Chapter 5

An Integrated Approach to Design Functionality with Security for Cyber-Physical Systems

Generally, a CPS is a kind of distributed system with safety-critical functionalities. Designing a secure, maintainable, and performance-efficient large-scale CPS is challenging. 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 [113] [13]. 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 and 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.

Hence, in this chapter, we propose a multi-tier distributed architectural model of CPS to integrate and organize the functionality and security of a large-scale CPS. The idea is similar to aspect-orientation [69] 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. Hence, we propose a fresh fault-tolerant leader election algorithm to elect the functionality and security leaders for CPS. 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. The proposed architectural model is evaluated 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 chapter is organized as follows. Section 5.1 presents the attack scenarios. Section 5.2 proposes the multi-tier architectural model and a pre-selected leader election algorithm for electing the functionality and security leaders. Section 5.3 presents the performance evaluation of the proposed architectural model and a case study on a smart healthcare system. Section 5.4 presents the summary.

5.1 Attack scenarios

In general CPS architectures [78, 44, 61, 63] the security vulnerabilities may exist at any of the architecture layers. As a result, security concerns are different at different layers. Different attacks like tampering, 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

5.2 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.

5.2.1 Formal description

The proposed architectural model consists of four layers where security is added as a cross-cutting concern as shown in FIGURE 5.1. This architectural model is designed and viewed as a distributed system with heterogeneous nodes as presented in FIGURE 5.2. Formally, the proposed CPS architecture is defined as a set of nodes (SN) 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 domains. 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 FIGURE 5.3.

5.2.2 Layers responsibilities

In the proposed architectural model (FIGURE 5.2), different computational responsibilities of the total work of automation, instrumentation, control and security are distributed. These responsibilities are performed by different types of homogeneous or heterogeneous nodes at different layers for different purposes, including sensing,

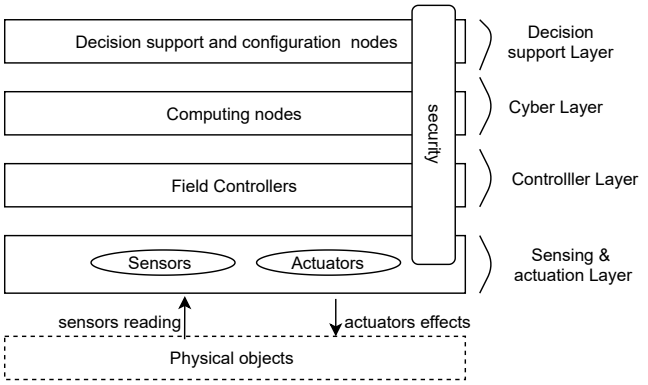

FIGURE 5.1: Layered representation of CPS architecture

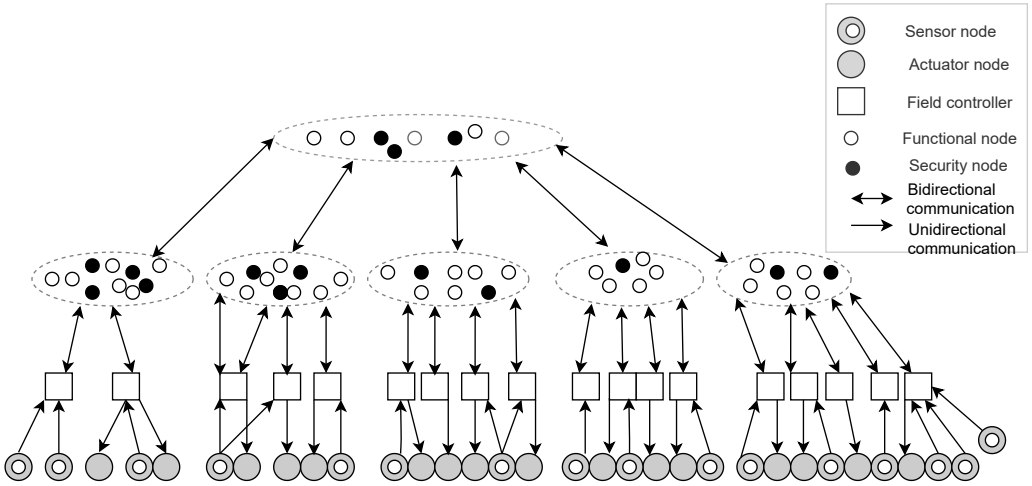

FIGURE 5.2: Clustered view of the proposed distributed CPS architectural model

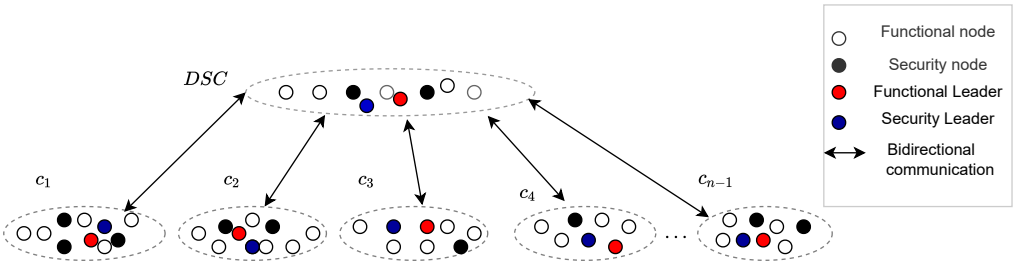

FIGURE 5.3: Clustered view of cyber layer and decision support layer of the proposed distributed CPS architectural model with functionality and security leaders

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.

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.

Controller layer consists of multiple programmable field controllers and shown as boxes. 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.

Both sensor and 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.

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 and 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.

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.

5.2.3 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. These 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, the 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 [21], and key management, including key generation, distribution, and storage. Moreover, these nodes monitor, detect and respond to the malicious events or abnormal behavior of functionality nodes [111] 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, the security and functionality leaders 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 as 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 in executing 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.

5.2.4 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 section 5.2.1, divided into m clusters (sub graphs) such that $c_1, c_2, \cdots, c_m \subseteq G$. Each cluster c_l has a unique id. d_l and rd_l are the diameter and radius of c_l are defines as $\lceil \frac{d_l}{2} \rceil \leq rd_l \leq d_l$. $\forall l, d_l < D$. 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 due to DoS attacks.

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 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 the transient leaders realizes the leader is failed, it communicates to the transient leaders to inform the leader's failure but can not 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 the equation (5.1)

$$Rk_i = \sum_{q=1}^{\lambda} \xi_{iq} \tag{5.1}$$

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. It is to be noted that while applying this model to any specific case, the benefit and cost attributes may be decided as required for the application. The measurement unit of the different attributes can be different, so we use the max-min normalization to normalize the attributes.

5.2.4.1 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 been failed. It is used to inform the transient leaders about the current leader's failure.

5.2.4.2 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

Algorithm 1: Chief Leader Election

`// When a node` i `initiates an election.`
$einit\_id \leftarrow node\_id_i$, $s\_em\_id \leftarrow node\_id_i$, $parent_i \leftarrow None$
Set the value of toe according to the type of the election.
Create an $em\langle einit\_id, s\_em\_id, toe\rangle$ message and send it to all the adjacent nodes.
`// When a node` j `gets a` $em\langle einit\_id, s\_em\_id, toe\rangle$ `message.`
if *(the received message is a* $em\langle einit\_id, s\_em\_id, toe\rangle$*)* **then**
  if *(*$(parent_j == Empty) \vee (einit\_id > parent_j)$*)* **then**
    $parent_j \leftarrow s\_em\_id$, $c\_ack\_id \leftarrow node\_id_j$, $s\_em\_id \leftarrow node\_id_j$
    Create an $ack\langle c\_ack\_id, einit\_id\rangle$ message and send it to the parent node.
    Send the $em\langle einit\_id, s\_em\_id, toe\rangle$ 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 $rank\langle r\_list\rangle$ message
    end
  else
    Discard the received message.
  end
end
`// When a node` j `gets a` $ack\langle c\_ack\_id, einit\_id\rangle$ `message.`
if *(the received message is a* $ack\langle c\_ack\_id, einit\_id\rangle$*)* **then**
  Insert the c_ack_id into the l_child_j.
end
`// When a node` j `gets` $rank\langle r\_list\rangle$ `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 $rank\langle r\_list\rangle$ message.
`// When the election initiating node that conducts the whole election (here node` i`) gets` $rank\langle r\_list\rangle$ `messages from its all child nodes.`
Node i arranges all the received node Ids (through $rank\langle r\_list\rangle$ 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 .
  $l\_id \leftarrow$ *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.
  $fun\_leader_i \leftarrow l\_id$, $flc\_list_i \leftarrow t\_list$
else
  Choose the best node as the chief security leader .
  $l\_id \leftarrow$ *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.
  $sec\_leader_i \leftarrow l\_id$, $slc\_list_i \leftarrow t\_list$
end
Create the $ld\langle l\_id, t\_list, toe\rangle$ message and sends it to all the adjacent nodes.
`// When node` j `gets` $ld\langle l\_id, t\_list, toe\rangle$ `message.`
if *(*$toe == 1$*)* **then**
  $fun\_leader_j \leftarrow l\_id, flc\_list_j \leftarrow t\_list$
else
  $sec\_leader_j \leftarrow l\_id, slc\_list_j \leftarrow t\_list$
end
Send the $ld\langle l\_id, t\_list, toe\rangle$ message to all the adjacent nodes except its sender.
If a node j gets the $ld\langle l\_id, t\_list, toe\rangle$ message multiple times, it processes the first received $ld\langle l\_id, t\_list, toe\rangle$ 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.
1  if (node i realizes that a chief leader is failed) then
2  |  create and send a lfmsg⟨failed_leader_id, tol⟩ message to all the nodes in lc_list
3  end
   // When every node of the lc_list gets a lfmsg⟨failed_leader_id, tol⟩
4  Every node in lc_list checks whether the leader (failed_leader_id ) is failed.
5  if (The leader (failed_leader_id ) is failed) then
6  |  if (|lc_list| > r/2) then
7  |  |  if (tol == 1) then
8  |  |  |  Select the best node from the flc_list and put the rest nodes in the t_list.
9  |  |  |  Declare the selected best node as the functionality leader by broadcasting a leader declaration
   |  |  |    message
10 |  |  else
11 |  |  |  Select the best node from the slc_list and put the rest nodes in the t_list.
12 |  |  |  Declare the selected best node as the security leader by broadcasting a leader declaration
   |  |  |    message
13 |  |  end
14 |  else
15 |  |  Invoke algorithm 1
16 |  end
17 end
```

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 is 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 its 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 wait 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 send 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 checks value of toe, if $toe = 1$ it updates the chief leader id as functionality chief leader and transient leaders list as the functionality transient leaders. If $toe = 0$, it updates the chief leader id as a chief security leader and transient leaders list as the 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 $lfmsg\langle failed\_leader\_id,\ tol\rangle$ 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 $ld\langle l\_id, t\_list\rangle$ 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.

5.2.4.3 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 $O(D)$.

5.2.5 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 [68] [86], identifies the non-responsiveness of attached sensor and actuator nodes using heart beat message [52], 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 [52] [68] [86].

At cyber and decision support layers, the functionality and security leader collaborate as co-routines [130] [116] 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 literature [66] [51] [101]. After confirming the suspected behaviour 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 communicate 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 section 5.2.4. 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.

5.3 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.

5.3.1 Case study

A smart hospital is a concept that uses emerging technologies of information and communications technology (ICT) to optimize and manage the healthcare operations and its functional requirements efficiently [83]. 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 FIGURE 5.4. 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 [30].

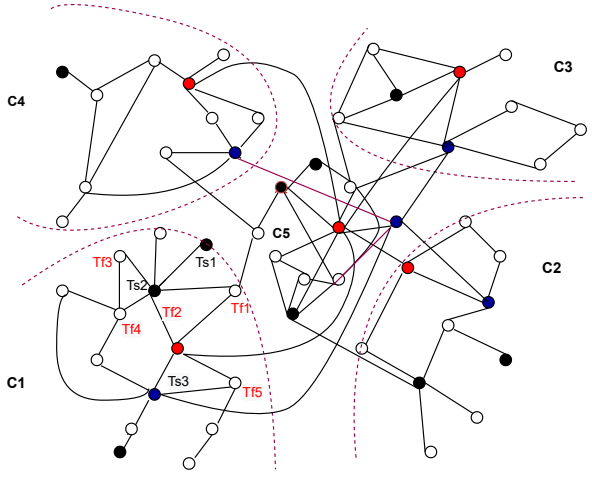

FIGURE 5.4: Cluster arrangement of a distributed hospital network with functionality and security leaders

The decision support layer represented as $c5$ (as shown in FIGURE 5.4) 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.

Suppose, in cluster $c1$, a set of functionality tasks $Tf = \{Tf1, Tf2, Tf3, Tf4, Tf5\}$ 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, security monitoring node, 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.

5.3.2 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.

TABLE 5.1: 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

(1) Purely centralized: In purely centralized system management mode, 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) get 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 5.1, and the simulation results are shown in FIGURE 5.5 and FIGURE 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).

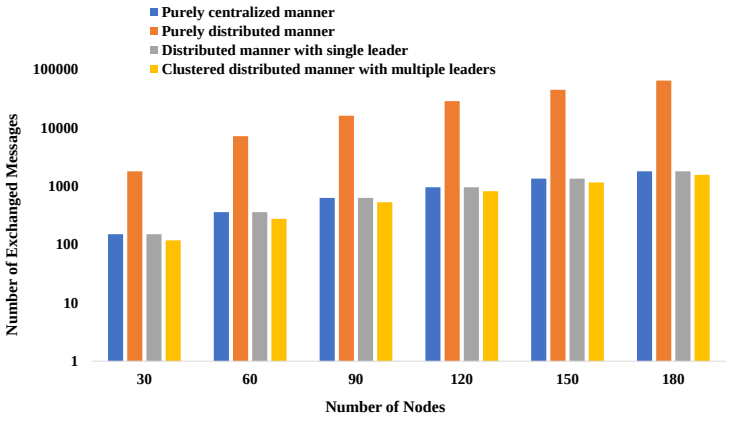

FIGURE 5.5: Comparison of the proposed system management manner with other possible system management manners based on the number of exchanged messages to complete the task.

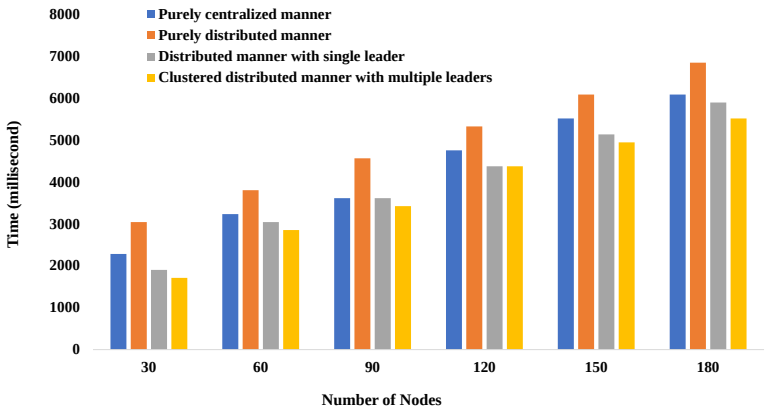

FIGURE 5.6: Comparison of the proposed system management manner with other possible system management manners based on the time required to complete the task.

In FIGURE 5.5, X-axis represents the number of nodes, and Y-axis represents the number of messages exchanged to complete the task. In FIGURE 5.6, X-axis represents the number of nodes, and Y-axis represents the time required to complete the task. FIGURE 5.5 and FIGURE 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 5.7 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 FIGURE 5.7, 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 5.7 concludes that if we manage the CPS in a distributed clustered manner with multiple leaders, the average response time of tasks get reduced. FIGURE 5.8 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 FIGURE 5.8, 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 these tasks, where real-time tasks have priority over non-real-time tasks. FIGURE 5.8 concludes that success rate of real-time tasks get increased if we manage the CPS in a clustered distributed manner with multiple leaders. From the simulation results shown in FIGURE 5.7 and 5.8, 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 [79] and the Shapiro-Wilk test [54] 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 FIGURE 5.9 and FIGURE 5.10, respectively. The Q-Q plots show that the average response time 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.

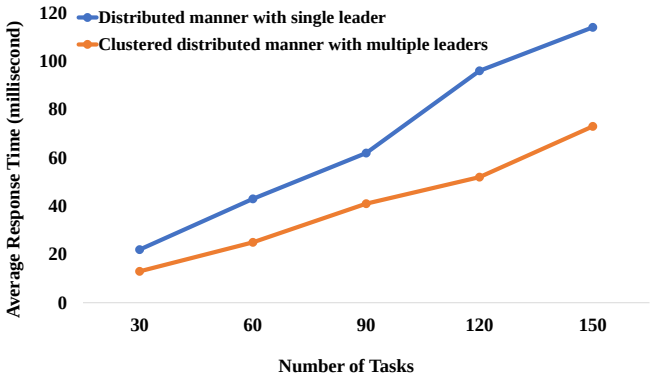

FIGURE 5.7: Comparison of the proposed system management manner with the distributed manner with a single leader based on the average response time of the task.

Moreover, the proposed distributed clustered manner with multiple leaders is more

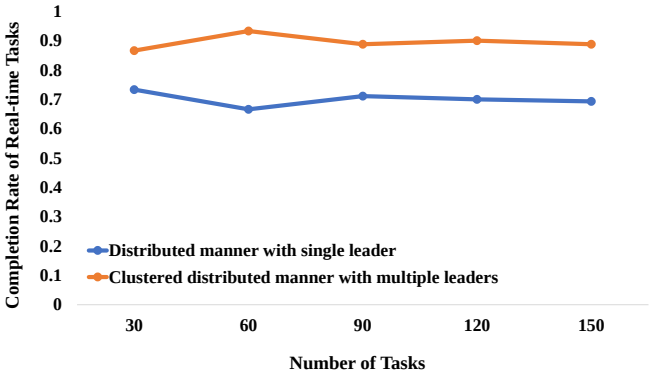

FIGURE 5.8: Comparison of the proposed system management manner with the distributed manner with a single leader based on the success ratio of real time tasks completion within deadline

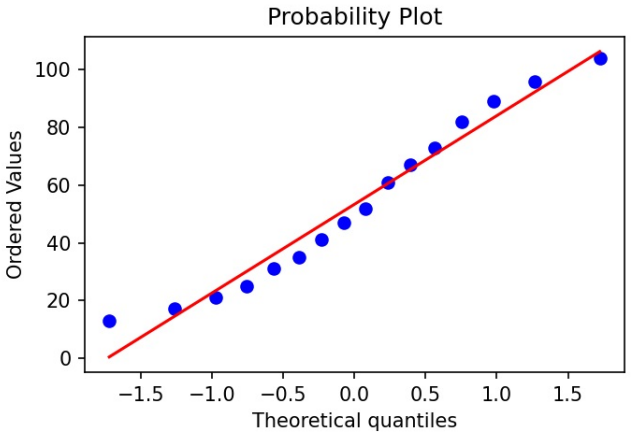

FIGURE 5.9: Quantile-Quantile plot (Q-Q plot) on the average response time of the tasks getting through the proposed system management manner.

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.

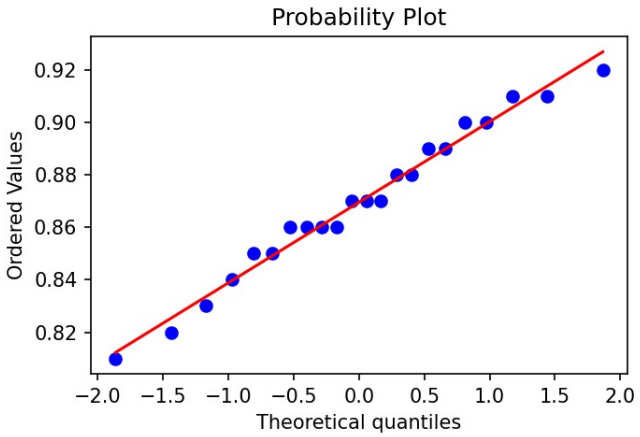

FIGURE 5.10: Quantile-Quantile plot (Q-Q plot) on the completion ratio of real-time tasks getting through the proposed system management manner.

5.4 Summary

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 CPS. Management of functional concerns and security concerns separately improve the maintainability and 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,

security 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.