Received: 15 January 2019 | Revised: 15 July 2019 | Accepted: 11 August 2019
DOI: 10.1002/ett.3753

RESEARCH ARTICLE

WILEY

A probability distribution function for investigating node infection and removal times

Tala Tafazzoli[1] | **Babak Sadeghiyan**[2]

[1]ICT Security Faculty, ICT Research Institute (ITRC), Tehran, Iran

[2]Computer and IT Department, Amirkabir University of Technology, Tehran, Iran

Correspondence
Tala Tafazzoli, ICT Security Faculty, ICT Research Institute (ITRC), Tehran, Iran.
Email: tafazoli@itrc.ac.ir

Present Address
End of Kargar Ave., Tehran, Iran

Abstract

Considering the important issue of computer infections by worms spread via networks, the theme of source identification has been a prominent research field that aims at investigating infection propagation including acquiring knowledge about the infection and the node removal times when a worm infection happens. This information helps in identifying the patient zero in the worm attack and may be used by computer forensic investigators and network administrators to spot the culprits and to identify related network vulnerabilities. In this paper, we tackle this problem by developing new probabilistic models based on Bayesian networks. We learn a probability distribution to calculate, at every time step, the probability that each node is infected by a scanning worm, using historical data and features extracted from the network and application layers. With the mentioned probability distribution, the node infection status can be inferred using feature values at each time step. We propose a four-step method to investigate the time of infection and removal of each node probabilistically. First, features are extracted and derived from network traffic data. There are no suitable training and test datasets publicly available for our tests; therefore, we developed the training and test datasets using simulations of the Code Red II worm. Second, a prior model is built using training data. Third, the probabilistic model is built by the estimation of distribution algorithm. Fourth, the infection probability of nodes is inferred given the probability distribution and feature values at each time step. It has already been shown that the number of infectious nodes can be probabilistically approximated backward in time through the stochastic Back-to-Origin Markov model. We combine our first model with the prior stochastic Back-to-Origin Markov model to develop our second model. To evaluate our first and second models, we conducted experiments that show that these models can pinpoint the source node and the infection time of nodes with acceptable accuracy. It should be noted that our method could be employed with other propagating worm types including ransomware worms.

1 | INTRODUCTION

Computer infections such as worms can rapidly spread via networks and cause damage to computers or can compromise them. The existing similarities between the spread of computer worms and diffusion processes have encouraged

Abbreviations: Computer worms, EDA, Infection time, Probability distribution function, Removal time.

researchers to adopt common methods to identify the origins of worm propagations.[1] Solving the source identification problem is of great interest in the field of network forensic investigation and administration. For example, finding the computers that originate a worm in a network may help to identify the culprit and detect security vulnerabilities in the network that have been exploited by the worm.

In the past few years, researchers have addressed the problem of identifying propagation sources in networks. In a previous study,[1] 13 source identification methods have been analyzed and classified into three main categories. These methods require either high computational complexity or high storage capacity or make some simplifications to obtain suboptimal solutions.[1] In order to overcome these limitations, here, we have proposed a new probabilistic approach.

To identify the worm origin, we adopted a step-by-step approach. In a previous research,[2] three models had been developed to probabilistically estimate the number of infectious nodes at each step of worm propagation, back in time, namely the deterministic Back-to-Origin model, stochastic Back-to-Origin model, and stochastic Back-to-Origin Markov model. In the Back-to-Origin approach, random scanning worms that are modeled using the Susceptible, Infected, and Removed (SIR) model, in which all nodes are susceptible except for one node, may subsequently become infected due to contact with infected nodes. Infectious nodes may be removed with some probability. According to the common characteristics of a random scanning worm and SIR epidemic model, in the present study, SIR modeling was used to assess the propagation of worm similar to another research.[3] We calculate the infection probability of the preferential scanning worms at each time step when time runs backwards.

In this paper, we have proposed a new method to automatically identify the time when the worm initially starts spreading in the network, so as to find the probability of infection and removal of the nodes at all times. The nodes which have been infected before others may be the origins of worm propagation. In order to achieve this, we reviewed the literature on malware detection and mitigation techniques to choose related features for the learning process. Then, we developed a probabilistic model using learning methods from historical feature data and employed it to infer the infection probability of nodes at each time step. Second, the learned model was combined with the stochastic Back-to-Origin Markov model to reduce the number of features. In this approach, we had no information about the infection status or the time of infection of nodes. This was made even more challenging by the fact that the infected machines were not detected by some antivirus software. We probabilistically estimated the origins of worm propagation.

In order to achieve this, we developed a four-stage approach for the learning algorithm. First, a number of features extracted from the network traffic were defined based on the performed literature review. Second, we learned a simple prior model of the distribution of features in network traffic using Bayesian network learning algorithms. In the third step, we constructed a probabilistic model using estimation of distribution algorithm (EDA)[4] based on the prior model learned in the second step and network features. In the fourth step, the infection status of nodes was probabilistically inferred. We could also determine the timeline of worm attack. Using the predefined stochastic Back-to-Origin Markov model[2] a secondary model was also developed using our first model and the stochastic Back-to-Origin Markov model to build the second model.

In this four-step method, a set of features were extracted from network traffic and we learned a model to estimate the infection status of nodes at different time steps. In order to reduce the number of features collected from the network, the stochastic Back-to-Origin model was combined with our model.[2] The nodes which had been infected before others were the possible origins of worm propagation. The most important achievement of the proposed method is to help digital investigators to identify the possible nodes responsible for entrance of a worm to the network (patient zero) and the probability of node infection times. In this way, we can investigate the patient zero of an outbreak to answer the following questions: when does the worm starts spreading in the network? Who was infected first? Which places and which machines have been affected by the worm? In this method, it is assumed that agents are installed on all nodes in the network to gather feature values. Comparing our method with other state-of-the-art methods in terms of utilized assumptions shows that it can estimate the origins without any knowledge about network topology, network graph, or infection status of nodes. The majority of available methods assume to have the infection graph which cannot be easily obtained and are more applicable to social networks as they provide the message path between members. In worm propagation, extracting the infection graph is a difficult task and indeed an unrealistic assumption. The time complexity of the present method was linear, being dependent on the average number of infected nodes. However, other available methods are associated with different complexities even of a degree of 2 or 3. Evaluating the precision and recall of our method showed that 96% of periods had been detected accurately by our method. Moreover, 92% of node infections had been correctly detected.

According to experimental outcomes, the first model was able to estimate the node infection and removal times with acceptable accuracies. Combining stochastic Back-to-Origin Markov model with the first model, we calculate the infection probability of nodes. The Back-to-Origin Markov model is developed for the spread of random scanning worms, but our first model is developed for localized scanning worms.

This paper is organized as follows. A number of related researches on source identification approaches have been presented in Section 2. The proposed method has been described in Section 3, whereas Section 4 deals with the process of dataset creation and feature selection. In Section 5, the prior Bayesian Network structure is learned. In Section 6, the EDA algorithm is used to optimize the Bayesian Network model. The node infection probability is extracted from the learned model in Section 7. A new combined model for infection distribution has been developed in Section 8. Experimental evaluation of our method has been presented in Section 9. Section 10 deals with some conclusions on the current paper and future research directions for this topic.

2 | RELATED WORKS

In recent years, researchers have introduced some methods to detect the information source in networks.[5-9] A few methods have been proposed to identify the origins of worm propagation, whereas other methods have been deveoped to find the most influential initial nodes in the spread of disease, information, rumor, or error in networks or the water pollution sources in water system.

In 2005, a method was proposed to identify the origin and reconstruct the random scanning worm propagation path.[10] The authors assumed that the worm spreads in tree-like networks and the complete graph of host communications is available. The latter case was a limiting assumption required high storage volumes. The algorithm worked through repeatedly sampling paths on the host communication graph backwards in time with random walks.

From 2010 to 2014, numerous approaches have been proposed based on different centrality measures to identify vital nodes over a contact network or infection graph. Based on the utilized assumptions, these studies are classified into the following categories.[1]

1. Origin identification methods with complete observations,[5,8] in which it is assumed that a complete snapshot of node states at a time t is available. The node state may be Susceptible, Infectious, or Removed.
2. Source identification methods with partial observations,[6,7] in which there is partial knowledge of network states at a time t, under partial observations of the network.
3. Methods based on sensor observations,[10] in which sensors are installed in the network and the dynamics of propagation, including node state, state transition time, and information propagation directions are collected.

Most of the mentioned studies also assume to have access to the contact network or infection graph. These assumptions are limiting and require high computational and storage volumes. In the following, we will review several state-of-the-art methods from the perspective of their assumptions.

Shah and Zaman[5] have introduced a method to identify the virus source in networks based on complete observation of the states of nodes at a time t. They also supposed to have the infection graph assuming the SI propagation model. The authors developed rumor centrality measure to identify the single source node of an epidemic disease. In the rumor centrality approach, the number of paths through which each infected node can spread through the network is determined and the snapshot is achieved. Some of the assumptions of these methods are far from reality.

In 2013, Zhu et al[6] identified an information source with a sample path-based approach. They assumed to have partial observations of network states and the infection graph. They measured the infection eccentricity of each node in the network, which was the maximum distance between a node and other infected ones. The node with the minimum infection eccentricity was referred to as the Jordan center of the network and the information source.

In a previous study,[11] which is relevant to the present research, a continuous time network diffusion model was learned based on historical diffusion traces. The source of an incomplete diffusion trace was then identified through maximizing its likelihood under the learned model. Each trace was a record of observed infection times within a population during a time interval. The framework traveled back to the past and identified the source node and its infection times. This approach required a full observation of the history of network traces for a sufficiently large time period. There are important differences between this approach and our study. For instance, in the mentioned research, it was assumed to have access to the contact network and partial observation of infection times, which are very limiting assumptions since it is very difficult to gain a knowledge about the node infection times and network graph. Furthermore, to solve the likelihood problem of an incomplete cascade with observed historical data, the authors used an approximation algorithm based on Monte Carlo approximation, which is different from our proposed method.

Alexandru and Dragotti[12] have introduced a technique to estimate the source location of epidemics on a general graph with a known topology. They assumed to have knowledge about the starting times of rumors and supposed that a small number of monitors are installed in the network. The authors also assumed to have knowledge about the pairwise infection transmission rate. It was demonstrated that the probability of infection of a node was a function of the distance from the source.

Cai et al[13] have provided an accurate estimates for the information spreading characteristics, using one or more sequential snapshots of the information spreading process. They also assumed to have access to the contact network, ie, the spreading rate, start time of infection, and location of information source through a four-step method.

The aim of the present study was to investigate the node infection probability while reducing the limitations and weaknesses of the state-of-the-art methods. For instance, current methods are time consuming and nonoperational with high error rate. They are associated with many open problems in topology, number of origins, number of networks, temporal dynamics, complexity, and scalability.[1] Most of these methods assume to have the infection graph or propagation graph, which is not a realistic assumption.

This paper presents a method for inferring the infection probability of a node at each time step back-in-time. To achieve this, we have built a discrete time probabilistic model with EDA algorithms using historical feature data. Then, the probability of node infection was determined based on the observed features at each time step. The nodes with highest probabilities of infection at earlier time points could be the origin of propagation. The proposed method reduces the required storage space and computational complexity in comparison with other methods.

In order to build the model to infer the status of the nodes, we used EDA algorithms for learning the Bayesian network structure. Among different available machine learning models such as regression algorithms, instance-based algorithms, decision-tree algorithms, etc, the Bayesian network is the most commonly used graphical model to represent the dependency relations between variables. Here, it has been used for reasoning about the state of the nodes. A brief review about the Bayesian networks and EDA algorithms has been provided in Section 2.2. To learn the model, we have chosen a set of features in the related literature. Available features in malware detection context have been reviewed in Section 2.3.

In this paper, we have adopted a previously developed Back-to-Origin model.[2] In this model, it is assumed to have a fully connected graph for interactions, where nodes represent computers and arcs represent neighboring relationships. Such a fully connected graph of interactions has been suggested to result in a homogeneous mixing.[14] Here, a random scanning worm operates in the application layer of the protocol stack. We have modeled application layer interactions as a graph where neighboring relations indicate that nodes can interact at application layer. In other words, in this model, nodes can directly interact with each other in a fully connected graph and at the application layer. In our problem, homogeneous mixing is held inside and outside of each subnet, and the assumptions used in another research[2] are applied for each subnet. This model has been briefly reviewed in Section 2.1.

2.1 | Related Back-to-Origin models

2.1.1 | Deterministic Back-to-Origin model

In the Back-to-Origin model, time runs backwards. In this model, β_B is the susceptibility rate, and γ_B represents the revivability rate. Removed hosts become infectious with rate γ_B, and infectious hosts become susceptible with rate β_B.[2] β_B is the rate at which effective <infective, susceptible> pairs convert an infective node to a susceptible one at each time unit and is defined as Equation (1).

$$\beta_B = \frac{\text{\# of infectives that become susceptible by one susceptible at each time unit}}{\text{Infected space}} \tag{1}$$

Removed nodes become infectious with rate γ_B. The deterministic Back-to-Origin model[2] is defined using the following set of differential Equations (2):

$$\begin{aligned} \frac{\mathrm{d}S(\tau)}{\mathrm{d}\tau} &= \beta_B S(\tau) I(\tau) \\ \frac{\mathrm{d}I(\tau)}{\mathrm{d}\tau} &= -\beta_B S(\tau) I(\tau) + \gamma_B I(\tau) \\ \frac{\mathrm{d}R(\tau)}{\mathrm{d}\tau} &= -\gamma_B I(\tau). \end{aligned} \tag{2}$$

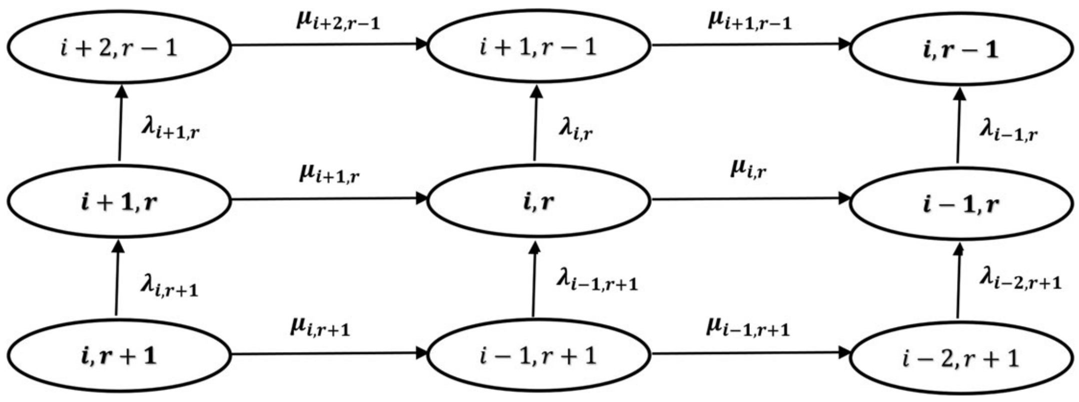

FIGURE 1 Transition diagram for stochastic Back-to-Origin Markov model

2.1.2 | Stochastic Back-to-Origin Markov model

Stochastic processes $I(t)$ and $R(t)$ are defined as the number of infective and removed nodes at time t.[15] The process $(i, r) = \{(I(t), R(t)); t \geq 0\}$ is a Markov process because the system state at the next time point only depends on the current state of the system and not on the previous one. Continuous-Time-Markov-Chain is considered with the state $(I(t), R(t)) = (i, r)$. Then, the transition rates at state (i,r) can be described as the following.

In this model, shown in Figure 1, a removed node becomes infectious with rate γ_B, so the transition rate to the state $(i+1, r-1)$ is given by

$$\lambda_{i,r} = \gamma_B i. \tag{3}$$

An infectious node can be vulnerable with rate β_B. The transition rate to the state $(i-1, r)$ is given by

$$\mu_{i,r} = \beta_B i(N - i - r). \tag{4}$$

Consider the following probability:

$$P_{i,r} = Pr\{I(t) = i, R(t) = r\}, \tag{5}$$

where $P_{i,r}(t)$ is the probability by which i nodes are infectious and r nodes are removed at time t. Using Equations (3) and (4), one can obtain the following difference-differential equation:

$$\frac{\mathrm{d}}{\mathrm{d}t}P_{i,r}(t) = \mu_{i+1}P_{i+1,r}(t) + \lambda_{i-1,r+1}P_{i-1,r+1}(t) - \nu_{i,r}P_{i,r}(t), \tag{6}$$

where $\nu_{i,r} = \lambda_{i,r} + \mu_{i,r}$.

The marginal distribution of $P_{i,r}(t)$ averaging over r provides the probability distribution of $P_i(t)$, which is the probability by which i nodes are infectious at time t.

2.2 | Probabilistic model building algorithms

In this research, we aimed at modeling the dependence relationships between variables by ascribing a directionality to each influence. Bayesian networks represents the conditional dependence properties of the variables to allow a compact and natural relationship.[16] Other types of graphical models such as undirected graphical models, local probabilistic models, Gaussian network models, etc, are useful in modeling a variety of phenomena with different characteristics, for example, when there is no directionality to the interaction between variables or there are dependence relationships between continuous variables, etc. Here, we have employed the Bayesian network approach; due to its similarity with our method, we employ this approach. To optimize model learning, we have deployed the probabilistic model building genetic algorithms. In the following sections, we have briefly described these concepts.

2.2.1 | Bayesian learning algorithms and tools

The aim of learning is to obtain an approximate representation of a high-dimensional distribution associated with many variables based on available data.[16] This process involves two important steps: parameter estimation and structure learning in Bayesian networks.

TABLE 1 Bayesian network learning algorithms and tools

Algorithm	Ref.	Matlab tool[15]
PC	18	learn_struct_pdag_pc
MWST	19,20	learn_struct_mwst
K2	21	learn_struct_k2
Markov Chain Monte Carlo (MCMC)	22	learn_struct_mcmc
Greedy search	-	learn_struct_gs2
Greedy search in the Markov equivalent space	23	learn_struct_ges

Different structure learning algorithms and tools available. Here, two well-known packages have been used for structure learning are used, including the Bayes Net Toolbox for Matlab (BNT)[17] developed by Murphy and the Bayes Net Toolbox Structure Learning Package (BNT-SLP)[15] written by Francois. Six important structure learning algorithms, which have been implemented in the package, are listed in Table 1. These algorithms attempt to search for causality relations between variables and then use the results to build the network structure.

2.2.2 | Estimation of distribution algorithms

Estimation of distribution algorithms (EDAs) are probabilistic model-building genetic algorithms that aim at estimating a distribution model for some sample data. EDAs are stochastic methods that build and sample explicit probabilistic models of candidate solutions. They run as a series of incremental updates of a probabilistic model.

In other words, at this step, EDA is used to learn the structure and parameters of Bayesian networks. In EDA, the new population of individuals is sampled from a probability distribution function, which is estimated from a dataset containing selected individuals from the previous generation.[4] In this manner, the relationships between the variables involved in the problem domain are explicitly captured and exploited. EDA consists of three main steps.

1. Selecting some individuals from a population of individuals;
2. Learning the joint probability distribution of the selected individuals;
3. Sampling from the learned distribution is repeated until a stopping condition is met.

These steps are repeated until a stopping criterion is satisfied.

There are a number of implementations for EDA.[24-27] Here, we used MATEDA 2.0,[28] which was first released in 2005 and allows optimization of single- and multiple-objective problems with EDA. A brief description of EDA has been provided in the following paragraph.

In EDA, the population for the first generation is initialized via a seeding method (line 4). In other generations, a new population is sampled from the learned model (line 9). In some situations, a repairing method is applied to replace the sampled population with better individuals (lines 10, 5). Then, the sampled population is evaluated (lines 11, 6). A local optimization method may be applied to the sampled population (lines 12, 7). In the first generation, the created population is just the current population. However, in other generations, a replacement algorithm is used (line 14). The new population is formed using the previous population, the selected population, and the sampled population. The model is then learned using one of the predefined learning algorithms utilizing the population (line 15). The EDA process continues and finally stops when there is no more improvement in the new generated population (line 18). The pseudocode of EDA has been shown in Figure 2.[28] The methods used in this code have been explained in Table 6.

It is notable that MATEDA 2.0 programs generally implement a particular class of EDAs and do not allow users to combine their own EDA components with the provided code.[29]

One important function used in MATEDA 2.0 is the fitness function. In learning algorithms, this function takes a candidate solution for the problem as the input and returns how "fit" the solution is with respect to the problem under consideration, as the output.[30] In MATEDA 2.0, individuals are ordered according to the ranking of their fitness values. During the replacement phase, sampled individuals with lower fitness levels are replaced with individuals from the previous generation with higher fitness values.

2.3 | Features

In this section, we have listed the features pointed out in related studies on malware detection approaches, so as to select a subset of relevant features for application in model construction. These are extracted from the layers of Transport Control

```
Set $t \leftarrow 0$
do {
    If $t = 0$
        Generate an initial population $D_0$ using a *seeding method*
        If requires, apply a *repairing method* to $D_0$
        Evaluate (all objectives of) population $D_0$ using an *Evaluation method*
        If required, apply a *local optimization method* to $D_0$
    Else
        Sample a $D_{sampled}$ population from the model using a *sampling method*
        If required, apply a *repairing method* to $D_{sampled}$
        Evaluate (all the objectives of) population $D_{sampled}$ using an *evaluation method*
        If required, apply *a local optimization method* to $D_{sampled}$
        Create $D_t$ population from population $D_{t-1}$, $D_{sampled}$, and $D_t^S$ using a *replacement method*
    Select a set $D_t^S$ of points according to a *selection method*
    Compute a probabilistic model of $D_t^S$ using a *learning method*
    $t \leftarrow t + 1$
    }
Until the evaluation of the *termination criteria method* is true.
```

FIGURE 2 Estimation of distribution algorithm[28]

Protocol/Internet Protocol (TCP/IP) protocol, ie, network layer, application layer, and logical network topology or the features used in host-based detection techniques.

The features can be listed as follows.

- The features that are used for detection and classification of worms at the internet layer, including the number of source and destination IP addresses, number of Internet Control Message Protocol (ICMP) packets, number of TCP packets, number of User Datagram Protocol (UDP) packets, number of SYN flags, number of ACK flags, number of RST flags, number of source ports, and number of differential packet sizes, port ratios, SYN ratios as previously mentioned.[31]
- Two types of features have been previously used[32] to identify SPAM email messages, including email features such as the number of attachments, MIME type of file attachment, etc, and the features calculated over a sending window such as the number of emails sent and the number of unique sender addresses.
- The methods proposed in some studies[5-8,33-35] assume to have the infection graph and subsequently identify the propagation source. The infection graph (virus graph) is a set of nodes and edges. If infection is transmitted from node *i* to node *j*, there is an edge between the two nodes in the graph.
- The exact state of all nodes in the network at time *t* has been sometimes considered as a feature.[5,8,33,34] In this manner, each node may have three states: susceptible, infected, and removed.
- In a previous research by Lokhov et al,[7] the nodes revealed if they had been infected with some probability at time *t*. In another study,[6] all infected nodes were recognized at time *t*, but susceptible or removed nodes could not be distinguished at this time. In both studies,[6,7] the state of nodes has been presented as a feature.
- The methods proposed in a few research works[36,37] use the contact network where diffusion takes place. The authors consider an edge in the contact network if there is a connection between two nodes.
- In some methods, sensors are used to allow collecting two features, ie, from which neighbor and at what time the sensor receives the information. The methods identify the information source.
- In another investigation,[10] it has been assumed that the contact graph, showing communications between hosts at the network layer, is available. The contact graph is a directed graph with a set of nodes and edges at time *t*. If there is a network flow between two nodes at that time, an edge is inserted between these nodes in the graph.
- Altarelli et al[38] have used a certain number of features of propagation characteristics, including the recovery probability for each node, the probability by which two nodes infect each other, the distribution of recovery time, and the transmission delay distribution.
- In another study,[11] historical diffusion traces were available, and the following features were extracted: which nodes got infected and when did this happen.
- Zhu et al[39] have defined the following features: the time elapsed before the subnet gets worm duplication, the number of infected hosts in each subnet at a moment, the bandwidth consumed by the worm inside subnet *i* to attack the outside (*bandwidth_out*), the bandwidth consumed by the worm outside subnet *i* to attack the subnet (*bandwidth_in*), and the bandwidth consumed by the worm inside subnet *i* to attack the subnet (*bandwidth_inside*) are used as features.
- In the previous research,[40] to predict the number of malware infections in a country the following features have been defined: the time when a file becomes infected with a malware, antivirus signature release time, and the patch release time.

- In another study,[41] 323 features were collected from a monitored computer, which could be classified into the following 11 main categories to detect worm activity in computers: ICMP, IP, memory, network interface, physical disk, process, processor, system, Transport Control Protocol TCP, thread, UDP.
- In an investigation by Tabish,[42] instructions or call sequences of an executable program have been mapped to a graph. Features were extracted from the constructed graph at the three following levels: vertex level, subgraph level, and graph level. Vertex level features could be divided into degree, path, and connectivity categories. At subgraph level, features (program instructions or call sequences) were extracted based on Markov Chain. The graph-level features were defined as clique number, average clustering coefficient, diameter, and average path length.
- In another study, the following classification features have been used[43]: the number of network connections (TCP, UDP, ICMP) from a node, entropy of the number of destination IP and ports connected by a host during each 50 sec intervals, and other host-based features such as CPU load. These features were used in an adaptive end-host anomaly detector, where a supervised classifier was employed.
- Moore and Zuev[44] have used the following features to classify traffic as common groups of applications, including flow duration, TCP port, packet interarrival time, payload size, the effective bandwidth based on entropy, and Fourier transform of packet inter-arrival time.
- The number of nodes that a host connects to, in a small predefined period of time, is called successors.[45]

3 | OUR METHOD

We have adopted the following method to infer the probability of node infections when the spread of a preferential scanning worm occurred.

1. The required features were first carefully defined, a dataset containing normal background traffic and preferential scanning worm traffic was created, and then, the features were extracted from the synthesized data.
2. Structure learning techniques were used to build a prior model based on the historical feature values from the dataset. Therefore, the most convenient algorithm for our problem was selected.
3. A probabilistic model was developed for inferring the infection status of nodes using EDAs, historical feature values, and the prior model. We calculated the Probability Density Function from the learned model.
4. The inference approach was used to extract the probability of infection of nodes based on feature values at each time step.
5. The scheme of stochastic Back-to-Origin Markov model[2] was combined with the probabilistic model of step 4 to reduce the number of required features based on the infection and removal rates and the number of infectious and removed nodes at a time step of worm propagation.

In the following sections, we describe the proposed approach.

4 | DATASET AND FEATURE EXTRACTION

4.1 | Dataset

Feature extraction and learning a model from network traffic data, a dataset is required containing normal background traffic and worm traffic at the same time. We aimed at using a worm with a local preference scanning method for generating the dataset and learning the model. Since such dataset not publicly available, a new dataset was generated.

A preferential scanning worm started scanning the network for vulnerable systems with one or multiple infectious sources. A malicious code was sent to neighbors in the same subnet with a higher probability and to nodes in different subnets with lower probability. In the network under investigation, computers were directly connected to each other, forming a homogeneous mixing. However, in the local subnet scanning, instead of a random selection of targets, worms preferentially scanned hosts on the local address space,[46] forming homogeneous mixing model inside and outside each subnet while constructing heterogeneous-mixing model in the whole network. Therefore, employing the Back-to-Origin model for a preferential scanning worm could result in errors in the model.

In the following, it has been demonstrated how a dataset was generated to build a model.

- The ISCX 2012 IDS dataset[47] has been generated in a physical testbed implementation using real devices. We included a subset of its normal traces (DAY 1) in our training and test datasets (40 seconds and 50 000 milliseconds, respectively).

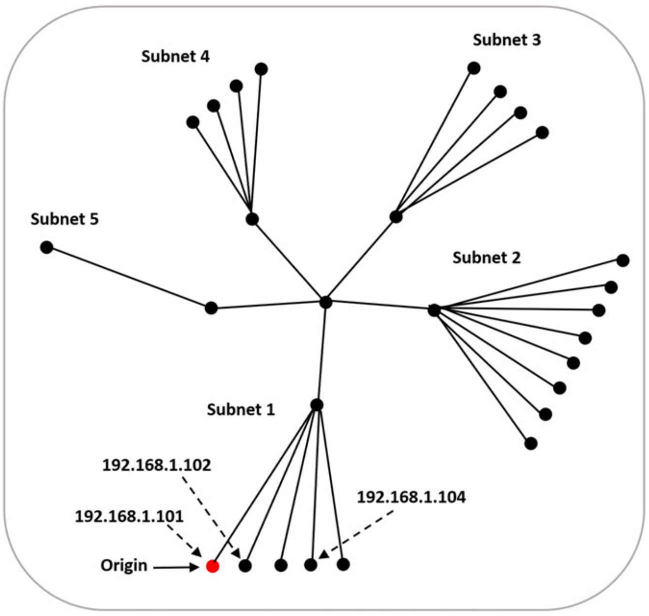

FIGURE 3 Simulation topology in the Georgia Tech Network Simulator

The ISCX testbed network included 22 workstations in five subnets with addresses ranging from 192.168.1.0/24 to 192.168.5.0/24.

- Georgia Tech Network Simulator (GTNetS) has been designed to allow for large-scale packet-level network simulations.[48] GTNetS can be used to study the behavior of internet worms under a variety of conditions[49] and allows capturing the packet traces. Worm transmission occurs through UDP and TCP worms. We generated simulations using the TCP local preference scanning worm in GTNetS with AlmostCodeRed2 class. In GTNetS, nodes move between two states, namely Susceptible and Infected (SI model). However, in the present research, each node could be always in one of the three following states: Susceptible, Infected, and Removed (SIR model). Therefore, we modified the GTNetS code to include the state R.

The available simulators can simulate network data, but most of them operate limitedly and are not able to simulate worm propagations. Furthermore, we did not have access to real data. Hence, our other choice was to use normal data from available datasets. Indeed, the datasets which contain internal network traffic data are not publicly available due to the privacy policies of organizations, which was needed for preference worm scan propagation. Other available datasets are fully anonymized and do not have statistical characteristics. At the beginning of this research, the following datasets were available: ISCX dataset,[47] DARPA,[50] LBNL,[51] UNSW-NB15,[52] BC,[53] and DEK-PKT,[54] among which we selected ISCX since it was newer than other available datasets.

The problem of simulating normal traffic with different topologies can be the subject of further investigation in a separate study. The ISCX dataset is a sample that models the behavior of users in the security center of UNB University and could be used as a good baseline for our modeling.

It is notable that we used the GTNetS simulator for packet level worm simulations. It was available at the time of this research and produces flow level captures for simulations. Although other simulators were also available to simulate ransomworms,[55,56] here, we did not intend to cover all available simulators. Other simulators could be used to complete the developed model.

To combine normal data with worm data, the worm in GTNetS should be propagated on a topology similar to ISCX topology. The simulation topology has been shown in Figure 3. At the beginning of simulation, one host was initially infected, as shown in red in the Figure.

A star topology is a topology used for connecting computers in the lower layers of protocol stack within a Local Area Network (LAN). Moreover, LANs with different topologies are connected to the internet in the lower layers of protocol stack. Propagation of a preferential scanning worm required selection of a vulnerable IP on the same subnetwork as the presently infected node with higher probability. This assumption was not related to the star-like topology and holds true for all internet topological structures.

We implemented a local preference scanning worm and captured packet level traces during simulations using predefined GTNetS functions in VC++. It has a higher probability of scanning an IP address within a subnet. Code red is an

TABLE 2 Worm scan parameters

Simulation parameters	Calculation method	Value
Local subnet scan prob.	Set	0.6
Remote subnet scan prob.	Set	0.4
Local subnet RTT	Measured	0.1 ms
Remote subnet RTT	Measured	0.2 ms
Total spread time	Measured	300 ms
Connections	Set	4
Local scan time	Measured	7 ms
Remote scan time	Measured	8 ms
Scan rate	Resulted	0.54
β	Resulted	0.02
γ	Resulted	0.01

example of preference scanning worm, and Code red II scans the local subnet with probability $p = 0.5$ and the remote class B network with probability $p = 0.375$ and Class C with probability $p = 0.125$. In the performed experiment, our network had 22 nodes, and we considered two classes of inside and outside. The probability of the worm to scan locally was set to 0.6 and the the probability of the worm to scan the remote subnet was set to 0.4.

In order to merge the traffic generated in GTNetS with ISCX traffic, we designed a network topology in GTNetS simulator similar to ISCX topology. The RTT of nodes was measured in ISCX testbed and used as the link delay of our simulated network in GTNetS. In ISCX testbed, we measured the time required for a SYN packet to travel from a source to a destination in the same subnet and in another subnet, and the time required for the ACK packet to be back again to the source (RTT). The average time for a SYN packet to be sent to the local subnet and the ACK packet to be back was 200 milliseconds. This value was 400 milliseconds for the remote subnet. Therefore, the RTT of the local subnet was set to 0.2 milliseconds, and the RTT of the remote subnet was set to 0.4 milliseconds. In GTNetS, a link is defined by delay. In this research, for a local subnet, the link delay was set equal to the local RTT divided by 4, and for the remote subnet, the link delay was set equal to the remote RTT divided by 8.

As previously described,[30] the infection rate of Code Red II was 358 per minute; therefore, Code Red II sent out on average $\frac{358}{60} = 5.965$ scans per second.[30] In GTNetS, a TCP worm can be configured by setting the number of simultaneous TCP connections that each infectious node can create. Each connection continues until the destination node becomes infectious, and then, new connections start. In our simulations, the number of connection threads was fixed to 4. According to the experiments performed in GTNetS, the spread and removal of Code Red II in the network with aforementioned topology and parameters took less than 200 milliseconds. Increasing the number of connections reduces the simulation time, which was not suitable for our experiments. The infection length and infection port were held at constant default values.

In GTNetS, we measured the time required for scanning. Based on the results, the required time duration was 7 milliseconds for the local subnet and 8 milliseconds for the remote subnet. The scan time was $7.4 = (0.6 \times 7 + 0.4 \times 8)$, where 0.6 was the probability of local subnet scanning and 0.4 was the probability of remote subnet scanning. Therefore, the scan rate was $0.54(\frac{4}{7.4})$ and $\beta = \frac{0.54}{22} = 0.02$. The removal rate was set to 0.01. Worm simulation parameters have been shown in Table 2. The second column in each row of the table represents whether the parameter was measured in simulations or kept constant.

We run 120 simulations for the described preferential worm of AlmostCodeRed2 class in GTNetS. In each simulation, network packets were captured for 300 milliseconds. With extracting features (see Section 4.2) from ISCX normal traffic and Code Red simulations in GTNetS and merging the features from both captures, we created the dataset and extracted the features that contained combined normal and worm traffic.

4.2 | Feature selection

In this step, we selected a subset of relevant features relevant for worm construction, as mentioned in Section 2.3. Our selected features have been presented in Table 3. Here, the approach used to select the features introduced in Section 2.3 will be described. No assumption was made about the infection graph, which presents the infection paths between nodes. We make no assumption about the infection graph. The features were collected for each node in the network. The nodes were numbered such that each number represented a unique node. We described the features of the ith node inferring its infectious state in previous time steps. As defined earlier, neighboring relations describe interactions between two nodes at the application layer. In the network topology presented in Section 4.1, each node can communicate with others, forming a fully connected model of relations. Node *j* is a neighbor of node *i*, the feature values of which are used to improve the inference of the *i*th node state.

- We selected 10 features that had been previously pointed out.[31] They have been listed in Table 3 in rows 1 to 6, 10 to 11, 17, and 18 of Table 3. We gather the features for the ith node. Other features introduced in this reference, ie, number of ICMP packets, number of RST flag and differential packet size are not selected as they have no influence on preference scan worms.
- We selected 10 features that had been previously pointed out.[31] They have been listed in Table 3 in rows 1 to 6, 10 to 11, 17, and 18 of Table 3. The features were gathered for the ith node. Other features introduced in previous research,[31] ie, the number of ICMP packets, number of RST flags, and differential packet sizes were not selected, as they have no impact on the preference scan worms.
- We also used the feature defined in another research,[39] ie, the number of infectious nodes in each subnet at each moment which is the seventh feature of Table 3. This feature was measured for node *i*.
- The infection state of node *i* was represented by a parameter named “infection,” as defined in row 9 of Table 3.
- As previously suggested,[43] the entropies of destination IPs and ports connected by node *i* were used as listed in rows 12 and 13 of Table 3.
- We employed *inf path to j* as a feature (as previously suggested[5-8,33-35]). This feature is set to 1 if there is an edge between the nodes *i* and *j* in the infection graph. Node *j* is a neighbor of node *i* in the fully connected graph of interactions, and its infection status is known. This edge exists if node *j* has been infected by node *i* or vice versa. It has been shown in row 19 of Table 3. This feature was used to investigate the infection path between the nodes *i* and *j*.
- In this research, *connection to j* was used as a feature. Node *j* is a neighbor of node *i* in the fully connected graph of interactions, with a known infection status. The connection flow between nodes *i* and *j* was investigated with this feature. The feature value is set to 1 when there is a connection between these nodes. This happens if the edge is a part of the contact graph of host communications at the network level.[10] It has been shown in row 20 of Table 3.
- *Inf Stat_j* was also used as a feature as shown in row 21 of the Table. It shows the infection state of node *j*. The state of all nodes state of all nodes have been previously considered as a feature.[5,8,33-35] In the next step, the dependency relations between features were learned.
- We selected the number of successors of node *i*[45] as a feature and called it *OUT Degree*. *OUT Degree* is the number of hosts that node *i* connects to in each millisecond. It has been shown in row 8 of Table 3.

TABLE 3 Feature list

no	Name	Description
1	*ACK*	# of received acknowledgement packets by *i*th host every msec
2	*DST IP*	# of unique destination addresses the *i*th host connected to every msec
3	*DST Port*	# of unique destination ports that ith host connected to every msec
4	*SRC Port*	# of unique source ports that *i*th host uses to every msec
5	*SYN CNT*	# of SYN packets sent by *i*th host every msec
6	*TCP CNT*	# of TCP packets sent by host *i* every msec
7	*INF CNT*	# of infectious nodes per subnet every msec
8	*OUT Degree*	# of connections made from *i*th host every msec
9	*Infection*	The status of *i*th host every msec (1=infectious; 0=susceptible or removed)
10	*SRC CNT*	# of unique SRC addresses in packets in subnet every msec
11	*UDP CNT*	# of UDP packets sent by node *i* every msec
12	*Ent node*	The entropy of the unique dst addresses that host *i* connects within 50 ms
13	*Ent port*	The entropy of the unique dst ports that *i*th host connects within 50 ms
14	*Bandwidth_in*	The bandwidth consumed by the hosts outside subnet to connect to the subnet
15	*Bandwidth_inside*	The bandwidth consumed by the hosts inside subnet to connect to the same subnet
16	*Bandwidth_out*	The bandwidth consumed by the hosts inside subnet to connect to the outside
17	*SYN ratio*	The ratio of SYN packets to unique dst addresses node *i* connects to every msec
18	*Port ratio*	The ratio of unique src ports to unique dst ports for node *i* every msec
19	*Inf Path to j*	If the infection spreads from node *i* to *j* or from node *j* to *i*, this feature is set to 1
20	*Connection to j*	If there is a network flow between node *i* and *j* at time *t*, this feature is set to 1 at that time.
21	*Inf Stat_j*	The infection status of node *j* every msec.

We built the model and independently used it on each node of the network. Using the Infection parameter, the infection status of the node under investigation, node *i*, was inferred, where *Inf Stat_j* was the infection status of a neighbor node (*j*) in the network and its infection status could be observed. For some features, it is required to use a sensor node in the network to record when and from which neighbor the information has been received. Other features[38,40-42,44] have been defined such that they are not compatible with the assumptions of our work, for example, assuming to have an antivirus or collect the features at host level or collect network flow traffic. We investigated the relationship between the following four features: the infection state of node *i*, the infection state of node *j*, the connection between nodes *i*, and *j* and the infection path between nodes *i* and *j*. Each feature has been fully explained in the related references. Other features mentioned in Section 2.3 were not employed as we were interested in the network level features, while host-based features and email features (except for the node state) were inconvenient for our problem. Preferential scanning worms propagate in a fully connected network.

Network packets were further processed to obtain the network features related to each host for 50 000 milliseconds. The correlations between features 1 to 8, 10 to 18, and 9 (infection state of nodes were examined. A change in the node status (feature 9) could increase or decrease the values of features 1 to 8. However, there was no distinct relationship between the value of feature 9 and those of features 10 to 18. Indeed, there was no correlation between these features and feature 9, and they were neglected for developing the model in the following steps.

5 | LEARNING THE PRIOR DISCRETE TIME BAYESIAN NETWORK STRUCTURE

In this step, we learned a Bayesian network structure to achieve the following goals.

- Selecting an appropriate learning algorithm among implemented ones, with respect to our problem and related data;
- Building a prior feature dependency graph and using as the initial directed acyclic graph (DAG) in the EDA algorithm in the next step.

As graph, a new heuristic approach was proposed for choosing the most convenient learning algorithm. The following steps were used to choose the best matching structure learning algorithm for our problem.

Heuristic 1:

1. Different structure learning algorithms were used to create the dependency graph between features.
2. The learned graphs were merged into one graph. In this regard, the frequencies of edges in each graph were combined in the merged graph.
3. The average number of edges was calculated in the merged graph.
4. The final DAG was obtained through keeping the edges with weights greater than the average value.
5. The appropriate algorithm was chosen based on the resulting DAG.

If an edge occurred several times between a pair of features in DAGs, it indicated the dependency between the two features with a higher probability. The algorithm that generated the most similar DAG to the final DAG was selected for the next step. The final DAG was given to the EDA algorithm (Section 6) as a prior structure. We will discuss all above mentioned steps in the subsequent sections.

5.1 | Structure learning algorithms

In this step, a Bayesian Network model was learned using 6 algorithms introduced in Section 2.2.1, utilizing BNT and BNT-SLP toolboxes. The raw data used for learning consisted of seven features for 50 000 records (*ACK*, *DST IP*, *DST Port*, *SRC Port*, *SYN CNT*, *TCP CNT*, and *Infection*). To build the model and select the tool in a reasonable time, seven features were used in this phase. Adding more features would increase the execution time of algorithms. The features were discrete. The raw data given to the algorithms have been shown in Table 1 based on which the conditional dependencies of features were found. A value of “1” was returned if adequate evidence of conditional dependence was found,; otherwise, “0” was returned. The output of each algorithm was a DAG; thus, six DAGs were obtained.

	ACK	*DST IP*	*DST Port*	*SRC Port*	*SYN CNT*	*TCP CNT*	*Infection*
ACK	0	4	2	2	2	5	1
DST IP	2	0	3	3	2	1	1
DST Port	2	1	0	1	2	2	1
SRC Port	1	4	2	0	2	1	1
SYN CNT	3	4	5	5	0	3	1
TCP CNT	2	3	2	3	4	0	2
Infection	1	2	1	1	4	4	0

TABLE 4 The combined directed acyclic graph graph

5.2 | Merged learned graphs

The obtained DAGs in Section 5.1 were merged in this step. The goal was to find the common Bayesian Network (BN) in the DAGs produced by the algorithms and select the best-matching algorithm with the final DAG. The merged DAG has been shown in Table 4. Each table cell shows the number of directed edges from one node to any other node in all six learned DAGs. This value was called edge weight.

5.3 | Average number of edges

In this step, we considered the average edge weights in all learned DAGs. The average edge weight is equal to the sum of all edge weights divided by the total number of edges between variables ($\frac{98}{42} = 2.33$), as there were seven vertices in each graph, and there were a total of $7 \times 7 - 7 = 42$ edges.

5.4 | Final DAG

In this step, we obtained our evaluation DAG, through comparing the weights of all edges of the merged graph with the average weight (calculated in Section 5.3). If the weight of an edge e (calculated in Section 5.2) was greater than the average edge weight, it was selected for the final DAG. Final DAG has been shown in Figure 4. The DAGs learned by different structure learning algorithms did not encode the same conditional dependencies. Therefore, if the number of occurrences of an edge in the learned structures was greater than the average edge weight, it was placed in the final DAG and the possibility of learning equivalent structures would increase.

5.5 | Algorithm selection

All DAG models learned by each algorithm were compared with the final DAG (calculated in Section 5.4). The number of edge mismatches for each algorithm has been shown in Table 5. As shown in this Table, gs2 and MCMC resulted in more equivalent classes for our dataset. The MCMC run ended after 700 iterations and the graph with the highest acceptance ratio was selected. These two algorithms have been summarized as the following.

- In the MCMC algorithm, a Markov chain was defined over the space of possible structures whose stationary distribution was the posterior distribution, $P(G|D)$. Then, a set of possible structures was generated through a random walk in this Markov chain. This process was continued until the chain converged to a stationary distribution.[16]
- Cooper and Herskovits have used a greedy algorithm to construct the output graph in stages. For each vertex X in the graph, the algorithm considered the effect of adding the parent set of X with each individual predecessor of X that was not already a parent of X. It selected the vertex that its addition to the parent set of X resulted in the highest increase in the posterior probability of the local structure consisting of X and its parents.[18]

The two aforementioned algorithms are used as search methods. We used the gs2 algorithm in the next step of our method. The DAG shown in Figure 5 is the output of gs2 algorithm. Based on our experiments, the gs2 algorithm constructed a better model for learning our Bayesian network structure using our dataset.

The final DAG contained 15 edges, although the DAG of gs2 algorithm had 10 edges. There were five edges in the final DAG, which were ignored by the gs2 algorithm.

5.6 | Analysis of the output of gs2 algorithm

Here, the output of gs2 algorithm was analyzed. As shown in Figure 5, there was a dependency between the infection status of a node and the number of TCP and SYN packets, confirming the fact that when a node gets infected, it sends

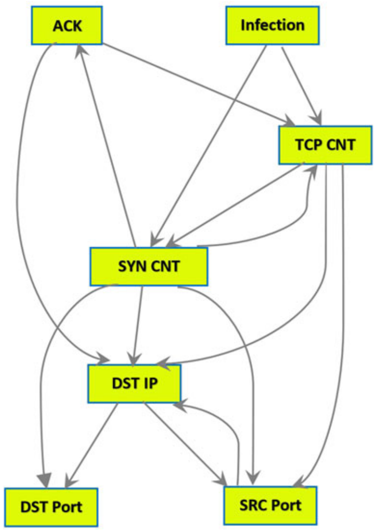

FIGURE 4 Final directed acyclic graph

TABLE 5 The number of mismatches between the directed acyclic graph (DAG) learned by each algorithm and the final merged DAG

Algorithm	# of mismatches
PC-DAG	26
MWST	13
K2	10
Monte Carlo Markov Chain	5
Gs2	5
Ges	12

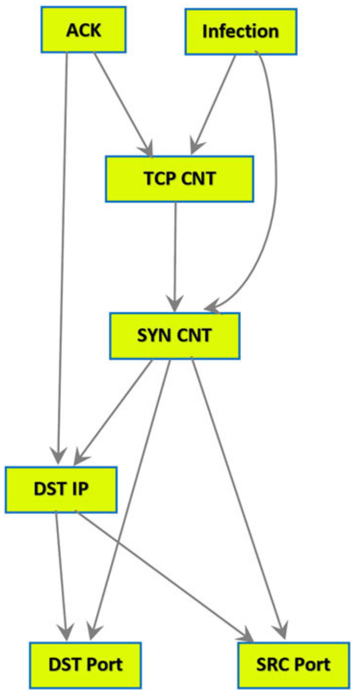

FIGURE 5 Output of gs2 algorithm

more flows. There was a dependency relation between TCP packet count and SYN packet count because increasing the number of TCP packets increased the number of SYN packets. Other dependency relationships between features could be justified based on the TCP/IP protocol. Figure 5 shows that connecting to different IP addresses increases the possibility of connection to different source and destination ports.

6 | OPTIMIZED DISCRETE TIME BAYESIAN NETWORK MODEL LEARNING USING EDA

In this paper, we adopted a stepwise approach to improve the worm infection probability distribution function. First, we developed a four step method to build a probability distribution function for estimating the infection status of nodes at each time step of worm propagation, when the spread of a preferential scanning worm happened. Then, we combined the previously developed stochastic Back-to-Origin Markov model[2] with our model.

In order to learn a prior Bayesian network from data, the greedy search algorithm was selected in the second step of our method (Section 5). In this step, using EDas, we optimized the worm infection probability distribution function using 12 features (ie, *ACK*, *DST IP*, *DST Port*, *SRC Port*, *SYN CNT*, *TCP CNT*, *INF CNT*, *OUT Degree*, *Infection*, *Inf Path to j*, *Connection to j and Inf Stat_j*).

6.1 | Application of EDA to our problem

We have proposed the use of EDA, which was able to estimate the joint probability distribution associated with 12 variables. We used the feature values extracted from three IP addresses (192.168.1.101, 192.168.1.102, 192.168.1.104) in our dataset for model learning. The three nodes generated 108 900 records within 40 seconds, which were used as the initial population and a Bayesian network was learned from the data. The following steps were followed.

1. A new population was sampled from the generated distribution. Sampling method served to generate the new population from the probabilistic model learned by EDA. Here, SampleBN was chosen as the sampling method.
2. Sampled individuals could be repaired through a repairing method.
3. Fitness was computed for each member of the repaired population. *fitness_ordering* was selected as the ordering method. Individuals were ordered according to the ranking of their fitness values.
4. In the sampled and repaired population, individuals with higher fitness values were selected, while those with lower fitness levels were replaced with individuals from the previous generation with higher fitness values. Replacement methods were used to combine the individuals generated in the current population with those from previous generations. *best_elitism* was used as the replacement method. It joined the selected individuals to the sampled individuals to form the next population.
5. The probabilistic model was constructed using the learning methods. Learning was achieved through model extraction from relevant data and a prior knowledge about the model structure. LearnBN was seleceted as the learning method to learn a Bayesian network from data.[33] We changed the LearnBN code and added it with the learn_struct_gs2 function, from BNT_SLP, as our learning method.
6. The EDA process was repeated until the termination criteria was met.

In our experiments, the iterations of the loop converged after 12 generations. The model did not change after 12 generations. The chosen functions in our EDA program have been listed in Table 6.

We considered a fitness function in order to evaluate generated samples from the learnt distribution in EDA. Twelve features (shown in Table 7) for each individual were evaluated in the fitness function. Based on the values of the observed features during normal network operations and worm propagation in learning dataset, thresholds were defined to identify normal and malicious feature values. These thresholds are given in Table 7. The fitness function pseudo code has been shown in Figure 6.

Formally, the fitness function *f* was defined as follows:

$$f(x) = \begin{cases} fit - const, & \text{Condition A} \\ not - fit - const, & \text{otherwise.} \end{cases} \tag{7}$$

TABLE 6 Estimation of distribution algorithm components

Sampling method	*Sample BN*	Samples a population of individuals from Bayesian Network (BN)
Replacement method	*Best Elitism*	Joins the selected individuals with the sample individuals to formthe next population.
Selection method	*Truncation − selection*	Samples a population of individuals from a Bayesian or Gaussian network
Learning method	*Learn − BN*	Learns a Bayesian Network from data using a greedy search algorithm (learn-struct-gs2) from Bayes Net Toolbox Structure Learning Package

TABLE 7 Fitness function thresholds

#	Feature	Threshold
1	*ACK*	≥ 2
2	*DST IP*	≥ 2
3	*DST Port*	≥ 2
4	*SRC Port*	≥ 1
5	*SYN CNT*	≥ 2
6	*TCP CNT*	≥ 2
7	*INF CNT*	≥ 0
8	*OUT Degree*	≥ 1
9	*Infection*	$= 1$
10	*Inf Path to j*	$= 1$
11	*Connection to j*	$= 1$
12	*Inf Stat_j*	$= 1$

```
Fitness_Function()
{
normal_weight ← 0;
infection_weight ← 0;

If (feature_value ≥ threshold)&&(Infection == 1)          // for features: 1,2,3,5,6,7,8
        infection_weight + +;
else if (feature_value < threshold)&&(Infection == 0)     // for features: 1,2,3,5,6,7,8
        normal_weight + +;

if (inf Stat_j == 1)&&(infection == 1)&&(feature_value == 1)   // for features: 10,11
        infection_weight + +;
else if (inf Stat_j == 0)&&(infection == 0)
        normal_weight + +;

if (infection == 1)&&(infection_weight ≥ 3)
        fitness_value ← 5;
else if (infection == 0)&&(normal_weight ≥ 6)
        fitness_value ← 5;
else
        fitness_value ← 3;
}
```

FIGURE 6 Pseudocode for our fitness function designed for this paper

Condtion A is if (infection_state > inf_thresh) or (normal_state > n_thresh).

$$infection\_state = \sum_{feature\_value} (feature\_value > infection\_threshold)\&\&(Infection == TRUE) \tag{8}$$

$$normal\_state = \sum_{feature\_value} (feature\_value > normal\_threshold)\&\&(Infection == FALSE), \tag{9}$$

where *f(x)* is the returned value of fitness function, indicating how the learnt feature values have been fitted with our previous generation. *fit_const* and *not_fit_const* are two constant values indicating that the sampled feature values are fitted or not fitted with the previous generation, respectively.

The rationale behind Equations (8) and (9) is that we first observed the thresholds for feature values in the normal and infection states. Then, we compared feature values of the generated population with *infection_threshold* and *normal_threshold*. A score was summed for the matching feature values. In case this sum (*infection_state* and *normal_state*) was higher than an experimentally predetermined threshold (*inf_thres* and *n_thresh*), which is set by experience, the sampled population was fitted with the previous generation (the function returns *fit_const*).

6.2 | Bayesian Network structure and parameters

We addressed the problem of learning a Bayesian Network structure and its parameters from the synthesized dataset introduced in Section 4.1 using EDA algorithm proposed in Section 6.1. Figure 7 presents a DAG, in which nodes represent random variables and edges represent the direct influence of one variable on another. Moreover, DAG is a compact representation of the conditional independence assumptions and a compact representation of the joint distribution between 12 features.

The joint probability distribution function at each time unit over all 12 features has been given in Equation (10). It was extracted from the Bayesian network structure (Figure 7), as the result of the EDA algorithm.

$$\begin{aligned}
&P_t(Infection, OUT\ \ Degree, INF\ \ CNT, TCP\ \ CNT, ACK, DSTIP, SRC\ \ Port, DST\ \ Port, SYN\ \ CNT,\\
&Connection\ \ to\ \ j, Inf\ \ Path\ \ toj, Inf\ \ Stat\_j) = P_t(Inf\ \ Path\ \ to\ \ j) \times P_t(Infection|Inf\ \ Path\ \ to\ \ j)\\
&\times P_t(Inf\ \ Stat\_j|Infection, Inf\ \ Path\ \ to\ \ j) \times P_t(Connection\ \ to\ \ j|Infection, Inf\ \ Stat\_j, Inf\vee'/Path\ \ to\ \ j)\\
&\times P_t(SRC\ \ Port|Connection\ \ to\ \ j, Inf\ \ Path\ \ to\ \ j, Infection) \times P_t(ACK|Connection\ \ to\ \ j, SRC\ Port,\\
&Inf\ \ \vee'/Path\ \ to\ \ j, Infection) \times P_t(TCP\ \ CNT|Connection\ \ to\ \ j, SRC\ \ Port, ACK) \times P_t(DSTPort|\\
&TCPCNT, Infection, Connection\ \ to\ \ j) \times P_t(INF\ \ CNT|Inf\vee'/\ \ Stat\_j, Connection\ \ to\ \ j, TCP\ \ CNT)\\
&\times P_t(SYN\ \ CNT|Connection\ \ to\ \ j, TCP\ \ CNT) \times P_t(OUT\ \ Degree|Connection\ \ to\ \ j, TCP\ \ CNT)\\
&\times P_t(DST\ \ IP|Connection\ \ to\ \ j, TCP\ \ CNT, SRC\ \ Port)
\end{aligned} \tag{10}$$

Since we aimed at deriving the infection probability of each node at all times, we calculated the marginal distribution of Equation (10) over the *Infection* feature in Equation (11). $P_t(Infection)$ was a stochastic process and showed the infection probability of node i at time t.

$$\begin{aligned}
P_t(infection) = &\sum_{OUT\ Degree}\ \sum_{Inf\ CNT}\ \sum_{TCP\ CNT}\ \sum_{ACK}\ \sum_{DST\ IP}\ \sum_{SRC\ Port}\\
&\sum_{DST\ Port}\ \sum_{SYN\ CNT}\ \sum_{Inf\ Path\ \ to\ j}\ \sum_{Inf\ Stat\_j}\ \sum_{Connection\ to\ j} \text{Equation (10).}
\end{aligned} \tag{11}$$

The first and last moments at which this probability exceeded a predefined threshold were considered as the infection and removal time of the node, respectively.

6.3 | Analysis of the learned structure

The structure of the learned network (Figure 7) indicates the dependency relationships between selected features. Figure 7 exploits the conditional dependence property between *Infection* variable and *Inf Path to j*, *Inf Stat_j*, *DST Port*, *SRC Port*, *ACK, and Connection to j* features. It is notable that *DST Port*, *SRC Port*, *Inf Stat_j*, *Connection to j*, and *ACK* were directly dependent on *Infection*. As the infection status of a node changes from susceptible to infected ($Infection = 1$), the values

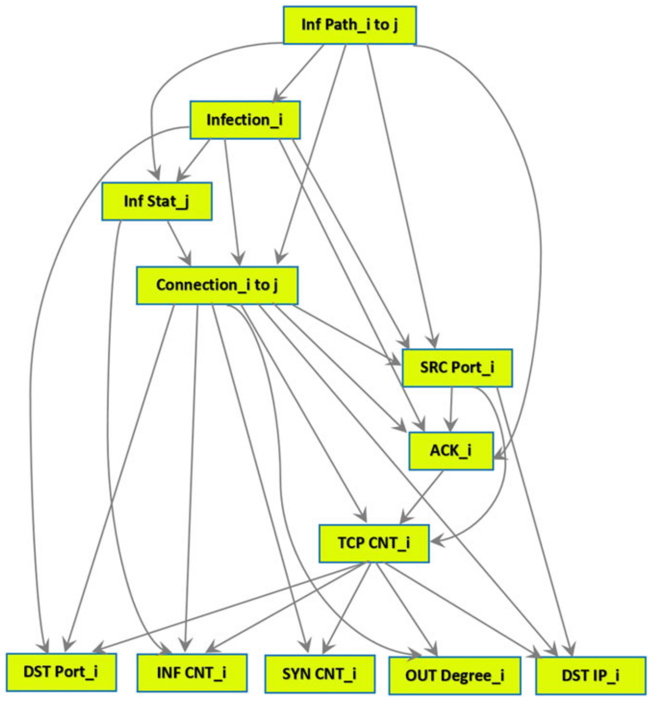

FIGURE 7 The worm infection probabilistic model

of the aforementioned six features may change (there may be an infection path between nodes *i* and *j* and the infection status of node *j* may change). If there was an infection path between *i* and *j*, the numbers of the following features could change: *SRC Port*, *ACK*, *TCP CNT* and *infection*. It could also affect the values of *Inf Stat_j* and *Connection to j* variables. Moreover, TCP/IP header features also had dependency relationships with each other, as in Figure 7.

As mentioned at the beginning of this section, a step-by-step approach was used to estimate the infection status of nodes at each time unit. The stochastic Back-to-Origin Markov model[2] estimates the number of infectious nodes at each time step when a random scanning worm spreads. The model developed at this step, estimated the infection probability of each node at each time step, when the spread of a preference scanning worm happened. At this point, we could combine the two models and use the stochastic Back-to-Origin Markov model to estimate the number of infectious nodes at each step and use the results as the value for (*INF CNT*) in Equation (11). This model has been developed in Section 8.

7 | INFERENCE

In probabilistic graphical models, the probability distribution function is generally applied for inference, answering queries using distribution as the model.[16] We made use of inference techniques to estimate the infection status of nodes at each time using Equation (11). There are many inference algorithms, each of which makes different tradeoffs between speed, accuracy, complexity, and generality.[23] Inference algorithms can be classified into the two following main categories: exact algorithms and approximate algorithms. Moreover, BNT toolbox implements many inference algorithms, including jtree_inf_engine, var_elim_inf_engine, enumerative_inf_engine, gaussian_inf_engine, quickscore_inf_engine, and pearl_inf_engine.

In BNT, each inference algorithm is treated as an object and is called an inference engine. An inference engine contains a BN and supports the enter_evidence and marginal_nodes methods. When enter_evidence is called, the engine may perform some processing. On the other hand, when marginal_nodes is called, the engine may do some query-specific processing.[23]

In this research, jtree_inf_engine was used as an implementation for the junction tree algorithm, which is the mother of exact inference algorithms[23] and provides an exact answer to the approximate problems, unlike approximate approaches that provide approximate answers to such problems. The idea of the junction tree algorithm is to find ways to decompose a global calculation into a linked set of local computations. The key point of this approach is the concept of locality.[57] Jtree_inf_engine implements this algorithm in BNT, Matlab.[17,58] It uses a greedy search procedure to find a proper order of nodes.

At this step, 11 feature values (*ACK*, *DST IP*, *SYN CNT*, *TCP CNT*, *SRC Port*, *DST Port*, *OUT Degree*, *INF CNT*, *Connection to j*, *Inf Path to j and Inf Stat_j*), extracted at a time step of worm propagation, were given to jtree_inf_engine to infer the infection probability of the node at that time. Inference engine took the learned BN from the third step and estimated the infection probability of the node at that time unit based on the feature values.

8 | COMBINATION OF THE PROBABILSTIC AND COMBINED MODELS

A secondary model was developed through combining the two models (ie, the probabilistic and the combined models). In order to reduce the number of collected features, the previously developed stochastic Back-to-Origin Markov model[2] was combined with the first model. In this way, there was no need to collect the number of infectious nodes at each time step in the network.

In this step, the number of infectious nodes at each time step of worm propagation was estimated using Equation (6). Therefore, we only needed to extract 10 features at each time step. These 10 features and the estimated number of infectious nodes were given to the inference engine (explained in the fourth step), and the infection probability of the node was estimated.

In order to use the stochastic Back-to-Origin Markov model, we needed to obtain the number of infectious and removed nodes at a time step of worm propagation and the infection and removal rates of worm propagation. In this way, there was no need to obtain the number of infectious nodes (*INF CNT* feature) at each time step, which reduces the number of required features.

9 | EXPERIMENTAL EVALUATION

In this section, some experiments were performed to assess the suitability of our models for infection probability estimation at every time unit. We tested the models with two scenarios as explained below.

9.1 | Scenario development

In order to evaluate the learned model, we generated test datasets for two different scenarios. These test datasets were a combination of Code Red II worm propagation traffic generated in GTNetS environment and ISCX normal traffic for 300 msec. Twelve features were extracted from test datasets. Test scenarios have been explained in the following paragraphs.

- Scenario 1: The goal of this scenario was to generate test datasets containing all selected features to evaluate the accuracy of the first model (Equation (11). In order to achieve this, Code Red II was propagated in the same network architecture shown in Figure 3, with network parameters similar to those explained in Section 4.1 in the GTNetS environment. Code Red II was propagated for 11 times (test numbers 1 to 11) in the network and the generated data were combined with normal ISCX traffic for a period of 300 milliseconds. In these tests, different source nodes were chosen. The source node for test number 1 was 192.168.1.101. The features were extracted from the test datasets. To evaluate the accuracy of the model, 11 features for two or three IP addresses were given to the learned model (Equation (11) to infer the state of nodes at each time step. Based on the principles explained in Section 9.2, the inferred infection periods of nodes were extracted from the test datasets. Based on the values of the "Infection" feature, the actual infection periods of nodes were also extracted.
- Scenario 2: The goal of this scenario was to generate a test dataset containing feature values for testing the accuracy of the second model, explained in Section 8. Similar to scenario 1, again, Code Red II was propagated in the GTNetS environment with the same architecture and network parameters for 300 milliseconds. The generated network data were combined with ISCX normal traffic (test number 1, scenario 2). The source node in this scenario was "192.168.1.101." Twelve features were extracted from this dataset. The values of 10 features for three IP addresses (192.168.1.101, 192.168.1.102, and 192.168.1.104), and the number of infected nodes at each time step were given to the model. In this scenario, the number of infectious nodes was estimated from the prior stochastic Back-to-Origin Markov model, using Equation (6), based on the number of infectious and removed nodes at a time step of worm propagation. The infection probabilities of three nodes were inferred at each time step. The inferred infection periods of the nodes were extracted from dataset based on the principles pointed out in Section 9.2.

9.2 | Experiments

The two scenarios explained in Section 9.1 were implemented. The probability distribution function created in Equation (10) was used for inference. We employed jtree_inf_engine in BNT toolbox as mentioned in Section 7. To calculate the marginal probability ($P_t(Infection)$), jtree_inf_engine was utilized to infer the infection probability of each node at time steps based on the datasets generated in the two mentioned scenarios. In the following, the results of analysis for the first and second scenarios will be explained in detail.

Figures 8 and 9 show the infection probability of three nodes for the first and second scenarios, ie, IP addresses 192.168.1.102 and 192.168.1.104, respectively. The principles used to discuss the figures have been explained in the following paragraphs.

- The first infection time of a node is the first time the infection probability of the node exceeds 0.7, and this probability is higher than the threshold (0.7) at some other time steps within the next 15 millisecond.
- When a node becomes infected, it remains infectious until removal. However, the node will not exhibit the same behavior during this period. On occasions during the infection period, some features may have larger values and the infection probability of the node may be more than the threshold (0.7). During this period, the infection probability of the node may be less than the threshold too. During the infection period, the feature values of the node change and perhaps become less than the threshold. Therefore, the infection probability of the node can decrease in some steps.
- A node is infected for more than 1 or 2 milliseconds; therefore, the peaks in Figures 8A, 8B, and 8C and Figures 9A, 9B, and 9C, for a period of 1 or 2 milliseconds, are outliers and are not considered.

- The node removal time is the last time the probability of infection is more than the threshold, and there is no increase in the probability of infection for a time period in the range of 1 to 15 milliseconds. The node will not be infected after this time.
- The distance between infection and removal times is called the infection period.
- When the infection probability of a node is lower than the threshold (0.7) and the corresponding time is outside the infection period, the node is healthy (being either susceptible or removed).

The covers of the node infection probability curves for the first and second experiments have been shown in Figures 8D, 8E, and 8F and Figures 9D, 9E, and 9F. The node infection probability was covered with the highest probability value within node infection and removal times. The node infection probability was more than threshold during the infection period and has been shown by the cover signal in Figures 8D, 8E, and 8F and Figures 9D, 9E, and 9F.

Two decision thresholds were chosen in this step.

- The estimated infection probabilities of nodes were compared while observing the Infection feature value at each time step of worm propagation in the training dataset to determine the threshold value for the feature. The estimated infection probability was higher than 0.7, when the value of Infection feature equated one.
- Comparing the predicted probability of the node status with the observed node status (Infection feature) in the training dataset showed that more than one time, the infection probability of the nodes exceeded the threshold level (0.7) during a 15-milisecond period. For each environmental condition, we needed to use a training dataset from this new environment to learn the parameters of BN structure and determine the decision thresholds.

We had also extracted the node infection and removal times from our test datasets for all scenarios as shown in Table 8. The location of nodes in the network have been indicated in Figure 3, and the source node in all scenarios is shown in Table 8.

In Figures 8A, 8B, and 8C, the results of inferred infection probability function for nodes 192.168.1.101, 192.168.1.102, and 192.168.1.104 are shown, where the dataset was generated according to Scenario 1, test number 1. In Figures 8D, 8E, and 8F, the cover of the curves in Figures 8A, 8B, and 8C are plotted, respectively. As it is shown in Table 8, the inferred

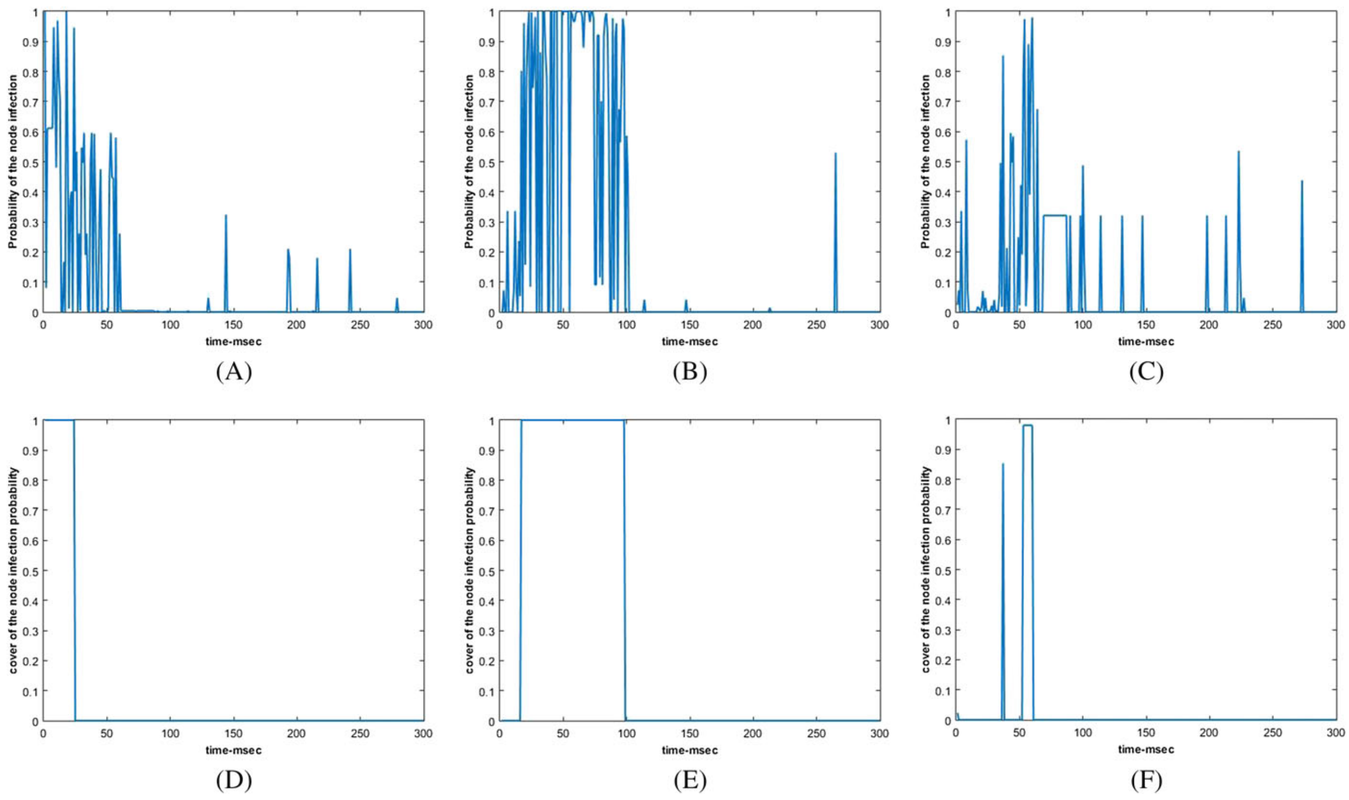

FIGURE 8 The infection probability of nodes inferred by the model of Equation (11) for Scenario 1, different IP addresses (A, B, C) and the covers of the probabilities (D, E, F). A, 192.168.1.101; B, 192.168.1.102; C, 192.168.1.104; D, 192.168.1.101; E, 192.168.1.102; F, 192.168.1.104

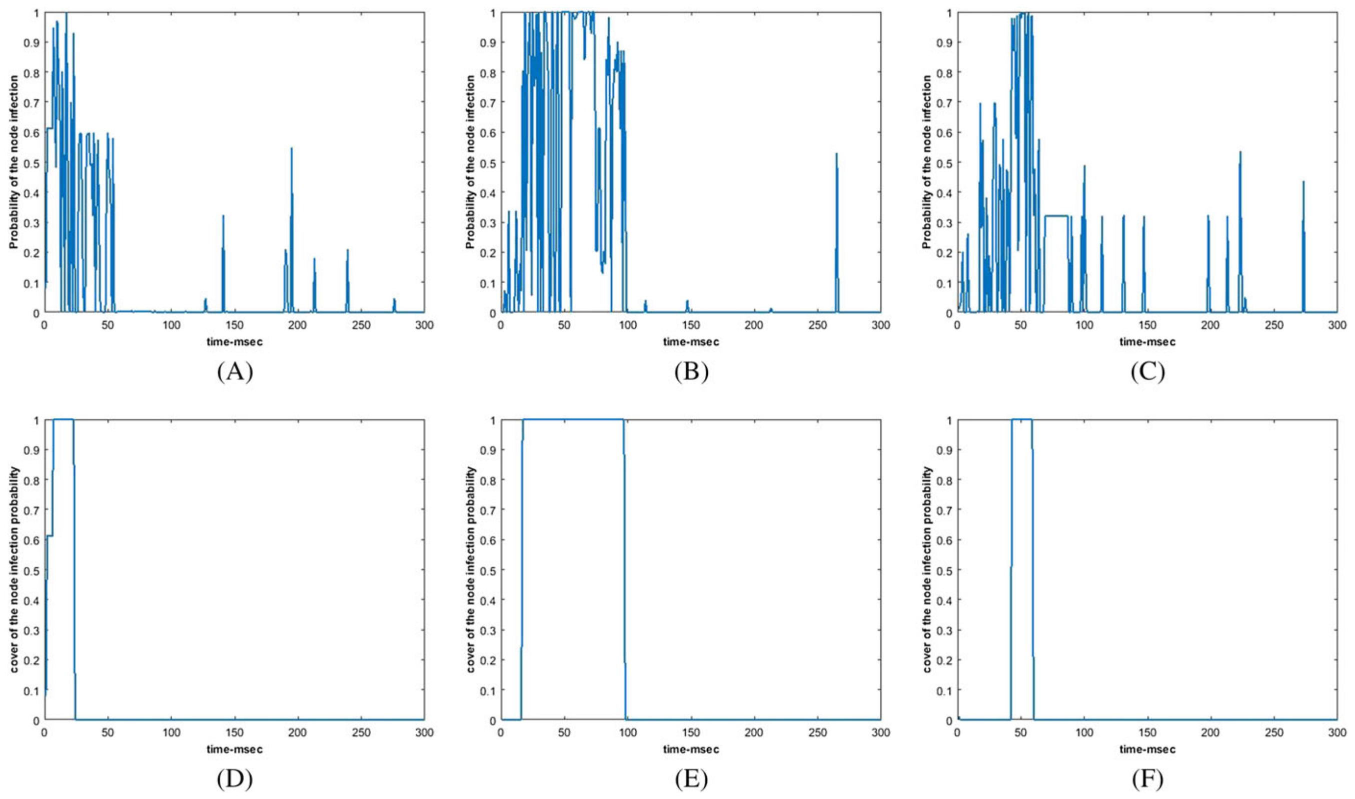

FIGURE 9 The infection probability of nodes inferred by the model of Equation (11) for Scenario 2, different IP addresses (A, B, C) and the covers of the probabilities (D, E, F). A, 192.168.1.101; B, 192.168.1.102; C, 192.168.1.104; D, 192.168.1.101; E, 192.168.1.102; F, 192.168.1.104

infection and removal times for node 192.168.1.101 were 1 and 24, respectively, while the actual infection and removal times were 1 and 28. We counted this case as a True Positive.

The inferred infection and removal times for node 192.168.1.102 were 16 and 98, while the corresponding actual times were 11 and 101, respectively. We also counted this case as a True Positive. Comparing the inferred infection and removal times of the node 192.168.1.104 in Scenario 1, test number 1, with real infection and removal times, Figures 8C and 8F, shows that the model has investigated the times correctly and this test is counted as True Positive.

The figures for probability distribution functions and the cover curves for test numbers 2 to 11 were not plotted but the results were shown in Table 8. In Scenario 1, test number 3, it was inferred that the node 192.168.3.117 had not been infected, while the actual data showed that the node has been infected at 56 and removed at 87. This case is a False Negative error. In Scenario 1, test number 5, it was inferred that the node with IP address 192.168.1.105 had not been infected but it was the origin of the propagation and has been infected and removed at 1 and 20. This case is another False Negative error.

In Scenario 1, test number 11, it has been inferred that the node 192.168.2.109 has been infected and removed at the seconds 40 and 67, but the node has not been infected in the actual scenario. Therefore, this case is a False Positive error. Other tests overlook True Positive results.

The results show that our algorithm could estimate the node's infection and removal times with high accuracy. The combination of two models gives high results in Scenario 2. In both models, the curve conforms to real data.

The True Positive and False Negative values were obtained through comparing the inferred infection and removal times shown in Figures 8D, 8E, and 8F and Figures 9D, 9E, and 9F (as indicated in Table 8) and the results of the tests performed in Scenario 1 (Table 8) using the actual infection and removal times extracted from datasets (see Table 9). The number of matches, mismatches, and False Positives in two scenarios are summed up in Table 9. Using these data, the precision and recall of the method were measured, as have been represented in Table 10. In this way, 96% of the infection periods of worm propagation were correctly detected, and 92% of the node infections were detected by our method.

TABLE 8 Infection and removal times of nodes in three scenarios

Scenario no.	Test no.	IP add.	Inferred infection Time, ms	Inferred removal Time, ms	Origin	Real infection time	Real removal time
1	1	192.168.1.101	1	24	*	1	28
1	1	192.168.1.102	16	98		11	101
1	1	192.168.1.104	53	60		46	59
2	1	192.168.1.101	1	23	*	1	28
2	1	192.168.1.102	17	92		11	101
2	1	192.168.1.104	43	58		46	59
1	2	192.168.1.103	1	28	*	1	23
1	2	192.168.2.110	34	78		40	80
1	3	192.168.2.111	1	40	*	1	38
1	3	192.168.3.117	–	–		54	87
1	4	192.168.3.115	1	35	*	1	35
1	4	192.168.4.120	80	115		67	110
1	5	192.168.1.105	–	–	*	1	20
1	5	192.168.4.118	20	40		16	45
1	6	192.168.2.107	1	35	*	1	30
1	6	192.168.3.117	50	70		45	80
1	7	192.168.4.119	1	25	*	1	27
1	7	192.168.1.103	90	115		88	120
1	8	192.168.5.122	1	18	*	1	18
1	8	192.168.1.101	35	80		36	76
1	9	192.168.1.101	1	23	*	1	20
1	9	192.168.2.106	43	56		40	60
1	10	192.168.3.116	1	30	*	1	30
1	10	192.168.2.108	50	66		51	67
1	11	192.168.1.102	1	27	*	1	28
1	11	192.168.2.109	40	67		–	–

TABLE 9 Match and mismatch resules for experiments

	Matches (true positive)	Mismatches (false negative)	False (false positive)	Total
Experiments	23	2	1	26

TABLE 10 The accuracy of our proposed method

	Precision	Recall
Experiments	$\frac{23}{23+1} = 0.96$	$\frac{23}{23+2} = 0.92$

10 | COMPARISON

In this paper, we built a model based on historical features of worm propagation and then inferred the infectious state of nodes in new experiments. In this section, the proposed model has been compared with some well-known origin identification methods.

In Section 2, we reviewed several source identification and propagation path reconstruction methods, among which only two methods dealt with worm infections, although the outcomes of other methods could also be used for this purpose. It is notable that the related data and implementation of other methods were not available. Thus, experimental evaluation of our method with other origin identification approaches was not possible. Therefore, in this section, we have compared our method with other methods based on their assumptions. Most origin identification and path reconstruction algorithms are based on limiting assumptions such as having the connection graph, or infection graph or the complete network connections. In this section, we compare our method with five state-of-the-art methods,[5,6,8,10,11] as newer and significant approaches in the field.

- The random MoonWalk algorithm[10] identifies the worm origin and reconstructs its propagation path on the basis of the following assumptions: all network connections are available, the worm propagates in a tree-like network, and there is only one origin in the network.

- Rumor Centrality,[5] Jordan center,[6] and NetSleuth[8] methods identify the origin based on the following assumptions: a snapshot of the network is available, in which the states of the nodes are determined and also the infection graph is given, which is not a realistic assumption. Furthermore, discrete-time sequential propagation models are considered.
- The Back-to-the-past method[11] identifies the origin and assumes to have historical diffusion traces, contact network, etc.

Our method was based on the stochastic model learned in the environment and needed to receive the mentioned feature values at each moment to investigate the infectious state of the nodes. The node that had been infected before others was the origins of worm propagation in the observed network.

On the other hand, the requirements for extracting and storing the infection graph or connection graph have not been explained in related studies.[5,6,8,11] The assumptions of our method seemed to be much simpler than extracting the infection graph and required less storage volumes in comparison with storing connection graph or network flows. Application of our model in a real environment with a different topology or for a different worm sample requires adjusting the model in the new environment or adjust the model parameters learned in the initial environment. It seemed that the learned structure does not change in different environments as it exhibited dependency relations between features that do not change in different conditions (discussed in Section 6.3). In order to tune the parameters in various contexts, one needs to use transfer learning methods[59,60] for parameter learning. For instance, in order to update model parameters to ensure accurate operation in a new environment, a probabilistic model can be used to map the learned parameters to new values suitable for inference in the new environment. In this way, the approach is scaled in the wild to provide an applicable method for identifying the origins of worm propagation. Although we developed the model in a small network, we can use it in large networks.

The time complexity of our proposed method is $O(K \times T \times t)$, where K is the average number of infected nodes, T is the average inference time and t is the investigation period. It is worth mentioning that the inference time is dependent on the number of features. As the number of features is constant, inference time is constant too, mentioned by T. Because T and t are constant values, the time complexity of the algorithm is $O(K)$, which is linear similar to a previous research,[8] though at $O(N^2)$ complexity has been reported in another study.[5]

Table 11 gives a comparison of the assumptions and the complexity of our method with other related methods.

We used detection accuracy and complexity as two metrics for comparing our method with mentioned state-of-the-art methods. From the perspective of detection accuracy, the comparison is as follows. The detection accuracy of each method was extracted from the simulation data reported in the corresponding paper and we did not perform any experiments on them. The detection accuracy of the random moonwalk algorithm,[10] measured in different simulations using various parameter levels, has been reported to be in the range of 0.472 to 0.614 for different host degree distributions. The virus

TABLE 11 A comparison of five state-of-the-art origin identification methods with our method

Paper and method	Approach	Epidemic model	Snapshot info	Infection network topology	Knowledge of network	Number of originators	Complexity
Our method	Probabilistic Backward modeling	N.A.	Network parameters	N.A.	N.A.	Any number	$O(K)$
Random Moonwalk[10]	Walking backwards in time and sampling	N.A.	The whole comm. net. at all times	Tree networks	Employed	Single Origin	Not Indicated
Virus source estimator[5]	Rumor Centrality	SI	Infected nodes at a snapshot	Tree networks	Infection graph	Single Origin	$O(N^2)$
NetSleuth[8]	MDL	SI	Infected nodes at a snapshot	General graph	Infection graph	Any number	$O(k(\epsilon_I + \epsilon_F + \nu_I))$
Back-to-the-past[11]	Historical diffusion model learning	SI	Incomplete diffusion traces	N.A.	Contact network	Single Origin	Not indicated
Information source detector[6]	Jordan Center	SIR	Complete snapshot	Tree networks	Network topology	Single Origin	Not Indicated

Abbreviations: SI, Susceptible and Infected; SIR, Susceptible, Infected, and Removed.

source estimator[5] error on different network topologies was almost below 4 hops in a 400 node virus graph, while the average virus graph diameters were eight and seven hops for the AS and power grid networks, respectively. In addition to finding the current number of seeds, Net-Sleuth is also able to identify good quality seeds with a high accuracy. Simulations in the Back-to-the-past method[11] show that when the number of cascades increases, the Top-10 success probability, which indicates that the source is among the Top-10 cases, increases to more than 0.8. The detection rate for information source detector[6] is almost 70%. The precision and recall of our method extracted from the experiments were more than 90%, being higher than the values measured in other methods.

The complexity of our algorithm was linear, while the virus source estimator has a complexity order of N^2. On the other hand, the time complexity of the NetSleuth algorithm is $O(k(\epsilon_I + \epsilon_F + v_I))$, in which K is the optimal seed set size, ϵ_I is the number of edges in the infected subgraph, and ϵ_F is the number of edges connecting the frontier set to infected nodes. Other methods have not been discussed in terms of their complexity.

Comparing the assumptions and complexity of our approaches with other methods indicates that our method needs less amount of prior knowledge; it requires a linear computational complexity with respect to the number of infected nodes and can handle infections with multiple origins. Our proposed method has no assumptions and limitations on network topology, network graph, and infection state of nodes.

11 | CONCLUDING REMARKS AND FUTURE WORKS

In this paper, we studied the problem of probabilistic investigation of node infection and removal time when the spread of a preferential scanning worm happens based on the features extracted from network traffic at each time step. The nodes whose infection occurred before others were the origins of worm propagation. In this way, we could better identify the patient-zero or initially infected node in worm propagation probabilistically. The idea was to extract the timeline of node infections from the probabilistic model.

In this paper, a step-by-step approach was adopted to develop a probability distribution function and probabilistically estimate the node infection state at each time. In order to achieve this, a probabilistic model was first learned from historical feature values over the network (11 features). According to the characteristics of Bayesian networks in modeling the dependency relations between variables, this model was used as the basis for model building. We have previously developed Back-to-Origin models to estimate the number of infectious nodes at each time step probabilistically when the spread of homogeneous random scanning worm happened. The first model was combined with the prior stochastic Back-to-Origin Markov model to develop a secondary model that investigates node infection times probabilistically, using 10 features. In the combined model, the time runs backwards.

In the present paper, a four-step method was used to probabilistically investigate the node infection and removal times for localized scanning worms. These steps included feature selection, prior model learning, optimizing, and building a model with EDA and inference. In order to develop the second model, model combination was performed in the fourth step.

Experiments confirmed the suitability of the probability distribution function in estimating the infectious state of nodes at each time step. The combination of prior Back-to-Origin model with our modeled to acceptable results similar to the first experiment.

Although the general approach of using EDA to learn the distribution model for a social network and employing Back-to-Origin model to find the sources might be applied to other cases, our learned model requires feature collection at the network level and uses these features to make inferences, which are appropriate for worm origin identification on TCP/IP networks. Our algorithm is not suitable for other applications such as finding the sources of rumor in social networks.

An interesting future work would be to extend our method to reconstruct the propagation path with partially observed feature values.

In order to use our historical model learned in a network environment with 22 nodes in a larger network, we need to map the feature thresholds to new values in new environment, which is the subject of our subsequent study.

ORCID

Tala Tafazzoli https://orcid.org/0000-0001-6206-4972
Babak Sadeghiyan https://orcid.org/0000-0002-5947-7570

REFERENCES

1. Jiang J, Wen S, Yu S, Xiang Y, Zhou W. Identifying propagation sources in networks: state-of-the-art and comparative studies. *IEEE Commun Surv Tutor*. 2017;19(1):465-481.
2. Tafazzoli T, Sadeghiyan B. A stochastic model for the size of worm origin. *Secur Commun Netw*. 2016;9(10):1103-1118.
3. Wang Y, Wen S, Xiang Y, Zhou W. Modeling the propagation of worms in networks: a survey. *IEEE Commun Surv Tutor*. 2014;16(2):942-960.
4. Larrañaga P, Lozano JA. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Norwell, MA: Kluwer Academic Publishers; 2002. *Genetic Algorithms and Evolutionary Computation*; vol. 2.
5. Shah D, Zaman T. Detecting sources of computer viruses in networks: theory and experiment. In: Proceedings of 10th International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS); 2010; New York, NY.
6. Zhu K, Ying L. Information source detection in the SIR model: a sample-path-based approach. *IEEE/ACM Trans Networking*. 2014;24(1):408-421.
7. Lokhov AY, Mézard M, Ohta H, Zdeborová L. Inferring the origin of an epidemic with a dynamic message-passing algorithm. *Phys Rev E*. 2014;90(1):012801.
8. Prakash BA, Vreeken J, Faloutsos C. Spotting culprits in epidemics: how many and which ones? Paper presented at: 2012 IEEE 12th International Conference on Data Mining; 2012; Brussels, Belgium.
9. Rozenshtein P, Gionis A, Prakash BA, Vreeken J. Reconstructing an epidemic over time. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD); 2016; San Francisco, CA.
10. Xie Y, Sekar V, Maltz DA, Reiter MK, Zhang H. Worm origin identification using random moonwalks. Paper presented at: 2005 IEEE Symposium on Security and Privacy (S&P); 2005; Oakland, CA.
11. Farajtabar M, Gomez-Rodriguez M, Zamani M, Du N, Zha H, Song L. Back to the past: source identification in diffusion networks from partially observed cascades. In: Proceedings of the 18th International Conference on Artificial Intelligence and Statistics; 2015; San Diego, CA.
12. Alexandru R, Dragotti PL. Rumor source detection in social networks using partial observations. Paper presented at: 2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP); 2018; Anaheim, CA.
13. Cai K, Xie H, Lui JCS. Information spreading forensics via sequential dependent snapshots. *IEEE/ACM Trans Networking*. 2018;26(1):478-491.
14. Kepart JO. *How Topology Affects Population Dynamics*. New York, NY: IBM Thomas J. Watson Research Division; 1992.
15. Leray P, Francois O. Bnt structure learning package: Documentation and experiments. Technical Report, Laboratoire PSI - INSA Rouen; 2008.
16. Koller D, Friedman N. *Probabilistic Graphical Models: Principles and Techniques*. 1st ed. Cambridge, MA: The MIT Press; 2009.
17. How to use Bayes net toolbox. https://www.cs.ubc.ca/~murphyk/Software/BNT/usage_02nov13.html
18. Spirtes P, Glymour C, Scheines R. *Causation, Prediction, and Search*. 2nd ed. Cambridge, MA: The MIT Press; 2001. *Adaptive Computation and Machine Learning*.
19. Chow CK, Liu CN. Approximating discrete probability distributions with dependence trees. *IEEE Trans Inf Theory*. 1968;14(3):462-467.
20. Heckerman D, Geiger D, Chickering DM. Learning Bayesian networks: the combination of knowledge and statistical data. *Machine Learning*. 1995;20(3):197-243.
21. Cooper GF, Herskovits E. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*. 1992;9(4):309-347.
22. Murphy KP. Active learning of causal Bayes net structure. 2001. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.20.8206
23. Chickering DM. Optimal structure identification with greedy search. *J Mach Learn Res*. 2003;3:507-554.
24. de la Ossa L, Sastry K, Lobo F. *X-ary Extended Compact Genetic Algorithm in C++*. IlliGAL report 2006013. Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory; 2006. https://illigal.org/2006/03/08/x-ary-extended-compact-genetic-algorithm-in-c/
25. Mateo J, de la Ossa L. *LiO: An Easy and Flexible Library of Metaheuristics*. Technical report. Albacete, Spain: Escuela Politecnica Superior Universidad de Castilla-La Mancha; 2007.
26. Pelikan M, Sastry K, Goldberg D. *Implementation of the Dependency-Tree Estimation of Distribution Algorithm in C++*. MEDAL report no. 2006010. St. Louis, MO: Missouri Estimation of Distribution Algorithms Laboratory (MEDAL); 2006. http://medal.cs.umsl.edu/files/2006010.pdf
27. Zhang Q, Zhou A, Jin Y. RM-MEDA: a regularity model-based multiobjective estimation of distribution algorithm. *IEEE Trans Evol Comput*. 2008;12(1):41-63.
28. Santana R, Echegoyen C, Mendiburu A, et al. *MATEDA: A Suite of EDA Programs in Matlab*. Research report EHU-KZAA-IK-2/09. 2009.
29. Santana R, Bielza C, Larrañaga P, et al. Mateda-2.0: estimation of distribution algorithms in MATLAB. *J Stat Softw*. 2010;35(7):41-63.
30. Genetic algorithms - Fitness function. https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_fitness_function.htm
31. Sarnsuwan N, Charnsripinyo C, Wattanapongsakron N. A new approach for internet worm detection and classification. In: Proceedings of the 6th International Conference on Networked Computing (INC); 2010; Gyeongju, South Korea.
32. Martin S, Sewani A, Nelson B, Chen K, Joseph AD. Analyzing behavioral features for email classification. In: Proceedings of the IEEE 2nd conference on Email and Anti-Spam; 2005; Palo Alto, CA.

33. Dong W, Zhang W, Tan CW. Rooting out the rumor culprit from suspects. In: Proceedings of the 2013 IEEE International Symposium on Information Theory (ISIT); 2013; Istanbul, Turkey.
34. Luo W, Peng Tay W, Leng M. Identifying infection sources and regions in large networks. *IEEE Trans Signal Process*. 2013;61(11):2850-2865.
35. Fioriti V, Chinnici M. Predicting the sources of an outbreak with a spectral technique. arXiv:1211.2333. 2012.
36. Pinto PC, Thiran P, Vetterli M. Locating the source of diffusion in large-scale networks. *Phys Rev Lett* 2012;109(6):068702.
37. Agaskar A, Lu YM. A fast Monte Carlo algorithm for source localization on graphs. In: Proceedings Volume 8858, Wavelets and Sparsity XV; 2013; San Diego, CA.
38. Altarelli F, Braustein A, Dall'Asta L, Lage-Castellanos A, Zecchina R. Bayesian inference of epidemics on networks via belief propagation. *Phys Rev Lett*. 2014;112(11):118701.
39. Zhu L, Feng L, Zhang Z. Predicting the propagation path of random worm by subnet infection situation using fuzzy reasoning. *Comput J*. 2012;55(4):487-496.
40. Kang C, Park N, Prakash BA, Serra E, Subrahmanian VS. Ensemble models for data-driven prediction of malware infections. In: Proceedings of the 9th ACM International Conference on Web Search and Data Mining (WSDM); 2016; San Francisco, CA.
41. Moskovitch R, Elovici Y, Rokach L. Detection of unknown computer worms based on behavioral classification of the host. *Comput Statics Data Anal*. 2008;52(9):4544-4566.
42. Tabish SM. *A Graph Theoretic Approach to Malware Detection* [thesis]. East Lansing, MI: Michigan State University; 2012.
43. Agosta JM, Diuk-Wasser C, Chandrashekar J, Livadas C. An adaptive anomaly detector for worm detection. In: Proceedings of the 2nd USENIX Workshop on Tackling Computer Systems Problems With Machine Learning Techniques (SYSML); 2007; Cambridge, MA.
44. Moore AW, Zuev D. Internet traffic classification using Bayesian analysis techniques. In: Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems; 2005; Banff, Canada.
45. Xiang Y, Li Q, Guo D. Online accumulation: reconstruction of worm propagation path. In: *Network and Parallel Computing: IFIP International Conference, NPC 2008, Shanghai, China, October 18-20, 2008. Proceedings*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2008:162-172.
46. Chen Z, Gao L, Kwiat K. Modeling the spread of active worms. Paper presented at: 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM); 2003; San Francisco, CA.
47. Shiravi A, Shiravi H, Tavallaee M, Ghorbani AA. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Comput Secur*. 2012;31(3):357-374.
48. Riley GF. The Georgia tech network simulator. In: Proceedings of the ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network Research (MoMeTools); 2003; Karlsruhe, Germany.
49. Riley GF, Sharif MI, Lee W. Simulating internet worms. Paper presented at: The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004. (MASCOTS); 2004; Volendam, The Netherlands.
50. 1998 Darpa intrusion detection evaluation dataset: simulation network hosts. https://www.ll.mit.edu/ideval/docs/hosts_1998.html
51. LBNL/ICSI enterprise tracing project. http://www.icir.org/enterprisetracing/Overview.html
52. Moustafa N, Slay J. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). Paper presented at: 2015 Military Communications and Information Systems Conference (MilCIS); 2018; Canberra, Australia.
53. Description of Bellcore traces. http://www.sfu.ca/~ljilja/TRAFFIC/Bellcore/bellcore.info
54. DEC-PKT - Hour-long traces of wide-area traffic. ftp://ita.ee.lbl.gov/html/contrib/DEC-PKT.html
55. Akbanov M, Vassilakis V, Logothetis MD. WannaCry ransomware: analysis of infection, persistence, recovery, prevention and propagation mechanisms. *J Telecommun Inf Technol*. 2019:113-124.
56. Symantec. *Internet Security Threat Report: ISTR Ransomware 2017*. An ISTR special report. 2017.
57. Jordan MI. The junction tree algorithm. 2011. https://people.eecs.berkeley.edu/~jordan/courses/281A-fall04/lectures/lec-11-16.pdf
58. https://github.com/IEDMS/BNT-SM/blob/master/lib/FullBNT/BNT/inference/static/@jtree_inf_engine/jtree_inf_engine.m
59. Velásquez R, Enrique Sucar L, Morales EF. Transfer learning for Bayesian networks. In: *Advances in Artificial Intelligence - IBERAMIA 2008: 11th Ibero-American Conference on AI, Lisbon, Portugal, October 14-17, 2008. Proceedings*. 2008:93-102. *Lecture Notes in Artificial Intelligence*; vol. 5290.
60. Soria E, Martin J, Magdalena R, Martinez M, Serrano A. Transfer learning. In: *Handbook of Research in Machine Learning Applications*. Hershey, PA: IGI Global; 2009.

How to cite this article: Tafazzoli T, Sadeghiyan B. A probability distribution function for investigating node infection and removal times. *Trans Emerging Tel Tech*. 2019;e3753. https://doi.org/10.1002/ett.3753