The Journal of Supercomputing (2022) 78:14813–14845
https://doi.org/10.1007/s11227-022-04481-9

An integrated approach of designing functionality with security for distributed cyber-physical systems

Dipty Tripathi[1] · Amit Biswas[1] · Anil Kumar Tripathi[1] · Lalit Kumar Singh[1] · Amrita Chaturvedi[1]

Accepted: 23 March 2022 / Published online: 9 April 2022

Abstract

In this work, we propose a multi-tier architectural model to separate functionality and security concerns for distributed cyber-physical systems. On the line of distributed computing, such systems require the identification of leaders for distribution of work, aggregation of results, etc. Further, we propose a fault-tolerant leader election algorithm that can independently elect the functionality and security leaders. The proposed election algorithm identifies a list of potential leader capable nodes to reduce the leader election overhead. It keeps identifying the highest potential node as the leader, whenever needed, including the situation when one has failed. We also explain the proposed architecture and its management method through a case study. Further, we perform several experiments to evaluate the system performance. The experimental results show that the proposed architectural model improves the system performance in terms of latency, average response time, and the number of real-time tasks completed within the deadline.

Keywords Distributed cyber-physical systems · Functional requirements · Security requirements · Aspect-orientation · Leader election · Fault-tolerance

✉ Amit Biswas
amitbiswas.rs.cse17@iitbhu.ac.in

Dipty Tripathi
diptytripathi.rs.cse17@iitbhu.ac.in

Anil Kumar Tripathi
aktripathi.cse@iitbhu.ac.in

Lalit Kumar Singh
lalit.rs.cse@iitbhu.ac.in

Amrita Chaturvedi
amrita.cse@iitbhu.ac.in

[1] Department of Computer Science and Engineering, Indian Institute of Technology (BHU), Varanasi, India

1 Introduction

These days cyber-physical systems (CPSs) are mostly network-ready extensions of traditional embedded systems which are connected to the Internet. The cyber nodes monitor and control the real-world physical devices and infrastructures to achieve better reliability and resource utilization [8, 26, 48]. The emerging CPS may range from small-scale industries to large-scale connected systems of diverse areas such as transportation, aerospace, entertainment, industrial control system, health care, and so on. Thus, the components of a CPS are geographically distributed, which consist of numerous sensing, actuation, and computational nodes. These nodes execute different software modules to perform multiple real-time and non-real-time jobs to achieve a common goal. However, these components may fail independently. This failure can be unintentional or intentional. The integration of cyber components enriches the intelligence and enhances the quality of services provided by physical infrastructure and devices. However, the automation and connectivity of all the networked computing devices increase the security risks [35, 47, 48, 51] by increasing intentional failure. In recent years, several powerful attacks have been launched on critical infrastructures. For instance, Stuxnet worm [1], Havex [40], and Triton [15] malware are deployed to target different CPSs. These attacks caused huge financial damage and physical injuries. The attackers try to dig out the existing security vulnerabilities present at any of the CPS architecture layers or any of its nodes. They may arise due to ignorance at the design time or the facilities provided to the external entities [49]. These vulnerabilities may be of different nature, such as open port, buffer overflow, weak password, access control, and adoption of standardized protocols and technologies with known weakness. The attacker may launch an active or passive attack to exploit the vulnerabilities and compromise the system's critical functionalities, availability, integrity, or confidentiality.

Hence, an integrated CPS architectural model is needed to organize security with functionalities to perform real-time jobs securely [29, 55]. Moreover, the occurrence of faulty nodes due to usual faults or security breaches demands the secure architecture to be fault-tolerant as well. Unfortunately, in earlier system design, security consideration takes a back-seat [9, 44] including existing CPS design and architectures. The authors in [20, 21, 26] presented reference architectures for designing CPSs in centralized fashion. In these architectures, the primary focus was on the necessary components, their responsibilities, and interactions. However, there is neither any discussion on the management and organization of cyber layer nor on CPS security. To reduce latency, monitor network traffic and reduce system management complexity, the authors [31] and [3] presented the centralized architectures of CPS. But, these architectures are designed without security arrangements. To facilitate secure data communication, the authors [50] presented software-defined networking (SDN)-based centarlized architecture of Internet of Things (IoT). However, the centralized architecture suffers from single point of failure. The authors [33] presented SDN-based data transfer security model middle box-guard (M-G) to manage the data flow with defined security policies. The authors [19] improved [50] by presenting distributed architecture as

Black SDN-IoT for smart city. The architecture integrates the network function virtualization (NFV) for monitoring the traffic data flow. However, the main focus in [3, 19, 31, 33, 50] was on the network layer and traffic security. To detect the intrusion into the system, the authors [16, 25, 32, 38, 56] presented the intrusion detection system-based CPS security frameworks. Though these approaches focus on security monitoring, they fail to present an architectural model to integrate security with functionality. To deal with cross-layer CPS security, the authors[57] presented a hierarchical architecture. They applied game-theory to analyze system security.

Hence, it is clear that to deal with these challenges, a security aware fault-tolerant distributed architectural model is needed. As distributed system has the capability to reschedule the job of a failed or compromised among the other non-faulty nodes node. This makes the system more fault-tolerant by avoiding a single point of failure [34]. In a recent study, the authors [27] also advocated that some sort of distribution is required to manage the functionality and security at physical and cyber level of CPSs. They stated that in [26], "a 5 level architecture, namely 5C-CPS, has been proposed for developing CPSs. There are many challenges associated with data security, privacy, centralization, etc., which require further development and progress". However, the authors do not present any explicit in-depth methodology to integrate and organize the functionality and security in a distributed manner. Hence, we propose design of distributed architectural model of CPS to integrate and organize the functionality and security. To design a distributed secure CPS, two major responsibilities need to be distributed and organized including:

- distribution of functionality tasks such as monitoring, control, execution, information gathering, and processing with aggregation of results.
- distribution of security tasks such as monitoring, data collection, buffering of security threat events, and aggregation of results for its control.

In a distributed system, to handle the responsibilities of task distribution-aggregation, communication, and task redistribution in case of a node failure, there is a need of a coordinating node known as leader [5, 39]. Depending upon the enormity and complexity of event monitoring for security and functionality delivered in a CPS, a single leader or separate leaders may handle the functionality and security responsibilities. If the system is smaller with few or delay-tolerant tasks, both responsibilities may be given to the same leader node. By nature, CPSs are complex and large real-time systems like rail management or smart city. If security and functionality are handled by one & the same leader, the leader node may face a heavy load to coordinate all the functionality and security activities simultaneously. Consequently, the deadline of the functional tasks may be overlooked, or security events may be missed, which is considered a failure in hard real-time systems. Moreover, monitoring and events related to security are pretty different from functionality and may be needed to integrate and update the existing system for instance, in the railway management system. By looking at the exigency and grave consequences of security and the time criticality of security mechanisms, there is a need to have a logical and physical separation between functionality and security.

Therefore, the objective of this paper is to propose a multi-tier distributed architectural arrangement to organize the functionality with the security of a large-scale CPS. The idea is similar to aspect-orientation [24], as security is designed, implemented, and maintained separately. It can be integrated along with the cyber part in CPS to improve the modularity and maintainability of the system. For that, we are bringing in the concept of leader(s) and leader election in CPS for the first time and facilitating the logical and physical separation by electing separate functionality and security leaders. Moreover, as most of the tasks are safety-critical and real time, there should be a way to elect a new leader immediately after a leader node is failed to minimize the adverse effect on real-time task coordination, system performance, and security.

The existing leader election algorithms [4, 6, 7, 37, 39] are not suitable for a large CPS as the proposed work elects a general leader without considering the need of functionality and security requirements of time-constrained real-time systems. Hence, we propose a fresh fault-tolerant leader election algorithm to elect the functionality and security leaders for CPS. The significant contributions of this paper are abridged as follows:

- We propose a distributed multi-tier architectural model to integrate and organize security and functionality. The model consists of four layers: sensing & actuation layer, controller, cyber layer, and decision support layer (CND). The bottom two layers follow fixed distribution since each field device and field controller performs dedicated tasks to respond to the real-time functional and security requirements. On the other hand, the cyber and decision support layers are managed in a purely distributed fashion, where computing nodes are divided into clusters. The functionality and security requirements in each cluster are distributed as two core tasks among the functionality and security nodes. These nodes are managed by functionality and security leader nodes, respectively. The security monitoring and response at each layer are handled by the same or upper layers.
- Along with this, we propose a fault-tolerant leader election algorithm for electing functionality and security leaders. Instead of electing only a single leader, a list of leader capable nodes is elected based on a predefined election criterion. Moreover, the general leader election process is itself vulnerable to initiate unnecessary leader election process. The proposed algorithm can also deal with this scenario, where a malicious node tries to initiate the election process unnecessary to target an unbiased leader. It achieves consensus among leader-capable nodes to start the election process.
- We evaluate the proposed architectural model by performing several experiments. The experimental results show that the proposed architectural model improves CPS performance in terms of latency, average response time, and the number of real-time tasks completed within the deadline.

The rest of the paper is organized as follows. Section 1 deals with related work for existing CPS architectures and the security approaches for CPSs. Section 3 presents the attack scenario. Section 4 proposes the multi-tier architectural model and a preselected leader election algorithm for electing the functionality and security leaders.

Section 5 presents the performance evaluation of the proposed architectural model and a case study on a smart healthcare system. Section 6 concludes the paper with future work.

2 Related work

In [48, 49], the authors focused on early security modeling to assess the security risks on CPS functionalities. However, these works do not provide any architectural or algorithmic solution to these risks. As the concept of smart cities is being developed as a CPS, Jalali et al. presented a three-layer architecture [20] for a smart city. The architecture includes the sensory, network, and control & service layers with the discussion of supporting technology for each layer. To manage the generated data in smart cities, Gaur et al. proposed a semantic web technology-based multi-level architecture [14] for a smart city. The architecture consists of data to service transformation layers such as data collection, data processing, data integration and reasoning, device control & alerts. In [26], the authors presented a more detailed and classic 5*C* CPS architecture which consists of connection, conversion, cyber, cognition, and configuration layers to optimize CPS roles and functions for manufacturing industries. Next, JR Jiang extended the 5*C* architecture proposed in [26] and presented it as 8*C* architecture [21] by adding customer, coalition, and content for broader adoption in industries. However, the authors did not mention the management procedure of these architectures [14, 20, 21, 26]. To reduce latency, monitor network traffic and reduce system management complexity, in [31] and [3], Liu et al. and Balta et al. presented two centralized architectures of CPS. However, centralized architecture increases the risk of a single point of failure. To deal with these challenges, Garofalo et al. presented a concept of a decentralized real-time system [13]. They applied the decentralized system to control urban drainage networks equipped with multiple sensors and a series of actuators. Moreover, the authors presented a gossip-based algorithm for achieving performance and fault-tolerance properties. However, there is a lack of provisions in [3, 13, 14, 20, 21, 26, 31] to make the system secure. In [43], the authors proposed a hybrid smart city cyber security architecture to analyze the threats and associated risks. To deal with security concerns in widely adopted networked and web-accessible CPSs, Zhu et al. presented a hierarchical architecture [57] for dealing with cross-layer CPS security. They applied game theory to evaluate the effect of possible strategies of attackers and defenders on system security. However, this is not a unified architectural model integrating functionality and security. There is no provision of being fault-tolerant. Tao et al. presented a cloud-based multi-tier architectural model [46] to enable interactions among different heterogeneous devices for IoT-based smart homes. Moreover, ontological constructs integrate security and privacy in the interaction process. Although the cloud supports the distributed architecture, the presented architectural model is managed in a decentralized manner but not in a purely distributed manner at the cyber level, limiting the model's breach tolerance and fault-tolerance capabilities. To facilitate secure data communication, Vandana et al. presented SDN-based centralized architecture [50] for IoT to ensure secure data communication. SDN can detect

anomalies and ensure some primary inhibition of communication network attacks. The SDN-based paradigm, in essence, describes a centralized control architecture where applications (the S in SDN) possess the intelligence of the system and fulfill many roles such as computing, decision-making, and reconfiguration (of devices) while leveraging the global view provided by a (logically) centralized controller. However, centralized architecture suffers from a single point of failure. In [33], Liu et al. presented SDN-based data transfer security model 'middlebox-guard' to manage the data flow through SDN with defined security policies. They mainly focused on the selection of the appropriate location of middlebox deployment and presented the algorithmic solution for the same although it is neither a unified architectural model to organize functionality with security nor a fault-tolerant model. In [19], the authors improved [50] by presenting distributed architecture as Black SDN-IoT for smart city. The architecture integrates the NFV to apply device virtualization and monitor traffic flow. However, the main focus in SDN-based approaches [19, 33, 50] is on the network layer and traffic security only, where security is the sole responsibility of the SDN controller. In this scenario, if the security controller of the SDN controller fails, the system security gets compromised. There is no mechanism for selecting the appropriate security controller node immediately. Lawal et al. presented real-time detection and mitigation approach of distributed denial of service attack on SDN [25]. However, the approach does not fit for large CPS. Moreover, the work does not provide any architectural or design solution for separating the functional and security concerns for CPS. In [56], the authors proposed a distributed intrusion detection system applied in multiple layers, including home area network, neighborhood area network, and wide area network for smart grid. Feng et al. considered connected and automated vehicles (CAVs) as distributed CPS and proposed a design for intelligent transport systems using information graphs [12]. The proposed design points out the security requirements and uses edge computing to process the information locally. However, the author does not provide a methodology to integrate and analyze the security measures with functionality. In [27], Lee et al. suggested a distributed architecture to overcome the centralized industrial network, security, and trust issue of CPS. They suggested that the security distribution should be at sensor level and computing level to take advantage of distributed computing in handling the performance and privacy concerns. However, the authors do not present any explicit explanation or in-depth methodology to organize and coordinate the functionality and security. In [52–54], the authors proposed the methodologies for privacy protection and handling the trust issues in information retrieval services hosted on cloud. These works present different algorithms to construct ideal dummy queries to meet the privacy model. However, these approaches are not designed for cyber-physical systems' privacy and security. Next, Liu et al. proposed hierarchically distributed intrusion detection for anomaly detection in industrial CPS [32]. The framework applies anomaly monitoring methods at each layer of CPS, including perceptual layer, data transmission layer, and application control layer. Similarly, in [42], the authors present a security framework to defend against cyberattacks for IoT, where the intrusion detection system is applied for IoT sensors network and Bluetooth protocol. The IDS detects cyber-attacks based on extracted features of Bluetooth and sensor signals, which are further used by different machine learning

classifiers. However, these works [32, 42, 56] do not consider the scenario where the security nodes may also be failed or be compromised by sophisticated and coordinated attacks.

Therefore, to the best of our knowledge, no work has been done yet that presents a distributed architectural model to integrate and organize security with functionality in existing CPSs. Moreover, who will coordinate the activity among heterogeneous nodes in CPS? How to implement adaptive functionality and security arrangements in case functional or security nodes are compromised by sophisticated and coordinated attacks for CPS? These are still open challenges that are not dealt with by the community. Hence, in Sect. 4, we present a distributed architectural model to coordinate and integrate the functionality and security, avoid a single point of failure, and increase fault tolerance at reduced communication latency in a CPS by bringing in the concept of fault-tolerant security and functionality leaders.

3 Attack scenario

In general CPS architectures [14, 20, 21, 26], the security vulnerabilities may exist at any of the levels. As a result, security concerns are different at different layers. Different attacks like tempering, spoofing, or denial of service may be launched at any of the layers to compromise the integrity, confidentiality, and availability of a node by performing ARP spoofing, false data/command injection attacks, smurf attacks, social engineering, replay attacks, infecting the firmware, or sniffing. As a result, the nodes may fail, become non-responsive, or behave in a faulty manner. Moreover, the sensitive information may be exfiltrated and sent to illegitimate nodes. Specifically, the attack scenarios (AS) include

AS(1) attack on sensors or actuators
AS(2) attack on field controllers or
AS(3) attack on computing nodes that perform specified functionality
AS(4) initiation of unnecessary leader election process

4 The proposed architectural model

The section presents the formal description of the proposed architectural model. Different layers of the proposed architecture and their responsibilities are also explained here. Then, the need and role of functional and security leaders and the proposed leader election algorithm are discussed in detail.

4.1 Formal description

The proposed architectural model consists of four layers where security is added as a cross-cutting concern as shown in Fig. 1. This architectural model is designed and viewed as a distributed system with heterogeneous nodes as presented in Fig. 2. Formally, the proposed CPS architecture is defined as a set of nodes (*SN*)

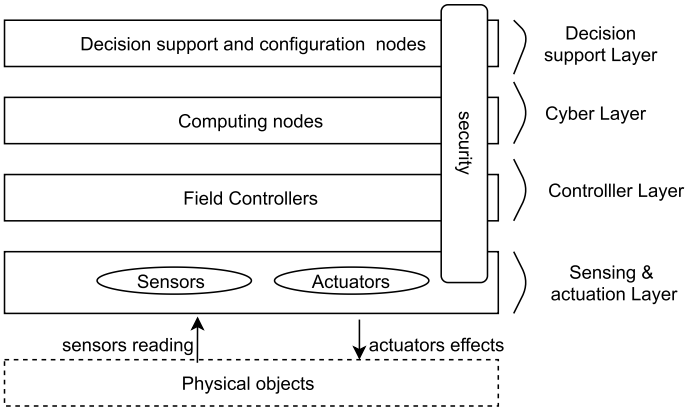

Fig. 1 Layered representation of CPS architecture

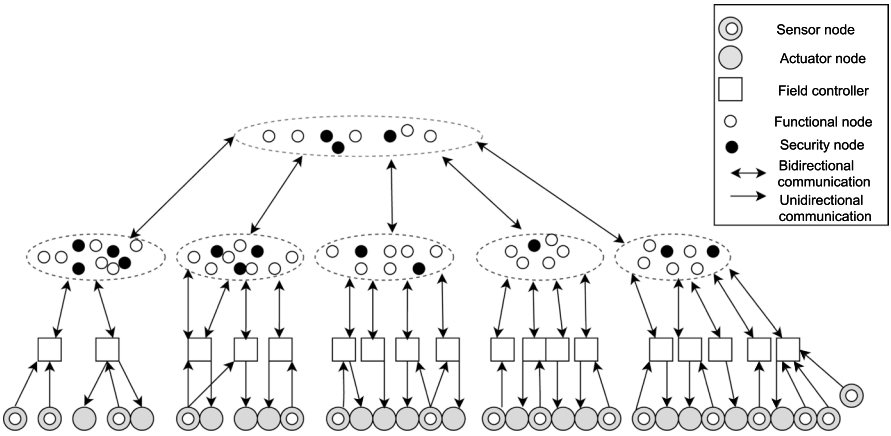

Fig. 2 Clustered view of the proposed distributed CPS architectural model

connected through an arbitrary network topology. $SN = \{S \cup AR \cup FC \cup CN\}$, where $S = \{s_1, s_2, \ldots, s_e\}$, $AR = \{ar_1, \ldots, ar_f\}$, $FC = \{fc_1, fc_2, \ldots, fc_g\}$, and $CN = \{FN \cup SN\}$, $FN = \{fn_1, \ldots, fn_h\}$ and $SN = \{sn_1, \ldots, sn_k\}$ where e, f, g, k and h are integer and $k < h$. The computing nodes are divided into set of non-overlapping clusters $C = \{c_1, \ldots, c_m\}$ such that each $c_l = \{fn \cup sn\}$ where $fn \subseteq FN, sn \subseteq SN$. The clustering is done on the basis of dependent and independent domain. A cluster in C is selected to make a higher level cluster called decision support cluster (*DSC*) to have a global view of system's functionality and intrusion monitoring and response requests. A cluster c_l communicates and coordinates with other clusters via *DSC*. The leaders in each c_l are responsible to establish the inter-cluster communication via *DSC* leaders as shown in Fig. 3. The meaning of all the variables and symbols used in this paper are given in Table 1.

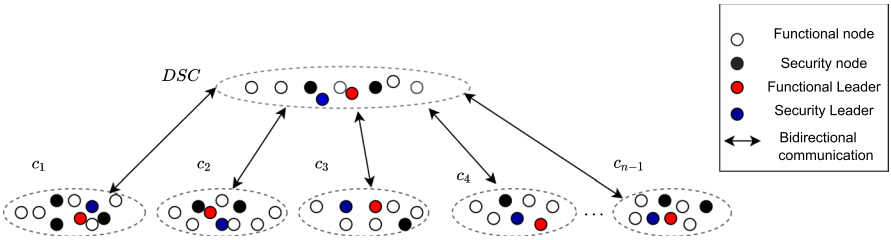

Fig. 3 Clustered view of cyber layer and decision support layer of the proposed distributed CPS architectural model with functionality and security leaders

4.2 Layers responsibilities

In the proposed architectural model (as shown in Fig. 2), different computational responsibilities of the total work of automation, instrumentation, control & security are distributed. These responsibilities and performed by different types of homogeneous or heterogeneous nodes at different layers for different purposes, including sensing, actuation, computing, and coordination. The responsibilities like sensing and actuation are hard-coded or fixed and performed by hardware entities like sensors, actuators, and micro-controllers at lower layers. The bottom two layers follow the fixed distribution. On the other hand, the top two layers follow floating distribution as the tasks may be distributed or reallocated on any computing node.

4.2.1 Sensor and actuation layer

Consists of numerous similar or different types of field devices, including sensor and actuator nodes and represented as gray circles. These nodes may be deployed for environmental and security monitoring. The layer is closer to the real world or physical equipment and infrastructure and responsible for observing and reacting. The sensor nodes perceive the system and environment state variables' value, and events send this information to the controller layer. The actuator nodes receive the control command to execute the required actions directed by upper layers.

4.2.2 Controller layer

Consists of multiple programmable field controllers shown as boxes in Fig. 2. The layer is responsible for performing purely real-time tasks. To respond to real-time functionality and security requirements, each controller receives, processes the sensor data, and instructs the actuator to change its state accordingly. The layer is also responsible for pushing the state information and control status onto the cyber layer and updating the control directives from the upper layer if required. The security at this level is embedded within the controller nodes to perform authentication of communicating nodes and verify sensor values with set values.

Table 1 Notations and their meaning

Notation	Description
SN	Set of nodes in the system
S	Set of sensors
AR	Set of actuators
FC	Set of monitoring & field controlling nodes
CN	Set of computing nodes
G	The graph consisting of all computing nodes
FN	Set of functional nodes
SN	Set of security nodes
C	Set of non-overlapping clusters
DSC	Decision support cluster
D	Diameter of G
R	Radius of G
c_j	j^{th} cluster of graph G
d_j	Diameter of cluster c_j
rd_j	Radius of cluster c_j
N	Number of computing nodes in graph G
$node\_id_i$	Id of the i^{th} node
n	Number of nodes in cluster c_j
λ	Number of attributes
Rk_i	Rank of node i
$eini\_id$	Election initiator Id
c_ack_id	*ack* message creator Id
r_list	A 2D list with two fields. First field contains node Id and second field contains rank of a node
$leader_i$	A node i stores the system leader Id in it
s_em_id	Id of an election message sender
flc_list_i	A node i stores the list of functionality leader capable nodes in it
slc_list_i	A node i stores the list of security leader capable nodes in it
l_id	Newly elected leader Id
t_list	List of transient leader
$parent_i$	Parent node of a node i
l_child_i	List of child nodes of a node i
$failed\_leader\_id$	Failed leader id
tol	Type of leader, tol=1 functionality leader, tol=0 security leader
ton_i	Type of node, if node i is a functionality node then $ton_i = 1$, if node i is a security node then $ton_i = 0$
toe	Type of election, if the election is initiated to elect the functionality leader then $toe = 1$, if the election is initiate to elect the security leader then $toe = 0$
fun_leader_i	Functionality leader Id stored by a node i
sec_leader_i	Security leader Id stored by a node i
Tf	Set of functional tasks
Ts	Set of security tasks

Both sensor & actuation and controller layers follow fixed distribution or have a limited scope of distribution. Hence, controller nodes are considered fixed leaders. Moreover, the redundant nodes are applied at these layers to make the system more fault or breach-tolerant.

4.2.3 Cyber layer

Interacts with the controller layer to monitor the system states and sends the control directives to the controllers. It receives massive real-time data and processes it to extract additional information for context awareness. The layer consists of several computing nodes, arranged as clusters and represented as ellipses. The nodes can collaborate and distribute the management-level operational decisions as tasks among themselves. Moreover, they aggregate and store the data at the local level and send the aggregated data & results to the decision support layer for a global view of the system. Clustering is done based on domain (region) separation to perform specific tasks in each domain and independent of physical proximity. It improves performance, system management, and security by identifying and localizing the system-level faults, isolating the attacked segment, and preventing the cascading failures of the region due to security threats. Each cluster performs some dependent and independent tasks. A region needs to interact with other regions to execute the dependent tasks but does not need any interaction to execute the independent tasks. The nodes communicate within the cluster to execute the independent tasks.

4.2.4 Decision support layer

Consists of multiple computing nodes that mainly communicate & coordinate with each cluster to obtain a global view of the entire system, although the layer can respond to the requests from the cyber layer in real-time. However, it primarily performs non-real-time operations for decision support, such as data correlation and more intense analytics. The layer is responsible for finding, observing, and predicting the CPS behavior, reliability assessment, machine health value, maintenance actions, configuration management, change in management policy and business rules, storage, visualization, auditing, and logging. It does not directly communicate with the controller layer, but it can direct the lower layer to send instructions. It provides operational support to lower layers by load balancing and task prioritization to avoid cascading failure due to overload. The cyber and decision support layers collect, process, and analyze the data to identify the changes in the environment and reconfigure the control decisions accordingly at the local and global situations, respectively.

Since threats may persist at any of the nodes in cyber-phyiscal layers, security of each layer is handled either on the same layer or at the upper layer.

4.3 Role of functionality and security leaders

The field controllers at the controller layer perform dedicated control tasks to respond to the real-time functionality and security requirements. They are

considered fixed leaders. The security module is also embedded within the controller node. It authenticates the attached nodes to establish secure communication, verifies the sensor's data corresponding to predefined set values to detect the unexpected deviation, identifies the non-responsiveness of attached sensor and actuator nodes, and intrusion attempts on the controller node itself. Moreover, redundant or diverse nodes are deployed at this layer to take up the role of failed leader node. At the top two layers, the functionality and security tasks are distributed as two core tasks in each cluster. The functionality nodes are responsible for performing the tasks related to functionalities such as monitoring, execution, storage, and control. The security nodes are responsible for authentication, encryption, secure storage [10], and key management, including key generation, distribution, and storage. Moreover, they monitor, detect and respond to the malicious events or abnormal behavior of functionality nodes [38] and the field controller nodes. The security monitoring nodes take a snapshot of functionality nodes at different times to monitor the discrepancy in their actual and expected behavior and generate alerts. These nodes are also responsible for responding to the detected malicious events by changing the system parameters.

The collaboration, coordination, and communication among the functionality and security nodes are managed by the functionality leader and security leader. The functionality leader coordinates the distribution and aggregation of functionality tasks among different functionality nodes. Similarly, the security leader is responsible for coordinating the distribution and aggregation of preventive and responsive security tasks among different security nodes to identify, prevent and respond to the malicious behavior of functionality nodes. It maintains a list of normal, suspicious, and compromised functional nodes along with the list of failed security nodes. These leader nodes of each cluster are called sub-leaders. If there are n clusters, there will be $2n$ sub-leaders. While designing a secure system, the functionality and security monitoring and response are two independent but coordinated tasks. Hence, the functionality and security leaders of each cluster act as co-leaders. The co-leaders are designed as co-routines to yield concurrency and communication. Further, they transfer the control to each other to execute the system functionalities securely and respond to malicious activities. To coordinate the system-level activities and to establish communication among the clusters, we elect the functionality and security leader at the decision support layer also and call these super-leaders. The communication request and data collected from each functionality and security sub-leaders are transferred to super-leaders to make the system self-aware and reconfigure the functionality and security policies. The election of sub-leaders and the super-leader avoid a bottleneck situation where a single leader may face a heavy load to coordinate all the functionality and security activities of the entire system. As the only system leader coordinates the region-wise dependent and independent tasks, the independent tasks take more time to complete because of the increased communication latency and response from an overloaded leader. Implementing sub-leaders reduces the unnecessary communication latency due to the need to coordinate with super-leader to execute region-wise independent tasks. Thus, the decisions at the local level reduce the upstream bandwidth demand as well. Consequently, the probability of missing the deadline of functionality tasks or security events is reduced.

4.4 The proposed leader election algorithm

In this section, the proposed leader election method is presented to elect the functionality and security leaders. We assume that the system is static and the set of computing nodes (CN) are arranged in a graph G and defined as $G = (CN, L)$, where L is the set of links of graph G. D and R are the diameter and radius of graph G and $\lceil \frac{D}{2} \rceil \leq R \leq D$. As G represents graph of CN, hence, G as specified in Sect. 4.1, divided into m clusters (sub graph) such that $c_1, c_2, \cdots, c_m \subseteq G$. Each cluster c_l has a unique id. The diameter and radius of the c_l are d_l and rd_l, respectively, and defined as $\lceil \frac{d_l}{2} \rceil \leq rd_l \leq d_l$, where $\forall l, d_l < D$. We also assume that each node has a unique id. A leader election algorithm runs to choose the leader when a system starts for the first time, or a leader node is failed, malfunctioned, or becomes non-responsive because of a DoS attack.

As in safety-critical systems, mostly real-time jobs need to be executed. Hence, the proposed algorithm identifies a list of leader capable nodes based on specified node selection criteria. Then, the highest potential (best) node is designated as the chief leader, and the remaining nodes are declared as transient leaders. In case of leader failure, one of the transient leaders instantly takes the responsibility of coordinating the management activities. The time to select a temporary leader is less than to elect a chief leader, so the presented algorithm reduces the election overhead. While selecting the list of potential leaders, the proposed algorithm also handles the threat scenario, where a malicious node can try to initiate the unnecessary leader election process to hamper the system performance by falsifying the information about leader failure. For this, if a node other than transient leaders realizes the leader is failed, it communicates to the transient leaders to inform the leader's failure but cannot initiate the leader election process by itself. The election process is started only when the transient leaders reach the consensus to start the election. The proposed algorithm always tries to elect good-quality leaders for the system. To do that, we introduce the concept of rank calculation of the nodes. Here, the higher rank indicates the higher-good quality. According to the system requirements, several quality attributes can be considered to calculate the rank of a node, for example, memory capacity, processing capacity, failure rate, degree, eccentricity, and so on.

Suppose the set of attributes that need to consider to calculate the rank is A and it contains λ attributes, $A = \{a_1, a_2, a_3, \ldots, a_\lambda\}$. Here, we assume that every node knows the possible maximum and minimum values of every attribute. $Max(a_q)$ and $Min(a_q)$ represent the maximum value and minimum value of an attribute a_q. Hence, the rank Rk_i of a node i is calculated using Eq. 1.

$$Rk_i = \sum_{q=1}^{\lambda} \xi_{iq} \tag{1}$$

$$\text{Here, } \xi_{iq} = \begin{cases} \frac{v_{iq} - Min(a_q)}{Max(a_q) - Min(a_q)}, & \text{if } a_q \text{ is a benefit attribute.} \\ \frac{Max(a_q) - v_{iq}}{Max(a_q) - Min(a_q)}, & \text{if } a_q \text{ is a cost attribute.} \end{cases}$$

where v_{iq} is the value of attribute a_q of a node i. The benefit attributes are those whose higher values are preferred, while cost attributes are those whose lower values are preferred during leader election. The measurement unit of the different attributes can be different, so we use the max-min normalization to normalize the attributes.

4.4.1 Message type

In the proposed election algorithm, we use the following five types of messages.

1. The election message: This message is represented as $em\langle eini\_id, s\_em\_id, toe\rangle$ and consists of the election initiator id, the *em* message sender's id and the type of election. It is used to initiate an election.
2. The acknowledgement message: This message is represented as $ack\langle c\_ack\_id, eini\_id\rangle$ and consists of the *ack* message creator id and the election initiator id. A node creates an *ack* message to respond to getting an *em* message.
3. The rank message: This message is represented as $rank\langle r\_list\rangle$. It is created by child nodes to pass their rank information to the parent node.
4. The leader declaration message: This message is represented as $ld\langle l\_id, t\_list, toe\rangle$ and consists of the elected leader id, the list of transient leaders and the type of election. It is used to declare the elected leader.
5. The failure information message: It is represented as $lfmsg\langle failed\_leader\_id, tol\rangle$ and consists of the failed leader's id and the type of leader that has failed. It is used to inform the transient leaders about the current leader's failure.

4.4.2 Leader election method

Algorithm 1 and Algorithm 2 are designed to elect the functionality and security leaders. Algorithm 1 explains the chief leader election method, while Algorithm 2 explains the transient leader election process. The election method elects the functionality or security leader according to the system need. When the system starts for the first time, any functionality or security node can run Algorithm 1 to elect a chief functionality or security leader, respectively. When the election is initiated to elect the functionality leader, the functionality nodes participate directly, and the security nodes participate indirectly by only forwarding the election messages. Consequently, the r leader capable nodes are selected from the functionality nodes only. The same things happen in the case of the security leader election. Algorithm 1 executes in two phases. In the first phase, the nodes build a tree using election message (*em*) and acknowledgement message (*ack*). A node i creates $em\langle eini\_id, s\_em\_id, toe\rangle$ to initiate the election process, where the *eini_id* and *s_em_id* are the same as *node_id*. Here, the boolean variable *toe* is used to represent the type of election. That means the election is started for electing functionality leaders or security leaders. If $toe = 1$, the election is for electing the functionality leader. On the other hand, if $toe = 0$, the election is for electing the security leader. As node i initiates election, it is considered as the root node of the tree where $parent_i = \phi$. It sends $em\langle eini\_id, s\_em\_id, toe\rangle$ to all the adjacent nodes and waits for $ack\langle c\_ack\_id, eini\_id\rangle$. When an adjacent

node j receives the $em\langle eini\_id, s\_em\_id, toe\rangle$, it creates $ack\langle c\_ack\_id, eini\_id\rangle$ message and sends it to node i. Here, the node i considers node j as is child node and node j considers node i as its parent node. Then, node j modifies and forwards the election message to its adjacent nodes, except its parent node (node i), and waits for the acknowledgement message. There may be two cases: (1) either it receives the election message from one node or (2) it receives the redundant election message from multiple nodes as duplicate messages. Hence, the receiving node checks if the election message is received for the first time by checking $eini\_id$, it considers the message sender node as its parent and sends back the *ack* message to it in response. Otherwise, it does not respond or send an acknowledgement message to the predecessor node. The steps repeat until the election message is circulated among all the nodes in the system.

In the second phase, all the nodes send their rank value to their parent node. To send its rank, a node j checks that if it is a leaf node or does not get any $ack\langle c\_ack\_id, eini\_id\rangle$ message from the adjacent nodes, it appends its id and rank in its rank list (r_list) and sends it to its parent node through rank message $rank\langle r\_list\rangle$. The parent node collects $rank\langle r\_list\rangle$ messages from all its child nodes, makes a list of nodes by sorting the collected child nodes' ranks and self-rank according to the rank value in descending order. Then, top r values are selected from the sorted list and sent to its parent node. The process is repeated until the root node gets the $rank\langle r\_list\rangle$ message from all its child nodes. The root node sorts the nodes to get the r leader capable nodes' list. If $toe = 1$, the top node of the r_list is declared as the chief functionality leader and the remaining $r - 1$ nodes are declared as the transient functionality leaders. If $toe = 0$, the top node of the r_list is declared as the chief security leader and remaining $r - 1$ nodes are declared as the transient security leaders. The root node broadcasts a $ld\langle l\_id, t\_list, toe\rangle$ message to declare the elected leader as well as the transient leaders. On receiving the $ld\langle l\_id, t\_list, toe\rangle$, node j

Algorithm 1: Chief Leader Election

```
// When a node i initiates an election.
eini_id ← node_id_i, s_em_id ← node_id_i, parent_i ← None
Set the value of toe according to the type of the election.
Create an em⟨eini_id, s_em_id, toe⟩ message and send it to all the adjacent nodes.
// When a node j gets a em⟨eini_id, s_em_id, toe⟩ message.
if (the received message is a em⟨eini_id, s_em_id, toe⟩) then
    if ((parent_j == Empty) ∨ (eini_id > parent_j)) then
        parent_j ← s_em_id, c_ack_id ← node_id_j, s_em_id ← node_id_j
        Create an ack⟨c_ack_id, eini_id⟩ message and send it to the parent node.
        Send the em⟨eini_id, s_em_id, toe⟩ to all adjacent node except the parent node and wait a certain amount of time to get the ack
          messages from those nodes.
        if (the node j is a leaf node or does not get any ack message from the adjacent nodes) then
            Insert self Id and rank in the r_list and send the list to the parent node through a rank⟨r_list⟩ message
        end
    else
        Discard the received message.
    end
end
// When a node j gets a ack⟨c_ack_id, eini_id⟩ message.
if (the received message is a ack⟨c_ack_id, eini_id⟩) then
    Insert the c_ack_id into the l_child_j.
end
// When a node j gets rank⟨r_list⟩ messages from its all child nodes.
if (|r_list| + 1 > r) then
    According to the rank value of the nodes, choose the best r nodes among the j^th node itself and its child nodes.
    Store the best r nodes' Id and rank value in r_list.
else
    Insert the self Id and rank value in r_list.
end
Send the r_list to the parent node through a rank⟨r_list⟩ message.
// When the election initiating node that conducts the whole election (here node i) gets
    rank⟨r_list⟩ messages from its all child nodes.
Node i arranges all the received node Ids (through rank⟨r_list⟩ messages) and self Id in descending order according to their rank value.
Choose the best r nodes among them.
if (toe == 1) then
    Choose the best node as the chief functionality leader .
    l_id ← the elected chief functionality leader Id
    Put the rest r − 1 node Ids in the t_list as the leader capable node for the transient functionality leader.
    fun_leader_i ← l_id, flc_list_i ← t_list
else
    Choose the best node as the chief security leader .
    l_id ← the elected chief security leader Id
    Put the rest r − 1 node Ids in the t_list as the leader capable node for the transient security leader.
    sec_leader_i ← l_id, slc_list_i ← t_list
end
Create the ld⟨l_id, t_list, toe⟩ message and sends it to all the adjacent nodes.
// When node j gets ld⟨l_id, t_list, toe⟩ message.
if (toe == 1) then
    fun_leader_j ← l_id, flc_list_j ← t_list
else
    sec_leader_j ← l_id, slc_list_j ← t_list
end
Send the ld⟨l_id, t_list, toe⟩ message to all the adjacent nodes except its sender.
If a node j gets the ld⟨l_id, t_list, toe⟩ message multiple times, it processes the first received ld⟨l_id, t_list, toe⟩ message and discards
  the rest.
```

Algorithm 2: Selection of a Transient Leader

```
// When a node i realizes that a chief leader is failed or non-responsive.
if (node i realizes that a chief leader is failed) then
    create and send a lfmsg⟨failed_leader_id, tol⟩ message to all the nodes in lc_list
end
// When every node of the lc_list gets a lfmsg⟨failed_leader_id, tol⟩
Every node in lc_list checks whether the leader (failed_leader_id ) is failed.
if (The leader (failed_leader_id ) is failed) then
    if (|lc_list| > r/2) then
        if (tol == 1) then
            Select the best node from the flc_list and put the rest nodes in the t_list.
            Declare the selected best node as the functionality leader by broadcasting a leader declaration message
        else
            Select the best node from the slc_list and put the rest nodes in the t_list.
            Declare the selected best node as the security leader by broadcasting a leader declaration message
        end
    else
        Invoke algorithm 1
    end
end
```

checks value of *toe*, if $toe = 1$, it updates the chief leader id as functionality chief leader and transient leaders list as functionality transient leaders. If $toe = 0$, it updates the chief leader id as a chief security leader and transient leaders list as transient security leaders. It is worth mentioning that if two or more nodes realize and initiate election simultaneously, the election message created by the node with highest id survives in the network. Thus, the node with highest id gets the scope to create the tree. On the other hand, the election messages created by the other nodes get discarded that helps to avoid multiple election trees formation.

Algorithm 2 runs to select the transient leaders. The leader failure may be realized by either leader-capable nodes or non-leader capable (normal) nodes. When a node realizes that the chief leader has been failed or become non-responsive, it creates a message *lfmsg*⟨*failed_leader_id*, *tol*⟩ and sends it to all the leader-capable nodes. Then, the transient leader nodes verify whether the chief leader has failed and initiating elections based on mutual consensus. If the leader is failed and the number of transient leader nodes is greater than *r*/2, the top alive node is selected from *t_list* as functionality or security leader based on the *tfl* value. After that, the *t_list* is updated and a *ld*⟨*l_id*, *t_list*⟩ message is broadcast to all the nodes. Otherwise, if the nodes in leader capable list (*lc_list*) are less than or equal to *r*/2, the nodes build the consensus to initiate election, and the highest leader capable node among them invokes algorithm 1 to elect a chief leader. Thus, the proposed algorithm also prevents any undetected compromised node from abusing the leader election process.

4.4.3 Complexity analysis

The complexity of the leader election algorithm is measured in terms of message complexity and time complexity. In this section, we calculate the message and time complexities of the proposed election algorithm considering a network of *N* nodes and *D* diameter.

Message complexity

As the nodes communicate by message passing, the message complexity depends upon the number of messages exchanged among the nodes during an election.

Best case: When the number of alive transient leader nodes is more than $r/2$, and one of them realizes the chief leader's failure, then it is the best-case scenario of the algorithm. In this case, the node that realizes the leader's failure informs the other transient leader nodes about the leader's failure. Then, all the transient leader nodes collaboratively elect the highest leader capable node from the list of transient leaders as the new leader and declare the elected leader by broadcasting the leader declaration message. Here, $O(r)$ messages are required to inform the leader's failure to all the transient leader nodes and $O(N)$ messages are required to broadcast the elected leader. $N \geq r$, hence in the best case, the message complexity of the proposed leader election algorithm is $O(N)$.

Worst case: When the number of alive transient leader nodes is less than $r/2$, and all the nodes realize the leader's failure concurrently, it becomes the worst-case scenario of our algorithm. In this case, all nodes initiate the election concurrently to identify the r leader capable nodes. Here, a maximum of $O(N^2)$ messages are exchanged to build the election tree. After that, $O(N)$ rank messages are exchanged for passing the ranks to the election conducting node. Finally, $O(N)$ leader declaration messages are exchanged to declare the leader. So, in this case, the message complexity is $O(N^2)$.

Time complexity

Time complexity quantifies the time required to elect a leader.

Best and worst cases: In the best case, $O(D)$ time is required to inform the leader's failure information to all the alive transient leader nodes, and $O(D)$ time is required to broadcast the leader declaration message. So, in the best case, the time complexity is $O(D)$. In the worst case, the time complexity depends on the election tree construction time, time to pass the ranks to the election conducting node, and to broadcast the leader declaration message. Each of these three steps takes $O(D)$ time. Hence, in the worst-case, the time complexity is also $O(D)$.

4.5 Resilience against cyber attacks

Since, cyber threats may persist at any of the nodes in cyber-physical layers, security of each layer is handled either on the same layer or at the upper layer. The field controllers perform dedicated control tasks to respond to the real-time functionality and security requirements at the controller layer. The security module is also embedded within the controller node to retaliate AS(1) and AS(2). It authenticates the attached nodes to establish secure communication, verifies the sensor's data with predefined set values to detect the unexpected deviation [23, 32], identifies the non-responsiveness of attached sensor and actuator nodes using heart beat message [17], and intrusion attempts on the controller node itself. Moreover, redundant or diverse nodes are deployed at this layer to take up the role of failed leader node. However, the methods of how these security mechanisms are implemented are already known and available in the literature [17, 23, 32].

At cyber and decision support layers, the functionality and security leader collaborate as co-routines [41, 45] to respond to the malicious events at the cluster level. To defend against AS(3), the model can retaliate the attacks on a functionality

node, functionality leader, security node, or security leader. Initially, the security leader maintains a list of nodes with normal status. When a security monitoring node observes a functionality node is behaving suspiciously, it informs the security leader. To detect and tolerate security monitoring node failure, the methods are already known and available in the literature [16, 22, 36]. After confirming the suspected behavior to be malicious, the security leader removes the node from the normal node list and adds it to the compromised node list. It sends a compromised node id to the functionality leader, which reallocates that node's responsibility among the least-loaded normal functionality nodes. The attack on security monitoring and control nodes is observed by the security leader, as it communicates with the security monitoring nodes periodically. If the security monitoring node does not respond, the security leader assumes it to be failed. In this situation, the security leader isolates the compromised node and reallocates the security task to the least loaded node. Similarly, when any security monitoring node attempts to communicate with the security leader and does not get any response, it communicates to one of the transient leaders. All the transient leaders would verify by sending the heartbeat message to the chief leader node and reach a consensus of whether the chief leader is failed due to attack as mentioned in Sect. 4.4. To defend against AS(4), the proposed algorithm can prevent the abuse of the leader election process itself. The algorithm can deal with the scenario where a malicious node tries to initiate the election process unnecessary to target an unbiased leader. Only the leader-capable nodes are responsible for verifying whether the chief leader has failed and initiating elections based on mutual consensus. Thus, the proposed architectural framework can tolerate or respond to the mentioned attack scenarios or exceptional conditions.

5 Performance evaluation of the proposed architectural model

In this section, we analyze and demonstrate the effectiveness of the proposed CPS architecture through several experiments. We show that the concept of clustering and separation of functionality and security leaders helps to improve the overall performance and security management. A distributed smart hospital management case study is considered to explain the proposed architecture, where multiple hospitals are connected as a medical-CPS.

5.1 Case study

A smart hospital is a concept that uses emerging technologies of information and communications technology (ICT) to optimize and manage the health care operations and its functional requirements efficiently [30]. Smart hospitals fall under the safety-critical domain as the safety of patients is at most priority. Any security failure in terms of denial of service or integrity failures of life support systems may lead to unsafe situations for the system. It may consequently create big chaos in patients' lives. Various sensors and data collection devices are deployed to monitor the environmental conditions, hospital resources, and

services. Different actuators respond as specified and controlled by controllers. In our proposed architectural model, each cluster with computing nodes represents a hospital. The computing nodes store and process the collected environmental, operational, and patient data (confidential and non-confidential) to perform different functionality and security monitoring tasks as shown in Fig. 4. The nodes do intra-cluster communication to perform the cluster independent tasks via the leader node. The nodes do inter-cluster communication to perform multiple clusters dependent tasks via cluster leaders. There may be various functionality tasks in hospital management, although, to demonstrate the effectiveness of our proposed approach, we are just demonstrating the example of treating the COVID patients and distribution, deployment, and administration of the vaccine for fighting with COVID-19 pandemic [11].

The decision support layer represented as $c5$ (as shown in Fig. 4) and it communicates to each cluster leader to optimize the availability of vaccines in each hospital by monitoring the lack or excess of the vaccine. Moreover, it stores the updated records of vaccinated persons and the total number of treated and active COVID patients and their distribution in each region/ hospital at the country level. The probable attack scenarios may include a denial of service attack, malfunctioning of life support system units in intensive care units, or an integrity attack on vaccinated person records. The attacker may delete the vaccine availability and distribution records, malfunctions computing nodes, breach patient records suffering from other critical diseases, disturb the HVAC control unit. Moreover, a successful integrity attack on a node that performs the staff-allocation task compromises its functionality in the critical time. As a result, the compromised node allocates a non-specialized doctor.

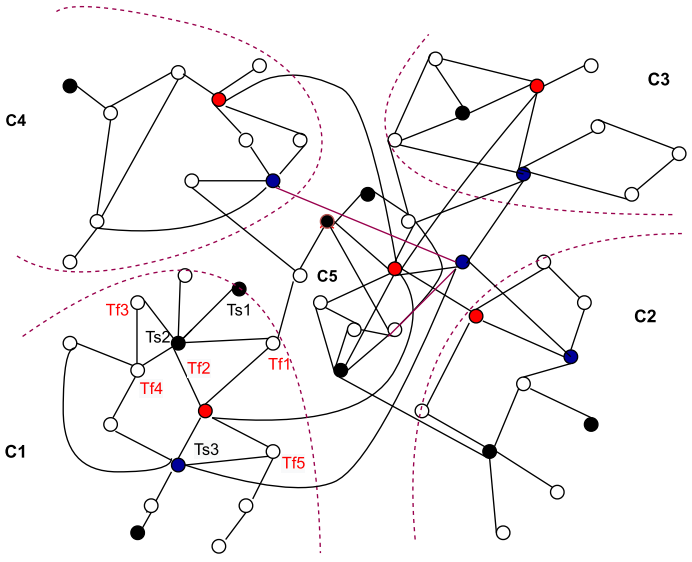

Fig. 4 Cluster arrangement of a distributed hospital network with functionality and security leaders

Suppose, in cluster $c1$, a set of functionality tasks $Tf = \{Tf1, Tf2, Tf3, Tf4, Tf5\}$ are running on functionality nodes. There are security tasks $Ts = \{Ts1, Ts2, Ts3\}$ running on security monitoring nodes. The security monitoring nodes observe the incorrect behavior of the functionality nodes by observing the deviation in allocated functionality tasks. It sends a message to the security leader on finding a node with suspicious behavior, which sends the message to the functionality leader and blocks the compromised node. The functionality leader reallocates the staff-allocation task to the least-loaded node to avoid further chaos.

5.2 Performance evaluation

To analyze the system management improvement, we have considered four different forms of system management, i.e., purely centralized, purely distributed without a leader, distributed with a single leader, and clustered distributed manner with multiple leaders (including functionality and security leaders). To evaluate the performance of the proposed architecture with each of these management forms, a p step task is considered. A p step task is defined as a task T that involves p steps to complete it. Suppose there are N nodes in the system.

(1) Purely centralized In this manner, a fixed central node controls and manages other nodes and all the system's activities. Here, the main problem is a single-point failure. When the fixed central node collapses, the whole system collapses. Hence, the fault-tolerance capacity of the system is minimum. To complete a p step task, a node exchanges p number of messages with the central node. The message complexity of completing this task is $O(p.D.N)$, and the time complexity is $O(D)$ where D is the network's diameter.

(2) Distributed without considering the leader There is no central node that controls and manages the system. Here, a node needs to send messages to all the other nodes to complete a task consistently. In this manner, the fault-tolerance capacity of the system is maximum, but the message and time complexities are very high. Here, the message complexity and the time complexity of completing a p step task are $O(p.N^2)$ and $O(D)$, respectively.

(3) Distributed with a single leader The system is managed in a distributed way by electing a node as the system leader, as discussed earlier. Here, the system is managed similarly to the centralized manner. The only difference is that the central node (the system controlling node) is fixed in a centralized manner, but here the central node is not fixed. If the central node is crashed, another node can be elected as the central node or the leader. The leader election overhead (extra cost) is associated with this manner. Here, the message complexity and the time complexity of completing a p step task are $O(p.D.N)$ and $O(D)$, respectively.

(4) Clustered distributed with multiple leaders In this manner, the CPS is managed in distributed manner but divided into multiple clusters. Each cluster has a functionality leader and security leader. Intra-cluster functionality and security tasks are managed by its functionality and security leaders, respectively. On the other hand, inter-cluster tasks are managed by the leaders of the clusters. Here, the message complexity and the time complexity of completing a p step inter-cluster task

are $O(p.d_{max}.N)$ and $O(D)$, respectively, where d_{max} is the maximum diameter of the clusters. The message complexity and the time complexity of completing a p step intra-cluster task are $O(p.d.n)$ and $O(d)$, respectively, where d is the cluster's diameter, and n is the number of nodes in the cluster.

Here, $d_{max} \leq D$ and $D < N$. So, $O(p.d_{max}.N) \leq O(p.D.N) < O(p.N^2)$. That means if we manage a CPS in distributed clustered manner with multiple leaders, to complete a task, the number of exchanged messages (network traffic) gets reduced. As $d < D$, then $O(d) < O(D)$.

We consider the COVID-19 vaccine distribution, deployment, and administration (as mentioned in the case study) task to simulate and evaluate the proposed architecture's effectiveness and performance with the management schemes as mentioned earlier. To simulate the proposed architectural model, we use *python 3.6* as a programming language, MPICH version 3.2, and *mpi4py* tool as a message passing interface. We have considered six different sizes of networks where nodes of each network are connected through an arbitrary network topology. All the networks details are given in Table 2, and the simulation results are shown in Figs. 5, 6. We perform the entire simulation in a single machine equipped with Intel (R) Core(TM) i7-3770 processor (3.40 GHz, 8 MB cache), 26 GB DDR3 RAM, 1TB 5400rmp HDD, NVIDIA GeForce graphics, running Ubuntu Linux Release 16.04 (xenial kernel 4.4).

In Fig. 5, X-axis represents the number of nodes, and Y-axis represents the number of messages exchanged to complete the task. In Fig. 6, X-axis represents the number of nodes, and Y-axis represents the time required to complete the task. Figures 5, 6 show that the number of exchanged messages among the nodes and the time required to complete the task in a purely distributed manner without a leader are highest. In contrast, the number of exchanged messages and time required to complete the task is least when the system is managed in a distributed clustered manner with multiple leaders for functionality and security. Figure 7 shows the average response time of the tasks when we manage the top two layers in a distributed manner with a single leader and in a distributed clustered manner with multiple leaders. In Fig. 7, X-axis represents number of tasks, and Y-axis represents the average response time of functionality and security tasks. Here, we have considered that out of total tasks, one-third are security tasks, and two-thirds are functionality tasks. Figure 7 concludes that if we manage the CPS in a distributed clustered manner with

Table 2 Details of the networks considered for the experiments

Network	Number of nodes	Number of edges	Diameter	Number of cluster
Network 1	30	52	8	3
Network 2	60	98	10	4
Network 3	90	176	12	5
Network 4	120	256	14	6
Network 5	150	290	16	7
Network 6	180	375	18	8

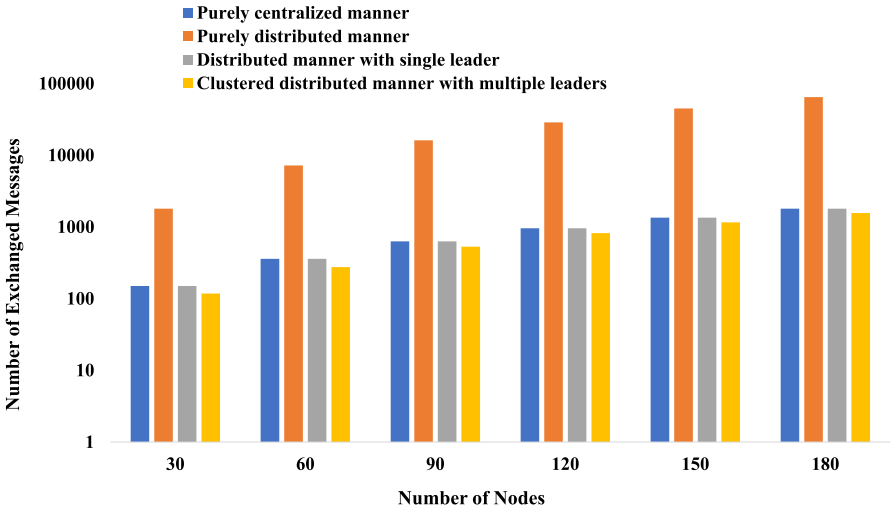

Fig. 5 Comparison of the proposed system management manner with other possible system management manners based on the number of exchanged messages to complete the task

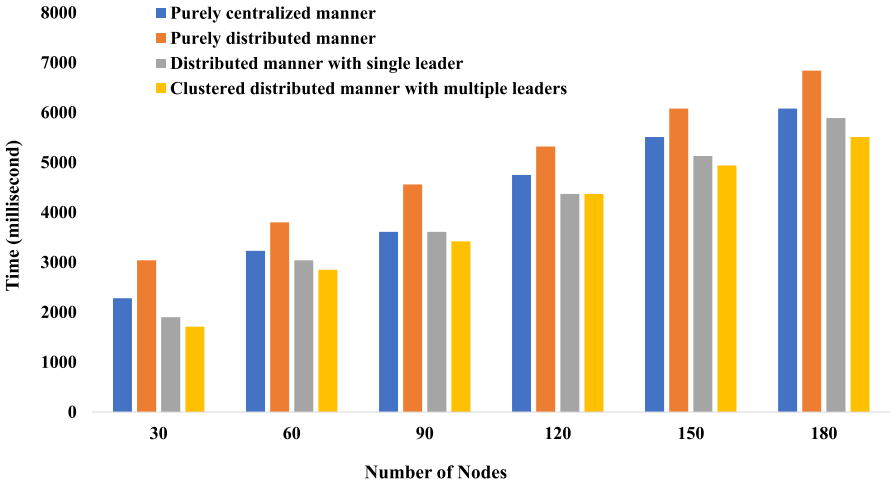

Fig. 6 Comparison of the proposed system management manner with other possible system management manners based on the time required to complete the task

multiple leaders, the average response time of tasks gets reduced. Figure 8 compares the completion rate of real-time tasks within a specified deadline while managing the cyber layer in a distributed manner with a single leader and in a clustered distributed manner with multiple leaders. In Fig. 8, X-axis represents number of tasks, and Y-axis represents the completion(or success) rate of real-time tasks within a specified deadline. Here, we have considered that out of total tasks, half are real-time task, and half are non-real-time tasks. We have used priority scheduling to schedule

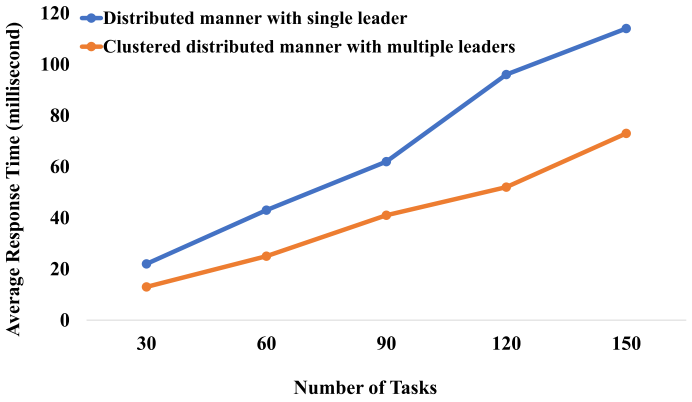

Fig. 7 Comparison of the proposed system management manner with the distributed manner with a single leader based on the average response time of the task

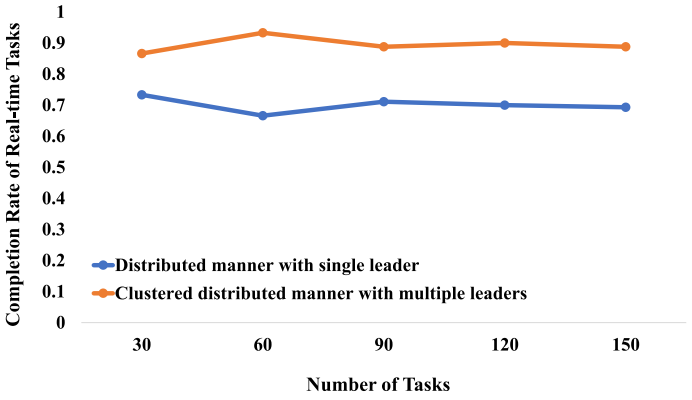

Fig. 8 Comparison of the proposed system management manner with the distributed manner with a single leader based on the success ratio of real time tasks completion within deadline

these tasks, where real-time tasks have priority over non-real-time tasks. Figure 8 concludes that success rate of real-time tasks gets increased if we manage the CPS in a clustered distributed manner with multiple leaders. From the simulation results shown in Figs. 7 and 8, it can be observed that the distributed clustered manner with separate functionality and security leaders is more efficient in terms of average response time and when the system needs to honor the deadlines of the real-time tasks. We performed two statistical tests, i.e., the Quantile-Quantile plot (Q-Q plot) test [28] and the Shapiro-Wilk test [18] on the completion ratio of real-time tasks and the average response time of the tasks through the proposed system management manner (clustered distributed manner with multiple leaders). The Q-Q plots of the average response time and the completion ratio of real-time tasks are shown in Figs. 9 and 10, respectively. The Q-Q plots show that the average response time

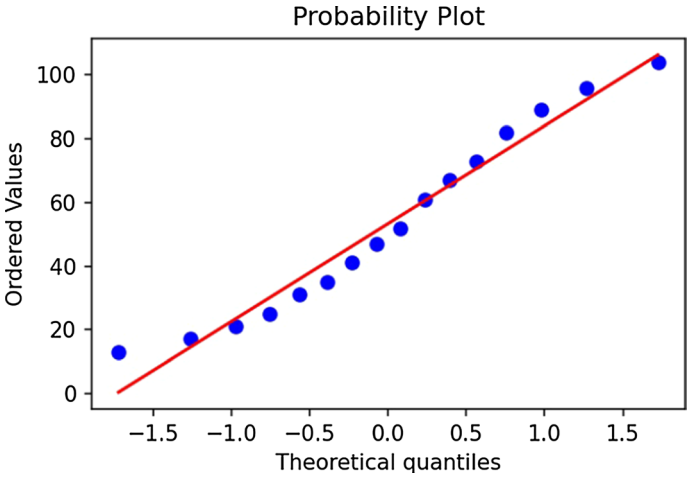

Fig. 9 Quantile-Quantile plot (Q-Q plot) on the average response time of the tasks getting through the proposed system management manner

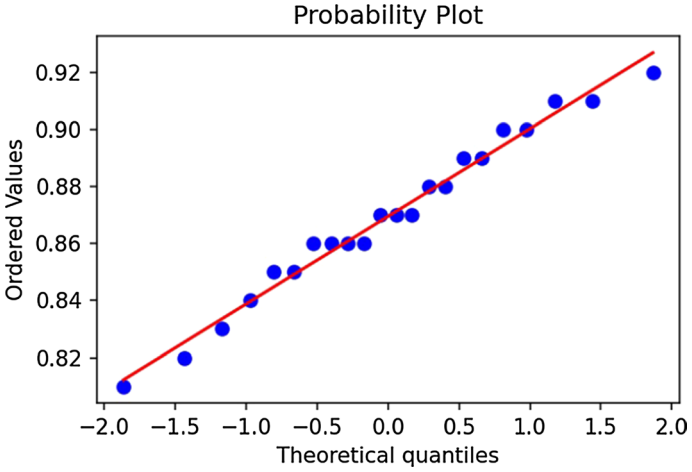

Fig. 10 Quantile-Quantile plot (Q-Q plot) on the completion ratio of real-time tasks getting through the proposed system management manner

and the completion ratio of real-time tasks follow the normal distribution. The p value of the Shapiro-Wilk test of the completion ratio of real-time tasks is 0.843. As $0.843 > 0.05$, the completion ratio of real-time tasks follows the normal distribution. On the other hand, the p value of the Shapiro-Wilk test of the average response time is 0.466. Here, 0.466 is also greater than 0.05, which means the average response time also follows the normal distribution.

Moreover, the proposed distributed clustered manner with multiple leaders is more fault-tolerant and maintainable as compared to other architectural

arrangements. The proposed arrangement avoids a single point of functionality and security failure in better ways and conquers the complexity of designing a secure CPS. When there is a change or update in security policy, it will not affect the system functionality. The proposed architectural model has the leader election overhead, but it is reasonable as it increases system performance and the fault-tolerance capability of distributed CPS.

Table 3 presents a comparative analysis of the proposed architectural arrangement with other architectural arrangements and philosophy. It shows the strength of the proposed work, which comes by including the objective neither considered nor analyzed in the existing works.

6 Conclusion and future work

A cyber-physical system (CPS) is a kind of distributed system with safety-critical functionalities. Nowadays, with the ever-growing technological expansion, large-scale CPSs are in demand. However, designing a secure, fault-tolerant, maintainable, and performance-efficient large-scale CPS is challenging. In this work, to improve the performance, maintainability, fault-tolerability, and security of a large-scale CPS, we propose a multi-tier architectural model of a cyber-physical system. We introduce the concept of separately managing the functional and security concerns by electing the functional and security leaders for better management of the proposed CPS. Management of functional concerns and security concerns separately improves the maintainability as well as the performance of the system. This separation is similar to the aspect orientation in design and implemented by exploiting the concept of leader(s) as available in the case of the distributed computing system. Here, the security and functionality leaders act as co-leaders to collaborate and communicate among themselves for preventing and responding to security threats. On the other hand, management of a CPS is done in a clustered distributed manner, which improves the system performance and makes the system more fault-tolerant. Along with this, we have proposed a fault-tolerant leader election algorithm. Unlike the existing algorithms, along with electing a leader, the proposed algorithm identifies and makes a list of leader-capable nodes. So that if a leader fails, the system can instantly elect a new leader from among the identified leader capable nodes to minimize the adverse effect on real-time task coordination, system performance, and security. Thus, the proposed architecture improves the maintainability, performance, security of the system and makes the system more fault-tolerant. Further, we perform several experiments by simulating the proposed architecture to evaluate its performance. The experimental results show that the proposed architectural model improves the system performance in terms of latency, average response time, and the number of real-time tasks completed within the deadline. The proposed architectural model has the leader election overhead, but it is reasonable as it increases system performance and the fault-tolerance capability of the distributed CPS. On the other hand, because security and functionality are separately scalable, the overhead of functionality nodes is minimized. This work does not present a scalability study of the proposed approach. However, it is not a limitation, but an elaborated study

Table 3 Comparative analysis with existing works

Existing Work	System Management	Target Work	Limitations	Security arrangement	Fault-tolerance	Performance	Maintainability
[14]	Not discussed	Presents semantic web technology driven CPS architecture	Not a unified architecture for integrating functionality and security	Not considered	Not discussed	Not discussed	Not considered
[26]	Not discussed	CPS reference architecture to describe its layers and roles	Not a unified architecture for integrating functionality and security	Not considered	Not discussed	Not discussed	Not considered
[20]	Not discussed	Three-layer architecture for a smart city	Not a unified architecture for integrating functionality and security	Partial	Not considered	Not evaluated	Not considered
[57]	Not discussed	Hierarchical architecture with game theory to deal with cross-layer CPS security	Not a unified architecture for integrating functionality and security	Considered	Not discussed	Not evaluated	Not considered

Table 3 (continued)

Existing Work	System Management	Target Work	Limitations	Security arrangement	Fault-tolerance	Performance	Maintainability
[31]	Centralized manner	Software-defined IoT architecture for smart urban sensing where Centralized controllers are designed to manage physical devices and provide APIs of data acquisition, transmission, and processing services	Not a unified architecture for integrating functionality and security. Moreover, the risk of a single point of failure	Not considered	Low due to the risk of single-point failure	Not evaluated	Not considered
[50]	Centralized network control	SDN-based architecture for IoT to ensure secure data communication using the inherent capability of SDN controllers	Only focus on the network layer and traffic security where security is the sole responsibility of the SDN controller. Moreover, the risk of a single point of failure	Considered	Low due to the risk of single-point failure	Low due to bottleneck	Not considered

Table 3 (continued)

Existing Work	System Management	Target Work	Limitations	Security arrangement	Fault-tolerance	Performance	Maintainability
[3]	Centralized manner	Centralized framework for system-level control and management of additive manufacturing fleets	Not a unified architecture for integrating functionality and security, Moreover, risk of a single point of failure	Not considered	Ow due to risk of single-point failure	Not evaluated	Not considered
[13]	Decentralized manner	Gossip-based algorithm for achieving performance and fault-tolerance properties	not a unified architecture for integrating functionality and security	Not considered	Considered	Considered	Not considered
[46]	Decentralized manner	A cloud-based multi-tier service-oriented architectural model with ontological constructs for interactions among different heterogeneous devices for IoT-based smart home	Who is coordinating the task distribution and aggregation is not discussed	Considered	No explicit arrangement is presented	Not evaluated	Considered
[43]	Decentralized manner	Hybrid smart city cyber security architecture to analyze the threats and associated risk	Not a unified architecture for integrating functionality and security concerns	Not considered	not considered	Not evaluated	Not considered

Table 3 (continued)

Existing Work	System Management	Target Work	Limitations	Security arrangement	Fault-tolerance	Performance	Maintainability
[19]	Distributed manner	Distributed architecture as Black SDN-IoT with SDN controller, device virtualization to control and monitor the traffic data flow	Only focus on the network layer and traffic security. Moreover, not a unified architecture for integrating functionality and security concerns	Considered	High	Analyzed	Medium
[27]	Distributed manner	Emphasized the need for distributed architectural management, where security distribution should be at sensor level as well as computing level to take advantage of distributed computing in handling the performance and privacy concerns	Not a unified architecture for integrating functionality and security concerns. Moreover, no discussion on how to coordinate, distribute and aggregate the functional and security tasks	Considered	No explicit arrangement is presented to achieve it	Not evaluated	Not considered
Proposed work	Clustered distributed manner	A unified architectural arrangement for integrating functionality and security concerns in existing system and new system	Can be extended for other dependability attributes	Considered	High, due to the proposed chief and transient leaders	Considered and evaluated	Considered with aspect-orientation

needs to be performed in the future. Moreover, the work related to application of learning in CPS may take benefit of similar approaches used in different domain including [2, 52–54] in future.

References

1. Albright D, Brannan P, Walrond C (2010) Did Stuxnet take out 1,000 centrifuges at the Natanz enrichment plant? Institute for science and international security
2. Bai B, Li G, Wang S, Wu Z, Yan W (2021) Time series classification based on multi-feature dictionary representation and ensemble learning. Expert Syst Appl 169:114162
3. Balta EC, Tilbury DM, Barton K (2018) A centralized framework for system-level control and management of additive manufacturing fleets. In: 2018 IEEE 14th International Conference on Automation Science and Engineering (CASE), IEEE, 1071–1078
4. Biswas A, Dutta A (2016) A timer based leader election algorithm. In: 2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), IEEE, 432–439
5. Biswas A, Tripathi AK (2021) Preselection based leader election in distributed systems. In: Proceedings. 14th international symposium on intelligent distributed computing, (accepted), Springer
6. Biswas A, Maurya AK, Tripathi AK, Aknine S (2021) Frlle: a failure rate and load-based leader election algorithm for a bidirectional ring in distributed systems. J Supercomput 77(1):751–779
7. Biswas A, Tripathi AK, Aknine S (2021b) Lea-tn: leader election algorithm considering node and link failures in a torus network. J Supercomput, 1–38
8. Bordel B, Alcarria R, de Rivera DS, Robles T (2018) Process execution in cyber-physical systems using cloud and cyber-physical internet services. J Supercomput 74(8):4127–4169
9. Camacho CR, Marczak S, Cruzes DS (2016) Agile team members perceptions on non-functional testing: influencing factors from an empirical study. In: 2016 11th International Conference on Availability, Reliability and Security (ARES), IEEE, pp 582–589
10. Castiglione J, Pavlovic D (2019) Dynamic distributed secure storage against ransomware. IEEE Transactions on computational social systems
11. Ciotti M, Ciccozzi M, Terrinoni A, Jiang WC, Wang CB, Bernardini S (2020) The covid-19 pandemic. Crit Rev Clin Lab Sci 57(6):365–388
12. Feng Y, Hu B, Hao H, Gao Y, Li Z, Tan J (2018) Design of distributed cyber-physical systems for connected and automated vehicles with implementing methodologies. IEEE Trans Ind Inform 14(9):4200–4211
13. Garofalo G, Giordano A, Piro P, Spezzano G, Vinci A (2017) A distributed real-time approach for mitigating cso and flooding in urban drainage systems. J Net Computer Appl 78:30–42
14. Gaur A, Scotney B, Parr G, McClean S (2015) Smart city architecture and its applications based on iot. Procedia Computer Sci 52:1089–1094
15. Gibbs S (2018) Triton: hackers take out safety systems in'watershed'attack on energy plant. The Guardian
16. Goodloe AE, Pike L (2010) Monitoring distributed real-time systems: a survey and future directions. National Aeronautics and Space Administration, Langley Research Center
17. Gouda MG, McGuire TM (1998) Accelerated heartbeat protocols. In: Proceedings. 18th International Conference on Distributed Computing Systems (Cat. No. 98CB36183), IEEE, pp 202–209
18. Hanusz Z, Tarasińska J (2015) Normalization of the kolmogorov-smirnov and shapiro-wilk tests of normality. Biom Lett 52(2):85–93
19. Islam MJ, Mahin M, Roy S, Debnath BC, Khatun A (2019) Distblacknet: a distributed secure black sdn-iot architecture with nfv implementation for smart cities. 2019 International Conference on Electrical. Computer and Communication Engineering (ECCE), IEEE, 1–6
20. Jalali R, El-Khatib K, McGregor C (2015) Smart city architecture for community level services through the internet of things. In: 2015 18th International Conference on Intelligence in Next Generation Networks, IEEE, 108–113
21. Jiang JR (2018) An improved cyber-physical systems architecture for industry 4.0 smart factories. Adv Mech Eng 10(6):1687814018784192

22. Kargl F, Klenk A, Schlott S, Weber M (2004) Advanced detection of selfish or malicious nodes in ad hoc networks. In: European workshop on security in Ad-hoc and sensor networks, Springer, 152–165
23. Keshtkarjahromi Y (2021) Method and system that determine malicious nodes in a distributed computation network. US Patent App. 17/069,077
24. Kiczales G, Lamping J, Mendhekar A, Maeda C, Lopes C, Loingtier JM, Irwin J (1997) Aspect-oriented programming. In: European Conference on Object-Oriented Programming, Springer, 220–242
25. Lawal BH, Nuray A (2018) Real-time detection and mitigation of distributed denial of service (ddos) attacks in software defined networking (sdn). In: 2018 26th Signal Processing and Communications Applications Conference (SIU), IEEE, 1–4
26. Lee J, Bagheri B, Kao HA (2015) A cyber-physical systems architecture for industry 4.0-based manufacturing systems. Manuf Lett 3:18–23
27. Lee J, Azamfar M, Singh J (2019) A blockchain enabled cyber-physical system architecture for industry 4.0 manufacturing systems. Manuf Lett 20:34–39
28. Lee JY, Woo JS, Rhee SW (1998) A transformed quantile-quantile plot for normal and bimodal distributions. J Inf Opti Sci 19(3):305–318
29. Leitão P, Colombo AW, Karnouskos S (2016) Industrial automation based on cyber-physical systems technologies: prototype implementations and challenges. Comput Ind 81:11–25
30. Lin CL, Chen JK, Ho HH (2021) Bim for smart hospital management during covid-19 using mcdm. Sustainability 13(11):6181
31. Liu J, Li Y, Chen M, Dong W, Jin D (2015) Software-defined internet of things for smart urban sensing. IEEE Commun Mag 53(9):55–63
32. Liu J, Zhang W, Ma T, Tang Z, Xie Y, Gui W, Niyoyita JP (2020) Toward security monitoring of industrial cyber-physical systems via hierarchically distributed intrusion detection. Expert Syst Appl 158:113578
33. Liu Y, Kuang Y, Xiao Y, Xu G (2017) Sdn-based data transfer security for internet of things. IEEE Internet Things J 5(1):257–268
34. Maurya AK, Tripathi D, Biswas A, Tripathi AK (2018) Design issues in distributed software. 2018 Fifth International Conference on Parallel. Distributed and grid Computing (PDGC), IEEE, 563–567
35. Moraitis G, Nikolopoulos D, Bouziotas D, Lykou A, Karavokiros G, Makropoulos C (2020) Quantifying failure for critical water infrastructures under cyber-physical threats. J Environ Eng 146(9):04020108
36. Mozafari SH, Meyer BH (2016) Efficient performance evaluation of multi-core simt processors with hot redundancy. IEEE Trans Emerg Top Comput 6(4):498–510
37. Pari SMA, Noormohammadpour M, Salehi MJ, Khalaj BH, Bagheri H, Katz M (2013) A self-organizing approach to malicious detection in leader-based mobile ad-hoc networks. In: 2013 IFIP wireless days (WD), IEEE, 1–3
38. Parsamehr R, Esfahani A, Mantas G, Radwan A, Mumtaz S, Rodriguez J, Martínez-Ortega JF (2019) A novel intrusion detection and prevention scheme for network coding-enabled mobile small cells. IEEE Trans Comput Soc Syst 6(6):1467–1477
39. Rahman MU (2019) Leader election in the internet of things: challenges and opportunities. arXiv preprint arXiv:191100759
40. Rrushi J, Farhangi H, Howey C, Carmichael K, Dabell J (2015) A quantitative evaluation of the target selection of havex ics malware plugin. In: Industrial control system security (ICSS) workshop
41. Şahin S, Gedik B (2018) C-stream: a co-routine-based elastic stream processing engine. ACM Trans Parallel Comput (TOPC) 4(3):1–27
42. Satam S, Satam P, Pacheco J, Hariri S (2021) Security framework for smart cyber infrastructure. Cluster Comput, 1–12
43. Sengan S, Subramaniyaswamy V, Nair SK, Indragandhi V, Manikandan J, Ravi L (2020) Enhancing cyber-physical systems with hybrid smart city cyber security architecture for secure public data-smart network. Future Gener Comput Syst 112:724–737
44. Singh P, Tripathi AK (2012) Exploring problems and solutions in estimating testing effort for non functional requirement. Int J Comput Technol 3(2b):284–290
45. Stroustrup B, Shopiro JE (1984) A set of C++ classes for co-routine style programming. AT & T bell laboratories

46. Tao M, Zuo J, Liu Z, Castiglione A, Palmieri F (2018) Multi-layer cloud architectural model and ontology-based security service framework for iot-based smart homes. Future Gener Comput Syst 78:1040–1051
47. Tripathi D, Maurya AK, Chaturvedi A, Tripathi AK (2019) A study of security modeling techniques for smart systems. 2019 International Conference on Machine Learning. Big Data, Cloud and Parallel Computing (COMITCon), IEEE, 87–92
48. Tripathi D, Singh LK, Tripathi AK, Chaturvedi A (2021) Model based security verification of cyber-physical system based on petrinet: a case study of nuclear power plant. Annals Nucl Energy 159:108306
49. Tripathi D, Tripathi AK, Singh LK, Chaturvedi A (2021b) Towards analyzing the impact of intrusion prevention and response on cyber-physical system availability: a case study of npp. Annals of Nucl Energy p 108863
50. Vandana C (2016) Security improvement in iot based on software defined networking (sdn). Int J Sci, Eng Technol Res (IJSETR) 5(1):2327–4662
51. Walker-Roberts S, Hammoudeh M, Aldabbas O, Aydin M, Dehghantanha A (2020) Threats on the horizon: understanding security threats in the era of cyber-physical systems. J Supercomput 76(4):2643–2664
52. Wu Z, Li R, Zhou Z, Guo J, Jiang J, Su X (2020) A user sensitive subject protection approach for book search service. J Assoc Inf Sci Technol 71(2):183–195
53. Wu Z, Shen S, Lian X, Su X, Chen E (2020) A dummy-based user privacy protection approach for text information retrieval. Knowl-Based Syst 195:105679
54. Wu Z, Li G, Shen S, Lian X, Chen E, Xu G (2021) Constructing dummy query sequences to protect location privacy and query privacy in location-based services. World Wide Web 24(1):25–49
55. Yaacoub JPA, Salman O, Noura HN, Kaaniche N, Chehab A, Malli M (2020) Cyber-physical systems security: limitations, issues and future trends. Microprocess Microsyst 77:103201
56. Zhang Y, Wang L, Sun W, Green RC II, Alam M (2011) Distributed intrusion detection system in a multi-layer network architecture of smart grids. IEEE Trans Smart Grid 2(4):796–808
57. Zhu Q, Rieger C, Başar T (2011) A hierarchical security architecture for cyber-physical systems. In: 2011 4th international symposium on resilient control systems, IEEE, 15–20