Air laptop to resemble a fog server. The fog server is typically a tethered (powered) machine with good computing capabilities that is one-network-hop away from edge nodes. On the other hand, an Arduino Uno R3 board was used to resemble the IoT node. We stacked a WiFi shield with the

[1]https://osf.io/mh6es/?view_only=f7dce520b1b64ce198ab039563c29e5f

TABLE III
HARDWARE AND SOFTWARE COMPONENTS USED TO BUILD THE ARCHITECTURE PROTOTYPE

Component name	Type	Role
MacBook Air	Hardware	Fog server.
Arduino Uno R3 [50]	Hardware	IoT device microcontroller.
Arduino WiFi shield [51]	Hardware	Connects microcontroller to the LAN.
Huawei wireless router	Hardware	Wireless LAN router.
Ditto [15]	Software	A framework for implementing DT Capabilities for IoT.
Docker [16]	Software	Container engine for building Ditto, data anomaly, and network intrusion containers.
Data anomaly container	Software	Container running the data anomaly model.
Network intrusion container	Software	Container running the network intrusion model.
Postman [52]	Software	Software platform used to call Ditto to create DTs.

Arduino Uno to provide it with WiFi capability and connected it along with the edge server to the same LAN using a Huawei wireless router. The prototype relied on Eclipse Ditto [15] for implementing the needed DT functionality. This functionality includes configuring DTs, online replication of DT state from the physical IoT node to the virtualized replica and observing DT state changes. To run Ditto, we deployed the prebuilt ditto docker images on the edge server. We also used Docker [16] to dockerize and deploy python scripts for loading the ML models used for performing online classification. Each script will load a pretrained model that is stored on disk in a pickle file [53] and listen to a transmission control protocol (TCP) socket to receive requests for record classification. We evaluated the performance of the overall architecture using the built prototype in Section V-C. DTs must be configured on Ditto before the mirroring and monitoring processes take place. Listing 1 demonstrates an example JavaScript Object Notation (JSON) configuration for creating a new DT in Ditto. This configuration pertains to the case of monitoring data anomalies based on the knowledge gained from the AnoML-IoT dataset as described in Section III-D. Note that we only cover the AnoML-IoT scenario in our online implementation, which is sufficient to measure the system performance of the architecture. Ditto requires to identify a *definition* clause, which should contain a unique identifier for the DT. This identifier will be used for later communication with the DT to update/observe its state features values as they change. The *attributes* section of the JSON document contains static information for the DT, such as its logical location, manufacturer, model, and so on. Finally, the *features* section tracks the online status of the DT. In the case of the AnoML-IoT scenario shown, it lists the readings of the four sensors as captured by the IoT node that will be monitored to detect anomalies in the environment. Section V-C evaluates the systems performance of the architecture while utilizing various datasets/ML classifiers combinations.

```
{"definition":"kw.edu.aasu:arduino:1.0",
 "attributes":{
  "manufacturer":"Arduino Inc",
  "location":"CS Dept. Corridor",
  "serialno":"1",
  "model":"Arduino Uno"
 },"features":{
  "temperature":{"properties":{"value": 0.0}},
  "humidity":{"properties":{"value": 0.0}},
  "light":{"properties":{"value": 0.0}},
  "loudness":{"properties":{"value": 0.0}}
 }}
```

TABLE IV
ACCURACY FOR DATA ANOMALY CLASSIFIERS BASED ON RF, SVM, AND MLP

	AnoML-IoT			DS2OS		
	RF($n=100$)	SVM	MLP	RF($n=100$)	SVM	MLP
Accuracy	0.989	0.938	0.966	0.993	0.993	0.993
Precision	0.989	0.952	0.977	0.986	0.984	0.984
Recall	0.983	0.899	0.942	0.893	0.887	0.894
F1-Score	0.986	0.921	0.958	0.934	0.930	0.934

B. Data Anomaly Models Implementation

We utilized two datasets for data anomaly detection. First, the AnoML-IoT dataset represents a scenario where the attacker is deliberately producing erroneous sensor values or data anomalies. Second, the DS2OS also included an attacker controlling the application as it contains data traces produced at the application level. However, instead of producing erroneous sensor values, in this dataset, the application is producing malicious calls for other application level services in the same environment. For each dataset, we ran three types of classifiers namely random forest (RF), support vector machines (SVM), and multilayer perceptron (MLP). The accuracy results are shown in Table IV. RF was chosen due to its ability of visualizing and studying the classification results. We kept the number of estimators to 100, the default number in Scikit Learn [54]. We also used SVM due to its efficiency with numerical as well as categorical features since the datasets contained both types of features. Finally, we used MLP as we wanted to also include a deep learning classifier in the experimental design. MLP is known to perform well with tabular data similar to the datasets we use. We have chosen to implement MLP with 3 hidden layers and 11 neurons at each layer as a starting point. We noticed that we were able to train the datasets efficiently with this choice. Optimizing MLP is an iterative process that is out of the scope of our work since our focus is to measure the system performance of the architecture.

We now turn into comparing the accuracy results for the three classifiers within each dataset as seen in Table IV. For the AnoML-IoT dataset, a relatively small dataset, we noticed that RF produced the best results in terms of the four metrics (i.e., accuracy, precision, recall, and F1-Score). However, the difference margin is small since, for example, the largest gap is between RF and SVM at the recall metric. Recall reflects the ability of the classifier to correctly flag all cases positive and negative. Note that this measure was the most difficult for all the three classifiers. For the DS2OS dataset, still Table IV, a dataset that is larger than the AnoML-IoT (refer to Section III-D for datasets description), all the three classifiers produced the

TABLE V
ACCURACY FOR NETWORK INTRUSION CLASSIFIERS BASED ON RF, SVM, AND MLP

	IoTID20		
	RF($n = 100$)	SVM	MLP
Accuracy	0.985	0.980	0.994
Precision	0.992	0.970	0.993
Recall	0.879	0.859	0.958
F1-Score	0.927	0.906	0.975

same accuracy of 99.3%. The accuracy is a common metric that reflects the ability of the classifier to correctly flag positive and negative instances. Also in this dataset, recall achieved the worst performance with RF producing the lowest result. Conversely, RF achieved the best result when it comes to precision.

C. Network Intrusion Models Implementation

For the network intrusion, we utilized the IoTID20 dataset, which consists of features extracted from *.pcap files containing *.pcaps for attack scenarios as well as benign scenarios. This dataset is a typical dataset to be used in intrusion detection solutions. Detecting malicious network behavior for an IoT node requires capturing the network packets produced by this IoT node and extracting features from those packets to be used by the classifier.

Table V reports the accuracy for the same three classifiers used with the data anomaly models when applied to the network intrusion dataset. Notice that MLP produced the best results across all the metrics, proving our design choice of using it with tabular data that is large in size. When we compare this result with data anomaly we see that RF produced the best results mostly in all cases. This result can be attributed to the size of the IoTID20 dataset, which is larger than the other two datasets. We note that the choice of which ML classifier not only impacts the accuracy but also affects important measures relevant to system efficiency such as model size, model loading time, and classification time. These measures are particularly important due to the distributed nature of our system architecture. Hence, we provide a complete system evaluation for the three datasets when utilizing the three classifiers in Section IV-D.

D. IoT Node Implementation

IoT nodes contain sensors and actuators to interact with the IoT environment. In addition, a networking module must be present through which the node can send and receive data and commands. We describe in this section an IoT node that we built to cover these requirements. The node represents the data anomaly in the AnoML-IoT scenario described in Section III. In this scenario, data from four sensors must be captured and mirrored to the DT namely temperature, humidity, loudness (i.e., microphone), and light sensors. Note that we did not implement the air quality sensor since we noticed that the values of air quality in the dataset are not changing. By looking at Fig. 3, we see that to add WiFi capability to the Arduino board, we stacked in WiFi shield (model: ESP13) on top of it. Afterwards, 5v, 3.3v, ground, and analog connections of the sensors were

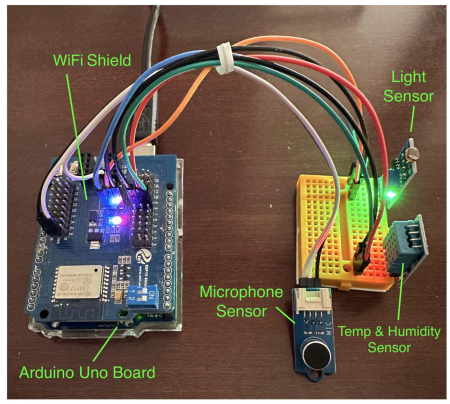


Fig. 3. IoT node implementation.

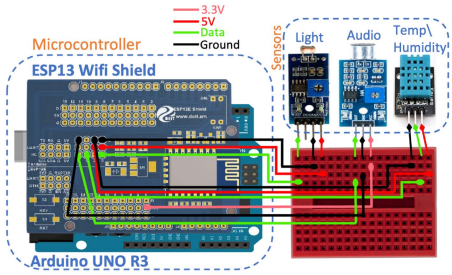


Fig. 4. Schematic diagram.

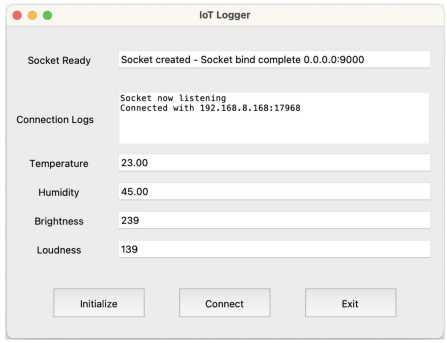


Fig. 5. Software.

connected to the ESP13 shield by means of jumper wires and a small breadboard. The schematic diagram for these connections is shown in Fig. 4. Notice from the figure that the light sensor (model: Photoresistor LDR Light Sensor Module) and temperature and humidity sensors (model: DHT11) needed to pull 5v current from the board. Whereas, the audio sensor pulled 3.3v current from the board to minimize the noise. The three sensor boards produced an analog output that was connected to the Arduino to be read over WiFi. As shown in Fig. 5, we adapted the python software and implementation from hackster.io [55] and changed it to display our four sensor values. Once the python software is connected to the node it can poll the sensor values periodically and push them to the DT via Ditto APIs. Note that the construction of the DT model occurs offline before

TABLE VI
TIME PERFORMANCE FOR DATA ANOMALY CLASSIFIERS BASED ON RF, SVM, AND MLP

	AnoML-IoT				DS2OS			
	Size	Loading (ms)	Fitting (ms)	Classification (ms)	Size	Loading (ms)	Fitting (S)	Classification (ms)
RF	2.4 MB	688.02	515.60	10.26	3.4 MB	28.84	51.99	6.60
SVM	41 KB	1.16	126.12	0.34	20.60 MB	25.19	284.58	2.76
MLP	17 KB	5.96	2,698.22	0.14	146 KB	6.30	56.71	0.819

TABLE VII
TIME PERFORMANCE FOR NETWORK INTRUSION CLASSIFIERS BASED ON RF, SVM, AND MLP

	IoTID20			
	Size	Loading (ms)	Fitting (S)	Classification (ms)
RF	26 MB	69.83	44.77	5.92
SVM	18.3 MB	13.85	1489.21	14.27
MLP	35 KB	2.48	128.00	0.67

posting sensors data by calling Ditto's API and posting the DT configuration JSON shown in Listing 1.

V. PERFORMANCE EVALUATION

We begin in Sections V-A and V-B by gauging the performance of the individual ML models. After that, in Section V-C, we measure the performance of the overall architecture when all the components are put together. Lastly, we compare the performance of our architecture with a reference implementation from the literature based on DT.

A. Data Anomaly Performance Evaluation

The choice of the dataset in terms of its size as well as the type of ML classifier utilized will have a significant impact on the performance of the architecture. The impact of these choices on the performance of the data anomaly model is shown in Table VI. We see from the table that the AnoML-IoT produced smaller model sizes compared to the DS2OS for all classifiers. This is normal since it is smaller in size with 6 K records compared to 350 K in the case of the DS2OS dataset. Consequently, the bigger size of the DS2OS models resulted a higher loading times compared to AnomML-IoT. Model loading time is an important metric as we cannot assume that the model will always be readily available in memory when needed. Therefore, we cover the performance of both loaded models and unloaded models for all scenarios when we measure the overall architecture performance in Section V-C. Beside the loading time, we report the fitting time for the two datasets across the three ML models. We see that the fitting time is very high for the larger DS2OS dataset. We also see that the ML model producing the highest fitting time for AnoML-IoT is MLP whereas the highest fitting time for the DS2OS is produced by SVM. This unpredictable performance for the fitting time is insignificant as the fitting process will take place offline. Finally, we see that time results were consistent for the classification time across the three ML classifiers with RF having the highest classification time and MLP having the lowest. Classification time is very low ($<$1 ms) in the case of MLP making it the most suitable for use in the distributed architecture.

B. Network Intrusion Performance Evaluation

We report the model size and time performance for the network intrusion model that is based on the IoTID20 dataset in Table V. Notice that both RF and SVM created large model sizes in the order of MBs leading to higher loading times. Hence, these models must be carefully introduced to the architecture to ensure that models are preloaded so as to avoid impacting the performance. In terms of fitting time, the results were in harmony with the results achieved with the DS2OS dataset. The lowest fitting time was achieved by RF and the highest by SVM. Also, for classification time, MLP produced the lowest results making it the most suitable for use in a distributed setting.

C. Architectures Performance Evaluation

Finally, we conducted an experiment to measure the performance of the architecture when different possibilities of individual components (i.e., datasets and classifiers) are utilized. Fig. 6 reports the total time for detecting an anomaly when a change in the state of a DT is observed. Notice that we measure the total time at the fog server level since it has the administrative role in the environment in detecting malicious nodes and blocking them. More specifically, the time reported in the figure is the time elapsed between receiving a change in the state from the DT until the classification result is received from data anomaly or network intrusion container to the controller. Experiments cover both scenarios of a preloaded models in memory waiting for classification queries (Loaded) versus only loading a model when the request is received (unloaded). We report the average time from 5 runs for each experiment with standard error bars. Notice from the figure that in most of the cases, a loaded model achieved better time performance when compared to the corresponding case of an unloaded model. However, there are few exceptions (e.g., DS2OS/SVM and DS2OS/MLP scenarios) where the time is comparable. We attribute such performance variability to two reasons. First, Ditto' unstable performance leading to variability form one run to another. Second, Docker's [16] cashing behavior as docker preloads layers from previously loaded images to optimize the performance. Overall, we noticed from Fig. 6 that when models are loaded the total time to detect an anomaly after observing a state changes is 500–600 ms. This reasonable additional time is justified by the added security layer to detect and block malicious nodes.

D. Comparing to a Baseline Solution

Plenty of works exist in the literature investigating anomaly detection in IoT, however, only a few examined systems performance under different design choices. The work by Morgan et al. [56] has the same objective as our work in evaluating

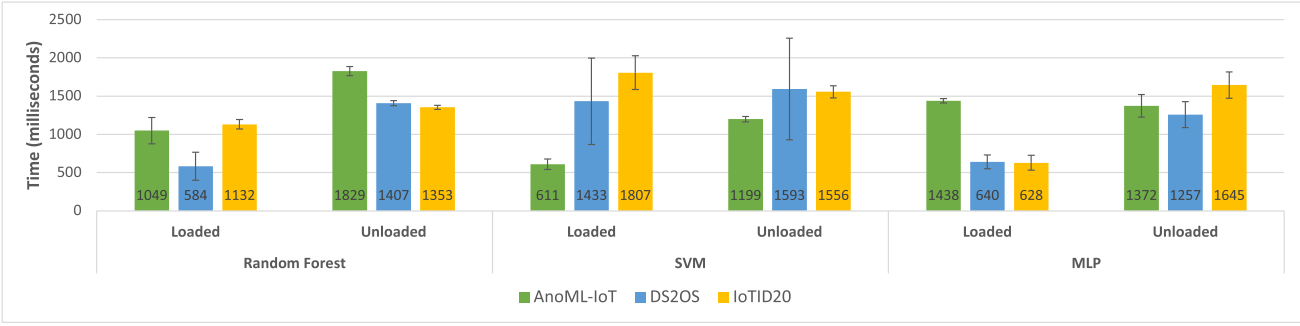


Fig. 6. Time elapsed between DT state change and receiving the classification result for combinations of datasets/ML classifiers/memory loading status.

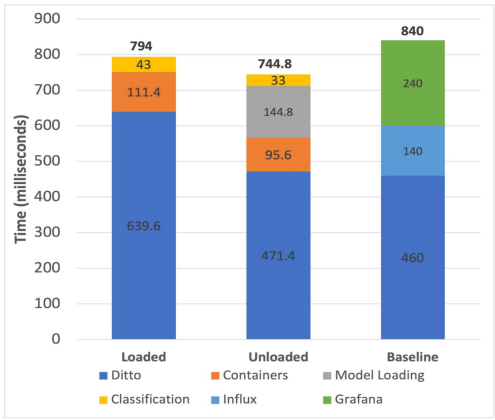


Fig. 7. Comparing the performance of DT architecture for anomaly detection using RF with baseline visualization scenario from [56].

the performance of a DT solution but for supporting data visualization application. They provide an alternative DT solution resembling a pipeline architecture. In addition to Ditto [15], their solution utilizes Hono [57] to integrate with IoT devices and Apache Kafka [58] for real-time streaming of IoT data before storing it into Influx database [59]. In Fig. 7, we plot the average time performance of their architecture, which we label as baseline, against the time performance of our architecture when RF with AnoML-IoT dataset is used with both loaded and unloaded scenarios. Notice from the figure that the Ditto's performance was almost the same when comparing their experiment (i.e., baseline) with our unloaded scenario. However, Ditto had arbitrary performance differences that caused the loaded scenario to perform worse than the loaded scenario. Posting sensor updates by means of Ditto incurs an additional cost of 400 ms on average compared to directly calling the ML models. However, this additional cost brings the various benefits of adopting the DT model discussed in Section I. Since our architecture employs docker, additional cost in our scenario comes from container communication with the cost of almost 100 ms for both loaded and unloaded scenarios. This cost can be compared to the cost of saving data to Influx at 140 ms. Finally, our application involved classification with minimal cost of 30–40 ms but their application involved visualization using Grafana costing 240 ms. It is clear that the choice of involved components, which is governed by the particular application, must be studied carefully as it has significant impact on the system performance.

VI. Discussion

This article discussed many benefits a DT-based node compromise detection brings, however, these benefits come at a cost. It is important to note that the proposed DT infrastructure does incur some performance overhead over a customized solution for node compromise detection, e.g., in general, all sensor data is transmitted in DT irrespective of whether it is relevant for node compromise detection or not, while only the data that is relevant for node compromise detection would be transmitted in a customized solution. On a related note, we have not addressed the scalability issue in this article. If the network gets congested, it would become increasingly difficult to keep the state of the DT in sync with the physical world. We note that we have addressed the scalability issues in DTs in [60], where we proposed a context aware communication control component running at the edge server that can dynamically control data transmission software in the sensing devices in the physical environment to cope up with current context such as a congested network. This technique can be used in the architecture proposed in this article. In addition to the scalability issue, we have not addressed the issue of DT framework security in this article, e.g., a brute force attack on the framework would compromise our solution. This can be addressed by using appropriate security mechanisms, such as blockchains to secure the framework [34]. We have used two models, data anomaly and network intrusion detection models to detect node compromises. It is important to note that both of these models are combined into one DT model, which is a significant benefit of the proposed DT-based node compromise detection. Furthermore, as discussed in [35], a DT-based architecture can be used to detect collaborative anomalies where individual data points could represent normality while as a group these points could represent an anomaly. In summary, to complete the coverage of this work, it is better to diversify the intrusion detection models used and study the impact on the performance versus the accuracy gain each model brings which we leave as a future work. Also, more experimental work is needed to study the impact of the communication overhead that can occur between the edge server and the cloud server.

VII. Conclusion

This article demonstrated the design, implementation, and evaluation of an architecture for compromised IoT node detection with three important features. First, the concept of DT is utilized to only replicate relevant information needed for malicious behavior detection. Second, classification is performed in the fog layer to enable low latency access. Third, the architecture combines data anomaly and network intrusion methods for better coverage of malicious behavior detection. Evaluation of the architecture shows good accuracy and negligible overhead. In the future, we plan to vary system components related to the communication/application and study the impact on the performance. For example, connectivity using bluetooth low energy (BLE) or long range communication (LoRa) can be explored to serve different applications. Also, we plan to explore system scalability under different scenarios.

References

[1] M. Noura, M. Atiquzzaman, and M. Gaedke, "Interoperability in Internet of Things: Taxonomies and open challenges," *Mobile Netw. Appl.*, vol. 24, pp. 796–809, 2019.

[2] M. Blackstock and R. Lea, "IoT interoperability: A hub-based approach," in *Proc. Int. Conf. Internet Things*, 2014, pp. 79–84.

[3] P. Desai, A. Sheth, and P. Anantharam, "Semantic gateway as a service architecture for IoT interoperability," in *Proc. IEEE Int. Conf. Mobile Serv.*, 2015, pp. 313–319.

[4] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[5] D.-Y. Jeong et al., "Digital twin: Technology evolution stages and implementation layers with technology elements," *IEEE Access*, vol. 10, pp. 52609–52620, 2022.

[6] M. Singh, E. Fuenmayor, E. P. Hinchy, Y. Qiao, N. Murray, and D. Devine, "Digital twin: Origin to future," *Appl. Syst. Innov.*, vol. 4, no. 2, 2021, Art. no. 36.

[7] S. A. Varghese, A. D. Ghadim, A. Balador, Z. Alimadadi, and P. Papadimitratos, "Digital twin-based intrusion detection for industrial control systems," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops Other Affiliated Events*, 2022, pp. 611–617.

[8] L. Raes et al., "Duet: A framework for building interoperable and trusted digital twins of smart cities," *IEEE Internet Comput.*, vol. 26, no. 3, pp. 43–50, May/Jun. 2022.

[9] M. Wu, Z. Song, and Y. B. Moon, "Detecting cyber-physical attacks in cybermanufacturing systems with machine learning methods," *J. Intell. Manuf.*, vol. 30, pp. 1111–1123, 2019.

[10] "Stuxnet." Accessed: Jun. 2024. [Online]. Available: https://www.malwarebytes.com/stuxnet

[11] K. Zetter, "A cyberattack has caused confirmed physical damage for the second time ever," 2015. [Online]. Available: https://www.wired.com/2015/01/german-steel-mill-hack-destruction/

[12] A. Kumar and T. J. Lim, "Early detection of mirai-like IoT bots in large-scale networks through sub-sampled packet traffic analysis," in *Proc. Int. Conf. Adv. Inf. Commun.*, 2020, pp. 847–867.

[13] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani, "Demystifying IoT security: An exhaustive survey on IoT vulnerabilities and a first empirical look on internet-scale IoT exploitations," *IEEE Commun. Surv. Tut.*, vol. 21, no. 3, pp. 2702–2733, Third Quarter 2019.

[14] R. Minerva, G. M. Lee, and N. Crespi, "Digital twin in the IoT context: A survey on technical features, scenarios, and architectural models," *Proc. IEEE*, vol. 108, no. 10, pp. 1785–1824, Oct. 2020.

[15] "Eclipse ditto." Accessed: Jun. 2024. [Online]. Available: https://eclipse.org/ditto/

[16] "Docker." Accessed: Jun. 2024. [Online]. Available: https://www.docker.com/

[17] Z. Jiang, Y. Guo, and Z. Wang, "Digital twin to improve the virtual-real integration of Industrial IoT," *J. Ind. Inf. Integration*, vol. 22, 2021, Art. no. 100196.

[18] A. Canedo, "Industrial IoT lifecycle via digital twins," in *Proc. 11th IEEE/ACM/IFIP Int. Conf. Hardware/Softw. Codesign Syst. Synth.*, 2016, pp. 1–1.

[19] L. R. Delfino, A. S. Garcia, and R. L. de Moura, "Industrial Internet of Things: Digital twins," in *Proc. SBMO/IEEE MTT-S Int. Microw. Optoelectron. Conf.*, 2019, pp. 1–3.

[20] Z. Zhao, L. Shen, C. Yang, W. Wu, M. Zhang, and G. Q. Huang, "IoT and digital twin enabled smart tracking for safety management," *Comput. Operations Res.*, vol. 128, 2021, Art. no. 105183.

[21] Y. Liu et al., "A novel cloud-based framework for the elderly healthcare services using digital twin," *IEEE Access*, vol. 7, pp. 49088–49101, 2019.

[22] D. D. Eneyew, M. A. Capretz, and G. T. Bitsuamlak, "Toward smart-building digital twins: BIM and IoT data integration," *IEEE Access*, vol. 10, pp. 130487–130506, 2022.

[23] C. Verdouw, B. Tekinerdogan, A. Beulens, and S. Wolfert, "Digital twins in smart farming," *Agricultural Syst.*, vol. 189, 2021, Art. no. 103046.

[24] N. Kharlamova and S. Hashemi, "Evaluating machine-learning-based methods for modeling a digital twin of battery systems providing frequency regulation," *IEEE Syst. J.*, vol. 17, no. 2, pp. 2698–2708, Jun. 2023.

[25] X. Lin, J. Wu, J. Li, W. Yang, and M. Guizani, "Stochastic digital-twin service demand with edge response: An incentive-based congestion control approach," *IEEE Trans. Mobile Comput.*, vol. 22, no. 4, pp. 2402–2416, Apr. 2023.

[26] R. Revetria, F. Tonelli, L. Damiani, M. Demartini, F. Bisio, and N. Peruzzo, "A real-time mechanical structures monitoring system based on digital twin, IoT and augmented reality," in *Proc. Spring Simul. Conf.*, 2019, pp. 1–10.

[27] A. Y. Khan, R. Latif, S. Latif, S. Tahir, G. Batool, and T. Saba, "Malicious insider attack detection in IoTs using data analytics," *IEEE Access*, vol. 8, pp. 11743–11753, 2019.

[28] H. Kayan, Y. Majib, W. Alsafery, M. Barhamgi, and C. Perera, "Aanoml-IoT: An end to end re-configurable multi-protocol anomaly detection pipeline for Internet of Things," *Internet Things*, vol. 16, 2021, Art. no. 100437.

[29] F. Cauteruccio et al., "A framework for anomaly detection and classification in multiple IoT scenarios," *Future Gener. Comput. Syst.*, vol. 114, pp. 322–335, 2021.

[30] G. Sebastian, P. Agustin, and E. M Jose, "IoT-23: A labeled dataset with malicious and benign IoT network traffic (version 1.0.0) [data set]. zenodo." Accessed: Jun. 2024. [Online]. Available: https://www.stratosphereips.org/datasets-iot23

[31] A. A. Diro and N. Chilamkurti, "Distributed attack detection scheme using deep learning approach for Internet of Things," *Future Gener. Comput. Syst.*, vol. 82, pp. 761–768, 2018.

[32] S. Raza, L. Wallgren, and T. Voigt, "Svelte: Real-time intrusion detection in the Internet of Things," *Ad Hoc Netw.*, vol. 11, no. 8, pp. 2661–2674, 2013.

[33] M. Shafiq, Z. Tian, A. K. Bashir, X. Du, and M. Guizani, "CorrAUC: A malicious Bot-IoT traffic detection method in IoT network using machine-learning techniques," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3242–3254, Mar. 2021.

[34] M. M. Salim, A. K. Comivi, T. Nurbek, H. Park, and J. H. Park, "A blockchain-enabled secure digital twin framework for early Botnet detection in IIoT environment," *Sensors*, vol. 22, no. 16, 2022, Art. no. 6133.

[35] P. Bascones, A. Voisin, P. Do, and M. A. Sanz-Bobi, "A collaborative network of digital twins for anomaly detection applications of complex systems. snitch digital twin concept," *Comput. Ind.*, vol. 144, 2023, Art. no. 103767.

[36] F. Piltan and J.-M. Kim, "Bearing anomaly recognition using an intelligent digital twin integrated with machine learning," *Appl. Sci.*, vol. 11, no. 10, 2021, Art. no. 4602.

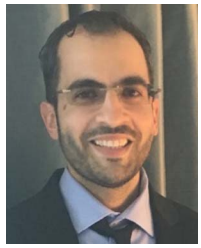
[37] Q. Xu, S. Ali, and T. Yue, "Digital twin-based anomaly detection in cyber-physical systems," in *Proc. 14th IEEE Conf. Softw. Testing, Verification Validation*, 2021, pp. 205–216.

[38] C. Yang et al., "Digital twin-driven fault diagnosis method for composite faults by combining virtual and real data," *J. Ind. Inf. Integration*, vol. 33, 2023, Art. no. 100469.

[39] H. Huang, L. Yang, Y. Wang, X. Xu, and Y. Lu, "Digital twin-driven online anomaly detection for an automation system based on edge intelligence," *J. Manuf. Syst.*, vol. 59, pp. 138–150, 2021.

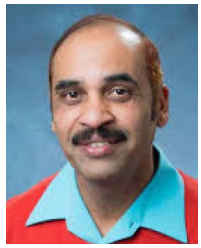
[40] Q. Lu, X. Xie, A. K. Parlikad, and J. M. Schooling, "Digital twin-enabled anomaly detection for built asset monitoring in operation and maintenance," *Autom. Construction*, vol. 118, 2020, Art. no. 103277.

[41] D. Gupta, O. Kayode, S. Bhatt, M. Gupta, and A. S. Tosun, "Hierarchical federated learning based anomaly detection using digital twins for smart healthcare," in *Proc. IEEE 7th Int. Conf. Collaboration Internet Comput.*, 2021, pp. 16–25.
[42] H.-J. Cha, H.-K. Yang, Y.-J. Song, and A. R. Kang, "Intelligent anomaly detection system through Malware image augmentation in IIoT environment based on digital twin," *Appl. Sci.*, vol. 13, no. 18, 2023, Art. no. 10196.
[43] A. Oluwasegun and J.-C. Jung, "The application of machine learning for the prognostics and health management of control element drive system," *Nucl. Eng. Technol.*, vol. 52, no. 10, pp. 2262–2273, 2020.
[44] R. Sahal, S. H. Alsamhi, J. G. Breslin, K. N. Brown, and M. I. Ali, "Digital twins collaboration for automatic erratic operational data detection in industry 4.0," *Appl. Sci.*, vol. 11, no. 7, 2021, Art. no. 3186.
[45] Z. Liu et al., "An integrated architecture for IoT Malware analysis and detection," in *Proc. 4th EAI Int. Conf. IoT Serv.*, 2019, pp. 127–137.
[46] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug. 2019.
[47] G. N. Schroeder, C. Steinmetz, R. N. Rodrigues, R. V. B. Henriques, A. Rettberg, and C. E. Pereira, "A methodology for digital twin modeling and deployment for Industry 4.0," *Proc. IEEE*, vol. 109, no. 4, pp. 556–567, Apr. 2021.
[48] F. Aubet and M. Pahl, "DS2OS traffic traces: IoT traffic traces gathered in a the DS2OS IoT environment." Accessed: Jun. 2024. [Online]. Available: https://www.kaggle.com/datasets/francoisxa/ds2ostraffictraces
[49] I. Ullah and Q. H. Mahmoud, "A scheme for generating a dataset for anomalous activity detection in IoT networks," in *Proc. Can. Conf. Artif. Intell.*, 2020, pp. 508–520.
[50] "Arduino," [Online]. Available: https://docs.arduino.cc/hardware/uno-rev3
[51] "Wifi shield." Accessed: Jun. 2024. [Online]. Available: https://docs.arduino.cc/retired/shields/arduino-wifi-shield
[52] "Postman api platform." Accessed: Jun. 2024. [Online]. Available: https://www.postman.com/
[53] "Pickle." Accessed: Jun. 2024. [Online]. Available: https://docs.python.org/3/library/pickle.html
[54] "Scikit." Accessed: Jun. 2024. [Online]. Available: http://scikit-learn.org
[55] kpower at hackster.io, "IoT with arduino and esp13 Wi-Fi shield." Accessed: Jun. 2024. [Online]. Available: https://www.hackster.io/umpheki/iot-with-arduino-and-esp13-wifi-shield-c93b08
[56] V. Kamath, J. Morgan, and M. I. Ali, "Industrial IoT and digital twins for a smart factory: An open source toolkit for application design and benchmarking," in *Proc. Glob. Internet Things Summit*, 2020, pp. 1–6.
[57] "Hono." Accessed: Jun. 2024. [Online]. Available: https://projects.eclipse.org/projects/iot.hono
[58] "Apache kafka." Accessed: Jun. 2024. [Online]. Available: https://kafka.apache.org/
[59] "Influxdb." Accessed: Jun. 2024. [Online]. Available: https://www.influxdata.com/
[60] K. Alanezi and S. Mishra, "Towards a scalable architecture for building digital twins at the edge," in *Proc. 1st ACM/IEEE Workshop Digit. Twins*, 2023, pp. 325–329.



Khaled Alanezi received the B.Sc. degree in computer engineering from Kuwait University, Kuwait City, Kuwait, in 2003 and the M.Sc. and Ph.D. degrees in computer science from the Department of Computer Science, University of Colorado, Boulder, Boulder, CO, USA, in 2012 and 2016, respectively.

He is currently an Assistant Professor with the College of Computing and Systems, Abdullah Al Salem University, Safat, Kuwait. His current research interests include edge computing, Internet of Things, and digital twins.



Shivakant Mishra received the Ph.D. degree in computer science from the University of Arizona, Tucson, USA, in 1992.

He is currently a Professor with the Department of Computer Science, The University of Colorado at Boulder, Boulder, CO, USA. His research interests include edge computing, cybersafety, mobile and pervasive computing, and large scale distributed computing.