Received: 11 July 2020 | Revised: 9 September 2020 | Accepted: 24 September 2020

DOI: 10.1002/dac.4646

RESEARCH ARTICLE

Random forest classifier-based safe and reliable routing for opportunistic IoT networks

Nisha Kandhoul[1] | **Sanjay K. Dhurandher**[2] | **Isaac Woungang**[3]

[1]Division of Information Technology, NSIT, University of Delhi, New Delhi, India

[2]Department of Information Technology, Netaji Subhas University of Technology, New Delhi, India

[3]Department of Computer Science, Ryerson University, Toronto, Ontario, Canada

Correspondence
Nisha Kandhoul,Division of Information Technology, NSIT, University of Delhi, New Delhi, India.
Email: nishakandhoul1990@gmail.com

Summary

Designing a safe and reliable way for communicating the messages among the devices and humans forming the Opportunistic Internet of Things network (OppIoT) has been a challenge since the broadcast mode of message sharing is used. To contribute toward addressing such challenge, this paper proposes a Random Forest Classifier (RFC)-based safe and reliable routing protocol for OppIoT (called RFCSec) which ensures space efficiency, hash-based message integrity, and high packet delivery, simultaneously protecting the network against safety threats viz. packet collusion, hypernova, supernova, and wormhole attacks. The proposed RFCSec scheme is composed of two phases. In the first one, the RFC is trained on real data trace, and based on the output of this training, the second phase consists in classifying the encountered nodes of a given node as belonging to one of the output classes of nodes based on their past behavior in the network. This helps in proactively isolating the malicious nodes from participating in the routing process and encourages the participation of the ones with good message forwarding behavior, low packet dropping rate, high buffer availability, and a higher probability of delivering the messages in the past. Simulation results using the ONE simulator show that the proposed RFCSec secure routing scheme is superior to the MLProph, RLProph, and CAML routing protocols, chosen as benchmarks, in terms of legitimate packet delivery, probability of message delivery, count of dropped messages, and latency in packet delivery. The out-of-bag error obtained is also minimal

KEYWORDS

machine learning, opportunistic IoT networks, random forest classifier, reliability, safety

1 | INTRODUCTION

IoT[1] can be defined as a collection of a large variety of internet connected devices and humans. Besides, OppNets[2] are a class of networks where devices communicate opportunistically, and the routes to carry the message from source to destination are constructed on the fly and on demand. In turn, OppIoT brings the benefits of IoT and OppNets together in a single architecture.[3] OppIoT networks exhibit the association of humans with their smart devices where the opportunistic contact nature of humans and human-based communities is utilized for message transmission.

In such systems, traditional safety primitives such as cryptography and trust-based schemes[4] are unable to handle the network security threats and ensuring superior network performance to a large extent. It has been proven that making use of ML techniques for such purpose can be a very effective solution.[5] Indeed, ML techniques such as Random Decision Forests[6] have been widely applied for improving the network safety and reliability including detection of malware, access control, and device authentication.[7] The random forest classifier (RFC)[8] comprise of a set of decision trees. Each tree is created taking a random sample from the input dataset. In the RFC technique, the randomization is basically attributed to the random sampling of data and random selection of input features for generating the individual base decision trees. During the training phase, multiple decision trees are randomly created using different subsets of data; and the strength of individual decision trees and correlation among base trees is considered as the measure of the generalization error.

In this paper, an RFC-based safe routing protocol (called RFCSec) for OppIoT to ensure better message communication and protect against packet collusion,[9] hypernova and supernova, and wormhole attacks,[10] is proposed, which proactively acts before any damage can be done to the network based on the past behavior of the nodes and a prior classification of the encountered nodes of a given node as malicious or benign. In this process, only those nodes who are benign and have high performance in terms of chosen parameters are qualified to forward the data packets toward its destination. It is worth noting that the packet collusion attack is a form of packet dropping attack, where the attacker tries to mask its dropping behavior by injecting fake packets in lieu of dropped packets. Besides, hypernova and supernova attacks are kinds of flooding attacks where the aim is the consumption of network resources. A supernova attacker will flood random packets in the network while a hypernova attacker will fake the delivery of randomly propagated packets to some unreal nodes for a longer period of time, resulting to resources wastage. Besides, a wormhole is a collaboration attack, in which an attacker tries to attract the maximum of the data packets to itself, then tunnels the captured data to some other node located at a distant point, which replays these data locally. As such, a wormhole peer is likely to be overloaded, causing a system crash due to a single point of failure. Our proposed RFCSec scheme successfully protects the network against the aforementioned attacks. Its main features include (1) the design of a RFC for node characterization as malicious or benign; (2) space and time efficiency by making use of B+ tree[11] for information storage; and (3) use of a hashing scheme[12] for message integrity purpose.

The paper is organized as follows. Section 2 provides some related work. In Section 3, the network and attack models are presented. In Section 3.2, the proposed RFCSec scheme is described. In Section 4, simulation results are presented. Finally, Section 5 concludes the paper.

2 | RELATED WORK

ML-based classification techniques have continuously proved their accuracy in the design of schemes for detecting the malicious attackers of various systems.[13] Alam et al[14] proposed an RF-based classier for detecting malware in android-based smartphones. Their work aimed at classifying the android applications as malicious or benign, and the detection accuracy was measured based on parameters such as number of randomly selected features, count of trees generated, and their depths. However, the overall working is not clear as in how the random forests were generated and on what basis apps were classified as malicious or benign. The features used were collected by running a single application at a time, making the work non-applicable to real-time monitoring of Android devices. Narudin et al[15] proposed an anomaly-based approach for detection of mobile malware. The work presented an extensive evaluation of various machine learning classifiers wherein their effectiveness for malware detection was studied. The work stated that any behavioral pattern varying from normal would be regarded as malignant. But it was not clarified, what feature values were considered as normal and how the anomalies were identified. Similarly, Chen et al[16] introduced a model for protecting against the distributed denial of service (DDos) attack for Domain Name Systems using RF for traffic classification, which successfully discriminated regular queries from abnormal ones, thereby identifying the attackers. The major shortcoming of this work is that it addresses only a single type of attack on DNS.

Gomes et al[17] presented an SDN-based RF classifier for locating the user's position in indoor environments using the signal strength from the wireless devices. Using a security filter, attacks were identified, and the random forest classifier was not allowed to interact with the IoT devices directly. The use of this additional security filter added overhead to the proposed system. Srinidhi[18] proposed a dynamic context information-based multicopy routing protocol for OppIoT, in which RF was used for classifying the nodes as reliable or non-reliable forwarder based on the present context and location information, for making the routing decision. However, the security aspect was not taken into consideration and the proposed system performed poorly in terms of average buffer time as compared to other multiple copy routing schemes. Kopp et al[19] proposed an anomaly detection method based on RF, which was able to identify any deviation of a sample data (qualified as anomaly) from the rest of the data, thereby distinguishing between normal and abnormal behavior of an expert system. The proposed system however had limited capabilities for anomalies that could be detected only when a specific subspace was considered as a whole. Li et al[20] proposed a multiple classifier system based on random forest, principal component analysis (PCA), and potential nearest neighbor. PCA was used for transforming features to add diversity. RF was used as an adaptive learning mechanism of *k* Potential Nearest Neighbors. A voting mechanism was also presented as replacement to the traditional majority vote. The proposed system was used for the detection of automobile insurance fraud. The model is quite complex and the information loss was not taken into consideration. Sharma et al[21] proposed a routing protocol for OppNets that used decision tree and neural networks to determine the probability of successful packet delivery. Their proposed system was trained using the node's predictability value, popularity, power consumption, speed, and location. The simulations were not performed on real dataset, and the security aspect was not addressed. Vashishth et al[22] proposed the use of a cascaded learning-based ML approach for routing in OppIoT. In their scheme, a logistic regression classifier was used as an input cascade to a neural network classifier. The use of multiple classifiers enhances the complexity and computation time. In addition to this, malicious behavior of nodes was not given any consideration.

The work conducted in literature either used too many techniques at a time like multiple classifiers, principal component analysis or assumed the use of additional hardware. The computations were really complex which made them unrealistic for implementation to the wide spectrum of OppIoT devices. In addition to this, the attacks were not proactively identified, and the attackers were not isolated from the routing procedure. There is no clear definition as in what feature values differentiate the normal nodes from the malicious ones. This motivated the authors to propose a model that is simple yet powerful enough to predict the node behavior using the random forest classifier. A detailed discussion is provided regarding the features taken into consideration and what differentiates the benign nodes from the malicious ones. In addition to this, the space and time complexity is addressed using B+ Trees at each node for storing the data. The security of the proposed scheme is further enhanced by using hashing for message security.

3 | DESIGN OF THE RFCSEC PROTOCOL FOR OPPIOT

Security of the OppIoT networks is a major challenge as the network is composed of a huge spectrum of devices, and the data are usually exposed to a wider unfamiliar audience. Most of the current security schemes for OppIoT in the literature are designed based on the prelude that the attacks are already present in the network. This raises the requirement of using ML techniques to preemptively predict the attacks based on the past behavior of nodes and some statistical analysis approach, so that such attacks can be avoided. This requirement justifies our motivation in proposing a RFC-based secured routing protocol (called *(RFCSec)*) for OppIoT to protect against packet collusion,[9] hypernova, supernova, and wormhole attacks.[10]

3.1 | Network and attack models

It is assumed that the OppIoT network is composed of *n* nodes which cooperate in message transmission and which have sufficient buffer and power. For message integrity purpose, each source node computes a hash value using a secure hash function and appends it to the generated message, and if that message is modified, the hash value will also change. Whenever two nodes encounter each other, they share some information such as the number of messages created, message copies, messages received, destination ids, malicious nodes encountered, packets forwarded, and packets dropped. These information are stored using a B+ tree. In our design, a malicious node can perform one of the following attacks:

- **Packet collusion attack**: This is a special form of packet dropping attack, where the attacker tries to mask its dropping behavior by injecting some fake packets in lieu of dropped packets.

- **Wormhole attack**: This is a kind of collaborative attack where an attacker (so-called wormhole) tries to attract the maximum of the data packets to itself, then tunnels them to some other node located at a distant point that it controls, which further replays these data locally.

- **Supernova attack**: This is a kind of flooding attack, where the attacker performs message flooding with the intent of wasting the network resources such as node's buffer space, energy and network bandwidth.

- Hypernova attack: This is a kind of flooding attack, where an attacker floods random messages in the network and pretends to be delivering them to some non-existent nodes. Therefore, these messages stay for a longer period of time in the network, which may lead to greater loss of resources as well.

One of the major challenges faced by OppIoT is the early detection of these attacks, so that they can be isolated from the routing process before causing any harm to the network. The proposed RFCSec scheme classifies the nodes encountered on the basis of their past behavior resulting in a smart selection of the message forwarders.

3.2 | Proposed RFCSec protocol

As the proposed RFCSec design makes use of the RFC design, it is worth presenting how it works.

3.2.1 | RFC design

RFC is an ensemble learning technique, which has been proven to be quite effective in performing the classification, regression, pattern recognition in skewed problems. It has proved to be superior to decision trees as they suffer from high variance. Decision trees are very sensitive to the data on which they are trained, and a small change can drastically impact the results. This is not the case with RFC as this technique comprises several decision trees that are constructed using a random sample drawn with replacement from the input database, leading to highly accurate results. The pruning method used in the RFC, and the selected attributes are used to decide the structure of the decision trees. Each tree then casts a unit vote for the most popular class in oreder to classify an input vector. The final class of the input is decided based on a majority voting mechanism. The working of the RFC is depicted in Figure 1.

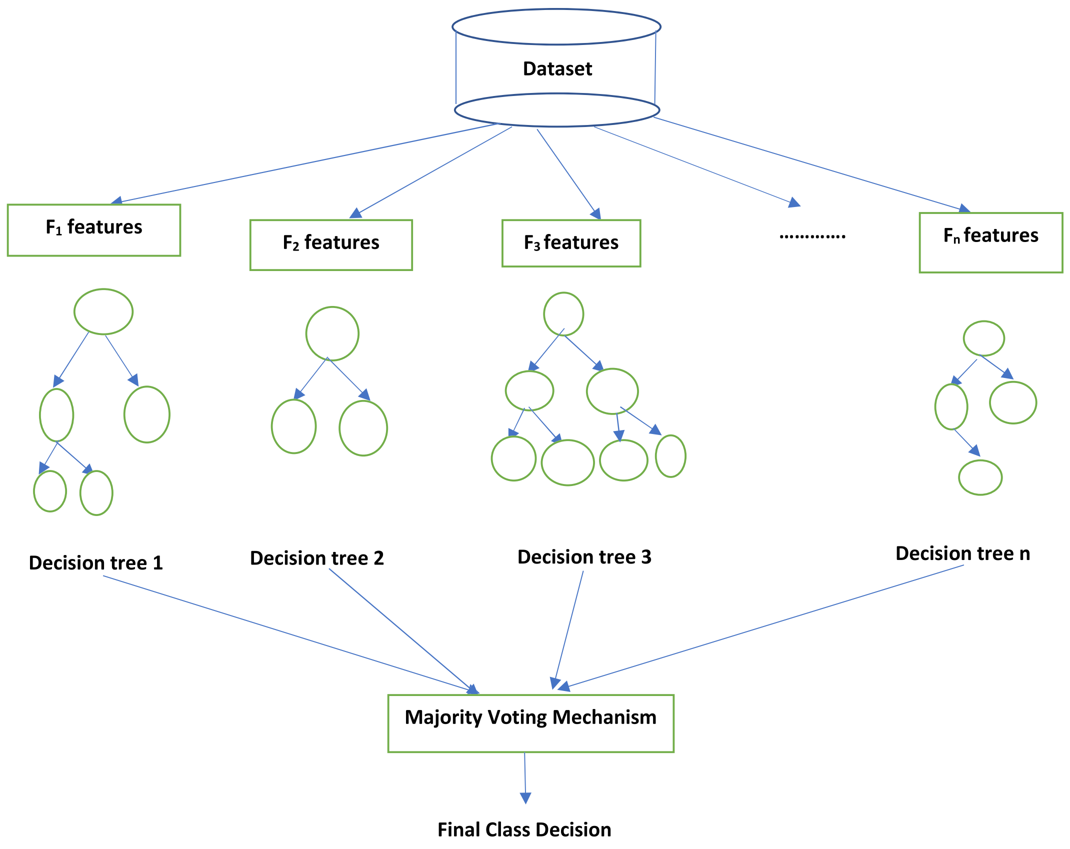

FIGURE 1 Random forest classifier

RFC works according to the following phases:

1. **Data preprocessing:** All the steps to convert the data into a more usable form including noise removal, handling of missing data and data transformation are done in this phase. If required, the data can be normalized and reduced if it is huge. Also, the attribute selection is done so that the most suitable one to be used for designing the protocol are collected. At the end of this phase, the dataset is divided into about 1/3 of it as testing dataset and 2/3 of it as training dataset. Of course, this ratio can be adjusted according to the requirements and data sample at hand.
2. **Training phase :** During this phase, the RFC is trained using the training dataset T such that

$$T = (X_i, Y_i)_{i=1}^{N} \tag{1}$$

where X_i are the set of M features, Y is the class response, and N are the number of training samples. Several decision trees are generated by taking a random sample from T with replacement. This is done by calling the Build_Decision_Tree method shown in Algorithm 1. The final outcome Y is decided based on the majority votes received from each decision tree, i.e.,

$$Y = Majority\_Vote(Y_j)_1^J \tag{2}$$

In this process, the Gini Index (GI)[23] is used as criterion for selecting the attributes as it is meant to measure the data impurity, and it is computed for every attribute of the dataset. If there are k classes in Y and the probability of getting the final output as *i*th class is P_i, then the GI for a given attribute is defined as

$$G.I.(attr) = 1 - \sum_1^k P_i^2 \tag{3}$$

Besides, the splitting attribute is the attribute with the largest reduction in the value of GI. If a decision tree is split into *J* trees, The Gini index split is obtained as

$$G.I._{Split} = \sum_{attr=1}^{F} \left(\frac{Q}{N}\right) * G.I.(attr) \tag{4}$$

where F is the total number of attributes, N is the size of the training dataset, and Q is the size of randomly selected subset.

Algorithm 1 Build_Decision_Tree(T, F, J)

```
1:  Input: Training dataset $T = (X_i, Y_i)_{i=1}^N$
2:  $J$ = Count of trees, $F$ = Number of features, $k$ = Number of classes in Y, $Q$ = Size of subspace and $P_i$ = Probability of getting the final output as $i^{th}$ class.
3:  Output : A set of decision trees.
4:  Begin
5:  for 1 ≤ j ≤ J do
6:      Use bagging and obtain a subset $T_p$ from T.
7:      Randomly select Q, subset of features F.
8:      for 1 ≤ f ≤ Q do
9:          Compute the Gini index (GI) for attribute $attr_f$ as: $G.I.(attr_f) = 1 - \sum_1^k P_i^2$
10:     end for
11:     Choose the splitting attribute as the one with minimum value of $G.I.(attr_f)$.
12:     Continue to split the tree till it grows to a maximum.
13: end for
```

3. **Testing phase:** RF has the lowest classification error compared to other traditional classification algorithms, and the error rate is dependent on the number of features selected per decision tree, correlation between trees, and strength of classification of each tree. In this work, the Out-of-Bag (OOB) error rate is utilized to measure the

performance of RFC on the dataset. During the sampling of the training set, 1/3 of the records (called OOB data) are left out of the sample. This OOB data are then used for performance validation of each tree during this testing phase. The error in the prediction of the class of OOB data is known as OOB, which eliminates the requirement of cross-validation or a separate test. The OOB error is computed as

$$OOB_{Err} = \frac{1}{NOB}\sum_{j=1}^{NOB} ERR(Y, Y_{Prdt}) \tag{5}$$

where NOB is the number of OOB samples, Y is the actual output class, and Y_{Prdt} is the class predicted by the RFC, and *ERR* function is the error function. The lower the OOB_{Err} value is, the better the prediction by the RFC is.

3.2.2 | RFCSec design

The proposed RFCSec works as follows. Whenever a source node generates a message, it uses a hash function for computing the hash value for the generated messages. The computed hash value is then appended to each packet before transmission. As stated earlier, every node maintains a B+ tree for storing the information, including the number of messages generated, number of message copies, number of messages received, message destination ids, number of packets forwarded and dropped, and number of encountered malicious hosts. The major advantage of B+ tree is efficient retrieval of data as they have large number of pointers to the child nodes, which reduces the number of I/O operations required to find an element in the tree. The space complexity for B+ tree is $O(n)$ and the time complexity is $O(\log n)$. B+ tree is even superior to hash tables where the search for an element can result in full table scan $O(n)$. Whenever two nodes encounter each other, the information stored in B+ tree are exchanged among each other in order to keep the tree updated. A decision is then taken whether the data packet should be forwarded to the encountered node or not. A foremost requirement is to check if the encountered node is malicious or benign. To achieve this, the proposed RFCSec scheme makes use of a RF-based classifier for predicting in which class (malicious or benign) an encountered node should belong based on its past behavior.

The following features are considered in the RFC design:

- Incoming packets (f1): This measures the count of data packets received at each node. The packet monitoring at a node is important since a very high value of packets count at a node is an indicator of an unusual behavior. This value could be due to the fact that the considered node is a very suitable carrier to carry the message to its intended destination or it is a malicious node such as a wormhole, which is attempting to attract the maximum traffic towards itself.
- Packets dropped (f2): This measures the number of data packets being dropped at each node in the network. If the count of packets dropped is very high for a particular node in the past, this indicates that it is not a suitable packet carrier and could be a possible malicious node.
- Outgoing packets (f3): This measures the number of packets being forwarded by each node. Every node monitors its neighboring nodes and keeps track of their forwarding behavior. It can be obtained as:

$$Out_{Pkt} = In_{Pkt} - Drop_{Pkt} \tag{6}$$

  So, a high value of Out_{Pkt} could be an indicator of a flooding attack.
- Packets modified (f4): This measures the number of packets being fabricated by a node in the network. Mod_{Pkt} value is calculated by the neighboring nodes. If the hash of the message is modified by a node, this is identified by the neighboring node. Packet fabricating node indicates a possible attacker and it is thus isolated.
- Packets successfully delivered (f5): This measures the number of successful delivered packets to the destination by a particular node. If this value is high, it is highly probable that in the future the chances of getting a packet delivered via that node will also be high.
- Buffer occupancy (f6): This indicates the amount of residual buffer at a node. The lower the occupancy is, the lower are the chances that a packet gets dropped, which makes the node become a suitable carrier for the packet.
- Packet destination count (f7): This is the count of each stored message's destination id. If a node is frequently forwarding most of the packets received to a given node only, this could be an indicator of a possible wormhole attack. Tracking the destination id count is also helpful in isolating the supernova attackers.

- Message copies (f8): This is the count of messages generated in the network along with their replicas. It is also helpful in identifying any type of flooding attacks.

In our proposed RFCSec design, these features are used to determine the output class of a node. Each node in the network can belong to one of the five classes: benign, hypernova, supernova, packet colluding, and wormhole, described as follows:

- **Benign class (c1):** If a node does not perform any malicious activity and cooperates with the neighboring nodes in message forwarding, it belongs to the benign class *c1*. For *c1* class, the value of features *f2, f6* are usually low and that of *f5* is high.
- **Packet colluding class (c2):** The nodes that belong to this class perform packet dropping and in order to prevent their detection, they add fake packets in lieu of real packets. For this class, the value of *f2* is very high and that of *f4* is also high as reported by the neighboring nodes due to the modified hash value.
- **Supernova class (c3):** These nodes flood the network with fake packets. For these nodes, the value of *f3* is way higher than that of *f1* and the value of *f8* is also very high.
- **Hypernova class (c4) :** The nodes of this class not only flood random packets in the network, but deliver them to fake destinations, thereby wasting the network resources. A node can be supernova if it belongs to the *c3* class and has a *f7* high value for certain nodes.
- **Wormhole class (c5):** Wormhole attackers belonging to this class try to attract the maximum traffic towards them and later tunnels them to some other destination node at a remote location that they control. For this class, the *f1* and *f6* values are usually very high and the destination count *f7* is high for specific collaborating nodes.

Using these features, the RFC is trained as stated above. Then, later on, as the simulation continues, a prediction is made on the node's behavior. A node can belong to any of the above defined classes. If the node is voted as benign, i.e. it belongs to *(c1 class)*, then it is qualified as relay node to which a packet can be forwarded. At each hop, the receiving node checks the hash value of the packet that it receives for ensuring the packet integrity. If the message is modified, the hash that was computed will not be as expected and the sender node will be added to the malicious node's list. Thereby, it will be prevented from participating further to the routing procedure. The modified packet will then be dropped to avoid further resource wastage of resources. On the other hand, if the hash value of the received packet is correct, the message routing will continue until the packet eventually reaches its intended destination. The complete routing procedure and malicious node detection are described in Algorithm 2.

4 | SIMULATION RESULTS

The proposed RFCSec protocol is simulated using the ONE simulator,[24] real data traces, and MalGenome,[25] a public dataset composed of a collection of 1,260 malware categorized in 49 different classes collated from 2010 to 2011. The Weka ML library[26] is used to train and deploy the RFC for the implementation of the RFCSec scheme.

4.1 | Simulation setup

The performance of the proposed RFCSec protocol is compared against that of the MLProph,[21] CAML,[22] and RLProph[27] protocols under varying number of nodes, percentage of malicious nodes in network, time-to-live (TTL) of packets, and buffer size of nodes. The considered performance metrics are average number of packets dropped, probability of message delivery, average delay and number of correct packets delivered to the destination.

- **Average packets dropped:** This metric represents the average count of packets dropped in the network from node's buffers during routing. A low value of this metric is desirable.
- **Message delivery probability:** This metric is an indicator of the number of messages getting delivered to their destination nodes successfully. This metric should be high for an efficient routing protocol.
- **Average delay:** This metric represents the average latency (delay) incurred in the message delivery to the destination nodes. This delay could be due to non availability of good carrier node that delivers or brings the message closer to the destination node. A low value of average latency is desirable.

Algorithm 2 RFCSec

```
1: Begin
2: Phase I: Initialization
3: Initialize the B+ tree of all hosts.
4: Calculate the hash of each message generated by the sender and append it to the packets.
5: Obtain the training dataset, $T = (X_i, Y_i)_{i=1}^{N}$ by drawing in-of-bag sample from dataset D.
6: Obtain the feature set, F.

   $F = (In\_Pkt, Pkt\_drop, Out\_Pkt, Pkt\_mdfy, Pkt\_Delvd, Pkt\_Destn\_Cnt, Msg\_copy)$.
7: Input the number of decision trees to be generated, $J = 1000$.


8: Phase II: Generation of the RFC
9: Call Build_Decision_Tree(T, F, J) for constructing and training RFC.
10: Test the Random forests generated using the remaining out-of-bag sample.


   $OOB_S = D - T$.
11: Choose the RF with the minimum out-of-bag error.

   $OOB_{Err} = \frac{1}{NOB} \sum_{j=1}^{NOB} ERR(Y, Y_{Prdt})$

12: Phase III: Identification of Encountered Node's Class
13: if $(B + Tree.Malicious\_Node$ contains $(E))$ then
14:     E has already been declared malicious, wait for a better carrier.
15: else
16:     Call RFC for predicting E's class.
17:     if (Node.Class IN (c2, c3, c4,c5)) then
18:         Malicious node encountered.
19:         Add node as $Malicious\_Node$ to the B+ Tree.
20:         Wait for a better forwarder till $packet.TTL$ expires.
21:     else
22:         Forward the packet to the benign node encountered.
23:     end if
24:
25: end if

27: Phase IV : Checking the packet integrity
28: for each packet p received at neighboring node do
29:     if (p.Hash == Computed_Hash(p)) then
30:         Packet Integrity intact.
31:         if $(p.Destination == Receiving\_Node)$ then
32:             Destination reached.
33:         else
34:             Continue routing, Goto Phase III
35:         end if
36:     else
37:         Packet p altered, Add Sender(p) to $B + Tree.Malicious\_Node$.
38:         Drop p.
39:         Goto Phase III until Destination is reached for all packets.
40:     end if
41: end for
```

- **Correct packet delivery:** This metric represents the count of messages getting correctly (non-fabricated) delivered to the destination nodes. A higher value indicates an efficient and integral routing protocol.

To assess the security performance of the studied protocols in the presence of malicious nodes, the OOB error and F1 score are used as metrics; the lower the value of the OOB error, the better the performance of the routing protocol is. A true positive t_p is obtained as outcome when the studied protocol predicts correctly if a node belongs to the malicious class. On the other hand, a true negative t_n is obtained as outcome when the studied protocol predicts correctly if a node belongs to the benign class. A false positive f_p is an outcome when the studied protocol makes an incorrect prediction about the benign class and classifies the node as malicious; and a False negative f_n is an outcome when the studied protocol incorrectly classifies a malicious node as benign. More precisely:

$$TruePositiveScore = \frac{t_p}{t_p + f_n} \tag{7}$$

$$FalsePositiveScore = \frac{f_p}{t_n + f_p} \tag{8}$$

$$TrueNegativeScore = \frac{t_n}{t_n + f_p} \tag{9}$$

$$FalseNegativeScore = \frac{f_n}{t_p + f_n} \tag{10}$$

Using the above-defined scores, the *Precision* metric is calculated as the ratio of the actual number of malicious nodes that are correctly classified to the total number of nodes that are classified as malicious.

$$Precision = \frac{t_p}{t_p + f_p} \tag{11}$$

The Recall metric is the ratio of the number of malicious nodes that are classified correctly to the total number of malicious nodes being classified correctly as malicious or incorrectly as benign.

$$Recall = \frac{t_p}{t_p + f_n} \tag{12}$$

Now, taking the harmonic mean of the Precision and Recall metrics, the F-Score is obtained, i.e.:

$$F\_Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{13}$$

A high value of *F_Score* is an indicator of a good performance of the classifier. It is also assumed that the network is composed of *n* nodes with a buffer size of 100 MB and the simulation area is set as 1000 m $*$ 1000 m. In addition, the Random Movement Model is used for depicting the movement of the nodes as obtained from the *MalGenome* dataset. The Bluetooth scanning granularity is 120 per second; the TTL for a message is 100 min and each simulation runs for 337 418 s. A message of size 500 Kb–1 Mb is created every 25–35 s.

Whenever a sender generates a message, the hash of the message is calculated using the SHA-256 algorithm,[12] which is then appended to the data packets. The search for a better carrier is then initiated until the packet reaches its intended destination. The simulation is performed in two phases: training phase and simulation phase. At the beginning of the training phase, all the nodes are assumed to be benign. As the simulation continues, information such as number of messages generated, count of message copies, number of messages received, messages destination ids, number of packets forwarded and dropped, and number of malicious hosts encountered, are stored in the B+ Tree at each node.

Whenever a node encounters another node, this information is shared among them to keep their values updated. These values stored in a B+ Tree at a given instance of time are shown in Table 1. From that table, it is clear that *n1* is a possible *Supernova* attacker as it is flooding the network with random packets. Similarly, *n5* is a possible *Hypernova* attacker as it is flooding the network with some packets, redirecting them simultaneously to some non existent node in the network. A possible instance of *Wormhole* attack is executed by nodes *n2*and *n121.* As for *packet collusion* attack, it is very difficult to detect as it tries to mask the packets dropped by inserting new packets as replacement. Nonetheless, a higher count of packets dropped like shown by *n124* could be an indicator of a possible packet collusion attack by that node.

During the training phase, the RFC is trained on the chosen feature set for making an accurate prediction regarding the correct class of the encountered node. The Weka library is used for this purpose. Besides, OOB_{Error} and *F_Score* are used for checking the accuracy of the predictions made. Table 2 represents the value of these quality metrics calculated by varying the number of decision trees and feature set size. Figure 2 depicts a graphical representation of the same. Next, the Random forest with the lowest OOB_{Error} and high *F_Score* is chosen for performing the training for further simulations. RFC then predicts the class where the encountered node belongs. If the neighboring node is benign, then the packet is forwarded to it. The packet's integrity is then checked at the receiving node side. If the packet has been altered, it is dropped and the sender node is added as malicious host to the B+ Tree. Otherwise, the message routing is continued till it reaches its destination.

TABLE 1 B+ tree values at a node at a given time instance

Node	*Msgs_Created*	*Msg_Copies*	*Msg_Received*	*Msg_Destn_Id*	*Pkt_Forwarded*	*Pkt_Dropped*	*Mal_Hosts*
n0	4	109	322	n2; n8; n65; n97	424	28	n1; n5; n28
n1	100	1000	28	n0; n1; …; n65; n97	1032	8	n5; n28; n87
n2	1	5	485	n96	485	1	n1; n28
n4	28	50	150	n10; n51; n72	204	15	n1; n2; n28
n5	10	2008	78	n509; n230; n345	345	18	n2; n28
…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…
n121	2	10	481		1	1	n1; n5
n122	40	220	45	n0; n95; n4; n100	390	60	n1; n2; n5
n123	78	450	59	n1; n8; n6; n10	424	28	n7; n28
n124	56	700	32	n5; n65	650	500	n1; n28; n5
n125	7	69	12	n45; n77	59	5	n2; n5

TABLE 2 Impact of the number of trees and features on the quality metrics

Trees	Features	OOB_{Error}	*tpr*	*tnr*	*fpr*	*fnr*	*Precision*	*Recall*	*F_Score*
500	4	0.0072	0.71	0.72	0.29	0.28	0.71	0.71	0.709
500	6	0.0071	0.75	0.77	0.25	0.23	0.75	0.765	0.765
500	8	0.0069	0.79	0.8	0.21	0.2	0.79	0.797	0.793
700	4	0.0070	0.81	0.82	0.19	0.18	0.81	0.81	0.81
700	6	0.0068	0.83	0.85	0.17	0.15	0.83	0.84	0.835
700	8	0.0067	0.85	0.86	0.15	0.14	0.85	0.858	0.853
900	4	0.0069	0.86	0.87	0.14	0.13	0.86	0.868	0.863
900	6	0.0067	0.89	0.91	0.11	0.09	0.89	0.9	0.894
900	8	0.0065	0.92	0.93	0.08	0.07	0.92	0.929	0.924
1000	4	0.0066	0.93	0.95	0.07	0.05	0.93	0.948	0.938
1000	6	0.0063	0.94	0.96	0.06	0.04	0.94	0.959	0.949
1000	8	0.0060	0.96	0.98	0.04	0.02	0.96	0.979	0.969

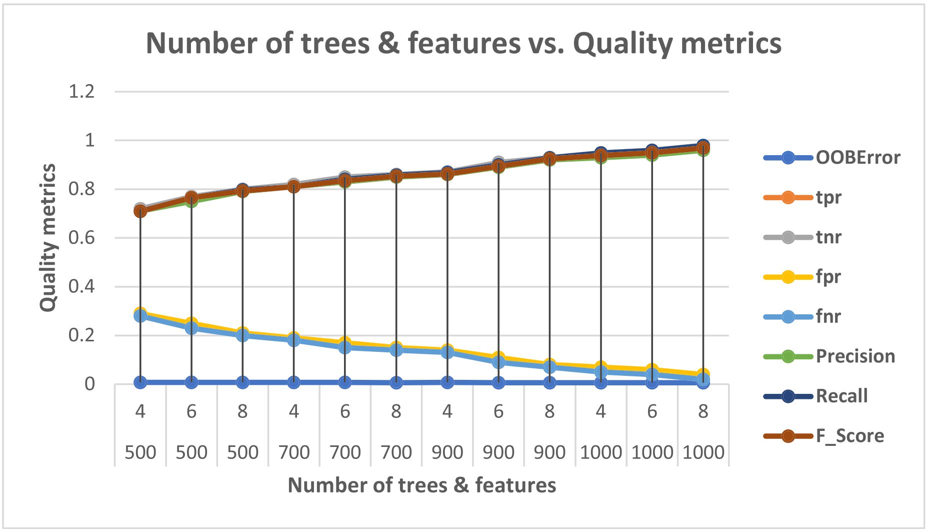

FIGURE 2 Number of decision trees and feature set size vs. quality metrics

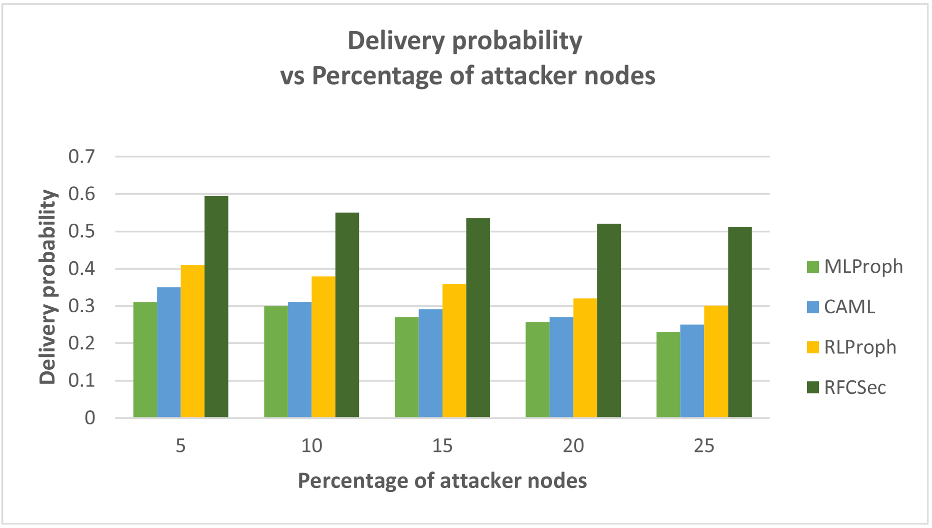

FIGURE 3 Delivery probability vs. percentage of attackers

4.2 | Simulation results

First, the number of malicious nodes in the network is varied from 5% to 25%. Figure 3 shows the impact of this variation on the packet delivery probability. It is observed that, when the number of malicious nodes in the network rises, the delivery probability falls for all studied protocols. RFCSec is designed to act proactively and isolate malicious attackers from participation in the routing of packets. Due to this isolation, packets are not forwarded to the malicious nodes, which helps in achieving high probability despite the rise in percentage of attackers in the network. It is noted that the average delivery probability obtained by RFCSec is 0.542, which is 53.4% better than that generated by RLProph, 84.3% better than that of CAML and 96.5% better than that of MLProph.

Second, the number of malicious nodes in the network is varied, and the impact of this variation on the number of legitimate packets reaching at the destination is investigated. The results are shown in Figure 4. It is observed that as the number of attackers rises, the number of legitimate packets reaching the destination tend to fall as more and more attackers are attempting to modify the data packets. The average count of legitimate packets getting delivered to the destination for RFCSec are the highest (i.e. 2038 in average), which is about 13.75% higher compared to that generated by RLProph, 21.04% higher compared to that obtained using CAML and 44.5% higher than that generated by MLProph.

Third, the number of malicious nodes in the network is varied and the impact of this variation on the number of dropped messages is studied. The results are shown in Figure 5. It is clear that with the rising of malicious nodes in the network, the total number of packets getting dropped also rises. It is found that the average number of packets dropped

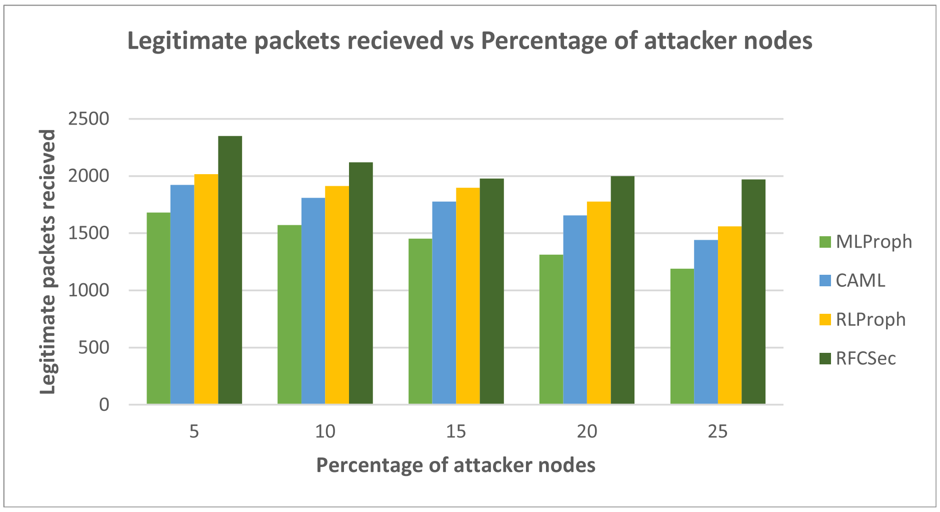

FIGURE 4 Legitimate packets received vs. percentage of attackers

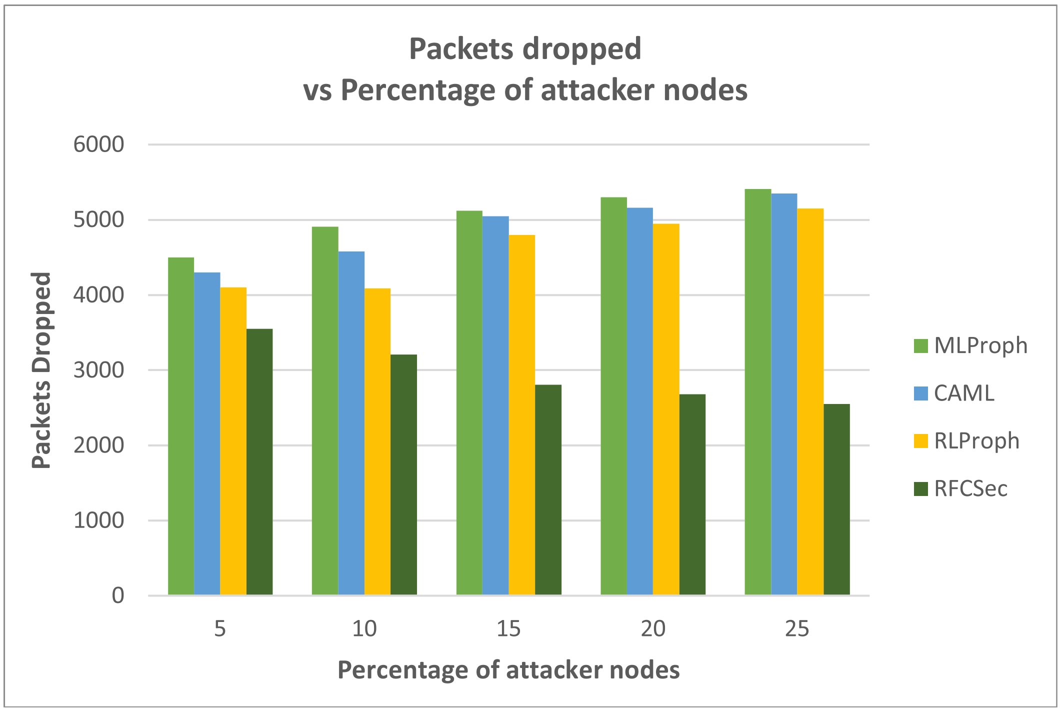

FIGURE 5 Messages dropped vs. percentage of attackers

for RFCSec is about 35.9% lower than that obtained using RLProph, 39.44% lower than that obtained with CAML and 41.36% lower than that obtained with MLProph.

Fourth, the impact of varying number of malicious nodes on the delay observed in the message delivery is investigated and the results are shown in Figure 6. It is observed that the average latency increases when the number of malicious nodes is increased. This might be due to the fact that the nodes have to spend more time identifying the benign nodes for message forwarding purpose. It is also observed that the average latency for RFCSec is 15.8% better than that of RLProph, 30.3% lower than that obtained for CAML and 28.77% lower than that generated using MLProph.

Table 3 summarizes the results obtained in Figure 3 to 6. It is obvious that our proposed RFCSec scheme secures the network against the studied attacks and produces superior results overall compared to the other protocols, in terms of the studied performance metrics.

Fifth, the percentage of malicious nodes is fixed to 5% and the buffer size of nodes is varied. The results are given in Figure 7. It is observed that when the buffer size is increased, the packet delivery is improved since an increased

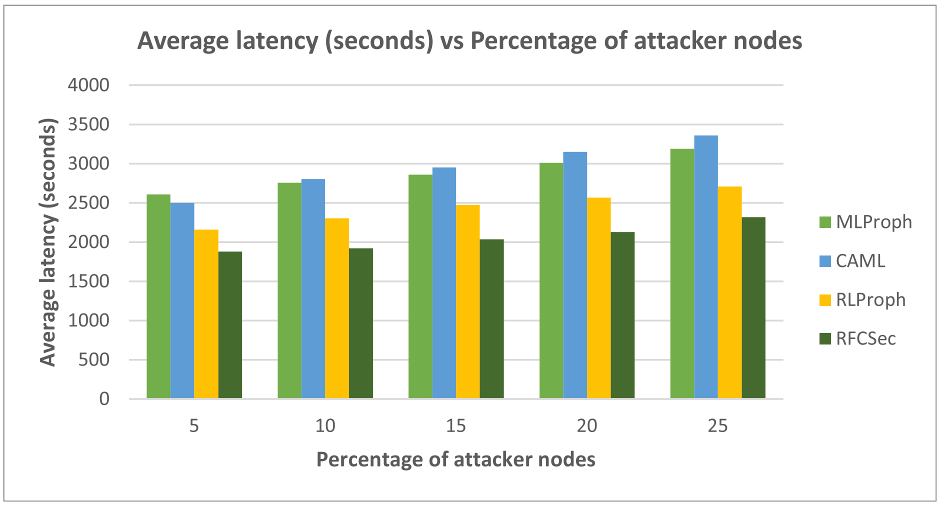

FIGURE 6 Average latency vs. percentage of attackers

TABLE 3 Average metric values across changing percentage of malicious nodes

Protocol	Average delivery probability	Average messages dropped	Average number of legitimate packets received	Average latency (seconds)
MLProph	0.276	5048	1441	2885.21
CAML	0.294	4888	1720	2952.41
RLProph	0.354	4618	1832	2441.11
RFCSec	0.542	2960	2083	2054.97

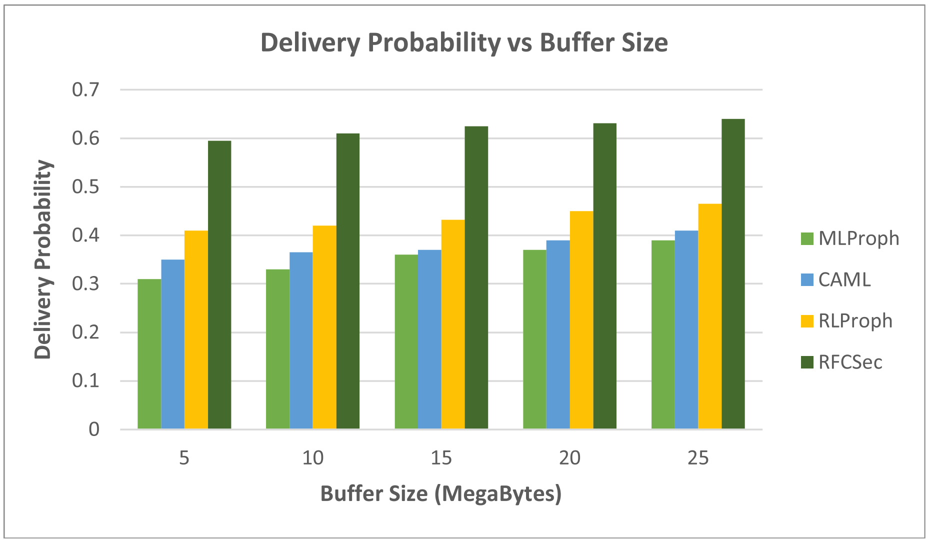

FIGURE 7 Delivery probability vs. buffer size

number of messages now reside in the buffer, which ultimately increases the packet delivery probability. The average packet delivery probability for RFCSec is 0.62, which is 42.4% superior to that produced by RLProph, 64.5% better than that generated by CAML and 76.1% better than that obtained using MLProph. Besides, Figure 8 shows the impact of varying buffer size on the number of legitimate packets received at destination. It is clear that the number of legitimate packets received at destination increases when the buffer size is increased as the overall packet delivery is improved. In fact, the average number of legitimate packets delivered for RFCSec is 13.6% higher than that obtained by RLProph, 21.04% higher than that obtained using CAML, and 44.5% better than that obtained using MLProph.

Sixth, the impact of varying buffer size on the number of packets dropped in the network is studied and the results are captured in Figure 9. It is observed that the number of packets dropped falls when the buffer size is increased

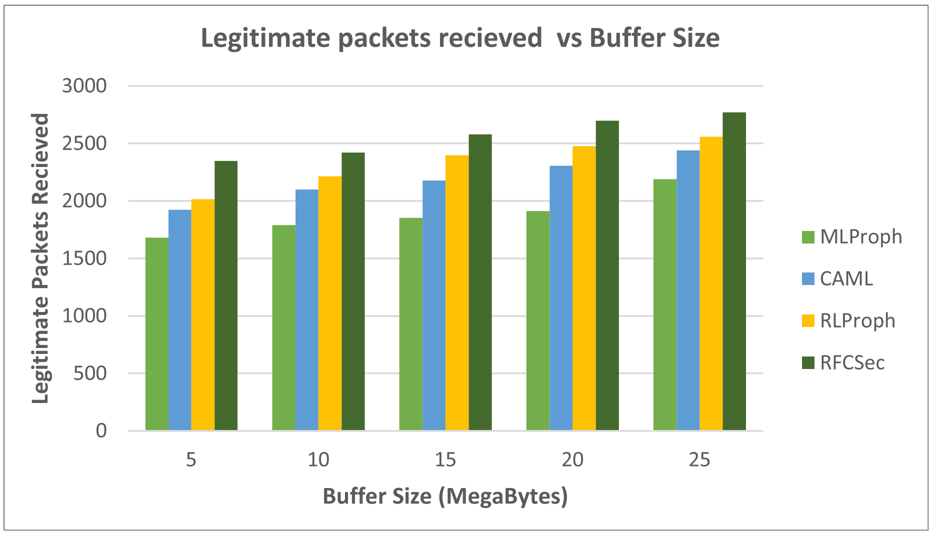

FIGURE 8 Correct packets received vs. buffer size

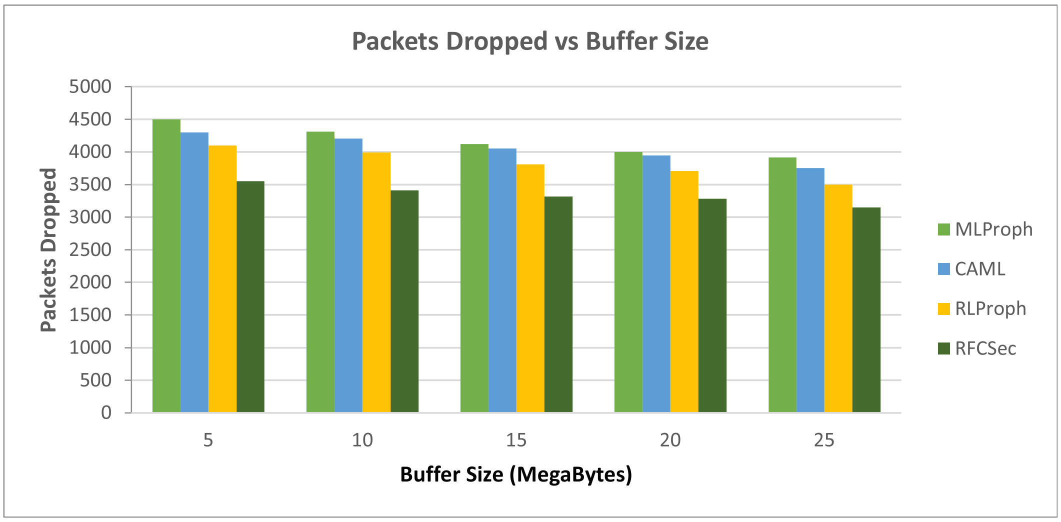

FIGURE 9 Messages dropped vs. buffer size

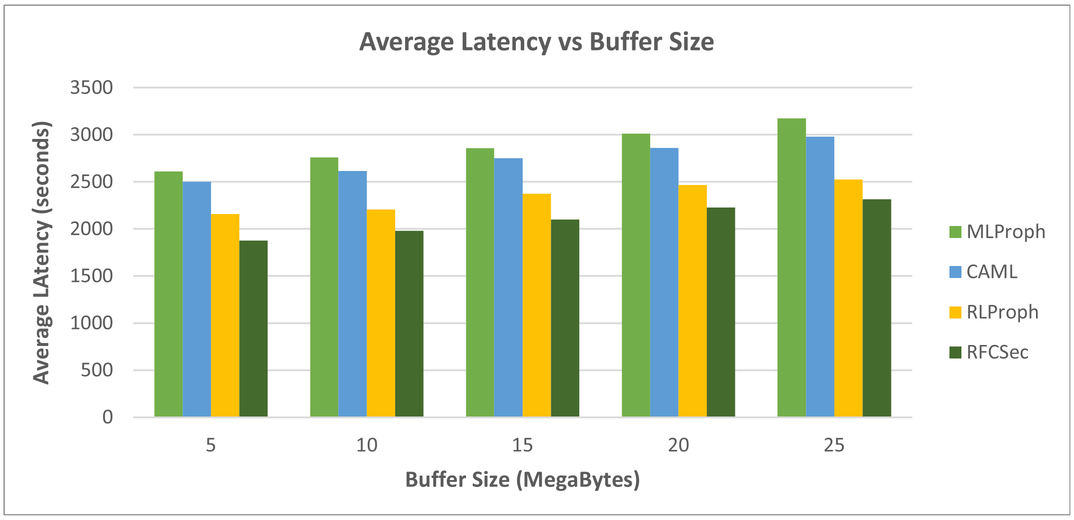

FIGURE 10 Average latency vs. buffer size

because with an enhanced buffer capacity, more packets end up residing in the buffer. The average number of packets dropped for RFCSec is about 35.9% lower than that of RLProph, 39.4% lower than that of CAML, and 41.3% lower than that of MLProph. Besides, Figure 10 shows the impact of varying buffer size on the latency in message delivery. It is found that the average latency for RFCSec is about 15.8% lower than that of RLProph, 30.3% lower than that of CAML, and 28.7% lower than that of MLProph. Table 4 summarizes the above results obtained under varying buffer size.

Finally, the message TTL is varied, and the impact of this variation on the message delivery probability is investigated. The results are illustrated in Figure 11. It is observed that as the TTL is increased, the average probability of message delivery tends to fall. This is due to the fact that the messages are now residing in the node's buffer for a longer

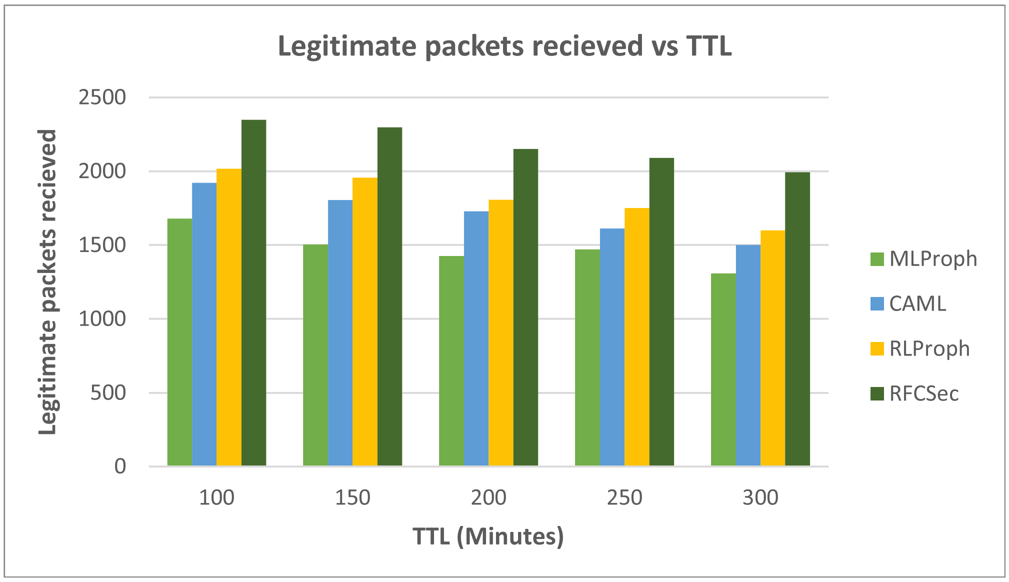

FIGURE 12 Correct packets received vs. TTL

TABLE 4 Average metric values across changing buffer size

Protocol	Average delivery probability	Average number of messages dropped	Average number of legitimate packets received	Average latency (seconds)
MLProph	0.352	4170	1885	2881.26
CAML	0.377	4051	2189	2741.41
RLProph	0.4354	3821	2332	2346.11
RFCSec	0.6202	3342	2563	2100.17

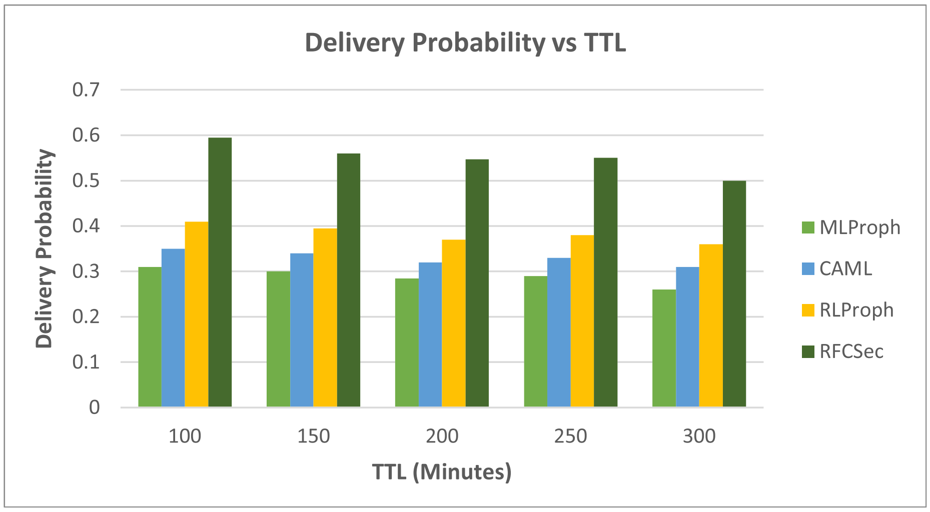

FIGURE 11 Delivery probability vs. TTL

period of time, enhancing its probability to get dropped as buffer is now occupied to a greater extent. It is found that the average message delivery probability for RFCSec is 0.554, which is about 43.7% higher than that obtained for RLProph, 40.04% higher than that obtained for CAML, and 90.4% higher than that obtained for MLProph. Besides, when the TTL is varied, the effect of this variation on the number of legitimate packets received at destination is shown in Figure 12. It is found that this number is for RFCSec is about 19.12% higher compared to that obtained using RLProph, 26.9% higher compared to that obtained with CAML and 47.2% higher compared to that obtained with MLProph. Besides, the number of packets dropped in the network when the TTL is varied is investigated; and the results are given in Figure 13. It is observed that the overall count of packets getting dropped in the network rises when the TTL of packets is increased. This is due to the fact that the packets are residing in the buffer for longer period of time. In fact, the average number of packets dropped for RFCSec is about 11.8% lower than that generated by RLProph, 16.13% lower than that generated by CAML, and 18.24% lower than that generated by MLProph. Finally, Figure 14 shows the impact of varying TTL on the delay observed in packet delivery. It is observed that when the TTL is increased, the average latency tends to increase as well since the packets are getting dropped by the node's. Indeed, the average latency obtained using RFCSec is about 13.39% lower than that generated by RLProph, 18.27% better than that

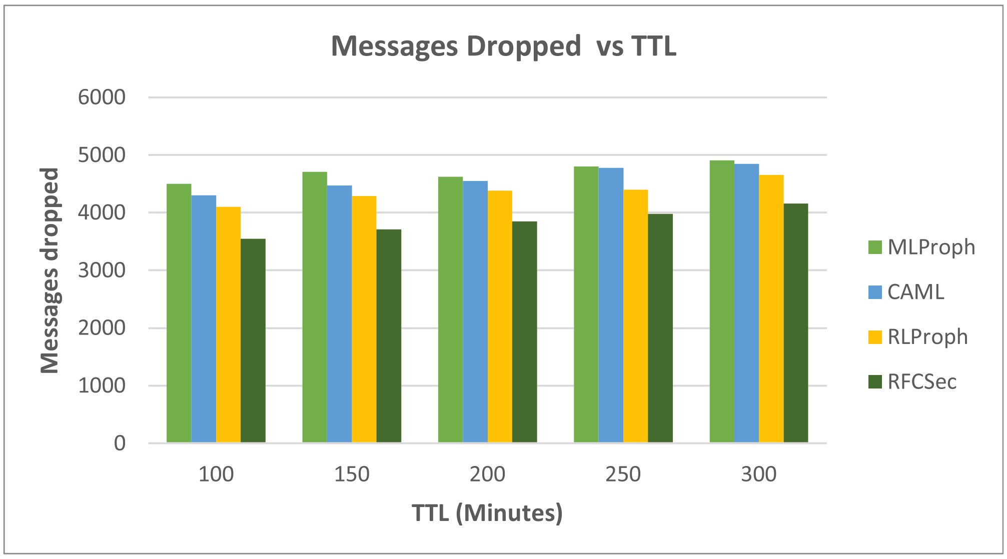

FIGURE 13 Number of messages dropped vs. TTL

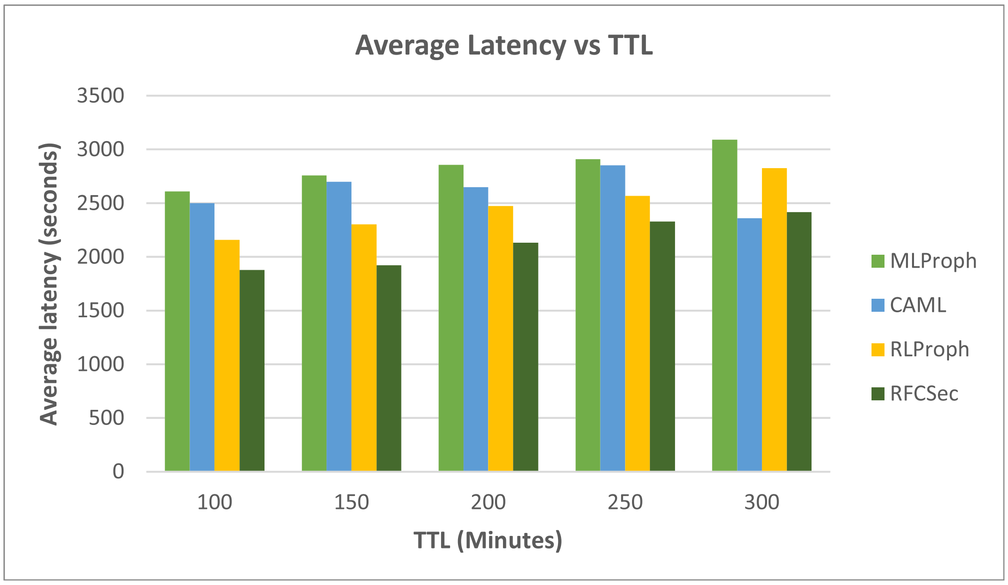

FIGURE 14 Average latency vs. TTL

TABLE 5 Average metric values across changing TTL

Protocol	Average delivery probability	Average number of messages dropped	Average number of legitimate packets received	Average latency (seconds)
MLProph	0.289	4708	1478	2845.21
CAML	0.33	4590	1714	2612.41
RLProph	0.383	4366	1827	2465.11
RFCSec	0.5504	3849	2176	2134.97

TABLE 6 Performance improvements by RFCSec

Metric	RLProph	CAML	MLProph
Delivery probability vs. malicious node %age	53.4%	84.3%	96.5%
Messages dropped vs. malicious node %age	35.9%	39.44%	41.36%
Legitimate packets delivered vs. malicious node %age	13.75%	21.04%	44.5%
Average latency vs. malicious node %age	15.8%	30.3%	28.77%
Delivery probability vs. buffer size	42.4%	64.5%	76.1%
Messages dropped vs. buffer size	35.9%	39.4%	41.3%
Legitimate packets delivered vs. buffer size	13.6%	21.04%	44.5%
Average latency vs. buffer size	15.8%	30.3%	28.7%
Delivery probability vs. TTL	43.7%	40.04%	90.4%
Messages dropped vs. TTL	11.8%	16.13%	18.24%
Legitimate packets delivered vs. TTL	19.12%	26.9%	47.2%
Latency vs. TTL	13.39%	18.27%	24.96%

obtained using CAML, and 24.96% lower than that generated using MLProph. Table 5 summarizes the above results obtained under varying TTL.

For all the above simulations, it was found that for our proposed RFCSec scheme, the OOB_{Error} is as low as 0.6%, and the *F_Score* is above 85% on average, which is an indication of a great performance in terms of accuracy of prediction. Table 6 summarizes the performance improvements achieved by RFCSec over RLProph, CAML, and MLProph.

5 | CONCLUSION

In this paper, a novel RFC-based safe and reliable routing protocol for OppIoT network has been proposed, which ensures message integrity by getting each sender node append a hash value to the message, which is then checked by each intermediate node. In this protocol, the space and time efficiency is also improved by using a B+ tree instead of a list to save the node's information; and the RFC is invoked for predicting the class to which any encountered node should belong (benign or malicious). Simulation results have shown that the Out-of-bag error obtained is minimal. In addition, our proposed RFCSec is superior to the other studied routing protocols in terms of delivery rate, count of packets dropped, delay in message delivery, and count of legitimate packets delivered to destination. As future work, our proposed RFCSec scheme can be further enhanced by including the user authorization and message encryption within its design. In addition, in the RFCSec scheme, the use of B+ tree for saving the information at the nodes is an important feature since it helps saving a lot of buffer space at each node and access time is much lower. Keeping these B+ trees up-to-date is a challenge since it adds some overhead to the network. It would be desirable to quantify this overhead.

The data that support the findings of this study are derived from public dataset, *MalGenome* available at https://www.malgenomeproject.org/, reference number.[25]

ORCID

Nisha Kandhoul https://orcid.org/0000-0003-0977-2982

REFERENCES

1. Atzori L, Iera A, Morabito G. The internet of things: a survey. *Comput Netw*. 2010;54(15):2787–2805.
2. Pelusi L, Passarella A, Conti M. Opportunistic networking: data forwarding in disconnected mobile ad hoc networks. *IEEE Commun Mag*. 2006;44(11):134–141.
3. Guo B, Zhang D, Wang Z, Yu Z, Zhou X. Opportunistic IoT: exploring the harmonious interaction between human and the internet of things. *J Netw Comput Appl*. 2013;36(6):1531–1539.
4. Paul AB, Biswas S, Nandi S, Chakraborty S. Matem: A unified framework based on trust and MCDM for assuring security, reliability and QoS in DTN routing. *J Netw Comput Appl*. 2018;104:1–20.
5. Canedo J, Skjellum A. Using machine learning to secure IoT systems. In: 14th Annual Conference on Privacy, Security and Trust (PST) IEEE; 2016:219–222.
6. Ho TK. Random decision forests. In: Proc. of 3rd International Conference on Document Analysis and Recognition, Vol. 1 IEEE; 1995: 278–282.
7. Ozay M, Esnaola I, Vural FTY, Kulkarni SR, Poor HV. Machine learning methods for attack detection in the smart grid. *IEEE Trans Neural Net Lear Syst*. 2015;27(8):1773–1786.
8. Breiman L. Random forests. *Mach Learn*. 2001;45(1):5–32.
9. Alajeely M, Doss R, Mak-Hau V, et al. Defense against packet collusion attacks in opportunistic networks. *Comput & Secur*. 2017;65: 269–282.
10. Dhurandher SK, Kumar A, Woungang I, Obaidat MS. Supernova and hypernova misbehavior detection scheme for opportunistic networks. In: IEEE 31st International Conference on Advanced Information Networking and Applications (AINA) IEEE; 2017:387–391.
11. B+ tree. https://en.wikipedia.org/wiki/B%2B_tree; 2020.
12. Appel AW. Verification of a cryptographic primitive: Sha-256. *ACM Trans on Program Lang Syst*. 2015;37(2):1–31.
13. Sharma S, Krishna CR, Sahay SK. *Detection of advanced malware by machine learning techniques*. Springer; 2019:333–342.
14. Alam MS, Vuong ST. Random forest classification for detecting android malware. In: IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing IEEE; 2013:663–669.
15. Narudin FA, Feizollah A, Anuar NB, Gani A. Evaluation of machine learning classifiers for mobile malware detection. *Soft Comput*. 2016;20(1):343–357.
16. Chen L, Zhang Y, Zhao Q, Geng G, Yan Z. Detection of DNS DDOS attacks with random forest algorithm on spark. *Procedia Comput Sci*. 2018;134:310–315.
17. Gomes R, Ahsan M, Denton A. Random forest classifier in SDN framework for user-based indoor localization. In: IEEE International Conference on Electro/Information Technology (EIT) IEEE; 2018:537–542.
18. Srinidhi NN, Sagar CS, Chethan SD, Shreyas J, Kumar SMDilip. Machine learning based efficient multi-copy routing for OppIoT networks. In: International Conference on Computational Intelligence, Security and Internet of Things Springer; 2019:288–302.
19. Kopp M, Pevnyċ T, Holeňa M. Anomaly explanation with random forests. *Expert Syst App*. 2020:113187.
20. Li Y, Yan C, Liu W, Li M. A principle component analysis-based random forest with the potential nearest neighbor method for automobile insurance fraud identification. *Appl Soft Comput*. 2018;70:1000–1009.
21. Sharma DK, Dhurandher SK, Woungang I, Srivastava RK, Mohananey A, Rodrigues JJPC. A machine learning-based protocol for efficient routing in opportunistic networks. *IEEE Syst J*. 2016;12(3):2207–2213.
22. Vashishth V, Chhabra A, Sharma DK. A machine learning approach using classifier cascades for optimal routing in opportunistic internet of things networks. In: 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON) IEEE; 2019:1–9.
23. Shouman M, Turner T, Stocker R. Using decision tree for diagnosing heart disease patients. In: Proc. of the Ninth Australasian Data Mining Conference, Vol. 121; 2011:23–30.
24. Keränen A, Ott J, Kärkkäinen T. The one simulator for DTN protocol evaluation. In: Proc. of the 2nd International Conference on Simulation Tools and Techniques ICST; 2009:1–10.
25. Zhou Y, Jiang X. Dissecting android malware: Characterization and evolution. In: 2012 IEEE Symposium on Security and Privacy IEEE; 2012:95–109.
26. Livingston F. Implementing Breiman's random forest algorithm into Weka. In: Ece591q Machine Learning Conference Papers, Vol. 27 Citeseer; 2005:1–5.
27. Sharma DK, Rodrigues JJ, Vashishth V, Khanna A, Chhabra A. Rlproph: a dynamic programming based reinforcement learning approach for optimal routing in opportunistic IoT networks. *Wireless Netw*. 2020:4319–4338.

How to cite this article: Kandhoul N, Dhurandher SK, Woungang I. Random forest classifier-based safe and reliable routing for opportunistic IoT networks. *Int J Commun Syst*. 2020;e4646. https://doi.org/10.1002/dac.4646