Noname manuscript No.
(will be inserted by the editor)

LDES: Detector Design for Version Number Attack Detection using Linear Temporal Logic based on Discrete Event System

Abhay Deep Seth · Santosh Biswas · Amit Kumar Dhar

Received: date / Accepted: date

Abstract The Internet Engineering Task Force (IETF) has defined routing protocols for Low Power and Lossy Networks (RPL) for constrained devices. RPL constructs DODAGs (Destination Oriented Directed Acyclic Graphs), to optimize routing. RPL ensures acyclic topology with the DODAG version number. However, the control message's DODAG version number is not authenticated. So, RPL is vulnerable to topological inconsistency attack known as DODAG Version Number (DVN) attack. DVN attack creates a packet delay, packet loss, cyclic topology, etc., in the network. This paper proposes a method for detecting DODAG version number attacks. Several existing schemes to defend against the DVN, such as cryptographic techniques, trust-based, threshold-based and mitigation are computationally intensive or require protocol modification. DVN does not change the packet format or sequence of packets, but can still perform attacks and hence fall under the category of stealthy attacks, which are difficult to detect using traditional Intrusion Detection System′s (IDS). Discrete-Event System (DES) based IDS have been applied in the literature for stealthy attacks that achieve low overhead, low false alarm rate, etc. However, the construction of DES-based IDS for network protocol may lead to errors, as modelling is manual. The resulting IDS, therefore, is unable to guarantee its correctness. This paper proposes Linear Temporal Logic (LTL) based DES paradigm to detect DVN. LTL-based paradigm facilitates formal verification of the DES-based IDS, and hence the correctness of the scheme is ascertained. The proposed technique is simulated using the Contiki cooja simulator. When the percentage of spiteful nodes in the network increases, the true positive rate, and packet delivery rate drops, while the false positive rate and control message overhead increase. The memory requirement for sending the packets and verifying the nodes is minimal. The LTL-based IDS has been formally verified using NuSMV to ensure the correctness of the framework.

Keywords Routing Protocol for Low Power and Lossy Networks (RPL) · Internet Engineering Task Force (IETF) · DODAG Version Number Attack (DVN) · Destination Oriented Directed Acyclic Graphs (DODAG) ·

Abhay Deep Seth
Indian Institute of Technology Bhilai
GEC Campus, Sejbahar, Raipur, Chhattisgarh
E-mail: abhays@iitbhilai.ac.in

Santosh Biswas
Indian Institute of Technology Bhilai
GEC Campus, Sejbahar, Raipur, Chhattisgarh
E-mail: santosh@iitbhilai.ac.in

Amit Kumar Dhar
Indian Institute of Technology Bhilai
GEC Campus, Sejbahar, Raipur, Chhattisgarh
E-mail: amitkdhar@iitbhilai.ac.in

1 Introduction

The Internet of Things (IoT) has been overgrowing. It connects a vast network of machines or objects over the Internet. These interconnected machines or objects such as sensors, actuators may consist of resource-constrained devices [1,2].

Various applications like smart hospitals, smart industrial monitoring, smart agriculture, smart irrigation, smart fighting, use IoT devices. In the IoT framework, a wide range of constrained devices with minimal memory, power, and processing capacities construct a Low Power and Lossy Networks (LLNs) [3]. LLNs have lossy links and low throughput. Therefore, LLN requires a routing protocol that can fulfill the need of LLN devices. However, there are already many existing routing protocols like OSPF (Open Shortest Path First) Version 2, RIP (Routing Information First) Version 2, DYMO (Dynamic Mobile ad hoc network On-demand routing), that are designed for other networks like

wireless, ad-hoc, MANET. Many researchers have applied these routing protocols for LLN devices, but such protocols lack the routing criteria (such as routing state, lossy response, control cost, link cost, node cost) required for LLN devices.

Thus, Routing Protocol for LLNs (RPL) [4–6] has been designed that passes all criteria associated with LLNs. RPL based network is open to various security attacks because of their limited potential and open and unguarded distribution. Major security attacks against RPL include internal attacks, e.g., Rank attack, DIS (DODAG Information Solicitation) flooding attack, topological inconsistency attack, and external attacks like DoS (Denial-of-Service), hijacking of IoT device, password-based attacks. A variety of security mechanisms have been taken into consideration for RPL [7]. All these mechanisms can secure or guard the network against external attacks [8,9,12], but it is also vital to secure the network from internal attacks [3,10,11,13,14,16].

This paper focuses on a topological inconsistency attack is referred to as DODAG Version Number attack. This attack falls into the category of stealthy attacks that are difficult to be detected by standard IDS. A spiteful node alters the version number by illegally incrementing the version number value in the respective field of the DIO control message format. Once the new increased version number of the DIO message format is received, this modification forces the rebuild of the new DODAG tree. The parent node sends DIO messages to their child nodes. In this way, the DIO messages are exchanged to build the new DODAG. The new DODAG formation can cause loss of reserved energy, unavailability of channels, and even form loops in the routing topology. There is no technique available that can assure the integrity of the DIO field, i.e., version number. Various trust-based [15], cryptography-based approaches (such as key exchange, signature-based approaches, hashing) have been proposed to ensure the security of version numbers in RPL networks. However, these approaches deliver the protocol as heavy-weight for LLN devices. Only a few strategies target to detect version number attacks but require protocol modification and hardware upgradation. [17,11]. Traditional signature and anomaly IDS [18,19,9] show high false alarms as stealthy attacks like DODAG version number attacks do not change the syntax of a network packet or result in a large statistical change of the network parameters. Therefore, in this paper, we apply the Discrete Event System (DES) based IDS technique to detect DODAG Version Number attack, which does not require a change in hardware or protocol and is low weight, yet able to handle stealthy attacks.

DES is a discrete automata model with event-driven dynamics [20–22] that has been used for the development of the IDS [23,8], to resolve the issues of change in protocol, augmentation in hardware, etc. The DES model is built for both normal and attack (referred to as failure) conditions on the network. The state estimator, also known as the DES detector, is designed, that is, the IDS. The detector evaluates whether the system traverses through the normal or failure condition-based DES model based on events created (triggered) by the system.

The Discrete Event System framework is automata-based because the specification is modeled for the network protocol as an automaton. The DES-based IDS has shown impressive results like low overheads, low false alarm rate, no protocol change, and also at the same time are effective against stealthy attacks. In DES-based IDS, the behavior of the network protocol under normal and failure conditions are described in terms of ordinary language, i.e., specification. The manual translation of common (natural) language into an automata-based DES for the network protocol is tiresome, so that it may lead to errors. As a result, proving the IDS's correctness becomes a difficult task. Therefore, the Linear Temporal Logic (LTL) based DES (LDES) is used to resolve the problems related to the correctness of the model in automata-based DES.

The LDES framework [24,20] is a user-friendly environment as it can capture the standard language specifications. This paper focuses on the LDES to construct an IDS that detects DODAG Version Number attacks.

The LDES IDS has the following features:

- Transformation of common language specification of DIO protocol to models. LDES renders modeling less vulnerable to errors.
- At any stage of IDS development, model checking [25–27] is performed to verify the correctness, i.e., from modeling to detector design.

The remaining paper is organized as follows. Section 2 presents the operation of the RPL protocol along with the version number attack. Section 3 consists of related work for the detection of version number attacks. The proposed scheme is discussed in Section 4. It gives a detailed working principle of the proposed IDS. It consists of an LTL-based DES framework and concerns constructing a DES detector for a version number attack. Section 5 contains the performance evaluation of various simulation scenarios and verifies the DES-based IDS using the NuSMV tool. The concluding remarks are given in Section 6.

2 Routing Protocol Low Power and Lossy Network (RPL)

RPL is a routing protocol for Low Power and Lossy Network, developed as a distance-vector routing protocol. The RPL protocol is primarily for IoT devices with limited capabilities and wireless sensors. The devices (nodes) use RPL to construct a Destination Oriented Acyclic Graph (DODAG)

[28] for communication. The DODAG consists of three different kinds of nodes: parent (in-between nodes), child (leaf nodes), and root (default gateway), as illustrated in Fig. 1. Each device in a DODAG tree has a rank that helps to reduce upward traffic flow while increasing downward traffic flow. In DODAG, the root node has rank zero, and the parent node′s rank is always lesser than its child to avoid loop creation. The RPL supports three kinds of traffic schemes. In P2P, the communication between two LLN devices occurs, whereas, in P2M, the traffic flows from a root node to LLN devices. In M2P, communication from LLN device(s) to root occurs.

The RPL consists of multiple DODAG networks, where the number of distinct wireless sensor nodes are linked to a DODAG′s gateway node, i.e., root [11]. Each DODAG in the network can be distinguished by some fields - DODAG version number, rank values, DODAG ID, and instance ID. To maintain and establish the DODAG routing, RPL uses ICMPv6 control messages: DODAG information object (DIO), DODAG Advertisement Object (DAO), DODAG Advertisement Object Acknowledgement (DAO-ACK), and DODAG Information Solicitation (DIS). DIO advertises the information to maintain and build the DODAG. The DODAG root begins the construction of DODAG towards the upward route by broadcasting DIO frames. The nodes select the sender as a parent on receiving the DIO frame, and the receiving node notifies the updated data to neighboring nodes in the next DIO. All nodes has an upward route towards the root on completing the DODAG construction process. A rank value is assigned to each node in DODAG with respect to the root node, which denotes the location of a node. A node selects the preferred parent from the parent list, which acts as a default gateway. A node selects the preferred parent to forward a packet towards the root. Upon unsuccessful transmission, the node selects any non-preferred parent to forward the packet, one after the other.

Fig. 1 depicts the DODAG construction using RPL. The root node (node A) delivers the DIO frame consisting of the DODAG′s data. Node C, B joins DODAG upon receipt of the DIO frame and responds to node A with the DAO frame. Then, node B forwards a DIO frame containing the latest DODAG data. On receipt of the DIO frame from node B, node D joins the DODAG. After joining DODAG, node D responds to node B with a DAO packet. Node B receives a DIS request response from node E, as no node has joined the DODAG. After node B joined the DODAG, node B sends the DIO frame to node E to join the DODAG. Now, after joining DODAG, node E responds with a DAO frame to node B. After receiving the frames, node B will sum up all the data and forward the DAO frame to its selected parent node A. Finally, node A attains all data about the nodes from their DAO frames to construct a downward route. Similarly, the remaining nodes will join the DODAG.

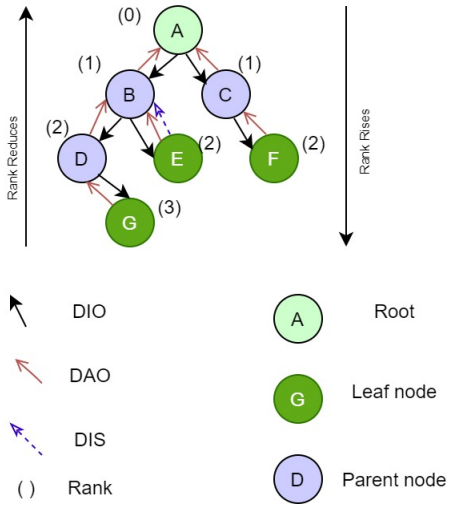
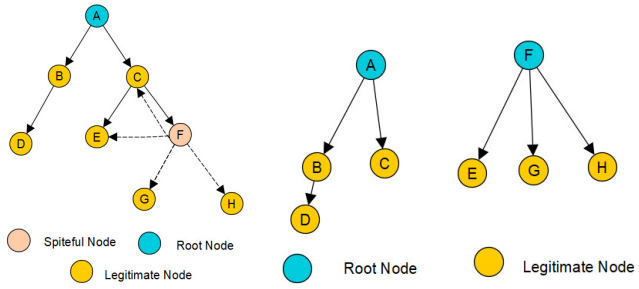


Fig. 1 DODAG Construction using RPL

In RPL, the trickle algorithm transmits the DIO periodically. A rank rule is used to avoid loop creation where a parent node always has a rank lesser than its child node. DODAG loops may occur when DODAG is cyclic. When inconsistencies arise in the DODAG (e.g., due to less power or lossy communication condition, nodes vanish from the network), RPL activates repair mechanisms. Two different repair mechanisms are available: (i) Global repair and (ii) Local repair. Whenever the preferred parent is unavailable, a local repair is triggered. RPL uses a local repair mechanism to find an alternate path to route the packets. This mechanism fails if multiple inconsistencies occur and activate the global repair mechanism. A global repair mechanism increments the DODAG version number to rebuild the whole DODAG tree. The DIO control message consists of the version number. A node checks its current version number with the new version number of the received DIO from its parent nodes. If the latest version number value is large, a node ignores its current rank value, resets the clock, and rebuilds a new DODAG. A global repair procedure in a new DODAG assures an acyclic topology.

The root node in the network uses a version number to manage the global repair process. The version number assures that all nodes in the DODAG are present with the routing state. The RPL routing attacks are categorized into three types, i.e., traffic, topology, and resources. In this paper, the version number attack has been considered. The Version Number Attack falls into the category of topological inconsistency RPL attack. In this attack, spiteful node illegally increments the version number field value in the DIO frame format [29]. A spiteful node forwards the DIO frame to all of its neighbors. When the neighboring nodes receive the DIO frame from the spiteful node with an incremented version number value, they form the latest DODAG tree. The latest DODAG tree formation creates a loop in the topology,



(a) DODAG Version Number Attack (b) Attacker Node becomes Root node

Fig. 2 Impact of DODAG Version Number Attack

resulting in a waste of node energy, increased overhead, and many more.

Fig. 2(a) shows the impact of DODAG Version Number Attack. The network consists of 8 nodes where there is one DODAG root (node A), one spiteful node (node F), and the remaining are legitimate nodes. In DODAG, only node A is allowed to update the Version number parameter in DIO message format. Node F joins DODAG as a legitimate node like other nodes, but it becomes spiteful after the DODAG becomes stable. Now, node F sends DIO (dotted arrows) messages to its neighboring nodes, i.e., Node C, E, H, and G (assuming in the transmission range). This DIO message consists of the modified version number. Node C rejects the received DIO message, as it is from the child node. Nodes E, H, and G also receive the DIO message with the updated version number. Upon receiving DIO, Nodes E, H and G update their existing version number and propagate it in the DODAG. It will result in new DODAG tree formation and breaks the single tree into two DODAG's as shown in Fig. 2(b). The new DODAG formation is one of the implications of DODAG version number attack, where Node F becomes root node (as shown in Fig. 2(b)) from the spiteful node (as seen in Fig. 2(a))

3 Related Work

Section 3.1 describes the study of the detection strategies for the DODAG version number attack. As already discussed, for stealthy attacks, e.g., the DODAG Version Number attack, DES-based techniques are implemented. Section 3.2 comprises the background of DES-based IDS techniques and their drawbacks.

3.1 DODAG Version Number RPL Attack Detection Techniques

The technique presented in SVELTE [30] consists of three modules for detecting routing attacks in a 6LoWPAN network. The first module collects the data of the IPv6 at the edge router, and the second module determines the invasion in the traffic. The third module prevents unwanted traffic from entering the 6LoWPAN network. The main issue of the scheme is that it shows high false positive rates.

Authors in [31] present a solution for attacks on VeRA-version number and rank authentication in RPL. This cryptography-based solution secures rank and version number fields using a hash chain in the DIO control message. However, the authors have neither evaluated network performance parameters nor discussed the paper's implementation.

Ahmet Arõsü et al. [17] have proposed mitigation techniques for the Version Number Attack. The technique eliminates the Version Number coming from the child nodes and allows the node to change its Version Number only when most neighbors with better ranks claim a Version Number update. However, the multiple Version Number attack condition has not been considered in this work.

Firoz Ahmed et al. [11] have proposed a technique for the detection of the version number attack using a cooperative verification technique. In this scheme, the node exchanges verification messages among the two-hop neighbors and collects the version number information from the two-hop neighbors to identify the attacker. The issue with this scheme is that every node should know the address of every two-hop neighboring node. Also, this scheme itself can become the source of the DoS attack by repeatedly sending the verification request messages to misuse the neighbors' resources.

The detection, as mentioned above, techniques are either the cryptographic schemes (involving resource overheads) or need protocol modification, software updation, etc.

Some attacks will require no change in protocol or header format and neither lead to any significant statistical deviation in the network parameters but can still exploit the vulnerabilities. These attacks are known as stealthy attacks, and DODAG Version Number attack falls in this category. For stealthy attacks, the DES-based IDS technique has been proposed in the literature that has shown impressive results in non-requirement of protocol modification, low resource overheads, low false alarm rate, etc.

In the following subsection, we discuss the DES-based IDS schemes and issues therein.

3.2 DES based IDS techniques

N. Hubballi et al. [32] proposed a LAN attack detection technique using Discrete Event Systems. ARP spoofing is a stealthy attack where signature and anomaly-based IDS have a high false alarm rate. In this work, the IDS detector is designed to detect ARP request and response spoofing using the probing scheme. The scheme has salient features

like no protocol change, low false alarm rate, etc. This technique only detects ARP attacks based on the IP-MAC pair conflicts.

In [21], the authors have designed the DES-based IDS detector for the de-authentication attack. The victim's access point gets disconnected from the network due to the de-authentication frame in the network. As for the IDS of ARP, in this case, also the DES-based IDS for de-authentication requires neither protocol change nor firmware upgradation.

Authors in [33] proposed an event-based detection approach to recognize impersonated IP packets. This approach uses proactive authenticity testing of the obtained packets. The active probing scheme uses discrepancies in the Time-to-Live values of the received packets to verify whether the earlier packet was impersonating or not. This method helps to identify impersonated IP packets with the help of the DES-based detector.

The critical problem with the DES-based system is that it begins with designing the model from the English language specification of the protocol under normal and attack conditions [20]. The state-transition model is built manually, which may lead to an inaccurate design. It is observed that inaccurate design can have a significant impact on the entire IDS.

In [24], Jiang and Kumar discussed the failure diagnosis problem for DES with LTL specifications. In the temporal logic setting, the diagnosability of DES is established. The issue of diagnostic testing is reduced to that of verification of the model.

The LDES framework has the facility of verification that is missing in the automata-based DES schema. So, the LDES schema is adopted for the detection of the Version Number attack in this paper.

The benefits of the LDES framework over automata-based DES are as follows.

- LTL schema offers a representation of the user-friendly common language specifications as compared to the automata-based DES [34].
- The proof for the correctness of the IDS can be done easily because the automated method, i.e., model checking, can be used for the verification of the LTL specification as compared to the detector of the state-based DES.

The main contribution of the proposed technique for version number attack detection is described below:

- To detect DODAG Version Number Attack, LTL-based DES framework IDS is proposed.
- The IDS designed for detecting DODAG Version Number attack does not require protocol modification can work with low resource overhead etc.
- The LTL-based DES IDS can be verified to check the correctness.

4 Proposed Technique LTL based DES IDS for Version Number Attack

This section provides an outline of the proposed technique to detect version number attacks as discussed in Section 2. The network under consideration comprises devices with limited power, energy, and processing. The network is assumed to be dense, and the root node is never compromised. After the formation of the DODAG tree, the spiteful nodes show unexpected (malicious) behavior.

4.1 Working Principle of the IDS

In the proposed technique, whenever a node accepts the DIO frame from adjoining nodes, a comparison of the DODAG version number (VNN) in the DIO message with the existing version number (VNO) of the accepting node is performed, as illustrated in Fig. 3. The node also checks whether the VNN is greater than VNO or not. If the VNN is greater than VNO, the receiving node(RN) sends the VNN and DIO Sender Node Address (DIO message sender abbreviates as DSNA) to the IDS node (located near root node); otherwise, RN ignores the received DIO message. Now the IDS is invoked to detect and verify the identity of neighboring node DSNA to determine whether the node is spiteful or not. For verification, the IDS node obtains the current DODAG version number with the help of Request (REQ) and Response (RSP) packets.

After joining the network, even if a spiteful node transmits DIO's without modifying the version number or other spiteful activity, the network functions cautiously. All nodes can observe the transmission behavior of the neighboring nodes, which ensures the neighboring node's honesty. Each node notices the neighboring node's transmission behavior by examining the DODAG version number sent by its neighbors in the DIO control message. The honesty of a node depends on whether a node has sent the correct version number in DIO or not to its neighboring nodes.

After receiving DIO messages from its neighbors, a node starts examining the DODAG version number field of the DIO control frame. If DIO has high Version Number (VNN) value than receiving node's existing Version Number (VNO), the receiving node (RN) sends VNN and DSNA to the IDS. The IDS is located near the root node. Upon receipt of this information, the identification procedure in IDS will be initiated to identify whether the neighboring node is malicious or not. For the confirmation of the neighboring node, whether it is malicious or not, the IDS sends REQ = (*destination node address, IDS address, AskVersionNumber*) message to know the version number. The REQ message is sent to the root node and DSNA asking for the current version number. Both nodes send responses to the IDS on receipt of the REQ message. The RSP = (*IDS address, Target address, Node's*

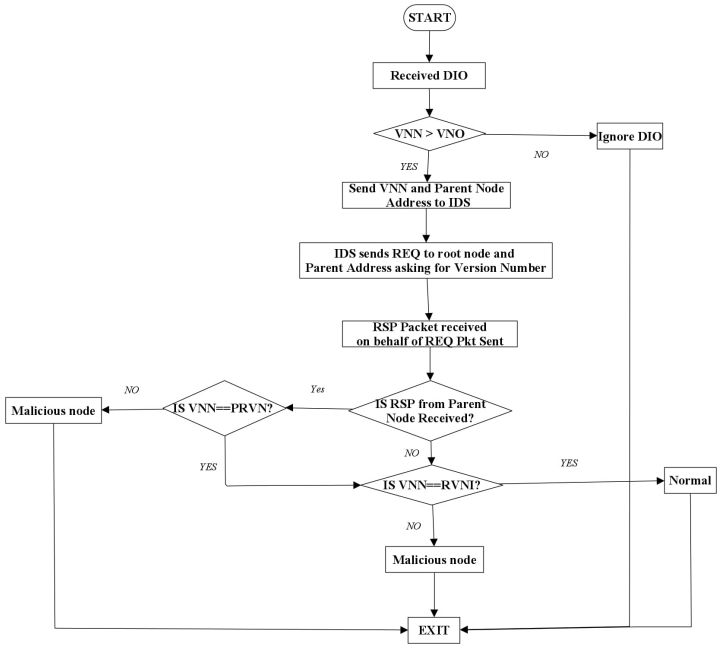


Fig. 3 Flow Diagram of IDS

latest Version Number) contains version number. Now, the IDS may receive version number from both the node's response messages, i.e., Root Node's version number(RVNI) and DSNA version number (PRVN). The IDS verifies whether the RSP message is received from DSNA or not. If the RSP from DSNA is received, IDS checks whether VNN is equivalent to PRVN or not. If the result is false, node (DSNA) is considered as a *Spiteful* node; otherwise, it compares VNN with RVNI. Even if the RSP message is not received at the IDS, it compares VNN with RVNI. If RVNI is equivalent to VNN, the node is considered as *Normal* node, otherwise as a *Spiteful* node. The confirmation node is validated by verifying the temporarily stored version number on the IDS with the version number received from the response messages. If the IDS receives two or more distinct version number values, it then considers the version number sent by the root node. We assumed that the root node is never compromised and always responds.

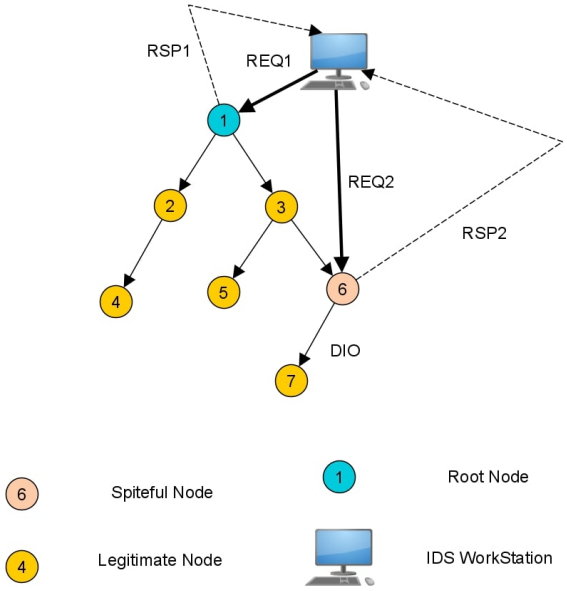


Fig. 4 Verification Procedure

Now, we demonstrate the working of the verification procedure through an example. Considering Fig. 4, node 7 receives the DIO control message with an increased version number from node 6. Now, node 7, instead of changing its existing version number with the new DIO version number (VNN), compares them to check if the new version is large than the old version number. Node 7 will send the new version number and node's 6 address to the IDS. After receiving the information sent by node 7, IDS sends the REQ1 packet to the root node and REQ2 to node 6, asking for the version number. After a while, IDS receives RSP1 and RSP2 from the root node and node 6, respectively. Finally, IDS compares the received version number in RSP1 and RSP2 with the VNN and decides whether node 6 is spiteful or normal.

Even if any spiteful node drops the REQ message or declines to forward the message due to communication breakdown, the scheme will work. It is so because other intermediate nodes have received the response messages other than the spiteful node. If any node does not send any response message, that node can be called spiteful. The response mes-

sage may not be transmitted due to link failure; in this condition, the node needs to send an error message to IDS.

4.2 Design of Detector using LTL-based schema

This section presents the steps to construct the detector with model variables using the LTL-based schema.

The tuples of the system model M_d for fault diagnosis (or attack detection)are defined as:

$$M_d = (S_i, S_{i0}, V, \Sigma, \tau, R, AP, L)$$

where,

- S_i is finite set of states.
- S_{i0} is set of initial states, $S_{i0} \subseteq S_i$
- V is the set of model variables.
- Σ is the set of events. Events can be observable or unobservable.
- τ is the set of transitions.
- $R \subseteq S_i \text{ X } \tau \text{ X } S_i$ is the transition relation.
- AP is the finite set of atomic propositions.
- L is a marking function $S_i \rightarrow 2^{AP}$ that marks each state with the set of atomic propositions which are valid at each state.

4.2.1 Steps for Diagnosability Test and Detector Design

The steps for the design of detector for M_d are as follows.

1. The first step consist of construction of FSA known as Buchi Automta, which shows the normal behaviour of the system obtained automatically through the LTL specification. The Buchi Automata accepts the input pattern if there exists a self-loop to atleast one of the final states. Buchi Automata consists of 5 tuples are defined as:

   $$B_f = (C_f, \Sigma_{AP}, R_f, q_0^f, F_i)$$

   where,
   - C_f is the set of states.
   - Σ_{AP} is the set of events
   - R_f is the transition relation, where $R_f \subseteq C_f \text{ X } \Sigma_{AP} \text{ X } C_f$.
   - q_0^f is the initial state.
   - $F_i \subseteq 2^{C_f}$ is the generalized Buchi automata.
2. This step tests the pre-diagnosability of the system model M_d. The pre-diagnosability test is performed to verify whether the fault occurrence can be observed within a finite time after its occurrence. The pre-diagnosability property will automatically hold if the LTL formula satisfies a safety property; it means either failure events are never triggered, or failure states are never visited. With this property, the infinite failure trace can be detected through the observations of the state traces. As a result, no detector can be constructed if a system cannot pass the pre-diagnosability test. The proposition synchronization of B_f and M_d constructs the automata T_1. T_1 is defined as follows:

   $$T_1 = (Q_1, \Sigma, \Im, R_1, Q_0^1, AP \cup F_i, L_1)$$

   - $Q_1 = C_f \text{ X } S_i$ is the set of states.
   - Σ is set of events.
   - $\Im$ is the set of transitions.
   - $R_1 \subseteq Q_1 \text{ X } \Im \text{ X } Q_1$ is the transition relation.
   - $Q_0^1 = \{(c_f, s) \in Q_1 \mid (q_0^f, L(s), c_f) \in R_f, s \in S_{i0} \}$ is the set of initial states.
   - $AP \cup F_i$ is the new set of proposition.
   - L_1 is the marking function.

   When every failure state has its own indicator, then the system model is called pre-diagnosable to a specification. So, if the states are visited infinitely often for every infinite proposition generated in T_1, then the system model is considered pre-diagnosable. The model T_1 should assure the LTL specification GFf_i where $f_i \in F_i$.
3. The diagnosability of M_d is tested in this step. The failure diagnosis approach captures both safety and liveness failures. The system model tests the diagnosability to the specification formula after completing the pre-diagnosability test. The fault diagnosis shows the detection and identification of the faulty states in the model. The diagnosis means either the fault has already occurred or predicts it will occur (liveness property). A detector can be constructed only after the system M_d is diagnosable. The construction of model for T_2 consist of 5-tuples as follows:

   $$T_2 = (Q_2, \Delta, \Im_m, R_2, Q_0^2),$$

   where,
   - $Q_2 = Q_0^1 \cup \{ q \in Q_1 \mid \exists (q', t, q) \in R_1 \}$ is the set of states.
   - Δ is the set of observable events.
   - $\Im_m$ denotes the set of observable equivalent transitions of $\Im$.
   - $R_2 \subseteq Q_2 \text{ X } \Im_m \text{ X } Q_2$ is the transition relation.
   - Q_0^2 is the set of initial states.

   Now design,

   $$T_2' = (Q_2, \Sigma, \Im, R_2', Q_0^2), \text{ where}$$

   $M^{-1}(lang(T_2)) = M^{-1}M(lang(T_1))$ are equivalent.
   Therefore, language generated through T_2' is all unobservable events are added with observable events equivalent to the language generated by T_1.
   Now from T_2', design T_3 as the event synchronization of T_2' and M_d.

   $$T_3 = (Q_3, \Sigma, \Im, R_3, Q_0^3, AP, L_3) \text{ where,}$$

   - $Q_3 = Q_2 \text{ X } S_i$ is the set of states.

- Σ is the set of events.
- $\Im$ is the set of transitions.
- $R_3 \subseteq Q_3$ X $\Im$ X Q_3 is the transition relation.
- $Q_0^3 = Q_0^2$ X S_{i0} is the set of initial states.
- AP is the set of atomic proposition.
- $L_3 = Q_3 \rightarrow 2^{AP}$ is the marking function.

The language accepted by $T_3 = lang(T_2^{'}) \cap lang(M_d)$. The diagnosibility can be checked using LTL model checking.

4. The construction of the detector $D = (T_2, M_{\Im_2})$ is the last step. If any transition is not generated in the language of model T_2 then the detector gives output as "fault".

4.3 LTL based DES model for Version Number attack and its Detector

This section presents the system model for version number attacks in normal and attack conditions. The LTL specification for normal behavior with its Buchi model and NuSMV model checking for the LTL formula are presented. The steps as discussed in section 4.2, i.e., the IDS construction, are introduced.

-Model variables

The Model variables used for DES modeling of Version Number attack are {VNO, VNN, DIOVN, DIOPVN, PRVN, RVNI}. The description of all model variables are as follows:

- *VNN* $\rightarrow$ New Version Number
- *VNO* $\rightarrow$ Old version Number
- *DIOVN* $\rightarrow$ Version Number obtained in current DIO frame
- *DIOPVN* $\rightarrow$ Version Number obtained in previous DIO frame
- *RVNI* $\rightarrow$ Version Number received in RSP1
- *PRVN* $\rightarrow$ Version Number received in RSP2

-Event Set

Based on changes in the network, the events are listed as follows.

- *DIO:* DODAG information Object Control Frame
- *RQP:* Request frame from IDS
- *RSP1:* Response frame from Root node
- *RSP2:* Response frame from Parent Node
- *FIN:* The IDS initiates a clock, after a RQP is sent. It maintains track of the T_{req} interval in which the responses should arrive. After time finishes, FIN event is triggered.
- *u:* unobservable event to model the occurence of attack.

-Transition

A transition is a tuple with three values i.e., σ, check(v), assign(v).

For example, $\langle$ DIO, VNN $>$ VNO, VNN $\leftarrow$ DIOVN && VNO $\leftarrow$ DIOPVN $\rangle$ is a transition, where, DIO event is triggered, with the assignment of DIOVN to VNN, and DIOPVN to VNO. There is a checking condition also that checks whether VNN is greater than VNO. $\langle$ RSP1, VNN==RVNI, - $\rangle$ is another transition where event RSP1 triggers after checking whether VNN is the same as RVNI or not. No assignment operation is performed in the third field of the transition.

-Proposition Set

The set of propositions are listed as follows.

- P1 $\rightarrow$ Source state for the detector.
- P2 $\rightarrow$ DIO frame is received. After any transition of DIO event occurs, proposition P2 is true.
- P3 $\rightarrow$ QRP has been sent. After any transition of QRP event occurs, proposition P3 is true.
- P4 $\rightarrow$ RSP2 frame is received having VNN == PRVN. After any transition of RSP2 event occurs, proposition P4 is true.
- P5 $\rightarrow$ RSP2 frame is received having VNN != PRVN. After any transition of RSP2 event occurs, proposition P5 is true.
- P6 $\rightarrow$ RSP1 frame is received having VNN == RVNI. After any transition of RSP1 event occurs, proposition P6 is true.
- P7 $\rightarrow$ RSP1 frame is received having VNN != RVNI. After any transition of RSP2 event occurs, proposition P7 is true.
- P8 $\rightarrow$ T_{req} expires. After any transition of FIN event occurs, proposition P8 is true.

-Mask Function

This function considers all the events viz. DIO, RQP, RSP1 and RSP2 are measurable. The event FIN is also measurable but event u is unmeasurable. All defined model variables are also observable. Therefore, Mask Function is as follows: *M(DIO) = DIO, M(RQP) = RQP, M(RSP1) = RSP1, M(RSP2) = RSP2, M(FIN) = FIN, M(u)* $= \varepsilon$.

The network model for version number attack under normal and faulty conditions are shown in Fig. 5. The model is represented as M_d and its description as follows:

$$M_d = (S_i, S_{i0}, \tau, AP, L, \Sigma, R, V)$$

where,

- S_i = { S1, S2, S3, S4, S5, S6, S7, S8, S9, S10 }
- S_{i0} = { S1 }
- τ = { t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11 }
- AP = { P1, P2, P3 , P4, P5, P6, P7, P8 }
- L is the marking function given by $L(S_i) \rightarrow 2^{AP}$ that can be seen from Fig. 5.
- Σ = { DIO, RQP, RSP1, RSP2, FIN, u}
- R shows the transition relation depicted in Fig. 5.
- V = {VNO, VNN, DIOVN, DIOPVN, PRVN, RVNI}

In system′s DES modeling, each transition is divided into three value tuples as depicted in Fig. 5. The transitions are represented in the form of t_i, instead of three tuples, to avoid complications in the figures.

For the both 'normal and failure (attack)' scenarios, the DES model is described, as depicted in Fig. 5 is given below.

4.3.1 Normal Scenario:

Initially, the system is normal and is in state S1. When there is a DIO frame, a transition from S1 to S2 occurs. In this transition, the model variables *VNN* and *VNO* are initialized by *DIOVN* and *DIOPVN*, respectively. Along with the initialization, it checks whether the *VNN* is greater than *VNO* or not. If the comparison is true, transition (t1) moves to the S2 state; otherwise, it remains in the initial state. The transition (t2) moves from S2 to S3 when the request *RQP* frame is received, asking for the version number. This QRP frame is received by the root node and Parent node (node sent the DIO frame). After receiving *RQP*, the root node and Parent node send the RSP1 and RSP2, respectively. The root node is assumed never to be compromised and reliable. When the response frame *RSP2* is received from the parent node, it verifies the received version number *PRVN* with the *VNN*. In the checking, if *PRVN* is equal to *VNN*, the transition moves from state S3 to S4. This transition is taken only when the frame is received within the required frame response time. Afterward, in this transition, the version number *RVNI* received from the response frame *RSP1* is compared with the *VNN*. If the comparison is true, then this condition is considered normal. The change of state from S4 to S5 occurs due to transition t4. *RSP2* cannot be received due to link failure or any other reason. In that case, the only *RVNI* version number is verified with *VNN*. If *RVNI* is same as *VNN*, the situation is considered as normal. So, transition(t4) moves from S3 to S6 state. Transition (t7) occurs from S6 to state S10 after the T_{req} time has passed and event *FIN* is fired. In the normal situation, the *VNN* should be same as *PRVN* and *RVNI*. The RSP2 may not be received; in that case, if *RVNI* is the same as *VNN* the situation is considered as 'normal'.

4.3.2 Attack Scenario:

When there is a DIO frame, transition (t1) from the initial state S1 to S2 occurs. This transition initializes *VNN* and *VNO* model variables, respectively, with *DIOVN* and *DIOPVN*. The transition also checks the equality of *VNN* with *VNO*. If the check is found true, transition moves to S2 state, otherwise remains at initial state S1. When the *RQP* event occurs, transition (t2) moves to S3. A *RQP* frame is sent to root node and Parent node asking for version number. The *RSP1* and *RSP2* events are triggered by root node and parent node, respectively. The *RSP1* and *RSP2* frames contain the version number *RVNI* and *PRVN*, respectively. Further, these responses are to arrive within T_{req} time after RQP is sent. A model moves from S3 to S4 on observation of the event *RSP2* by transition t3. After transition t2, responses from the attacker arrives (i) transition (t3) having *PRVN* the same as *VNN* (ii) transition (t6) not having *PRVN* same as *VNN*, where system moves to state S8. It may be noted that *RSP2* event for transition (t6) is considered malicious. But *RSP2* event for transition (t3) proceeds with response *RSP1*. Enabling of transition (t5) is only dependent on *RSP1*. The model variable equality is ensured by checking *RVNI* with *VNN*. In case *RSP1* arrives from attacker, the *RVNI* is not equal to *VNN* in transition (t5). The transition (t5) fires on receipt of single response *RSP1* within T_{req}. It checks *RVNI* with the *VNN* and t5 moves to state S9. The transitions (t7) is enabled when event *FIN* occurs and system moves to state S10.

4.4 LTL Specification

The LTL specification for the version number attack in the normal scenario is formulated. In terms of events, the specification of the system's non-failure scenario (no version number attack) is described as:

Either any response with unmatched VNN-PRVN and VNN-RVNI should not be encountered, or any response (RSP1 and RSP2) frame should not be detected after the QRP request frame is sent before the system terminates.

Therefore, according to the transition set, the specification is considered as: Either t5 and t7 (caused by RSP1 and RSP2) should not be detected, or t7 (caused by FIN) does not occur just after transition t2 before the system terminates.

It can be observed from the proposition set that P5 and P7 are the propositions that are true when transition t6 and t5 occur; respectively, P3 corresponds to transition t2 and P8 is true when t7 exists, and the system terminates.

The specification is described in the form of LTL formula f_1 as:

$$\mathbf{((\neg P5 \wedge \neg P7) \wedge (P3 \rightarrow X \neg P8)) \cup P8}$$

Using LTL, the system′s non-failure scenario is stated logically and unambiguously. The conversion of the LTL formula into a Buchi Automata is done automatically. Due to this, the LTL-based framework provides a proper and suitable way to state specifications compared to the state-based schema.

4.5 DES detector for Version Number Attack

The construction steps for the detector are as follows:

1. **Construction of Buchi Automata**

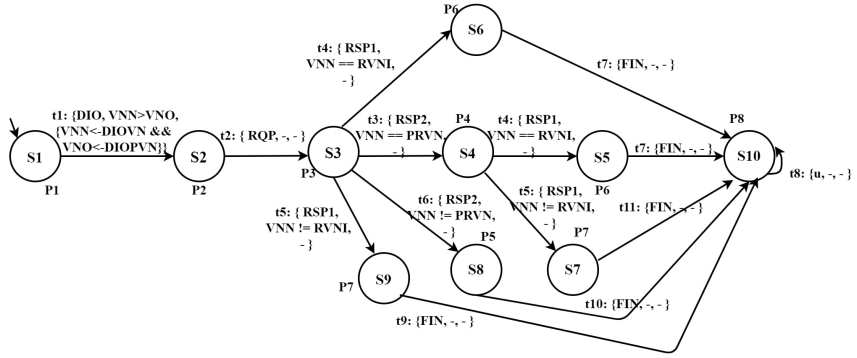


Fig. 5 System model for LTL diagnosis

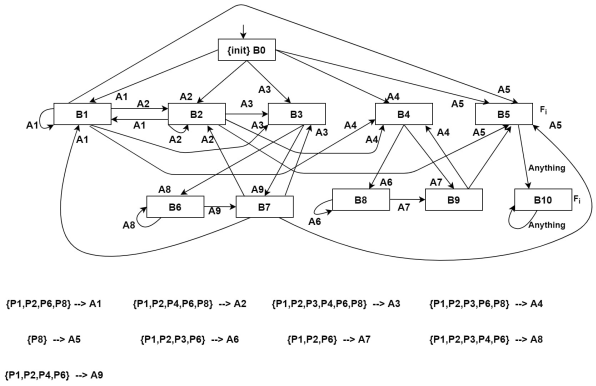


Fig. 6 Buchi Automata B_f for f_1

The Buchi Automata B_f is obtained from the given LTL specification as follows.

$B_f = (C_f, \Sigma_{AP}, R_f, q_0^f, F_i)$ is shown in Fig. 6 where,

- C_f = { B0, B1, B2, B3, B4, B5, B6, B7, B8, B9, B10} is the set of states.
- Σ_{AP} = { P1, P2, P3, P4, P6, P8} is the set of propositions defining the model corresponds to the event. The set of propositions are represented as A_i where, $i \geq 1$ (e.g., { P1,P2,P3,P4 } is written as A1).
- R_f is the transition relation, where $R_f \subseteq C_f$ X Σ_{AP} X C_f.
- q_0^f = B0 is the source state.
- F_i = { B5, B10 } are the final states that satisfies the specification expressed through F_1.

As shown in Fig. 6, B_f shows five transitions emerging from the source state B0. All the transitions satisfy ($\neg$P5 $\wedge$ $\neg$P7). Furthermore, the states that can be reached via proposition P3,i.e., B3 & B6 and B4 & B8, do not have any outgoing paths of P8. This suffices (P3 $\rightarrow$ X($\neg$P8)) sub-part of f_1. After reaching state B5, f_1 is satisfied, and as a result, any subsequent proposition leads to final state B10. B_f is a non-deterministic automaton which means, various transitions for the same set of propositions exist. It can be verified manually that "B_f accepts all proposition traces over AP satisfying f_1". As a result, the specification is represented through an automaton.

2. **Test of Pre-diagnosability using Proposition Synchronization of Buchi Automata and the model**

The model is considered to be pre-diagnosable, whenever every infinite proposition trace encounters final states of the automaton infinitely often. A Proposition Synchronization of the M_d and B_f is used for the testing of Pre-Diagnosibility. Fig. 7 depicts the synchronization where Y_{i_j} denotes (B_i, s_j), $B_i \in C_{f_1}$ and $s_j \in S_i$. The proposition synchronization is represented as model T_1 for f_1. The T_1 consists of 7-tuples is shown as:

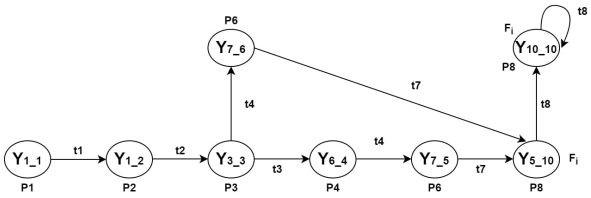


Fig. 7 Model T_1 for f_1

$$T_1 = (Q_1, \Sigma, \Im, R_1, Q_0^1, AP \cup F_i, L_1)$$

- Q_1 = {Y_{1_1},Y_{1_2},Y_{3_3},Y_{6_4},Y_{7_5},Y_{7_6},Y_{5_10},Y_{10_10}}
- Σ = set of events { DIO, RQP, RSP1, RSP2, FIN, u}
- $\Im$ = { t1, t2, t3, t4, t7, t8 } is the set of transitions.
- R_1 is the transition relation as shown in Fig. 7.
- Q_0^1 = { Y_{1_1} }
- $AP \cup F_i$ represents the set of proposition.
- L_1 represents marking function, which is shown in Fig. 7.

The construction of pre-diagnosibility model T_1 is shown in Fig. 7, Q_0^1 = {(c_{f_1}, S) $\in$$Q_1$|($q_0^{f_1}$, L(S), c_{f_1}) $\in$ R_{f_1}, S $\in$ S_{i0} }. Therefore Q_1^0 = {(B1S1), (B2S1), (B3S1), (B4S1)} as $L_1(S_1)$ = P1, $(B0, P1, B1) \in R_{f_1}$, (B0,P1,B2) $\in$ R_{f_1}, (B0,P1,B3) $\in$ R_{f_1}, (B0,P1,B4) $\in$ R_{f_1} and S_1 $\in$ S_{i0}. So, the initial states are {Y_{1_1},Y_{2_1},Y_{3_1},Y_{4_1}}. The initial states i.e., second Y_{2_1},third Y_{3_1} and fourth Y_{4_1}, get merged with initial state Y_{1_1}.

The state Y_{1_2} is reachable by the transition t1 from the initial state Y_{1_1}(= {B1, S1}) i.e., ((B1,S1), t1,(B1,S2)) $\in R_1$ as L(S2) = P2, (B1,P2,B1) $\in R_{f_1}$ and (S1,t1,S2) $\in$ R. We can also say, (Y_{1_1}, t1, Y_{1_2}) $\in R_1$. Y_{1_1} is labeled as P1, i.e., P1 $\in L_1(Y_{1\_1})$ as P1 $\in L_1$(S1). Y_{1_2} is marked as P2, i.e., P2 $\in L_1(Y_{1\_2})$ as P2 $\in L_1$(S2). According to this, T_1 is constructed and all equivalent states are merged.

As mentioned in Section 4.2, the system model is pre-diagnosable only if T_1 visits the F_i states infinitely for every proposition trace generated by T_1. In terms of model checking, T_1 must satisfy the LTL specification GFF_i. This model checking is performed by NuSMV tool for pre-diagnosibility (is shown in Section 5).

3. **Test for Diagnosibility** After completing the pre-diagnosability test, the system model proceeds to test the diagnosability to specification formula.

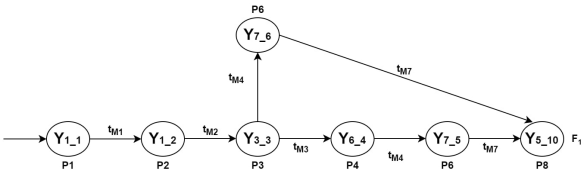


Fig. 8 Model T_2 for f_1

Synthesis of Masked T_1: With the use of mask function M, T_2 is constructed as shown in Fig. 8.

$$T_2 = (Q_2, \Delta, \Im_m, R_2, Q_0^2), \text{ where}$$

- $Q_2 = \{Y_{1\_1}, Y_{1\_2}, Y_{3\_3}, Y_{6\_4}, Y_{7\_5}, Y_{7\_6}, Y_{5\_10}\}$.
- $\Delta = \{DIO, RSP, RQP1, RQP2, FIN\}$.
- $\Im_m = \{t_{M1}, t_{M2}, t_{M3}, t_{M4}, t_{M7}\}$. t_{Mi} denotes the observable event equivalent of transition t_i (of M_d).
- R_2 is the transition relation, as shown in Fig. 8.
- $Q_0^2 = \{Y_{1\_1}\}$.

The set Q_2 contains all those states of T_1 where the transition leading into that state is triggered having an observable event. Hence, Y_{10_10} state (in T_1) gets eliminated as t8 is the only transition that leads to Y_{10_10} state whose observed event is ε.

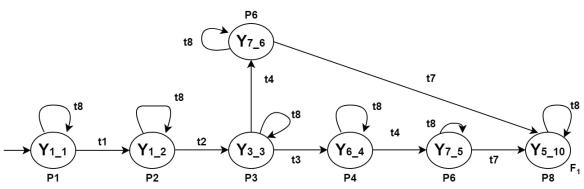


Fig. 9 Model $T_2^{'}$ for f_1

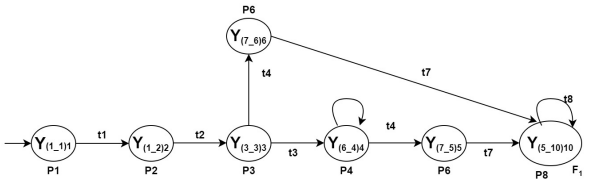


Fig. 10 Model T_3 for f_1

Unmasking T_1: Now from T_2, the construction of $T_2^{'}$ is done. $T_2^{'}$ accepts $M^{-1}(lang(T_2))$. The $T_2^{'}$ is represented using 5-tuples shown in Fig. 9 as follows:

$$T_2^{'} = (Q_2, \Sigma, \Im, R_2^{'}, Q_0^2), \text{ where}$$

- $Q_2 = \{Y_{1\_1}, Y_{1\_2}, Y_{3\_3}, Y_{6\_4}, Y_{7\_5}, Y_{7\_6}, Y_{5\_10}\}$.
- $\Sigma = \{DIO, RSP, RQP1, RQP2, FIN, u\}$.
- $\Im = \{$ t1, t2, t3, t4, t7, t8 $\}$.
- $R_2^{'}$ is the transition relation, as shown in Fig. 9.
- $Q_0^2 = \{Y_{1\_1}\}$.

In a comparison of $T_2^{'}$ with T_2, self-cycle transitions are an addon to all states of $T_2^{'}$. For example, (Y_{1_1}, t8, Y_{1_1}) $\in R_2^{'}$ as $Y_{1\_1} = Y_{1\_1}$ and t8 transition triggers the event ε.

$$T_3 = (Q_3, \Sigma, \Im, R_3, Q_0^3, AP, L_3) \text{ where,}$$

- $Q_3 = \{Y_{(1\_1)1}, Y_{(1\_2)2}, Y_{(3\_3)3}, Y_{(6\_4)4}, Y_{(7\_5)5}, Y_{(7\_6)6}, Y_{(5\_10)10}\}$
- Σ = set of events { DIO, RQP, RSP1, RSP2, FIN, u}
- $\Im = \{$ t1, t2, t3, t4, t7, t8 $\}$ is the set of transitions.
- R_3 is the transition relation as shown in Fig. 10.
- AP is the set of proposition, like M_d .
- $L_3 = Q_3 \rightarrow 2^{AP}$ is the marking function, which is shown in Fig. 10.

All the traces of M_d, which has similarly observed traces in T_1, are produced through T_3. Therefore, all unobservable event u has self-loop transitions on states Y_{1_1}, Y_{1_2}, Y_{3_3}, Y_{6_4}, Y_{7_5}, Y_{7_6}, Y_{5_10} are eliminated although any of the state does not associate to R(i.e., not present in M_d) and remaining transitions from $T_2^{'}$ are present. This way, T_3 is constructed.

If M_d is diagnosable then all infinite traces generated through T_3 satisfy the f_1. LTL model checking of the model T_3 with specification f_1 through NuSMV model checker is presented in Section 5.

4. **Final Detector** The system model M_d is diagnosable, so its detector can be designed. It can be seen in Fig. 8 that T_2 accompany M_{τ_2} to construct the final detector where,

$M_{\tau_2} : \Delta^* \rightarrow \{\text{failure}\}$ is a partial function where $\forall$ s $\in$ Δ^*, M_{τ_2}(s) = failure if s is not produced by T_2.

Using mask M, the detector notices the traces of transitions that are generated by M_d. If same trace is not generated by T_2, the detector shows that attack has occurred as

output. For example, consider t1, t2, t3, t5, t7 as the transition trace generated through M_d, then its equivalent transition trace t_{M1}, t_{M2}, t_{M3}, t_{M5}, t_{M7} is not generated through T_2. The transition trace t_{M1}, t_{M2}, t_{M3}, t_{M5}, t_{M7} corresponds to P1, P2, P3, P4, P7 that violates $((\neg P5 \wedge \neg P7) \wedge (P3 \rightarrow X\neg P8)) \cup P8$. Thus the detector shows the attack for a trace as a output. On the other side, for transitions t1, t2, t3, t4, t7 the equivalent transition trace is t_{M1}, t_{M2}, t_{M3}, t_{M4}, t_{M7} that is generated by T_2. The trace t_{M1}, t_{M2}, t_{M3}, t_{M4}, t_{M7} corresponds to P1, P2, P3, P4, P6 which coheres to $((\neg P5 \wedge \neg P7) \wedge (P3 \rightarrow X\neg P8)) \cup P8$. Thus, the detector shows normal for this case as output. Any transition that does not satisfy the property i.e., any transition is not generated in the language of the model T_2 then detector shows output as attack.

5 Performance Evaluation

The performance of the Version Number detection technique has been evaluated through simulation and the NuSMV model checker.

In this work, we considered the random topology with the increased percentage of the spiteful node to analyze the performance changes in the metrics. The simulation topology consists of 20 nodes, i.e., root, IDS, and remaining sensor nodes, as shown in Fig. 11. All nodes are placed in a 100x100m area. In random topology, all nodes are placed randomly except the IDS node. The IDS node lies near the root node. Here, node 1 and node 2 are considered as the root node and IDS node, respectively, as shown in Fig. 11(a). In other scenarios, the simulation setup has nodes 19 and 20 as spiteful nodes, depicted in yellow. Fig. 11(b) has only node 20 as a spiteful node, and the scenario shown in Fig. 11(c) has nodes 19 and 20 as spiteful nodes. In such a way, we have considered there scenarios wherein the number of spiteful nodes increases by 5%.

Simulation Environment

The detection technique uses the Contiki 2.7 operating system and its Cooja simulator for performance evaluation. Contiki is an open-source and multi-tasking operating system for wireless sensor networks. The Contiki 2.7 update contains the Contiki RPL protocol [11]. Cooja is a simulator for the simulation of the networks of sensors running over the Contiki operating system.

On a plane square, the Tmote Sky motes are introduced. All motes, i.e., motionless motes, are called static motes. For all motes, the communication and interference range is 50 and 100 meters, respectively. The simulation is verified 20 times for each scenario, and the average values for all parameters are determined. In Table 1, the simulation parameters and their values are given.

By increasing the number of spiteful nodes in random topology, the performance of the detection scheme has been assessed. For an attack or normal case, each simulation runs for 50 minutes. It is assumed that 50 minutes are sufficient to determine the network's performance and the network's DODAG becomes stable within three minutes.

Table 1 Cooja Simulator Parameters

Parameter	Value
Operating System	Contiki 2.7
Simulator	Cooja
Radio medium	UGDM
Simulation area	100m x 100m
Node Type	Tmot Sky
MAC layer	ContikiMAC/6LoWPAN
Number of nodes	20
Number of malicious nodes	Up to 25%
Physical layer	IEEE 802.15.4

Performance Parameters:

The following performance parameters are considered:

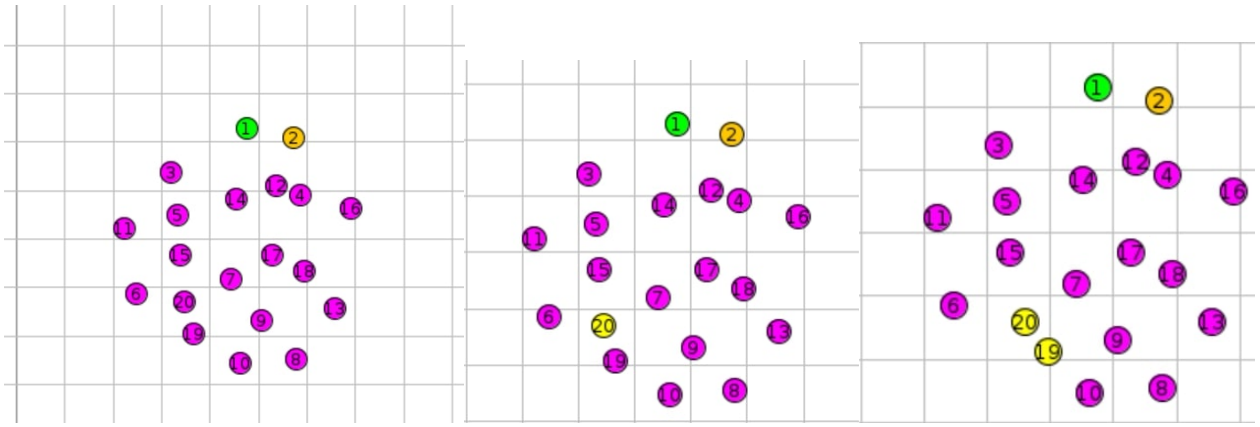
- **Resource Requirements:** Average power consumption, and Memory requirements.
  Average power consumption is calculated based on the average of the total power consumption of nodes except root, IDS, and attacker nodes. Memory requirement shows memory (RAM and ROM) needed to implement the DVN detection technique, i.e., the IDS.
- **Network Parameters:** Packet Delivery Rate (PDR) and Control Message Overhead.
  The ratio between the number of packets delivered (D) to the total number of packets sent (S) from source node to the destination is known as Packet Delivery Rate (PDR). PDR is computed according to the Equation 1.

$$\textit{Packet Delivery rate } (PDR) = \frac{D}{S} \tag{1}$$

  Control Message Overhead refers to the total number of control packets sent for the path establishment of the node as well as the verification of the spiteful node.
- **Accuracy:** True Positives Rate (TPR), False Positives Rate (FPR).
  True Positive Rate (*TPR*) is calculated as per Equation 2. In Equation 2, *TP* is the number of true positives and *P* refers to the total number of positive cases. A true positive *TP* is a case where our proposed solution detects the legitimate node correctly as legitimate.

$$TPR = \frac{TP}{P} \tag{2}$$

  False Positive Rate (*FPR*) is computed as per Equation 3. In Equation 3, *FP* is the number of false positives while *N* refers to the total number of negative cases. *FP* is a



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Fig. 11 Random Topology for the Simulation

case in which our proposed mechanism detects legitimate node incorrectly as spiteful node.

$$FPR = \frac{FP}{N} \tag{3}$$

Table 2 Accuracy Metrics Values

% of Spiteful Node	TPR	FPR
5	99.7	0.55
10	98.8	1.02
15	97.7	1.9
20	96.5	2.1
25	95.4	2.35

Analysis on Simulation Results:
The performance of the simulation parameters in terms of accuracy metrics, i.e., True Positive Rate (TPR) and False Positive Rate (FPR) with the distinct percentage of spiteful nodes are shown in Table 2, and resource requirements like average power consumption and memory utilization are shown in Fig. 12 and Table 3, respectively.

As seen from Table 2, the true positive rate drops when the percentage of spiteful nodes rises. It happens because an increased percentage of spiteful nodes may collide, and therefore, IDS can not detect all spiteful nodes. It is showed that the proposed scheme functions well with near about 99.7 and 95.5 true positive rate when the percentage of the spiteful node is 5 and 25, respectively. Table 2, it is also shown that false-positive rates rise with the increased percentage of the spiteful nodes. It happens when there is no path to send the version number value to the IDS node to verify the version number.

The simulation uses the emulated sky motes [35] which utilizes the MSP430 controller to determine the memory usage. The controller consists of 10 KB RAM and 48 KB ROM. The verification technique requires the extra overhead of sending the probing packets and verifying the node,

Table 3 Memory Utilization

Scenario	RAM (Byte)	ROM (Byte)
Contiki RPL	7530	47363
Contiki RPL + IDS	7857	48106

which requires extra RAM and ROM usage. In this case, an additional 327 bytes and 743 bytes of RAM and ROM are required, as shown in Table 3. It may be noted that this memory requirement is reasonable. The IDS stores the DIO message VNN and DIO sender's address to verify the spiteful node through the REQ and RSP message.

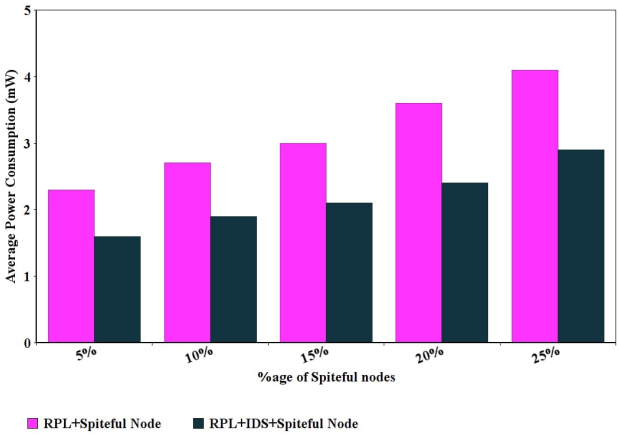


Fig. 12 Avg. Power Consumption vs Percentage of Spiteful Nodes

The average power consumption results are shown in Fig. 12, which correspond to the RPL+Spiteful node and RPL+IDS+Spiteful node environments. It can be seen that the power consumption increases with the increase in the percentage of spiteful nodes. It happens since packets are dropped due to a global repair mechanism. This causes the

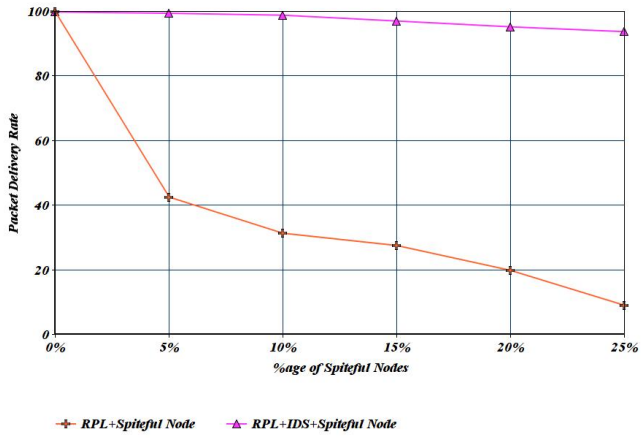


Fig. 13 Packet Delivery Rate vs Percentage of Spiteful Nodes

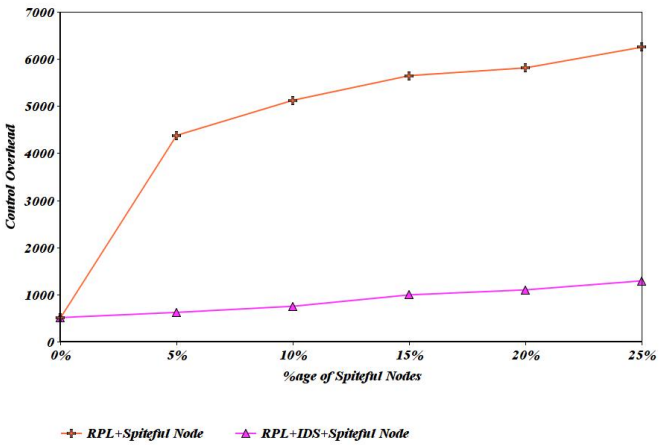


Fig. 14 Control Message Overhead vs Percentage of Spiteful Nodes

nodes to waste energy due to the repeated transmissions of the packets. Therefore, it leads to high power consumption. When the efficiency of the IDS is checked, we can see that power consumption has been reduced compared to the RPL+Spiteful node environment.

The performance of the network parameters, i.e., Packet Delivery Rate and Control Message Overhead against the distinct percentage of spiteful nodes, are shown in Fig. 13 and Fig. 14, respectively. The PDR performs better when there is no spiteful node in the network, i.e., higher than 99. Nevertheless, when 5% of the total nodes are spiteful in the network, the PDR value drops drastically to nearly 40 percent from 99 percent. The PDR value depreciates more with the increase of spiteful nodes. However, our proposed detection technique works efficiently, as it reduces the effect of the Version Number Attack. Fig. 13 shows that the PDR of the proposed technique is better in the RPL+IDS+Spiteful scenario than the RPL+Spiteful scenario. It happens because our proposed technique detects the spiteful node more accurately after the verification of the suspect node. The verification node does not update the new Version Number sent from the suspect node until the verification procedure gets complete and the suspect node gets verified as normal. The packet is sent to the root node from the path of the suspect node. If the suspect node is verified as a spiteful node, then the verification node does not update the new version number, and the packet is not sent via the suspect node.

The performance of the Control Message Overhead (CMO) can be seen in Fig. 14. The CMO of the RPL+Spiteful environment increases drastically even when 5% spiteful nodes of total nodes are present. The CMO increases with the increase of the spiteful nodes in the RPL+Spiteful environment. It happens because of the version number attack behavior. However, the proposed solution′s control overhead is not proportional to the percentage of spiteful nodes, as the proposed scheme prevents DIO′s from the broadcasting process, which broadcasts for the reconstruction of the DODAG. As a result, the CMO is much lower in the RPL+IDS+Spiteful environment than the RPL+Spiteful environment, as shown in Fig. 14.

Table 4 Comparison of Proposed Approach with Related Work

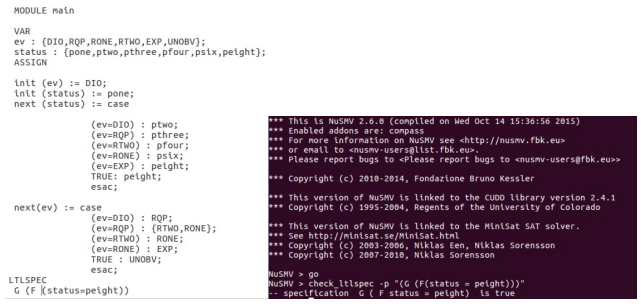
Mechanism	PDR (in %)	Overhead (Pkts/sec)	Topology
Lightweight Mitigation [17]	96.24	5045	Random, grid-centre
Distributed and cooperative Verification [11]	97	1500	Random
VeRa [31]	-	-	Random
Proposed Approach	98.7	1100	Random

Table 4 represents the comparison of studies in the related work with the proposed approach. It is shown in the table that the proposed approach provides better network performance, i.e., in terms of PDR and CMO. The proposed solution supports random topology and performs better than the studies in the related work. The PDR and CMO obtain the best result, i.e., 98.7% and 1100 pkts/sec, respectively.

Model Checking:
NuSMV [36] is a platform for describing the models and validates LTL formula(s) for these models. It accepts the program (describing a model) in the form of text as input. It shows the output as 'true' over the given LTL specification; if the specification holds, it otherwise generates a trace where the specification is not satisfied. As discussed in section 4.3, the LTL specification used for the normal condition is as follows.

$$((\neg \mathbf{P5} \wedge \neg \mathbf{P7}) \wedge (\mathbf{P3} \rightarrow \mathbf{X} \neg \mathbf{P8})) \cup \mathbf{P8}$$

Pre-diagnosibility Model Checking: The pre-diagnosability testing of the system model M_d (as depicted in Fig. 5) is



(a) NuSMV Code for Pre-diagnosibility (b) Output for Pre-diagnosibility testing

Fig. 15 Pre-diagnosibility testing of the model

(a) NuSMV Code for Diagnosibility (b) Output for Diagnosibility testing

Fig. 16 Diagnosibility testing of the model

done through NuSMV model checking. Fig. 15(a) depicts the code snap for the pre-diagnosability of M_d. The variables *status* and *ev* are of enumeration type. Both variables *status* and *ev* are initialized with proposition value (pone) and event (DIO) that can be seen in Fig. 15(a). The variable *ev* is assigned the value among DIO, QRP, RONE, RTWO, EXP, UNOBV of different events. Proposition values like pone, ptwo, three, pfour, psix, peight are referred through the variable *status* as depicted in Fig. 15(a). The definition of a transition in NuSMV is not supported, so the code snap consists of the triggering events of transitions (for example, DIO denotes $t1$). Furthermore, the propositions are presented in words (e.g., P1 is represented through Pone; similarly, other propositions are represented). The cases are evaluated where P8 is true for the pre-diagnosability test. The result is true for the specification G(F(P8)) that can be seen in Fig. 15(b). So, it can be said that the system is pre-diagnosable, as model T_1 satisfies the specification.

Model checking Test for Diagnosability: For testing the diagnosibility, the event synchronization of the system model and $T_2^{'}$ model together constructs the model T_3 as depicted in Fig. 10. The code snap for diagnosibility testing and output corresponding to T_3 are shown in Fig. 16 with the help of NuSMV model checker.The system model is said to be diagnosable with respect to T_3, as, model satisfies the specification $((\neg P5 \wedge \neg P7) \wedge (P3 \rightarrow X\neg P8)) \cup P8$.

As T_3 is formally verified, the diagnoser can be constructed (as per Section 4.5), and the resulting IDS can be considered free of errors.

6 Conclusion

The LLN devices are constrained in terms of low resources, high packet loss, etc. The RPL protocol uses DODAG version number to form DODAG (acyclic) topology. However, there is no technique to authenticate the DODAG version number. So, RPL has become vulnerable to DODAG version number attacks. DVN requires neither any change in packet format nor the sequence of packets and hence falls into the category of stealthy attacks. DES-based IDS has been applied for stealthy attacks that achieve low false alarm rates, low overhead, etc. The construction of DES-based IDS for the network protocol is done manually, which may lead to errors. So the designed IDS does not guarantee its correctness. Therefore, the LTL-based DES schema has been used to design and formally verify the proposed DES-based DODAG version number detection scheme that deals with its correctness.

The proposed detection scheme has been simulated in the Contiki cooja simulator. The simulation results show the accuracy of 99.7% and 95.4% when the percentage of spiteful nodes are 5% and 25%, respectively. The true positive rate decreases with the increase of spiteful nodes, whereas the false positive rate increases with spiteful nodes. It happens since there is no path to send the version number value to the IDS node for its verification. In sending the packets and verifying them, the nodes require extra 327 bytes and 743 bytes of RAM and ROM, respectively, which are minimal when compared to the memory available in IoT nodes.

We plan to advance the proposed research work using a test-bed environment to validate the results in future work. Future research could be to analyze the version number attack collaborated with other attacks.

Compliance with Ethical Standards

Conflict of Interest All author declares that they have no conflict of interest.

Ethical Approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. S. Ziegler, C. Crettaz, L. Ladid, S. Krco, B. Pokric, A. F. Skarmeta,A. Jara, W. Kastner, M. Jung, Iot6 – moving to an ipv6-based future iot,in: A. Galis, A. Gavras (Eds.), The Future Internet, Vol. 7858, SpringerBerlin Heidelberg, Berlin, Heidelberg, 2013.
2. L. C. El Ksimi, A., A new ipv6 security approach for a local network.,in: B. M. E. M. Khoukhi, F. (Ed.), AIT2S 2018, Vol. 66, Springer LNNS,Cham, 2019.
3. A. Mayzaud, R. Badonnel, I. Chrisment, A Taxonomy of Attacks in RPL-based Internet of Things, International Journal of Network Security 18 (3)(2016) 459 – 473.

4. R. Alexander, A. Brandt, J. Vasseur, J. Huii, K. Pister, P. Thubert, P. Levis,R. Struik, R. Kelsey, T. Winter, RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks, RFC 6550 (Mar. 2012). doi:10.17487/RFC6550.
5. L. J. Chugh K, Lasebae A, Case study of a blackhole attack on 6 lowpan-rpl., in: SECURE 2012, Sixth International Conference Emerging Secure Information, System Technology, 2012, pp. 157–162.
6. H. Deng, W. Li, D. Agrawal, Routing security in wireless ad hoc net-works, IEEE Communications Magazine 40 (10) (2002) 70–75.doi:10.1109/MCOM.2002.1039859
7. M. Ammar, G. Russello, B. Crispo, Internet of things: A survey on the security of iot frameworks, Journal of Information Security and Applications 38 (2018) 8–27. doi:https://doi.org/10.1016/j.jisa.2017.11.002.
8. L. Babun, K. Denney, Z. B. Celik, P. McDaniel, A. S. Uluagac, A survey on iot platforms: Communication, security, and privacy perspectives,Computer Networks 192 (2021) 108040. doi:https://doi.org/10.1016/j.comnet.2021.108040
9. C. A. de Souza, C. B. Westphall, R. B. Machado, J. B. M. Sobral, G. dos Santos Vieira, Hybrid approach to intrusion detection in fog-based iot environments, Computer Networks 180 (2020) 107417. doi:https://doi.org/10.1016/j.comnet.2020.107417.
10. A. Mayzaud, A. Sehgal, R. Badonnel, I. Chrisment, J. Schönwälder ,Mitigation of Topological Inconsistency Attacks in RPL based Low Power Lossy Networks, International Journal of Network Management (Jun.2015). doi:10.1002/nem.1898.
11. F. Ahmed., Y. Ko., A distributed and cooperative verification mechanism to defend against dodag version number attack in rpl, in: Proceedings of the 6th International Joint Conference on Pervasive and Embedded Computing and Communication Systems - PEC, (PECCS 2016), INSTICC, SciTePress, 2016, pp. 55–62. doi:10.5220/0005930000550062.
12. Aufner, P. The IoT security gap: a look down into the valley between threat models and their implementation. Int. J. Inf. Secur. 19, 3–14 (2020). https://doi.org/10.1007/s10207-019-00445-y
13. T. Tsao, R. Alexander, M. Dohler, V. Daza, A. Lozano, M. Richardson, A Security Threat Analysis for the Routing Protocol for Low-Power and Lossy Networks (RPLs), RFC 7416 (Jan. 2015). doi:10.17487/1085RFC7416.
14. F. Medjek, D. Tandjaoui, N. Djedjig, I. Romdhani, Multicast dis attack mitigation in rpl-based iot-llns, Journal of Information Security and Applications 61 (2021) 102939. doi:https://doi.org/10.1016/j.jisa.2021.102939.
15. Ferraris, D., Fernandez-Gago, C. TrUStAPIS: a trust requirements elicitation method for IoT. Int. J. Inf. Secur. 19, 111–127 (2020). https://doi.org/10.1007/s10207-019-00438-x
16. N. Djedjig, D. Tandjaoui, F. Medjek, I. Romdhani, Trust-aware and co-operative routing protocol for iot security, Journal of Information Security and Applications 52 (2020) 102467. doi:https://doi.org/10.10951016/j.jisa.2020.102467.
17. A. Arıs , S. B. Ors Yalcın, S. F. Oktug, New lightweight mitigation techniques for rpl version number attacks, Ad Hoc Networks 85 (2019) 81–91. doi:https://doi.org/10.1016/j.adhoc.2018.10.022.
18. G. Simoglou, G. Violettas, S. Petridou, L. Mamatas, Intrusion detection systems for rpl security: A comparative analysis, Computers Security 104 (2021) 102219. doi:https://doi.org/10.1016/j.cose.2021.102219.
19. L. Rosa, T. Cruz, M. B. de Freitas, P. Quiterio, J. Henriques, F. Caldeira, E. Monteiro, P. Simoes , Intrusion and anomaly detection for the next-generation of industrial automation and control systems, Future Generation Computer Systems 119 (2021) 50–67. doi:https://doi.org/10.1016/j.future.2021.01.033.
20. M. Mitra, P. Banerjee, F. A. Barbhuiya, S. Biswas, S. Nandi, Ids for arp spoofing using ltl based discrete event system framework, Networking Science 2 (2013) 114–134. doi:10.1007/s13119-013-0019-1.
21. A. D. Seth, S. Biswas, A. K. Dhar, De-authentication attack detection using discrete event systems in 802.11 wi-fi networks, in: 2019 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), 2019, pp. 1–6. doi:10.1109/ANTS47819.2019.9118100.
22. M. Agarwal, S. Biswas, S. Nandi, Discrete event system framework for fault diagnosis with measurement inconsistency: case study of rogue dhcp attack, IEEE/CAA Journal of Automatica Sinica 6 (3) (2019) 789–806. doi:10.1109/JAS.2017.7510379.
23. I. F. Kilincer, F. Ertam, A. Sengur, Machine learning methods for cyber security intrusion detection: Datasets and comparative study, Computer Networks 188 (2021) 107840. doi:https://doi.org/10.1016/j.comnet.2021.107840.
24. S. Jiang, R. Kumar, Failure diagnosis of discrete-event systems with linear-time temporal logic specifications, IEEE Transactions on Automatic Control 49 (6) (2004) 934–945. doi:10.1109/TAC.2004.829616.
25. J. Edmund M. Clarke, O. Grumberg, D. Kroening, D. Peled, H. Veith,Model Checking, MIT Press, 1999, cambridge, MA, USA.
26. M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, D. Teneket-zis, Diagnosability of discrete-event systems, IEEE Transactions on Automatic Control 40 (9) (1995) 1555–1575. doi:10.1109/9.412626.
27. K. Hofer-Schmitz, B. Stojanovic, Towards formal verification of iot protocols: A review, Computer Networks 174 (2020) 107233. doi:https://doi.org/10.1016/j.comnet.2020.107233.
28. G. Montenegro, J. Huii, D. Culler, N. Kushalnagar, Transmission of IPv6 Packets over IEEE 802.15.4 Networks, RFC 4944 (Sep. 2007).doi:114010.17487/RFC4944.
29. Z. A. Almusaylim, N. Jhanjhi, A. Alhumam, Detection and mitigation of rpl rank and version number attacks in the internet of things: Srpl-rp, Sensors 20 (21) (2020). doi:10.3390/s20215997.1145
30. S. Raza, L. Wallgren, T. Voigt, Svelte : Real-time intrusion detection in the internet of things, Ad hoc networks 11 (8) (2013) 2661–2674. doi:10.1016/j.adhoc.2013.04.014.
31. A. Dvir, T. Holczer, L. Buttyan, Vera - version number and rank authentication in rpl, in: 2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems, 2011, pp. 709–714. doi:10.1109/MASS.2011.76.
32. N. Hubballi, S. Biswas, S. Roopa, R. Ratti, S. Nandi, Lan attack detection using discrete event systems, ISA Transactions 50 (1) (2011) 119–130. doi:https://doi.org/10.1016/j.isatra.2010.08.003.
33. N. Hubballi, N. Tripathi, An event based technique for detecting spoofed ip packets, Journal of Information Security and Applications 35 (2017)32–43. doi:https://doi.org/10.1016/j.jisa.2017.04.001.
34. A. Pnueli, The temporal logic of programs, in: 18th Annual Symposium on Foundations of Computer Science (sfcs 1977), 1977, pp. 46–57. doi:10.1109/SFCS.1977.32.
35. http://www.eecs.harvard.edu/konrad/projects/shimmer/references/tmote-sky-datasheet.pdf,moteivCorporation,Tmote Sky: Datasheet.
36. NuSMV[Online].Available:http://www.nusmv.fbk.eu.