Threat Management in Data-centric IoT-Based Collaborative Systems

MUHAMAD FELEMBAN, KFUPM, Saudi Arabia

EMAD FELEMBAN, Umm Al-Qura, Saudi Arabia

JASON KOBES, Northrop Grumman

ARIF GHAFOOR, Purdue University, USA

In this article, we propose a threat management system (TMS) for Data-centric Internet-of-Things-based Collaborative Systems (DIOtCSs). In particular, we focus on tampering attacks that target shared databases and can affect the execution of the DIOtCS services. The novelty of the proposed system is to isolate the damage caused by tampering attacks into data partitions. We formulate the partitioning problem as a cost-driven optimization problem, prove its NP-hardness, and propose two polynomial-time heuristics. We evaluate a TMS experimentally and demonstrate that intelligent partitioning of the database improves the overall availability of the DIOtCS.

CCS Concepts: • **Security and privacy** → **Malware and its mitigation**; **Database and storage security**;

Additional Key Words and Phrases: Threat management, Internet-of-Things, database systems, cost-driven optimization

ACM Reference format:

Muhamad Felemban, Emad Felemban, Jason Kobes, and Arif Ghafoor. 2019. Threat Management in Data-centric IoT-Based Collaborative Systems. *ACM Trans. Internet Technol.* 19, 3, Article 42 (August 2019), 19 pages.

<https://doi.org/10.1145/3323232>

1 INTRODUCTION

The Internet-of-Things (IoT) interconnects physical devices and objects that can sense, communicate, and actuate. A basic example of such devices is the HVAC (Heating, Ventilation, and Air Conditioning) control system in smart homes [1]. The IoT plays a remarkable role in other domains, including intelligent transportation [16] and smart grids [30]. It is expected that the number of IoT devices will reach 50 billion by 2020 as a result of the fourth industrial revolution (Industry 4.0) [32, 35]. IoT devices generate a vast amount of data that can be used to perform data-centric services. In

This research was supported by grants from Northrop Grumman Corporation and US National Science Foundation (NSF) Grant IIS-0964639.

Authors' addresses: M. Felemban, COE P.O Box 5065 KFUPM, Dhahran, 31261, Saudi Arabia; email: mfelemban@kfupm.edu.sa; E. Felemban, Umm Alqura University, AlAwali Makkah 24381, Saudi Arabia; email: eafelemban@uqu.edu.sa; J. Kobes, Northrop Grumman; email: jason.kobes@ngc.com; A. Ghafoor, Purdue University, 610 Purdue Mall, West Lafayette, IN, 47907, USA; email: ghafoor@purdue.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2019 Association for Computing Machinery.

1533-5399/2019/08-ART42 \$15.00

<https://doi.org/10.1145/3323232>

Fig. 1. Example of DIoTCS.

such services, data is collected, processed, disseminated, and shared among collaborative users to extract useful information and make decisions. The fundamental characteristic of the data-centric services is that users are oblivious about the location where the data resides. To this end, cloud computing has been used to support data-centric services for large-scale organizations [34].

In this article, we consider Data-centric IoT-Based Collaborative Systems (DIoTCSs) that use the Cloud as a platform for storing data and executing workflows. A workflow is a set of dependent data-centric tasks that need to be executed in real time in a particular order to achieve certain business or operational functionality. Several cloud-based data-storage solutions have been proposed to support the execution of workflows. Examples of such data storage include Relational Database Management Systems (RDBMSs), NoSQL DBMS, and Resource Description Framework (RDF)-based data storage [10]. Each task can be performed by executing a set of atomic transactions that read data, perform computation, and update data storage accordingly. An example of DIoTCSs can be found in several applications, including health care management [33] and manufacturing [8].

Since data is the most valuable asset of an organization, it is imperative to design DIoTCSs such that the confidentiality, integrity, and availability of the data are protected. DIoTCSs can be exposed to a large amount of vulnerabilities due to several reasons, including implementation flaws and no or weak authentication in IoT devices [36]. A prominent example of attacks is the Mirai Botnet, which exploits vulnerabilities in IoT devices and network components and launches a distributed denial-of-service attack [5]. In this article, we focus on data-tampering attacks that aim at deliberately modifying—i.e., destroying, manipulating, or editing—data through unauthorized channels. Such attacks can lead to severe consequences, including making wrong decisions and missing tasks' execution deadlines [15]. The following example illustrates a data-tampering attack scenario on a DIoTCS.

Motivating Example. Figure 1(a) depicts a DIoTCS for a manufacturing organization with different departments, e.g., Marketing, Design, Logistics, Assembly, and Manufacturing, that access a cloud-based shared database. The objective is to execute business and operational workflows to design, manufacture, and deliver products. Figure 1(b) depicts a fictitious example of a workflow pertaining to monitoring the safety of the shop floor. The IoT sensors periodically collect and update the database with real-time data regarding the physical state of the assembly line machines. For this purpose, a “monitoring” workflow executes a task that continuously checks the collected data to find any incidents. In the case that an incident occurs, the workflow executes two parallel tasks: (1) shutting down the assembly line machines and (2) triggering the safety alarm and alerting

the safety department. Suppose that a malicious employee tampers with the database to cause an incident. Such action can lead to bad consequences on the DIoTCS. For example, it can result in halting the assembly line(s), which can result in loss of productivity.

1.1 Contribution

In this article, we propose a threat management system (TMS) that counters the data-tampering attacks on DIoTCSs. We propose a novel data partition-based response and recovery mechanism for DIoTCSs. The framework is based on an adaptive access (admission) control mechanism that responds to attacks by selectively blocking the part of the data that has been tampered with. The access control provides a fine-grained control mechanism to allow a graceful degradation of the services and maintain a desired level of availability.

The contribution of this article is summarized as the following. First, we introduce a new model, termed a *time-constraint transaction execution graph* (TTEG), that captures the execution dependencies and time constraints among transactions in DIoTCSs. Second, we propose the concept of containment regions (CRs) that defines the extent of damage over shared data in DIoTCSs. We formulate the CR partitioning problem (CRPP) as a cost-driven optimization problem. We prove that the CRPP is NP-hard and, accordingly, propose two heuristics to provide a polynomial-time solution. Finally, we present data partition-based response and recovery mechanisms that use the resulting CR assignment to improve threat response and recovery in terms of availability and response time.

1.2 Article Organization

The rest of the article is organized as follows. In Section 2, relevant background is discussed. In Section 3, the definition and formulation of CRPP are introduced. The problem is shown to be NP-hard; accordingly, two heuristics are proposed. Section 4 describes the design and implementation of a TMS. The performance evaluation and experimental results are provided in Section 5. The related work is presented in Section 6, and Section 7 contains our conclusions.

2 PRELIMINARIES

In this section, we present the architecture and describe the threat model for a DIoTCS.

2.1 The Architecture of a DIoTCS

Figure 2(a) illustrates the architecture of a DIoTCS that consists of three layers [1]: (1) the perception layer, (2) the network layer, and (3) the application layer. The perception layer represents the physical devices—i.e., sensors and actuators—that aim at collecting data, processing the collected data, transmitting the information to upper layers, receiving commands from the application layer, and actuating devices in the physical world. The network layer is responsible for routing the data to the application layer. The network layer interconnects heterogeneous IoT devices using various communication protocols, e.g., 6LoWPAN and Zigbee [1].

The cloud-based application layer provides high-level services that are required by the DIoTCS. Several protocols can be used to exchange data between the application layer and the perception layer, including the Constraint Application Protocol (CoAP), Advanced Message Queuing Protocol, Message Queue Telemetry Transport (MQTT), and Data Distribution Service (DDS) [21]. In addition, the application layer provides a platform to execute the business and operational workflows, including the Workflow Management System (WfMS) and Business Process Management System (BPMS) [12].

Within the application layer, we identify two sublayers: the data management and service management sublayers. The data management sublayer is responsible for managing the data collected

Fig. 2. DIoTCS architecture.

by IoT devices. Typically, IoT applications receive data from a diverse set of sensors and in different formats, e.g., relational, time-series, and location data. As a result, different types of data management solutions—e.g., RDBMS, NoSQL DBMS, RDF-based DBMS, Online Analysis Processing (OLAP), graph processing, and stream processing—need to support DIoTCS to provide real-time data ingestion and analysis [6].

The service management sublayer executes the workflows. In particular, an application service uses the stored data to summarize raw sensory measurements and eventually make decisions and react accordingly. Therefore, there is a strong interaction between the service management and data management components in a DIoTCS. The application layer provides other features as well, including data analysis, user authentication, and access control to IoT devices.

In this article, we consider only relational data that can be stored in RDBMSs, which application services can interact with by executing transactions to read/update data. Figure 2(b) depicts a conceptual view of the interaction among IoT devices, data management, and service management planes. According to this figure, there are three types of transactions. First, the upstream transactions generated by IoT sensors are used to update the database with the real-time status of the physical objects. Second, the downstream transactions are used to trigger IoT actuators. Third, the workflow transactions perform business or operational functionalities on the collected data. A formal definition of the transaction is presented in Section 3.1.

2.2 Threat Model for DIoTCS

In this article, we consider data-tampering attacks on databases [3]. In particular, we focus on attacks that target cloud-based shared data among collaborative users (the middle plane in Figure 2(b)). The attacks can be performed either by exploiting application vulnerabilities—e.g., SQL injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF) [29]—or by physical manipulation of IoT devices [1]. Owing to the collaborative environment, the damage in the shared database plane can affect the benign tasks accessing the shared data. Moreover, owing to the natural interaction between the cyber and physical world in IoT applications, such tampering attacks can pose a threat to the business and safety of organizations.

3 DATA-DRIVEN CONTAINMENT REGION PARTITIONING

In this section, we propose a model to represent the execution dependencies and time constraints among transactions in DIoTCSs pertaining to the top plane in Figure 2(b). Subsequently, we present

Fig. 3. An example of a transaction execution graph and a possible CR partition with $k = 3$.

the data-driven containment region problem and formulate it as a cost-driven optimization problem. We prove that the problem is NP-hard and present two polynomial time heuristics to solve the problem.

3.1 Time-Constrained Transaction Execution Graph Model for DIoTCSs

DIoTCSs execute business and operational workflows that generally consist of a set of tasks, where each task executes one or more transactions on a shared database. We model the execution of the transactions as a weighted graph, i.e., $TTEG = \{V, E, W\}$, where V represents the set of transactions T , E is a set of edges representing dependencies among transactions, and W is a function over the set of edges, i.e., $W : E \rightarrow \mathcal{R}^2$, which models the labels on the edges, as discussed below.

In the database context, a transaction t_i is a partially ordered set of operations, with $<_i$ as ordering relation [7]. The operations are performed on a set of n objects denoted by $D = \{o_1, o_2, \dots, o_n\}$. The read and write (update) operations of a transaction $t_i \in T$ on an object $o \in D$ are denoted by $r_i[o]$ and $w_i[o]$, respectively. The set of objects that are read and updated by t_i are denoted by R_i and W_i , respectively. The size of t_i is $|t_i| = |R_i \cup W_i|$. Formally, t_i is defined as the following [7]:

- (1) $t_i \subseteq \{w_i[o], r_i[o] | o \in D\} \cup \{a_i, c_i\}$,
- (2) if $w_i[o], r_i[o] \in t_i$, then either $w_i[o] <_i r_i[o]$ or $r_i[o] <_i w_i[o]$,
- (3) $a_i \in t_i$ if and only if $c_i \notin t_i$, and
- (4) for any operation $p \in t_i$, $p <_i t$, where t is either a_i or c_i (whichever in t_i)

where a_i and c_i denote the *abort* and *commit* operations of t_i , respectively. A history H over a set of m transactions $T = \{t_1, t_2, \dots, t_m\}$ is a partial order with the ordering relation $<_H$, where $H = \cup_{i=1}^m t_i$ and $<_H \supseteq \cup_{i=1}^m <_i$. Two transactions t_i and t_j in H are *dependent* if (i) t_i is directly followed by t_j , i.e., $t_i <_H t_j$; and (ii) $W_{t_i} \cap R_{t_j} \neq \emptyset$. The dependency between t_i and t_j is denoted by $t_i \rightarrow t_j$ and reads “ t_j depends on t_i .” In general, t_j depends on t_i if (i) $t_i <_H t_{k_1} <_H \dots <_H t_{k_n} <_H t_j$, and (ii) $(W_{t_i} - \cup_{l=1, \dots, n} W_{t_{k_l}}) \cap R_{t_j} \neq \emptyset$.

An edge—say, $e_{ij} \in E$ —between two transactions, t_i and t_j , exists if $t_i \rightarrow t_j$. The label on each edge e_{ij} between two transactions t_i and t_j is a pair $\langle SO_{ij}, \delta_{ij} \rangle$. The first label is SO_{ij} , which represents the set of shared objects between t_i and t_j , i.e., $SO_{ij} = W_{t_i} \cap R_{t_j}$. The second label is δ_{ij} , which represents the upper time limit, i.e., deadline, within which a transaction t_j has to be executed after transaction t_i is executed. Such time constraints can be obtained from the quality-of-service (QoS) requirements of real-time applications [15]. Failure to meet these time constraints may result in an incorrect or unsafe operation of the DIoTCS. For example, the execution graph model for a set of transactions $T = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}\}$ is depicted in Figure 3(a), where the first four transactions in T represents the tasks depicted in Figure 1(b). In Figure 3(a), t_2 depends

on t_1 because t_2 reads 25 data objects that are previously updated by t_1 . The time constraint between t_1 and t_2 is 500ms, which indicates that transaction t_2 must be executed within 500ms after the successful execution of t_1 .

We refer to the transaction that tampers with the data as a **malicious** transaction. A transaction is termed **affected** if it directly (or indirectly) depends on a malicious transaction. For example, assume that transaction t_3 is malicious in Figure 3(a). Then, transactions t_4 , t_5 , t_7 , and t_6 are affected transactions. All malicious and affected transactions are invalid transactions that can cause damage to the database.

3.2 CR Partitioning Problem

The objective of the CRPP is to contain the damage within a single CR without severely affecting the execution time constraints of the transactions. This can be achieved by partitioning the TTEG into k subgraphs, i.e., k CRs. The damage can be contained within a CR by minimizing the “outgoing” edges to other CRs. Alternatively, the damage can be contained by minimizing the number of shared objects represented with $|SO|$ on these outgoing edges. Such shared objects are termed as **boundary** objects. During runtime, the access to the boundary objects is delayed in order to stop the damage from propagating to other CRs. This mechanism is explained in Section 4.1. However, delaying access to the boundary objects can result in violating the execution time constraints. Accordingly, CRPP is defined as the following.

Definition 3.1 (Containment Region Partitioning Problem). Given the TTEG that is generated by a set of transactions T over a set of n objects, the CRPP is to partition TTEG into k CRs such that the number of boundary objects and number of transactions violating their execution time constraints are minimized.

3.2.1 Problem Formulation. We formulate the CRPP as a dual-objective optimization problem. The first objective function focuses on damage containment, which can be achieved by minimizing the number of shared objects, represented by $|SO_{ij}|$, in the crossing edges among CRs. Accordingly, the objective function can be defined as the following.

$$f_1(X) = \frac{\sum_{\ell=1}^k \sum_{i=1}^m \sum_{j=i+1}^m |SO_{ij}| x_{i\ell} (1 - x_{j\ell})}{\sum_{e \in E} |SO_e|}, \quad (1)$$

where X is the transactions to CR assignment matrix, i.e., $x_{i\ell} = 1$ if the i^{th} transaction is assigned to the ℓ^{th} CR and $\sum_{e \in E} |SO_e|$ is the sum of the number of shared objects among all transactions.

The second objective function is to minimize the effect of partitioning on the transactions' execution time constraints. This can be achieved by partitioning the TTEG such that the crossing edges among CRs has the maximum delay weights. Formally, the objective function is expressed as the following.

$$f_2(X) = \frac{\sum_{\ell=1}^k \sum_{i=1}^m \sum_{j=i+1}^m \delta_{ij} x_{i\ell} (1 - x_{j\ell})}{\sum_{e \in E} \delta_e}. \quad (2)$$

Consequently, the CRPP is formulated as a cost-driven optimization problem using Equations (1) and (2) as the following.

$$\underset{X}{\text{Minimize}} \quad f_1(X) - f_2(X) \quad (3)$$

subject to

$$\left| \sum_{i=1}^m x_{ij} |t_i| - \sum_{i=1}^m x_{i\ell} |t_i| \right| \leq 2\epsilon \frac{m}{k} \quad \forall j \neq \ell \in \{1, \dots, k\} \quad (4)$$

$$\sum_{j=1}^k x_{ij} = 1 \quad \forall i \in \{1, \dots, m\} \quad (5)$$

$$\sum_{i=1}^m x_{ij} \geq 1 \quad \forall j \in \{1, \dots, k\} \quad (6)$$

$$x_{ij} \in \{0, 1\} \quad (7)$$

$$\forall i \in \{1, \dots, m\}, \forall j \in \{1, \dots, k\}.$$

The solution for the CRPP is the assignment matrix X . Constraint (4) enforces a balanced-transactions assignment to CRs. Constraint (5) enforces the assignment of a transaction to only one CR, while Constraint (6) prevents generation of empty CRs. Constraint (7) limits the values of X to be binary. An example of an assignment that results from solving the CRPP with $k = 3$ is depicted in Figure 3(b). In this figure, transactions t_1, t_2 , and t_3 are assigned to CR_1 ; transactions t_4, t_5 , and t_6 are assigned to CR_3 ; and transactions t_7, t_8, t_9 , and t_{10} are assigned to CR_2 . Note that the CRPP is similar to the (k, v) Balanced Graph Partitioning Problem [4], which is NP-hard.

THEOREM 3.1. *The CRPP is NP-hard.*

PROOF. We first formulate the decisional CRPP as the following: *Given the TTEG, does there exist a valid partition into k CRs such that $f_1(X) - f_2(X) \leq Q$?* The proof is by reduction from the partition problem. First, a valid solution to the decisional CRPP can be checked in polynomial time. Thus, the CRPP is in NP. Second, we show that the CRPP contains the Partition problem as a special case. Let the set of items A in the Partition problem be the set of transactions T , i.e., an item $a \in A$ in the Partition problem is a transaction $t_i \in T$ in the CRPP. This implies that $s(a) = |t_i|$, $\forall a \in A$, and $t_i \in T$, where $s(a)$ is the size of item a and $|t_i|$ is the size of the transaction t_i . Also, let $\epsilon = 0$. Consequently, Constraint (4) will be $|\sum_{i=1}^m x_{ij}|t_i| - \sum_{i=1}^m x_{i\ell}|t_i|| \leq 0$. We restrict the set of transactions T to be nonoverlapping, i.e., $t_i \cap t_j = \emptyset, \forall i \neq j$. As a result, there are no shared objects between any pair of transactions, i.e., $SO_e = 0, \forall e \in E$. Moreover, let $\delta_e = 0$ for every $e \in E$. Thus, $w(e) = \{0, 0\}, \forall e \in E$. Finally, let $k = 2$ and $Q = 0$. We claim that there exists a solution to the decisional CRPP problem if and only if there exists a solution to the Partition problem over set A , i.e., $\sum_{a \in A} s(a) = \sum_{a \in A'} s(a)$.

Suppose that there exists a valid solution to the decisional CRPP. In that case, the solution is also a valid solution to the Partition problem because $|\sum_{i=1}^m x_{i1}|t_i| - \sum_{i=1}^m x_{i2}|t_i|| \leq 0$, i.e., $|\sum_{i=1}^m x_{i1}|t_i| \leq \sum_{i=1}^m x_{i2}|t_i|$, which implies that $\sum_{a \in A} s(a) = \sum_{a \in A'} s(a)$. In the other direction, suppose that there exists a solution to the Partition problem. In this case, the solution is also a valid solution for the decisional CRPP since $f_1(X) - f_2(X) \leq 0$ and other constraints are satisfied. $\square$

3.2.2 Control of CRs in a Collaborative System. The CR assignment matrix is stored in the CR assignment table that is used by the TMS when the DIoTCS encounters attacks. More details about the response and recovery mechanisms are provided in the next section. The CRs can be implemented using physical or logical partitioning of the database. Physical partitioning of the database implies placing CRs in different machines, e.g., using cloud computing platforms. In this article, we assume that the DIoTCS uses a shared database in which all of the transactions are executed. CRs can be implemented as logical partitions using Access Control Lists (ACLs). Fine-grained ACLs—e.g., Oracle VPD [9] and MS SQL [26], which control access at the object level—are common in commercial and open-source DBMSs.

3.3 Heuristics for CR Partitioning

We propose two greedy-based heuristics to solve the CRPP. All algorithms take a TTEG and the number of CRs as inputs. The output is the transactions-to-CRs assignment. The algorithms start

with an empty CR assignment and iteratively assign transactions to CRs based on a greedy decision that optimizes the objective function. The first heuristic is termed an Edge-Based Assignment (EBA), which attempts to minimize the value of the objective function by searching for the best CR for each transaction. The second heuristic is termed a Neighbor-Based Assignment (NBA), which traverses the TTEG in a depth-first order and assigns each transaction to the smallest CR. Detailed discussion about each algorithm is presented in the following sections.

3.4 Edge-Based Assignment (EBA)

An EBA is presented in Algorithm 1. The algorithm starts with an empty CR assignment set $\mathcal{S}$ and empty assigned transactions set A (Line 1). Subsequently, the algorithm loops until all transactions are assigned (Lines 2–28). For each transaction t , the set of neighboring transactions (i.e., $InNeighbors$) that have an edge coming into t is found (Line 4). If $InNeighbors$ is empty, then t is assigned to the smallest CR. Otherwise, the CRs of each “assigned” transaction $t_n \in InNeighbors$ are stored in $feasibleCR$ (Lines 11–15). If $feasibleCR$ is not empty, then t is assigned to the “best” CR in $feasibleCR$ that has the minimum value of objective function (Lines 16–19). Otherwise, t is assigned to the best CR in $\mathcal{S}$ (Lines 20–23).

LEMMA 3.1. *The complexity of the EBA is $O(k|T|^3)$, where $|T|$ is the number of transactions.*

PROOF. The loop in Lines 2 to 28 assigns a single transaction to each empty CR. Thus, the loop has a running complexity of $O(|T|)$. During each iteration, the set of $InNeighbors$ is found (Line 4), which has a running complexity of $O(|T|)$. Assigning the transaction and evaluating the objective function in Lines 17 and 21 have a total running complexity of $O(|T|^2)$. Thus, finding the best CR among k CRs has a running complexity of $O(k|T|^2)$. The overall running complexity of the EBA is $O(|T|(|T| + k|T|^2))$, which is approximately $O(k|T|^3)$. $\square$

3.5 Neighbor-Based Assignment (NBA)

An NBA is listed in Algorithm 2. The algorithm starts with an empty CR assignment set $\mathcal{S}$, an empty assigned transactions set A , and an empty stack (Line 1). The NBA iterates over all of the transactions in T (Lines 2–31). During each iteration, a transaction t is selected, is removed from T , and is added to the stack (Lines 3–4). Subsequently, the NBA loops until the stack is empty (Lines 5–31). After a transaction t_i is popped from the stack (Line 6), the sets of $OutNeighbors$ and $InNeighbors$ are found (Lines 8–9). The NBA adds each transaction $t_n \in OutNeighbors$ into the stack (Lines 10–12). If $InNeighbors$ is empty, t_i is assigned to the smallest CR (Lines 13–15). Otherwise, the CRs of each assigned transaction $t_j \in InNeighbors$ are stored in $feasibleCR$ (Lines 18–22). If $feasibleCR$ is not empty, then t is assigned to the smallest CR in $feasibleCR$ (Lines 23–25). Otherwise, t is assigned to the smallest CR in $\mathcal{S}$ (Lines 26–28). The assigned transaction t_i is added to A (Line 29).

LEMMA 3.2. *The complexity of the NBA is $O(|T|^2)$, where $|T|$ is the number of transactions.*

PROOF. The loop in Lines (2–31) iterates over all of the transactions in T . For every transaction $t \in T$, the NBA iterates over the $InNeighbors$ transactions by adding them to the stack (Lines 5–30). Thus, the combined running complexity of the inner and outer loop (Lines 2–31) is $O(|T|)$. During each iteration, the set of $InNeighbors$ is found (Line 4), which has a running complexity of $O(|T|)$. Assigning and evaluating the objective function in Lines 24 and 27 have constant running complexity. Thus, finding the best CR among k CRs has a running complexity of $O(k)$. The overall running complexity of NBA is $O(|T|(|T| + k))$, which is approximately $O(|T|^2)$. $\square$

ALGORITHM 1: Edge-Based Assignment

Input: TEG, k
Output: $S = \{CR_1, \dots, CR_k\}$

```

1. Let  $S = \emptyset, A = \emptyset$ 
2. for  $t \in T$  do
3.   Let  $assigned = false$ 
4.   Let  $InNeighbors = \{t_j \in T | e_{ji} \in E\}$ 
5.   if  $InNeighbors$  is empty then
6.     Assign  $t$  to the smallest CR
7.      $assigned = true$ 
8.   end
9.   else
10.    Let  $feasibleCR = \{\}$ 
11.    for  $t_j \in InNeighbors$  do
12.      if  $t_j \in A$  then
13.        Add the CR of  $t_j$  to  $feasibleCR$ 
14.      end
15.    end
16.    if  $feasibleCR$  is not empty then
17.      Assign  $t$  to  $CR_{best} \in feasibleCR$  that minimizes the objective function and satisfies the
        constraints
18.      Add  $t$  to  $A$  and set  $Assigned = true$ 
19.    end
20.    else
21.      Assign  $t$  to  $CR_{best} \in S$ , which minimizes the objective function and satisfies the
        constraints
22.      Add  $t$  to  $A$  and set  $Assigned = true$ 
23.    end
24.    if  $assigned == false$  then
25.      Assign  $t$  to the smallest  $CR \in S$ 
26.    end
27.  end
28. end
29. return  $S$ 

```

4 A THREAT MANAGEMENT SYSTEM FOR DIOTCS

In this section, we present the architecture of a TMS and its integration with a DIoTCS. Figure 4 depicts such integration. Note that the TMS is composed of five components: the CRPP solver, CR table, admission controller, response subsystem, and recovery subsystem. In addition, the TMS is integrated with an Intrusion Detection System (IDS) that analyzes the transactions and alerts the TMS when a committed malicious transaction is detected [28]. We assume that the IDS has a detection delay of Δms after the malicious transaction is committed. The CRPP solver creates the TTEG model and generates the CR assignment using one of the heuristics proposed in the previous section. The CR assignment is stored in the CR table, which is used by the admission controller to regulate the admission of the transactions to the shared database. The functionalities of the response subsystem include collecting the timing information about the malicious transactions and populating the initial set of damaged objects. The status of a damaged object is stored in a special table termed the Damaged Objects Table (DOT). A transaction that attempts to read an

object from the DOT is blocked. The recovery subsystem recovers the damages by executing the compensating transactions. The recovered data objects are immediately removed from the DOT in order to increase the availability of the DIoTCS. The TMS maintains a transactions log that records the transaction ID, object ID, before image (the previous value), after image (the current value), and timestamp for each read-and-update operation. The transactions log is used to provide the correct versions for the damaged data objects during the recovery procedure. In the following sections, we present detailed information about the functionality of each component. The procedure of each subsystem is listed in Algorithm 3.

ALGORITHM 2: Neighbor-Based Assignment

Input: TEG, k
Output: $\mathcal{S} = \{CR_1, \dots, CR_k\}$

1. Let $\mathcal{S} = \emptyset, A = \{\}, \text{Stack} \leftarrow \{\}$
2. **while** T is not empty **do**
3. $t \leftarrow$ pick a transaction t from T
4. Add t to Stack and remove from T
5. **while** Stack is not empty **do**
6. $t_i \leftarrow$ pop from Stack
7. $CR_{target} = -1$
8. Let $OutNeighbors = \{t_j \in T | \exists e_{ij} \in E\}$
9. Let $InNeighbors = \{t_j \in T | \exists e_{ji} \in E\}$
10. **for** $t_n \in OutNeighbors$ **do**
11. Add t_n to Stack and remove from T
12. **end**
13. **if** $InNeighbors$ is empty **then**
14. Assign t_i to the smallest CR
15. **end**
16. **else**
17. Let $feasibleCR = \{\}$
18. **for** $t_j \in InNeighbors$ **do**
19. **if** $t_j \in A$ **then**
20. Add the CR of t_j to $feasibleCR$
21. **end**
22. **end**
23. **if** $FeasibleCR$ is not empty **then**
24. Assign t_i to smallest $CR \in feasibleCR$
25. **end**
26. **else**
27. Assign t_i to smallest CR
28. **end**
29. Add t_i to A
30. **end**
31. **end**
32. **end**
33. **return** $\mathcal{S}$

4.1 Admission Controller

The main functionality of the admission controller is to regulate the execution of the transactions. In particular, the admission controller checks whether the transaction is requesting to access either

Fig. 4. TMS architecture and monitoring of DloTCS transactions.

boundary or damaged objects. Access to boundary objects is controlled according to a *delayed-access* mechanism, in which the boundary objects are locked for sufficient time to ensure that the updating transactions are not malicious. We set the delayed-access mechanism timer to be twice the IDS detection delay, i.e., 2Δ . If the IDS identifies a transaction that updates a boundary object before the timer expires, then the boundary object is added to the DOT. Transactions that request to access damaged objects in the DOT are blocked until the objects are recovered and released from the DOT by the recovery subsystem, as discussed below.

4.2 Response Subsystem

The objective of the response subsystem is to prevent benign transactions from reading damaged objects that are updated by a malicious transaction t_m . In particular, the IDS triggers the response subsystem when a t_m is detected as malicious. Subsequently, the response subsystem collects various time information about the t_m , including its commit timestamp t_m^c and its detection timestamp t_m^d , that are provided by the IDS. Then, the response subsystem extracts the CR(s) of the t_m (denoted by the CR_{t_m}) from the CR table. All objects in the CR_{t_m} that have been updated during the period t_m^c and t_m^d are added to the DOT.

4.3 Recovery Subsystem

The objectives of the recovery subsystem are to (1) identify the *correct* and *complete* set of affected transactions and (2) execute the compensating transactions accordingly. The set of affected transactions is correct and complete if and only if the set contains no transactions that are falsely identified as affected and contains every affected transaction caused by the attack. In order to identify the set of affected transactions, the recovery subsystem temporarily blocks new transactions to prevent them from reading any undiscovered damaged objects. We implement the blocking operation using a locking algorithm. When the lock is acquired by the recovery subsystem, the admission controller blocks all incoming transactions until the recovery subsystem releases the lock. During the time in which the lock is acquired by the recovery subsystem, the correct and complete set of affected transactions is identified. Subsequently, undamaged objects are released from the DOT.

The recovery procedure is performed in two phases. First, the recovery subsystem executes compensating transactions to undo the effect of the malicious and affected transactions. The recovery subsystem uses the transactions log table to find the correct version of the damaged objects. The damaged objects are updated with the values of the most recent versions before the execution of the malicious transaction. Second, the recovery subsystem executes compensating transactions to

ALGORITHM 3: Procedures for TMS Components in Figure 4

```

1. Function Admission Controller ( $t_i$ )
2.   /* Invoked when a transaction  $t_i$  is executed */
3.    $RW_{t_i} \leftarrow$  the read/write objects set  $t_i$ 
4.   while  $RW_{t_i} \cap BT$  or  $RW_{t_i} \cap DOT$  do
5.     | Block  $t_i$ 
6.   end
7.   Execute  $t_i$ 
8. Function Response Subsystem ( $t_m, t_m^d$ )
9.   /* Invoked when IDS detects a transaction  $t_m$  as malicious */
10.   $t_m^c \leftarrow$  get commit time of  $t_m$ 
11.  Find  $CR_{t_m}$  the set of CRs spanned by  $t_m$ 
12.  for objects updated between  $t_m^c$  and  $t_m^d$  do
13.    | if object is assigned to an  $CR \in CR_{t_m}$  then
14.      |   Add object to  $DOT$ 
15.    | end
16.  end
17. Function Recovery Subsystem ( $t_m, CR_{t_m}$ )
18.  /* Invoked when the response subsystem is terminated */
19.  Block new transactions in  $CR_{t_m}$ 
20.  Find the complete and correct set of affected transactions
21.  Resume new transaction in  $CR_{t_m}$ 
22.  Remove undamaged objects from  $DT$ 
23.  Undo  $t_m$  and affected transactions
24.  Redo affected transactions

```

re-execute each affected transaction. The information required to re-execute the transactions is maintained in the transactions log. At the end of the recovery procedure, the recovered data objects are removed from the DOT. Also, the admission controller is signaled to resume the admission of the blocked transactions.

5 PERFORMANCE EVALUATION

In this section, we present the performance evaluation of the TMS. First, we discuss the synthetic workload used for such evaluation. Second, we evaluate the performance of the proposed CR demarcation heuristics. Finally, we conduct an extensive system evaluation of the TMS using a synthetic workload and present the performance results.

5.1 Synthetic Transactional Workload

Although several benchmarks have been developed to evaluate the performance of DBMSs, e.g., TPC-C,¹ no benchmark has been developed to evaluate the performance of intrusion management systems in DBMSs. In order to evaluate the performance of a TMS, we have developed a workflow-based transactional workload benchmark that generates workflows consisting of dependent transactions. The dependency among the transactions in the workflow amplifies the potential damage that can be caused by the malicious transactions. In essence, executing dependent transactions shortly after malicious transactions can induce the spread of damage. Thus, the proposed

¹<http://www.tpc.org/tpcc/spec/tpcccurrent.pdf>.

benchmark allows assessment of the capability of intrusion management systems in managing and recovering from damage without degrading the performance of the DIoTCS.

The proposed benchmark simulates a collaborative manufacturing application, as discussed in Section 1. Each transaction resembles a service performed either by an IoT device or using a WfMS. The benchmark consists of a single table and has three forms of transactions: *fan-out*, *fan-in*, and *mesh*. A fan-out transaction reads data from one object and updates N objects; a fan-in transaction reads M data objects and updates a single object; a mesh transaction reads data from N objects and updates M objects. The benchmark is implemented on an OLTP-benchmark testbed [14].

We now explain the process of generating a workflow-based workload of m transactions. We use three parameters to characterize the workload: (1) the number of workflows w , (2) the maximum number of transactions in each workflow w_{max} , and (3) the maximum transaction size $Size_{max}$. First, each workflow is modeled as an ℓ -ary tree, where each node represents a transaction. We choose the value of ℓ from a uniform distribution with a range of $[1, \frac{w_{max}}{w}]$. Then, the size of each transaction in the workflow is chosen using a uniform distribution with a range of $[2, Size_{max}]$. The form of the transaction is randomly selected from fan-out, fan-in, and mesh.

For each pair of dependent transactions, two additional variables are used, i.e., SO and δ . SO is chosen from a uniform distribution with a range of $[1, 10]$. We model δ as an exponential random variable with parameter λ . The values used for λ are 0.001, 0.0006, and 0.0005. Transactions are generated based on these four parameters. For the experiments, we generate workloads with 100 workflows and 5,000 transactions.

5.2 Evaluation of the CR Demarcation Heuristics

In this section, we evaluate the performance of the proposed heuristics using a synthetic transactional workload. We compare the performance of the proposed heuristics with an approximation algorithm to solve the (k, v) balanced graph partitioning problem [4]. The algorithm, known as Kernighan-Lin and denoted by KL, has a runtime complexity of $O(|T|^3 \log |T|)$. Two metrics are used to evaluate the performance of the heuristics: the number of boundary objects and an assignment fairness index. The fairness index is used to assess the performance in terms of size balancing among CRs. For this purpose, we use Jain's fairness index [19].

$$\mathcal{J}(CR_1, CR_2, \dots, CR_k) = \frac{(\sum_{i=1}^k |CR_i|)^2}{k \cdot \sum_{i=1}^k |CR_i|^2}, \quad (8)$$

where $|CR_i|$ is the number of transactions assigned to the i^{th} CR.

In the first experiment, we vary the number of CRs while keeping the value of λ to be 0.001. The number of boundary objects increases as k increases, as illustrated in Figure 5(a). This is owing to the fact that as k increases, the number of contained transactions in each CR is reduced. As a result, the number of crossing edges among CRs increases and, thus, increases the number of boundary objects. As expected, owing to high complexity, the performance of KL is better than that of the EBA and NBA and produces a lower number of boundary objects. Note that the EBA produces less boundary objects as compared with the NBA. This is because the EBA performs a greedy decision to reduce the number of boundary objects when assigning a transaction to a CR. In terms of the fairness index, KL outperforms the NBA and EBA while the NBA outperforms the EBA in terms of the fairness index, as illustrated in Figure 5(b). This is because the NBA is designed to produce balanced assignments. The fairness index reduces as the value of k increases.

In the next experiment, we study the effect of varying the value of λ with $k = 10$. The results are illustrated in Figure 6. The number of boundary objects remains constant as the value of λ changes in the case of NBA. This is owing to the fact that, for the same value of k , the value of

(a) Number of boundary objects (b) Fairness index

Fig. 5. Effect of k on a workload of 5,000 transactions with $\lambda = 0.001$.

(a) Number of boundary objects (b) Fairness index

Fig. 6. Effect of λ on a workload of 5,000 transactions with $k = 10$.

λ has no effect on the number of boundary objects. However, in the case of the EBA, the number of boundary objects increases as the value of λ decreases. The reason is that, as the value of λ decreases, the average value of δ increases (recall that $\delta \sim \text{Exp}(\lambda)$). In this case, the value of f_2 is maximized by forcing the edges to cross CRs and, thus, increases the number of boundary objects.

5.3 TMS Performance Evaluation

The experiments are conducted on a Dell Power Edge R420 server with a 6-core Intel E5-2620v3 CPU, 120 GB of RAM, and Ubuntu Server 16.0 LTS OS. We use PostgreSQL 9.5 as the host DBMS. The TMS is implemented using Java and connected to PostgreSQL using JDBC. We use 20 workers (threads) for the admission controller and an unlimited pool of threads for the response and recovery procedures. We assume that the transactions are submitted according to a Poisson distribution with an arrival rate of 10. Malicious transactions are injected into the workload in a uniform distribution over time. The number of malicious transactions is based on the attack intensity π , which is a ratio to the total number of transactions.

We evaluate the performance of the TMS using five metrics: (1) the number of affected transactions, (2) the number of blocked transactions, (3) the average recovery time, (4) the average response time, and (5) the number of temporal constraints violations. The number of blocked transactions indicates the performance of the TMS in terms of availability. The number of affected transactions is used to evaluate the cost of damage. The average recovery time is computed based on the time taken to recover the affected transactions, while the average response time is computed based on the response time of all committed transactions. The number of temporal constraints violations counts the number of transactions that failed to meet the execution time constraints. For the rest of the experiments, we set $\lambda = 0.001$ and $\Delta = 100$ ms, unless stated otherwise. We compare the performance of the TMS using the proposed heuristic, i.e., EBA, NBA, and the approximation algorithm, i.e., KL, with the approach, termed the Intrusion-Tolerant Database (ITDB), given in [22]. We refer to the TMS with $k = 1$ as “OneCR”. Note that, for the case of OneCR and the ITDB, there are no boundary objects; thus, the delayed access mechanism is not active. However, the difference between the ITDB and OneCR is that the ITDB does not block new transactions while identifying the set of affected transactions, which results in performing multiple scans over the transactions log. In each experiment, we run the TMS with various values of k , i.e., 10, 20, 30, and 40. We did not notice any further improvement in the performance beyond 40 CRs.

5.3.1 Effect of Attack Intensity. In this experiment, we compare the performance of the TMS with the ITDB as the value of the attack intensity π increases. We use $k = 10$ for the TMS using EBA, NBA, and KL assignments. The results are depicted in Figure 7. We observe that the number of affected transactions increases as the number of malicious transactions increases (Figure 7(a)). The reason is that, for the same intertransaction dependency, increasing the malicious transactions generates more affected transactions. Note that the number of affected transactions and average

Fig. 7. Effect of π with workload of 5,000 transactions, $k=10$, $\delta=100ms$, and $\lambda=0.001$.

Fig. 8. Effect of Δ with a workload of 5,000 transactions; $\pi=10\%$ and $\lambda=0.001$.

recovery time in the TMS are reduced by at least 50% and 40%, respectively, as compared with the ITDB. The reason is that the TMS employs a delayed-access mechanism that takes a proactive approach to stop transactions from spreading the damage, which results in fewer affected transactions as compared to the ITDB. In addition, the TMS temporarily blocks new transactions while identifying the set of affected transactions. As a result, OneCR performs a single scan over the transactions log as compared with multiple scans performed by the ITDB. However, the downside of the delayed-access mechanism is the increase in the number of blocked transactions (by at most 83%) and the increase in average response time (by at most 60%), as illustrated in Figures 7(b) and 7(d), respectively. We observe that KL exhibits less response time overhead as compared with EBA and NBA. This is because KL generates the least number of boundary objects.

Note that the average response time exhibits an increase as the attack intensity increases (Figure 7(d)). The reason is that as the value of π increases, the number of affected transactions and the number of blocked transactions increase. Thus, the blocking time of an incoming transaction, as discussed in Section 4.1, is prolonged.

5.3.2 Effect of IDS Detection Delay. Figure 8 illustrates that increasing the intrusion detection delay from $\Delta=100ms$ to $\Delta=1,000ms$ degrades the performance of the TMS with $k>1$ as compared with OneCR and the ITDB. The reason is that as Δ increases, the time required to release (unlock) a boundary object increases. Recall that the delayed-access mechanism time is 2Δ , as discussed in Section 4.1. As a result, the number of arriving transactions that request to read blocked boundary objects increases. Note that, for the same Poisson arrival rate of 10, the number of affected transactions, number of blocked transactions, and average recovery time do not change as Δ increases. However, the average response time increases as Δ increases owing to the delayed-access mechanism. In conclusion, OneCR is more suitable for systems with poor IDS performance, i.e., an IDS with a long detection delay. However, for systems with good IDS performance (i.e., $\Delta<200ms$), using a TMS can achieve better performance in terms of damage containment (in terms of affected transactions) with a low response time overhead.

5.3.3 Effect of Number of CRs. In this experiment, we evaluate the performance of a TMS with various values of k , i.e., 10, 20, 30, and 40. We compare the results of an EBA, NBA, and KL with

Fig. 9. Effect of k with workload of 5,000 transactions; $\pi = 10\%$, $\Delta = 100\text{ms}$, and $\lambda = 0.001$.

Fig. 10. Effect of k , π , Δ , and λ on the number of temporal constraints violations.

OneCR. The experiment is conducted using $\Delta = 100\text{ms}$, $\lambda = 0.001$, and $\pi = 10\%$. As demonstrated in Figure 9, increasing k improves the performance of the TMS. The number of affected transactions is reduced by at least 50%. Consequently, the average recovery time decreases as k increases. However, the TMS incurs an increase of at least 75% in response time overhead, as illustrated in Figure 9(d). This is due to the increase in the number of boundary objects. We observe that the increase in the response time overhead is marginal as the value of k increases beyond 10.

5.4 Number of Violations of Temporal Constraints

In this experiment, we study the performance of a TMS with respect to the number of violations of the temporal constraints as specified by the DiIoTCS application while varying the values of various parameters, including k , π , Δ , and λ . Figure 10 highlights the interplay between two underlying processes: the application execution process and the response and recovery process in the wake of attacks. The interplay illustrates the trade-off between damage containment and the satisfaction of temporal constraints. We observe that the number of transactions that violates the time constraints increases as the values of k , π , and Δ increases. The reason is that, as the values of k and Δ increase, the access delay mechanism blocks transactions temporarily in favor of damage containment. On the other hand, increasing the value of π increases the number of affected transactions and, thus, increases the recovery time. Subsequently, this increases the blocking time of incoming dependent transactions beyond their execution time constraints.

From the trade-off point of view, we observe that the partitioning-based TMS reduces the number of affected transactions by 50% with $k = 10$ (Figure 9(a)). On the other hand, an increase in the response time results in a 20% increase in the number of transactions violating their execution time constraints. Nevertheless, increasing the average execution time constraints among transactions reduces the number of violating transactions as illustrated in Figure 10(d). In essence, we observe that in order to achieve high “threat-resilient” DiIoTCS applications, their temporal constraints must be consonant with the IDS detection delay and the temporal dynamics associated with the threats embedded in the transaction workload. In particular, if the threat intensity is high, transactions with shorter temporal constraints will incur a high number of violations.

6 RELATED WORK

The IoT is becoming a mainstream infrastructure for several applications, including smart grids, intelligent transportation, health care management, and manufacturing [1, 8, 33]. Several cloud-based solutions have been proposed to support large-scale storage and processing of data in IoT applications. For example, the authors of [35] propose an architecture for IoT applications that is based on information-centric networking to better accommodate data-centric services. In [16], the authors present IoT- and cloud-based platforms for intelligent transportation services. In [31], the authors propose an architecture for cloud computing and an IoT-based cloud manufacturing service system. The proposed system aims at realizing the transformation from production-oriented manufacturing to service-oriented manufacturing.

Security risks in IoT include attacking data confidentiality, integrity, and availability. A broad span of research has addressed the preservation of data confidentiality, including work on identity authentication [18], access control [2], authorization [13], and encryption-based techniques [20]. The protection of data integrity has been addressed by providing secure communication [17] and trust management protocols [24]. The availability of DIoTCSs is protected by providing intrusion detection techniques against attacks [23]. Our work aligns with protection of data integrity and availability against malicious intrusion attacks on DIoTCSs. However, no previous work has addressed the problem of post-intrusion management in the IoT.

Several solutions have been proposed for intrusion recovery in database-backed applications. A generic intrusion-tolerant architecture for web servers that uses redundancy and diversification principles is proposed in [27]. In [11], WARP is proposed to recover from intrusions in web applications by rolling back the database and replaying subsequent legitimate actions to correct the state of the DBMS. In [25], an intrusion recovery tool for database-backed applications running in Platform-as-a-Service clouds is proposed. In [3], a suite of recovery algorithms are proposed to unwind the effect of malicious transactions for offline and online recovery. Our work is designed as a middle-layer between the data management layer and the application layer, which makes it suitable for DIoTCSs.

7 CONCLUSION

DIoTCSs are vulnerable to a wide number of threats that can degrade their performance. In this article, we propose a partitioning-based threat management system for DIoTCSs that can endure intense attacks with minimum effect on availability and response time. We present a new model that captures the dependencies and execution time constraints among the transactions in DIoTCSs. We present a data-driven partitioning scheme, termed *containment regions*, with the objective to limit the extent of the damage with low response time overhead. We formulate the CR partitioning problem as a cost-driven optimization problem, prove its hardness, and propose two heuristic solutions. We present the architecture of a TMS and conduct various experiments to evaluate its performance. We show that, under intense malicious attack, a TMS incurs response time overhead of at most 83%. However, the reduction in the number of affected transactions and recovery time is up to 50% and 40%, respectively.

ACKNOWLEDGMENTS

The authors wish to acknowledge facility support provided by KFUPM.

REFERENCES

- [1] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. 2015. Internet of Things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys and Tutorials* 17, 4, 2347–2376.

- [2] Abdulrahman Almutairi, Muhammad Sarfraz, Saleh Basalamah, Walid Aref, and Arif Ghafoor. 2012. A distributed access control architecture for cloud computing. *IEEE Software* 29, 2, 36–44.
- [3] Paul Ammann, Sushil Jajodia, and Peng Liu. 2002. Recovery from malicious transactions. *IEEE Transactions on Knowledge and Data Engineering* 14, 5, 1167–1185.
- [4] Konstantin Andreev and Harald Racke. 2006. Balanced graph partitioning. *Theory of Computing Systems* 39, 6, 929–939.
- [5] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. 2017. Understanding the Mirai Botnet. In *USENIX Security Symposium (USENIX'17)*. 1092–1110.
- [6] Vaibhav Arora, Faisal Nawab, Divyakant Agrawal, and Amr El Abbadi. 2017. Multi-representation based data processing architecture for IoT applications. In *IEEE 37th International Conference on Distributed Computing Systems (ICDCS'17)*. IEEE, 2234–2239.
- [7] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. 1986. Concurrency control and recovery in database systems. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- [8] Zhuming Bi, Li Da Xu, and Chengen Wang. 2014. Internet of Things for enterprise systems of modern manufacturing. *IEEE Transactions on Industrial Informatics* 10, 2, 1537–1546.
- [9] Kristy Browder and M. Davidson. 2002. The virtual private database in Oracle9iR2. *Oracle Technical White Paper, Oracle Corporation* 500.
- [10] Hongming Cai, Boyi Xu, Lihong Jiang, and Athanasios V. Vasilakos. 2017. IoT-based big data storage systems in cloud computing: Perspectives and challenges. *IEEE Internet of Things Journal* 4, 1, 75–87.
- [11] Ramesh Chandra, Taesoo Kim, Meelap Shah, Neha Narula, and Nickolai Zeldovich. 2011. Intrusion recovery for database-backed web applications. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*. ACM, 101–114.
- [12] Chii Chang, Satish Narayana Srirama, and Rajkumar Buyya. 2017. Mobile cloud business process management system for the Internet of Things: A survey. *ACM Computing Surveys* 49, 4, 70.
- [13] Simone Cirani, Marco Picone, Pietro Gonizzi, Luca Veltri, and Gianluigi Ferrari. 2015. IoT-OAS: An OAuth-based authorization service architecture for secure services in IoT scenarios. *IEEE Sensors Journal* 15, 2, 1224–1234.
- [14] Djellal Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudre-Mauroux. 2013. OLTP-Bench: An extensible testbed for benchmarking relational databases. *Proceedings of the VLDB Endowment* 7, 4, 277–288.
- [15] Jianhua Fan, Xianglin Wei, Tongxiang Wang, Tian Lan, and Suresh Subramaniam. 2017. Deadline-aware task scheduling in a tiered IoT infrastructure. In *IEEE Global Communications Conference (GLOBECOM'17)*. IEEE, 1–7.
- [16] Wu He, Gongjun Yan, and Li Da Xu. 2014. Developing vehicular data cloud services in the IoT environment. *IEEE Transactions on Industrial Informatics* 10, 2, 1587–1595.
- [17] Christine Hennebert and Jessye Dos Santos. 2014. Security protocols and privacy issues into 6LoWPAN stack: A synthesis. *IEEE Internet of Things Journal* 1, 5, 384–398.
- [18] Pengfei Hu, Huansheng Ning, Tie Qiu, Houbing Song, Yanna Wang, and Xuanxia Yao. 2017. Security and privacy preservation scheme of face identification and resolution framework using fog computing in Internet of Things. *IEEE Internet of Things Journal* 4, 5 (2017), 1143–1155.
- [19] Raj Jain, Arjan Duresi, and Gojko Babic. 1999. *Throughput Fairness Index: An Explanation*. Technical Report. Department of CIS, The Ohio State University.
- [20] Heshan Kumarage, Ibrahim Khalil, Abdulatif Alabdulatif, Zahir Tari, and Xun Yi. 2016. Secure data analytics for cloud-integrated Internet of Things applications. *IEEE Cloud Computing* 3, 2, 46–56.
- [21] Jie Lin, Wei Yu, Nan Zhang, Xinyu Yang, Hanlin Zhang, and Wei Zhao. 2017. A survey on Internet of Things: Architecture, enabling technologies, security and privacy, and applications. *IEEE Internet of Things Journal* 4, 5 (2017), 1125–1142.
- [22] Peng Liu, Jiwei Jing, Pramote Luenam, Ying Wang, Lunquan Li, and Supawadee Ingsriswang. 2004. The design and implementation of a self-healing database system. *Journal of Intelligent Information Systems* 23, 3, 247–269.
- [23] Aleksandar Milenkoski, Marco Vieira, Samuel Kounev, Alberto Avritzer, and Bryan D. Payne. 2015. Evaluating computer intrusion detection systems: A survey of common practices. *ACM Computing Surveys* 48, 1 (2015), 12.
- [24] Michele Nitti, Roberto Girau, and Luigi Atzori. 2014. Trustworthiness management in the social Internet of Things. *IEEE Transactions on Knowledge and Data Engineering* 26, 5, 1253–1266.
- [25] David R. Matos, Miguel L. Pardo, and Miguel Correia. 2017. Rectify: Black-box intrusion recovery in PaaS clouds. In *Proceedings of the 18th International Middleware Conference*. ACM.
- [26] Art Rask, Don Rubin, and Bill Neumann. 2005. Implementing row- and cell-level security in classified databases using SQL Server 2005. *MS SQL Server Technical Center*. Technical Report.
- [27] Ayda Saidane, Vincent Nicomette, and Yves Deswarte. 2009. The design of a generic intrusion-tolerant architecture for web servers. *IEEE Transactions on Dependable and Secure Computing* 6, 1, 45–58.

- [28] Asmaa Sallam, Elisa Bertino, Syed Rafiul Hussain, David Landers, R. Michael Lefler, and Donald Steiner. 2017. DB-SAFE: An anomaly detection system to protect databases from exfiltration attempts. *IEEE Systems Journal* 11, 2, 483–493.
- [29] Satish M. Srinivasan and Raghvinder S. Sangwan. 2017. Web app security: A comparison and categorization of testing frameworks. *IEEE Software* 34, 1, 99–102.
- [30] Qie Sun, Hailong Li, Zhanyu Ma, Chao Wang, Javier Campillo, Qi Zhang, Fredrik Wallin, and Jun Guo. 2016. A comprehensive review of smart energy meters in intelligent energy networks. *IEEE Internet of Things Journal* 3, 4 (2016), 464–479.
- [31] Fei Tao, Ying Cheng, Li Da Xu, Lin Zhang, and Bo Hu Li. 2014. CCIoT-CMfg: Cloud computing and Internet of Things-based cloud manufacturing service system. *IEEE Transactions on Industrial Informatics* 10, 2, 1435–1442.
- [32] Martin Wollschlaeger, Thilo Sauter, and Juergen Jasperneite. 2017. The future of industrial communication: Automation networks in the era of the Internet of Things and Industry 4.0. *IEEE Industrial Electronics Magazine* 11, 1, 17–27.
- [33] Boyi Xu, Li Da Xu, Hongming Cai, Cheng Xie, Jingyuan Hu, and Fenglin Bu. 2014. Ubiquitous data accessing method in IoT-based information system for emergency medical services. *IEEE Transactions on Industrial Informatics* 10, 2, 1578–1586.
- [34] Xun Xu. 2012. From cloud computing to cloud manufacturing. *Robotics and Computer-integrated Manufacturing* 28, 1, 75–86.
- [35] Hao Yue, Linke Guo, Ruidong Li, Hitoshi Asaeda, and Yuguang Fang. 2014. Dataclouds: Enabling community-based data-centric services over the Internet of Things. *IEEE Internet of Things Journal* 1, 5, 472–482.
- [36] Nan Zhang, Soteris Demetriou, Xianghang Mi, Wenrui Diao, Kan Yuan, Peiyuan Zong, Feng Qian, XiaoFeng Wang, Kai Chen, Yuan Tian, et al. 2017. Understanding IoT security through the data crystal ball: Where we are now and where we are going to be. *arXiv preprint arXiv:1703.09809*.

Received January 2018; revised December 2018; accepted February 2019