Vulnerability-Aware Resilient Networks: Software Diversity-based Network Adaptation

Qisheng Zhang, Jin-Hee Cho, *Senior Member, IEEE,* Terrence J. Moore, *Member, IEEE*, Ing-Ray Chen, *Member, IEEE*

***Abstract*—By leveraging the principle of software polyculture to ensure security in a network, we propose a vulnerability-based software diversity metric to determine how a network topology can be adapted to minimize security vulnerability while maintaining maximum network connectivity. Our proposed metric estimates the software diversity of the node using the vulnerabilities of software packages installed on nearby nodes on attack paths reachable to the node. Our software diversity-based adaptation (SDA) scheme employs the diversity of each node for edge adaptations. These adaptations include the removal of edges that expose high security vulnerability as well as the potential addition of edges between certain nodes with low vulnerabilities associated with them. To validate the proposed SDA scheme, we conduct extensive experiments comparing our approach with counterpart baseline schemes in real networks. Our simulation results demonstrate that SDA outperforms these existing counterparts. We discuss insights into these findings in terms of the effectiveness and efficiency of the proposed SDA scheme under three real network topologies with vastly different network densities.**

***Index Terms*—Software polyculture, software diversity, shuffling, network resilience, network adaptation, epidemic attacks.**

I. Introduction

A. Motivation

Inspired by the close relationship between the diversity of species and the resilience of ecosystems [53], information and software assurance research has evolved to include the concept of *software diversity* for enhanced security [20, 21, 33, 34, 47]. Due to the dominant trend of software monoculture deployment for efficiency and effectiveness of service provisions, attackers have been granted significant advantages in that acquiring the intelligence needed to compromise a single software vulnerability enables the capability of efficiently compromising other homogeneous system components, such as operating systems, software packages, and/or hardware packages [56]. To deny this advantage, the concept of diversity has been applied in the cybersecurity literature [31]. Randomization of software features has been used to thwart cyber attacks by increasing uncertainty towards a target system whose critical information was known to an attacker previously. The concept of moving target defense (MTD) [24, 42] has been proposed to change the attack surface in order to increase uncertainty and confusion for attackers and software diversity-based security mechanisms have also been used as part of MTD techniques.

Qisheng Zhang, Jin-Hee Cho, and Ing-Ray Chen are with the Department of Computer Science, Virginia Tech, Falls Church, VA, USA. Email: {qishengz19, jicho, irchen}@vt.edu. Terrence J. Moore is with US Army Research Laboratory, Adelphi, MD, USA. Email: terrence.j.moore.civ@mail.mil.

Research has shown that software diversity is closely related to enhancing the immunization of a computer system that halts multiple outbreaks of malware infections simultaneously occurring with heterogeneous and sparse spreading patterns [46]. Hence, the rationale that software diversity reduces malware spreading is quite well known and has been validated for its effectiveness to some extent [20, 21]. This underlying philosophy encompasses a simple principle: *software polyculture enhances security* [20]. Due to the accessibility to the Internet, which enables the distribution of individualized software and cloud computing with the computational power to perform diversification, massive-scale software diversity is becoming a realistic and practical approach to enhance security [33]. In general, software diversity-based approaches have already been applied in various domains, such as operating systems [49], firewalls [40], intrusion detection systems [52], and malware detectors [22]. In particular, a common example of software diversity is the use of various kinds of operating systems (OSs) as a MTD strategy, such as Linux-based OSs, Microsoft Windows-based OSs, FreeBSD-based OSs, Apple iOSs, Android OSs, or Apple macOSs in a given network [25]. Although the benefit of software diversity seems obvious, the secure and transparent implementation of automatic software diversity is highly challenging [34]. In addition, no prior work has considered software diversity metrics as the basis to adapt a network topology to balance network connectivity and system security where each node's software vulnerability is incorporated into estimating each node's software diversity.

In this work, we are interested in developing a software diversity metric to measure a node's software diversity based on software vulnerabilities of intermediate nodes on attack paths reachable to the node.

B. Research Problem

In this work, we develop a software diversity metric for measuring a network topology in terms of minimizing security vulnerabilities against epidemic attacks (e.g., malware/virus spreading) while maintaining a sufficient level of network connectivity to provide seamless service availability. The proposed software diversity metric can be used to make decisions related to which two nodes should be disconnected or connected in order to construct an improved network topology meeting these two goals, minimizing security vulnerability and maximizing network connectivity. However, identifying the optimal network topology requires an exponential solution complexity [55]. In this work, we propose a heuristic method called software diversity-based adaptation (SDA) to generate a better network topology that is resilient against epidemic attacks with a sufficiently high network connectivity where the deployment cost is acceptable. We leverage percolation theory [46], which has been used to describe the process or paths of some liquid passing through a medium. We use site and bond percolation from this theory to model and analyze attack processes and defense or recovery processes. *Site percolation* (i.e., removing a node) [46] is used to model an attacker's behavior in compromising another node, wherein the node being percolated corresponds to the node being compromised (infected) by the attacker. This leads to the disconnection of all edges around the node to reflect its failure or its isolation by an intrusion detection system (IDS). *Bond percolation* is used to model the adaptation of edges between nodes such that connected nodes with high security vulnerability (e.g., two connected nodes have the same software package installed or a neighbor node has high software vulnerability) are disconnected while disconnected nodes with low or no security vulnerability (e.g., two disconnected nodes using a different software with low software vulnerability) can be connected in a given network.

C. Key Contributions

We made the following **key contributions** in this work:

- This work is the first that takes a multidisciplinary approach by considering both the computer science's software diversity to enhance cybersecurity and percolation theoretic network resilience techniques to study the effect of interconnectivity on network connectivity under epidemic attacks. To be specific, we develop network adaptation strategies that determine whether to add or remove edges between two nodes in a given network, aiming to minimize network vulnerabilities against epidemic attacks while maintaining maximum network connectivity. Given that each node is installed with a set of software (we call it a 'software package'), we investigate network resilience and vulnerability depending on how a network topology is connected under epidemic attackers who can exploit the vulnerabilities based on their knowledge on software vulnerabilities.
- We develop a novel software diversity metric that measures a node's software diversity level, representing both the vulnerabilities of attack paths reachable to the node and the network connectivity. To minimize computational complexity in estimating the node's software diversity based on attack path vulnerability, we utilize the only the k-hop local neighborhood of the node. This approach provides a lightweight method to compute each node's software diversity. To prove the effectiveness of this software diversity metric, we use it as the criterion to determine whether to add or remove an edge between two nodes.
- Although most software diversity-based network topology adaptations are studied by shuffling the types of software packages [55, 56], our work takes one step further by changing network topology, which is proven much more effective than its software shuffling counterpart (e.g., graph coloring) in reducing vulnerability to epidemic attacks while maximizing the network connectivity. Further, we broaden the concept of software diversity by both maintaining the use of different software in adjacent nodes and minimizing security vulnerability in each node's ego network (i.e., local network within k-hop), which has not been considered in the state-of-the art. In addition, our proposed software diversity-based network adaptations are lightweight showing acceptable operational cost while achieving minimum security vulnerability and maximum network connectivity, which opens a door for the applicability in resource-constrained, contested network environments.
- We validate the outperformance of the proposed SDA strategy by conducting a comprehensive comparative performance analysis with the following six schemes (see Section V-B): non-adaptation, random adaptation, graph-coloring, and three variants of the proposed SDA strategies. We analyze the effect of key design parameters such as network density, attack density, and the number of software packages available on four performance metrics (see Section V-A), i.e., the size of the giant component, the fraction of undetected compromised nodes, software diversity levels, and defense cost (i.e., shuffling plus network topology adaptation costs). We validate the outperformance of our SDA scheme in three real network topologies covering dense (high), medium dense, and sparse (low) networks [36]. Further, to profoundly understand the effect of various network characteristics, we conduct sensitivity analysis under a random graph using the Erdös-Rényi (ER) network model and analyze the results. Due to space constraints, we placed these results for the ER network in Sections C.2–C.3 of the appendix file.

We will discuss the answers to the research questions in Section VI and summarize our findings in Section VII.

We conducted preliminary work in [9] to evaluate the performance of the software diversity-based network adaptation algorithms. In this current effort, we substantially extended [9] by providing the following additional contributions:

- We substantially enhanced the proposed software diversity metric as shown in Eq. (4). We considered two types of vulnerabilities to estimate a node's software diversity: The node's vulnerability from the software package installed and the k-hop attack paths reachable to the node. This allowed us to capture an individual node's software diversity in terms of vulnerabilities at both the node-level and the network-level.
- We also introduced the software diversity ranking threshold to allow more flexibility of identifying sufficient candidates for edge adaptations compared to using the software diversity threshold in [9].
- We substantially extended our simulation experiments by using three real world network topologies representing dense, medium dense, and sparse networks and four different metrics measuring both security and performance of a given network. We also conducted extensive experiments for an in-depth sensitivity analysis varying the values of key design parameters.

II. Background & Related Work

This section provides an overview of related work and background literature in terms of the percolation theory studied for network resilience in Network Science and the software diversity-based approaches studied for system security in Computer Science.

Percolation Theoretic Network Resilience: Percolation theory has been substantially used to investigate network resilience (or robustness) in Network Science. *Site percolation* and *bond percolation* are commonly used to select a node or an edge to remove or add, to model the choice of nodes to immunize in the context of epidemics on networks, such as disease transmission, computer malware/virus spreading, or behavior propagation (e.g., product adoption) [14, 46]. Recently, percolation theory was leveraged to develop software diversity techniques particularly to solve a software assignment problem [55, 56] because how nodes are connected matters in propagating malware infection while choosing nodes or edges to add or remove is exactly following the concept of site or bond percolation in percolation theory [46].

The origins of percolation theory come from the mathematical formalization of statistical physics research on the flow of liquid through a medium [19]. Percolation theory has been substantially applied to networks to study connectivity, robustness [5, 46], reliability [37], and epidemics [6, 44]. The percolation process was studied in computer science under the notion of "network resilience" [11, 45, 51], independent of its development in the statistical physics literature. More recent developments in the physics literature have profoundly influenced studies in computer science. These contributions have incorporated the recognition that networks are not derived from a random structure, and failures of nodes, whether from attacks or due to dependent cascades, are not uniformly random [2]. Hence, significant interest has developed in removal processes that model targeted attacks on the network using a centrality metric. In the network science domain, the degree of network resilience is commonly measured based on the size of the giant component (i.e., the largest connected component in a given network), which gives a clear sense of how the network is connected with a portion of existing nodes even after a certain number of nodes or edges are removed. Percolation theory has been used to model various processes on networks in the context of network failures or attacks, e.g., connectivity, routing, and epidemic spreading [11, 45].

Software Diversity-based Cybersecurity: Many approaches have been explored to validate the usefulness of software or network diversity to ensure network security. Chen and May [7] investigated the usefulness of software diversity to enhance security. Huang et al. [27, 28] solved a software assignment problem by isolating nodes with the same software to minimize the effect of epidemic worm attacks. Franz [18] proposed an approach to introduce compiler-generated software diversity for a large scale network, aiming to create hurdles for attackers and eliminate any advantage of knowing the vulnerabilities of a single software. Homescu et al. [23] presented a large-scale automated software diversification to mitigate the vulnerabilities exposed by software monoculture. Yang et al. [55, 56] proposed a software diversity technique to combat sensor worms by solving a software assignment problem, given a limited number of software versions available. The authors used percolation theory to model the design features of software diversity to defend against sensor worms.

Zhang et al. [58] developed a resilient, heterogeneous networking-based system where a single solution was common to increase interoperability. Recently, *network diversity* is proposed as a security metric to measure network resilience against zero-day attacks [57]. Inspired by the network diversity metrics [57], Li et al. [39] further developed the network model and diversity metric based on vulnerability similarity, configuration constraints and multi-label hosts. Hosseini and Azgomi [26] mathematically analyzed the malware propagation under a network with six different types of nodes in an epidemic model. They proved a positive correlation between network security and the degree of network diversity. Prieto et al. [48] proposed an optimal software assignment algorithm with multiple software packages to enhance network resilience under attacks.

Although the above works discussed the concept of software diversity to ensure system security, their aim is to solve a software assignment problem by shuffling different types of software packages among nodes without changing the network topology. Unlike the software assignment approach, we aim to generate an optimal network topology that is resilient against epidemic attacks while maximizing network connectivity. The proposed software diversity metric is designed for each node to make a decision on whether to add or remove an edge based on the vulnerabilities on the attack paths reachable to the node [8, 30].

III. System Model

This section discusses our system model in terms of the network model, the node model, the attack model, and the defense model.

A. Network Model

In this work, we assume that our proposed network adaptation strategies are applicable in networks with multiple controllers that can govern a partition of nodes in the network. Typical examples include a software-defined network (SDN) where each node can be instructed by the one or more SDN controllers it is assigned to [32], an edge computing Internet-of-Things (IoT) system with some edge devices or nodes available to perform high computing tasks [38], a wireless sensor network with multiple cluster headers [29], and a hierarchical mobile ad hoc network with decentralized controllers in charge of governing a subset of the nodes [10]. In a reconfigurable network [12, 13, 16, 25, 35, 50, 59], regional controllers (e.g., SDN controller(s), or any other network controller(s) in general) can change the flow table in the programmable switches to reconfigure a logical network topology.

Periodic information exchange between nodes and the regional coordinators is required to ensure seamless operations of the system. However, since each node's software diversity value, which is used to make a decision on edge adaptation (i.e., adding/removing edges), is computed locally by each node, a regional coordinator will only need to rank the software diversity values of neighbor nodes around a target node, and inform the target node of which edges to add or remove based on the estimated ranks. Moreover, the ranking operation of the neighbor nodes around a target node is only periodically performed by a regional coordinator and will not require high communication overhead for each node to communicate with the regional coordinator.

A temporal network is an undirected network for which the topology evolution (or change) occurs due to node failures or nodes being compromised by attackers. In addition, the network may change its topology when adaptation strategies are performed by connecting between two nodes or disconnecting all the edges associated with compromised nodes to mitigate the spread of infection over the network. We use an index to label a node (e.g,, node i) and characterize its attributes as described in Section III-B.

An edge between nodes can be on and off depending on the dynamics caused by node failures, node recovery, or edge adaptations (i.e., an edge can be added or removed). We maintain an adjacency matrix $\mathbf{A}$ in order to keep track of direct or indirect connectivities (i.e., edges) between nodes where $a_{ij} = 1$ indicates there exists an edge between nodes i and j while $a_{ij} = 0$ indicates that no edge exists.

In order for each node to efficiently estimate its software diversity by considering the vulnerabilities of attack paths reachable to the node, it only considers neighboring nodes within k-hop distance from itself. This k-hop local network is used for each node to estimate its software diversity value by considering the vulnerabilities of attack paths within its local network.

Although we utilize a sufficiently small value of k (e.g., 1 or 2), it does not underestimate the vulnerabilities of possible attack paths because using smaller k means that an attacker is nearby within the local network. For example, if an attacker wants to compromise a particular target node, it may try multiple attack paths where each attack path has a set of intermediate nodes. When the attack path is long, it means the vulnerability of the target node is low as the attacker needs to compromise all the intermediate nodes in order to finally compromise the target node. However, when the path length is small, it does not necessarily decrease the attack path vulnerability because the attacker is close to the target node.

We assume that software packages installed in each node and the associated vulnerabilities information are given to the regional coordinator in the initial network deployment period. In addition, we assume that each node is also well informed about the software vulnerability information associated with the software packages installed in the neighboring nodes in its k-hop local network. We assume that the changes of network topology are mainly made by node failures or network adaptations in this work.

Adding or removing an edge between two nodes requires secure communications between them. Even if they are within wireless range of each other but don't share a secret key for secure communications, they are not logically connected. In this work, generating an optimal network topology which is resilient against epidemic attacks with maximum network connectivity is based on a logical network topology.

B. Node Model

Each node i is characterized by its attributes as follows:

- Node i's status on whether it is active or not (i.e., has sufficient energy and responsiveness regardless of being compromised or not), denoted by na_i, indicating whether it is alive ($= 1$) or not ($= 0$), respectively;
- Node i's status on whether the node is compromised ($= 1$) by an epidemic attack or not ($= 0$), denoted by nc_i;
- Node i's software package installed, representing the diversified package or the version of the same software providing the same functionality. In this work, we adopt the well-known software diversification approach called N-*version programming* [3, 4]. This concept means that a software has multiple independent implementations. While these different implementations of the software can still provide the same functionalities, since the implementations are different they naturally have different bugs or vulnerabilities. Following this concept, we model the node's software package installed, denoted by s_i, with a limited number of software packages available, N_s, so that s_i is an integer, ranged in $[1, N_s]$. The node's software package type stays the same during the adaptation process and our SDA algorithm aims to answer how to adjust edges between two nodes to minimize security vulnerability while maximizing network connectivity. However, a regional coordinator should have knowledge of the software package information of the nodes under its region during the deployment phase;
- Node i's degree of software diversity, sd_i, whose physical meaning is how different node i's software package is from its neighbors. The computation of node i's software diversity are described in Eq. (4); and
- Node i's software vulnerability (sv_i) is the same as the software vulnerability of the software package (comprising multiple software products) installed in node i. The software vulnerability of software package s, denoted by vul_s, is estimated by:

  $$vul_s = 1 - \prod_{k \in S} (1 - v_k), \tag{1}$$

  where S refers to a set of software products in software package s, k refers to the kth software product in S, and v_k is vulnerability of the kth software product estimated based on CVSS scores divided by the maximum CVSS value (i.e., 10). vul_s is calculated as above because software package s is vulnerable when any single software product contained within is vulnerable. Suppose node i is installed with software package s_i. Then, node i's software vulnerability (sv_i) is equal to the software vulnerability of software package s_i (vul_{s_i}).

Based on the above five attributes, node i is characterized by:

$$\mathbf{node}(i) = [na_i, nc_i, s_i, sd_i, sv_i]. \tag{2}$$

If attacker j targets vulnerable node i (i.e., a node that has not been compromised before), which is one of its direct neighbors, the probability that node j infects node i, denoted by β_{ji}, is estimated based on the probability that node j can exploit the vulnerability of node i's software package, s_i. We estimate this probability based on node i's vulnerability to node j, estimated by [20]:

$$\beta_{ji} = \begin{cases} 1 & \text{if } \sigma_j(s_i) > 0; \\ sv_i & \text{otherwise,} \end{cases} \tag{3}$$

where σ_j is a vector of software packages attacker j has learned about their security vulnerabilities. For example, attacker j knows the vulnerabilities of software packages 1 and 3 among 5 packages available. It is denoted by $\sigma_j = [1, 0, 1, 0, 0]$. In this case, the sum of σ_j indicates the total number of software packages for which attacker j knows the security vulnerabilities and so can exploit. Note that it is a dynamic value learned after node j compromises node i via reconnaissance even if their installed software packages are different, i.e., $s_i \neq s_j$. Here sv_i refers to the vulnerability of software package s_i, which can be estimated based on the degree of a Common Vulnerabilities and Exposures (CVE) with a Common Vulnerability Scoring System (CVSS) severity score [1, 17].

C. Attack Model

This work deals with two stages of attack behaviors: An outside attacker before the node is compromised and an inside attacker after the node is compromised but undetected.

(1) Node Compromise by Epidemic Attacks: We consider the so called *epidemic attack* which describes an attacker's infection behavior based on an epidemic model, called the SIR (Susceptible-Infected-Removed) model [46]. That is, an outside attacker can compromise the nodes directly connected to itself, its direct neighbors, without access rights to their settings or files. Typical example scenarios include the spread of malwares or viruses. Botnets can spread malwares or viruses via mobile devices. A mobile device can misuse a mobile malware, such as a Trojan horse, thus acting as a botclient to receive commands and controls from a remote server [43]. Further, worm-like attacks are popular in wireless sensor networks where the sensor worm attacker sends a message to exploit the software vulnerability in order to cause a crash or take control of sensor nodes [55, 56]. Attacker j can compromise its direct neighbor i when node i uses a software package that attacker j can exploit because the attacker knows the vulnerability of the software package. This case happens when s_i is the same as s_j or attacker j learned s_i's vulnerability in the past (i.e., $\sigma_j(s_i) > 0$). When attacker j is installed with a particular software package, s_j, we assume that attacker j knows the vulnerability of its own software package, s_j. Attacker j can learn the vulnerabilities of other software packages although it needs to commit more time and resources to obtain the information of their security vulnerabilities. Node i's vulnerability by attacker j based on these two cases is reflected in Eq. (3). When node i is compromised, node i's status is changed from 'susceptible' to 'infected' indicating that node i is now an attacker. Then, node i can infect other nodes and learn their software vulnerabilities, which are unknown to it. The attack procedures are described in Algorithm 8 of the appendix file.

TABLE I
KEY NOTATIONS AND THEIR MEANINGS

Notation	Meaning
na_i	Node status for active (= 1) or failed (= 0)
nc_i	Node status for compromised (= 1) or legitimate (= 0)
s_i	Software package installed in node i
sd_i	Software diversity value of node i
sv_i	Software vulnerability of node i
vul_s	Software vulnerability of software package s
σ_j	A vector of software packages attacker j learned about their security vulnerabilities
β_{ji}	Probability that node j compromises node i
AP	Attack path
ap_i	A set of attack paths available to node i
apv_{ij}^k	Vulnerability of an attack path j to node i, given the maximum hop distance k
$SD_{\text{diff}}^A(i,j)$	Software diversity difference introduced by adding an edge between nodes i and j
$SD_{\text{diff}}^R(i,j)$	Software diversity difference introduced by removing an edge between nodes i and j

(2) Malicious Behavior of Compromised Nodes Undetected by the IDS: Even if an intrusion detection system (IDS) is assumed to be placed in this work (see Section III-D below), an attacker may not be detected by the IDS and the inside attacker can perform malicious behaviors such as packet dropping attacks (e.g., gray or black hole attacks), data exfiltration attacks, or denial-of-service (DoS) attacks to compromise the security goals in terms of loss of confidentiality, integrity, and availability [15, 54].

D. Defense Model

We assume that a system is equipped with an IDS, which detects infected (i.e., compromised) nodes. When an infected node is detected by the IDS, we denote the detection probability with γ representing the removal probability in the SIR model. The response to the detected node will be performed by disconnecting all the edges connected to the detected attacker, which corresponds to removing the node from the system based on the concept of *site percolation*. Note that the development of an IDS is beyond the scope of this work. We simply consider the IDS characterized by a false negative probability $1 - \gamma$.

IV. SOFTWARE DIVERSITY BASED ADAPTATION ALGORITHM DESIGN

In this section, we describe our proposed software diversity based adaptation (SDA) algorithm design in detail. SDA uses software diversity as a key determinant to select edges to percolate (i.e., add or remove) for mitigating the spreading of compromised nodes by attackers and also to maximize the network connectivity for network resilience.

A. Software Diversity Metric

A node's vulnerability is commonly computed based on the software package installed [20, 21]. However, if the node is connected with many other nodes that are directly or indirectly connected, its potential vulnerability is not simply restricted by the vulnerability of its software package. We use a broader concept of node vulnerability by incorporating the vulnerabilities of attack paths reachable to each node. To better capture the relationship between node vulnerability and network topology, we utilize an attack path AP an attacker can take to successfully compromise a target node. That is, in order to compromise the target node, the attacker needs to compromise all intermediate nodes on the attack path. Hence, we estimate each node's software diversity value as the probability that a node is robust against vulnerabilities from attack paths APs reachable to the node.

To this end, we consider the shortest paths (i.e., maximum k-hop distance paths) between boundary nodes (i.e., nodes in the boundary of a target node's local network) to a target node as attack paths. In addition, to reduce the complexity of measuring each node's software diversity, we use a limited number of attack paths, denoted by l, where each path has at most k-hop distance. Target node i's software diversity based on l attack paths within k-hop distance from node i, denoted by $sd_i(k,l)$, is defined by:

$$sd_i(k,l) := \prod_{j \in \mathbf{ap}_i}^{l} (1 - apv_{ij}^{k}), \tag{4}$$

where $\mathbf{ap}_i$ is a set of attack paths available to node i ranked based on their highest vulnerability and apv_{ij}^{k} is the vulnerability of the attack path j to node i with maximum hop distance k. In order to consider the maximum number of nodes associated with the attack paths, we consider disjointed attack paths (i.e., the j's in $\mathbf{ap}_i$) from the boundary nodes to node i.

B. Software Diversity based Bond Percolation for Network Adaptation

The design objective of SDA is to decide which edges to add or remove in order to maximize the size of the giant component (i.e., the largest network cluster in a network) for maintaining network connectivity and to minimize the fraction of nodes being compromised due to epidemic attacks with minimum defense cost defined in Section V-A.

We have two tasks to determine which edges to remove or add as follows:

1) Estimate the gain or loss as a result of removing or adding an edge. This is determined from the difference between a node's current software diversity value and its expected software diversity value if the edge adaptation is made between nodes i and j. To determine if adding an edge between nodes i and j is beneficial, we compute the software diversity (SD) difference by comparing the SD before and after edge adaptations between nodes i and j:

$$SD_{\text{diff}}^{A}(i,j) = (sd_i - sd'_i) + (sd_j - sd'_j), \tag{5}$$

where, for conciseness, $sd_i = sd_i(k,l)$ and $sd_j = sd_j(k,l)$, which are defined in Eq. (4). sd'_i and sd'_j are the expected software diversity values of nodes i and j after an edge is added. The most promising candidate edge to be added should be an edge with the lowest $SD_{\text{diff}}^{A}(i,j)$. The expected software diversity value of node i after addition of an edge with node j is simply obtained by

$$sd'_i = sd_i(1 - sv_i \cdot pv_j), \tag{6}$$

Algorithm 1 Software Diversity-based Adaptation (SDA)

$N \leftarrow$ The total number of nodes in a network
$\mathbf{DN} \leftarrow$ A vector containing the number of removed edges per node
$\mathbf{A} \leftarrow$ An adjacency matrix for a given network with element a_{ij} for $i,j = 1,\dots,N$
$\mathbf{S} \leftarrow$ A vector of software packages installed over nodes with element s_i for $i = 1,\dots,N$
$\mathbf{SV} \leftarrow$ A vector of the vulnerabilities associated with software packages
$k \leftarrow$ A hop distance given in a node's local network
$l \leftarrow$ A maximum number of attack paths considered for estimating a node's software diversity
$\rho \leftarrow$ A threshold referring to the fraction of edges to be removed when $\rho < 0$ and added when $\rho > 0$
$\mathbf{A}' \leftarrow$ An adjacency matrix after edges are adapted in Step 1
$\mathbf{A}'' \leftarrow$ An adjacency matrix after edges are adapted in Step 2

$\mathbf{A}'' = \mathbf{SDA}(\mathbf{DN}, \mathbf{A}, \mathbf{S}, \mathbf{SV}, k, l, \rho)$

Step 1: $\mathbf{A}' = \mathbf{SDBA}(\mathbf{DN}, \mathbf{A}, \mathbf{S}))$ ▷ Remove edges between two nodes with the same software package based on Algorithm 1 of the appendix file).

Step 2: Add or remove edges locally based on the ranks of the software diversity differences estimated in Eqs. (5) and (7) (Algorithms 4 and 5 of the appendix file)
$\mathbf{SD} \leftarrow$ A vector of software diversity where each element, $sd_i(k,l)$, refers to node i's software diversity value when at most l number of attack paths are considered where each attack path has at most k-hop length.
$\mathbf{PV} \leftarrow$ A vector of estimated maximal attack path vulnerabilities associated with each node. ▷ Algorithm 2 of the appendix file.
candidate $\leftarrow$ A set of edge candidates ▷ Algorithms 4 and 5 of the appendix file.
$\mathbf{T}^{local}, T^{global} = \mathbf{setEAB}(\mathbf{DN}, \mathbf{A}', \rho)$ ▷ Set edge adaptation budget based on Algorithm 3 of the appendix file.
if $\rho > 0$ **then**
  candidate $= \mathbf{GEAC}(\mathbf{A}', \mathbf{SD}, \mathbf{SV}, \mathbf{S}, \mathbf{PV}, \mathbf{T}^{local})$
  ▷ Algorithm 4 in the appendix file.
else
  candidate = $\mathbf{GERC}(\mathbf{A}', \mathbf{SD}, \mathbf{SV}, \mathbf{S}, \mathbf{PV}, \mathbf{T}^{local})$
  ▷ Algorithm 5 in the appendix file.
end if
$\mathbf{A}'' = \mathbf{AdaptNT}(\mathbf{A}', \mathbf{candidate}, \mathbf{T}^{local}, T^{global}, \rho)$
▷ Algorithm 6 in the appendix file.
return $\mathbf{A}''$

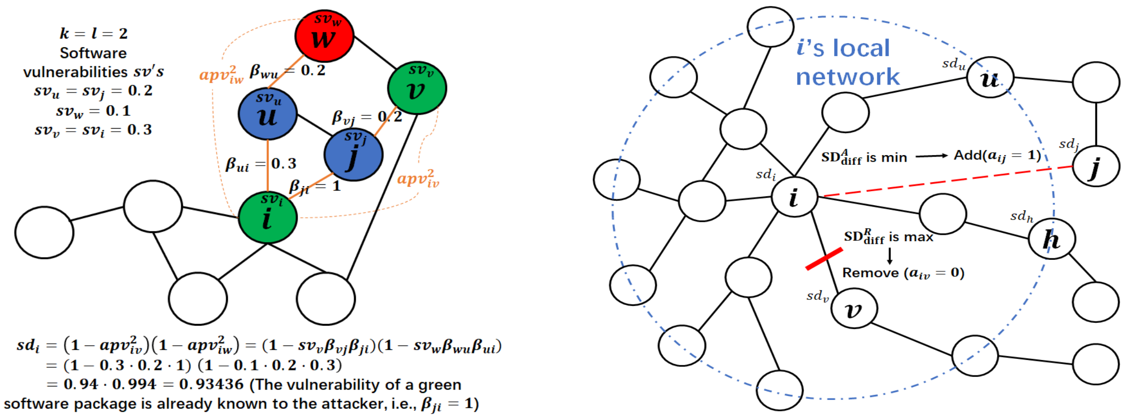
where sv_i is the software vulnerability of the software package installed in node i (i.e., s_i) and pv_j is the estimated attack path vulnerability apv of an attack path from node j to a boundary node in node j's local network. That is, $sv_i pv_j$ is the estimation of attack path vulnerability apv of an attack path from node i to a boundary node in node i's local network through node j. sd'_j is similarly obtained. To determine if removing the edge between nodes i and j is beneficial, we compute the software diversity difference by:

$$SD_{\text{diff}}^{R}(i,j) = (sd'_i - sd_i) + (sd'_j - sd_j), \tag{7}$$

where now sd'_i is computed by:

$$sd'_i = sd_i/(1 - sv_i \cdot pv_j). \tag{8}$$

Here the division by $(1 - sv_i pv_j)$ represents the extent of reducing the vulnerability by removing an edge between



(a) Software diversity estimation of node i (b) Edge adaptations based on software diversity differences

Fig. 1. Example of the software diversity-based adaptation strategies: (a) The estimation of node i's software diversity value; and (b) The edge adaptation based on the software diversity difference in Eqs. (5) and (7).

nodes i and j based on Eq. (4). sd'_j is similarly obtained. The most promising candidate edge to be removed should be an edge with the highest $SD^R_{\text{diff}}(i, j)$. See Algorithm 4 (Generates Edge Addition Candidates or GEAC) and Algorithm 5 (Generates Edge Removal Candidates or GERC) of the appendix file for details on how we generate edge candidates for edge addition and removal, respectively.

2) Estimate how many edges each node can adapt, i.e., remove or add. Based on the rationale that high centrality nodes (e.g., high degree) may expose high vulnerability in terms of security and network connectivity, we minimize the difference between the maximum degree and minimum degree by adding more edges to nodes with lower degree while deleting edges to the nodes with higher degree. Based on this principle, we develop a heuristic method to estimate how many edges should be adapted per node. See Algorithm 3 (Set Edge Adaptations Budget or SetEAB) of the appendix file for detail.

Algorithm 1 details our proposed software diversity-based adaptation (SDA) algorithm. In Step 1, it executes SDBA (see Algorithm 1 of Appendix A) to remove edges between two nodes with the same software package. Then, it makes the decision to add or remove edges locally based on the ranking of software diversity differences estimated in Step 2 using Eqs. (5) and (7), with the objective to best satisfy both security vulnerability and network connectivity requirements. It first sets up the edge adaptation budget based on Algorithm 3 in Appendix A. Then, the SDA generates edge adaptation candidates based on the ranking of software diversity differences (see Algorithms 4 and 5 of Appendix A). As the last step, the SDA adapts the network topology based on edge adaptation candidates and network topology constraints (see Algorithm 6 of Appendix A).

Removing an attack path may increase a chance for attackers to use other attack paths. However, this can make the average vulnerability of existing attack paths significantly plummet because the attack paths are not easily exploitable by the attacker. In addition, we choose adjusting edges (removing or adding) to achieve both security and performance goals instead of using firewalls because networks could face significant performance degradation with high security level firewalls [41].

Fig. 1 illustrates the SDA algorithm execution with an example network where distinct software packages are marked with distinct colors. Fig. 1 (a) illustrates how node i estimates its software diversity value when $k = l = 2$. Fig. 1 (b) illustrates how node i determines whether to add or remove edges based on the software diversity differences, SD^A_{diff} and SD^R_{diff}, based on Eqs. (5) and (7), respectively.

C. Practical Operations of the SDA Algorithm

Several practical real-world examples of network topology adaptation (by reconfiguration) are given below. In an SDN, since its key merit is flexible manageability that can separate data plane from control plane, an SDN controller has been commonly used to reconfigure a logical network topology in its flow table so that packets can be forwarded based on routing instructions given from the SDN controller at node levels [25]. Generating virtual network topologies, called "virtual topology design", in optical networks is well-known to optimize service provision [16, 59]. In wireless sensor networks, network topology reconfiguration has been frequently considered for accurate estimates of sensed data by sensors where a gateway provides each node its next node to which a packet is forwarded [13, 35]. Moreover, by opening sectionalizing and closing tie switches of the network, power distribution systems perform efficient and effective network reconfigurations to minimize their power loss [12, 50].

An example of day-to-day operations can include network configurations, parameter updates, and maintenance operations upon attacks or outages when the proposed SDA algorithm is applied in a given network. Since there is a potential for service degradation while SDA is actively being applied, there is a tradeoff between the frequency of implementation of SDA thereby enhancing network survivability and the service performance of the system. We currently envision an infrequent active implementation to limit any effect on network service during the execution of the adaptations. These operations can be performed based on the following procedures: (1) *Network Configurations*: A network needs to be configured with the key design parameters that the SDA algorithm requires. For instance, we need to configure the values of key parameters (see Table II) affected by the network density and system constraints of the current state of the network. (2) *Parameter Updates*: As network and environmental conditions may vary due to network topology changes (e.g., node mobility or failure) or attacks, each node needs to calculate its software diversity value based on the change and dynamics periodically. We assume that each node's software diversity value will be updated upon an event or time interval; (3) *Maintenance*: Each node will inform its software diversity value to the regional coordinator periodically. We assume that all network configuration information is backed up and can provide redundancy to maintain reliability and resilience. Under some situations caused by power outage, operational failures, or successful internal and external attacks, the backup information of the network configuration will be used. Since each node performs periodic calculation of its software diversity value based on the changed network conditions and relays this value to the regional coordinator, no additional overhead will be generated.

V. Experimental Setup

In this section, we describe the performance metrics, the counterpart baseline schemes against which our proposed SDA algorithm (i.e., Algorithm 1) is compared for performance comparison, and the simulation environment setup for performance evaluation.

A. Performance Metrics

We use the following performance metrics:

- **Software diversity (SD)**: This metric measures the mean software diversity for all nodes in a network. Since node i's software diversity, i.e., sd_i, is computed based on Eq. (4), the mean software diversity for all nodes in the network is obtained by:
$$SD = \frac{\sum_{i=1}^{N} sd_i}{N}. \tag{9}$$
Recall that k is used to determine node i's local network and thus is the maximum possible hop distance from node i to all other neighboring nodes in its local network. Higher software diversity is more desirable to ensure high system security.
- **Size of the giant component (S_g)**: This metric captures the degree of network connectivity composed of non-compromised (uninfected), active nodes in a network. S_g is computed by:
$$S_g = \frac{N_g}{N}, \tag{10}$$
where N is the total number of nodes in the network and N_g is the number of nodes in the giant component. Higher S_g is more desirable, implying higher network resilience in the presence of epidemic attacks.
- **Fraction of compromised nodes (P_c)**: This metric measures the fraction of the number of compromised nodes due to epidemic attacks over the total number of nodes in a network. This includes both currently infected (not detected by the IDS) and removed (previously infected and detected by the IDS) nodes. P_c is computed by:
$$P_c = \frac{N_c}{N}, \tag{11}$$
where N_c represents the total number of compromised nodes after epidemic attacks on a network (i.e., the original network under No-Adaptation and an adapted network under all adaptation schemes). See Section V-B for a listing of counterpart baseline schemes against which our proposed SDA algorithm is compared for a comparative performance analysis.
- **Defense cost (D_c)**: This metric measures the defense cost associated with the following defense strategies employed by an adaptation scheme: (1) edge adaptations (i.e., adding or removing edges) to isolate detected attackers (or compromised nodes) by the IDS; (2) edge adaptations to maximize software diversity by each node based on the value of the software diversity metric in Eq. (4); and (3) shuffling operations based on the fraction of nodes whose software package is randomly shuffled over the total number of nodes. D_c is computed by:
$$D_c = \frac{\mathrm{sum}(|\mathbf{A} - \mathbf{B}|)}{\mathrm{sum}(\mathbf{A} + \mathbf{B})} + \frac{N_{SF}}{N} \tag{12}$$
In the first term, the numerator refers to the differences of edges between the adjacency matrix of an original network $\mathbf{B}$ and that of an adjusted network $\mathbf{A}$ after edges adaptations are made. The denominator is the sum of the addition of the two matrices. In the second term, N_{SF} is the number of nodes whose software packages are shuffled and N is the total number of nodes. Note that when a node's software package is shuffled but stays with its original software package, it is excluded from counting toward N_{SF}. This shuffling cost is estimated only when shuffling a software package is used such as random graph coloring, which is compared against our proposed SDA scheme in our work. Lower defense cost is more desirable.

B. Counterpart Baseline Schemes for Performance Comparison

In this work, we compare the performance of our proposed SDA scheme against No-adaptation (No-A), Random adaptation (Random-A), and Random graph coloring (Random-Graph-C) counterpart baseline schemes for a comparative performance analysis.

Our SDA scheme uses the software diversity-based metric in Eq. (4) to select an edge to remove or add based on the concept of bond percolation, as discussed in Section II. To be specific, SDA first removes all edges between two connected nodes with the same software package as shown in Step 1 of Algorithm 1 (i.e., executing Algorithm 1 in the appendix file). Then SDA decides a set of edges to be added or removed given ρ (the percentage of edges to be added if $\rho > 0$ or to be removed if $\rho < 0$) as shown in Step 2 of Algorithm 1. The effect of ρ on performance will be analyzed in Section V-C3 to identify the optimal ρ value that can best balance security and network connectivity. We experiment with various ρ values in the range of $[-1, 1]$ where -1 means removing all edges (such that no edges exist in the network) and 1 means fully restoring edges removed from Step 1. For example, SDA with $\rho = 1$ means fully restoring edges lost from Step 1 while SDA with $\rho = 0$ refers to only executing Step 1 (removing edges between two nodes with the same software package). SDA with $\rho = 0.6$ means only restoring 60% of the edges lost in Step 1 while SDA with $\rho = -0.6$ means removing 60% of edges in the network after Step 1. What edges to remove or add (see Step 2 of Algorithm 1) significantly affects network security and resilience.

Below we briefly discuss the three counterpart baseline schemes to be compared against our proposed SDA schemes:

- **No-adaptation (No-A)**: This represents the case in which no adaptation is applied, thus showing the effect of attacks on the performance of the original network. However, we allow an IDS to detect attackers. When the IDS detects compromised nodes with probability γ, all edges connected to the detected attacker will be disconnected in order to isolate the attackers, ultimately resulting in mitigating the spread of compromised nodes in the network. Therefore, when No-A is used, the adaptation cost can be high because the number of edges disconnected is affected by the network topology, which is one of the key factors impacting the degree of network vulnerability.
- **Random adaptation (Random-A)**: This scheme first removes an edge between two nodes with the same software package (i.e., executing Algorithm 1 in the appendix file) and then randomly adds edges between nodes with a different software package (see Algorithm 7 in the appendix file). In this scheme, we add the same number of edges lost due to the execution of Step 1.
- **Random graph coloring (Random-Graph-C)**: This scheme uses a simple rule for each node to shuffle its software package with the least common software package without changing any network topology. As a special case, when a node has many neighbors, it may choose the least common software package of those used among its neighbors. It may occur that a node shuffles to its original software package. In such a case, when the shuffled software package is the same as the original software package, we do not count it toward the shuffling cost in Eq. (12). We treat this scheme as an adaptation scheme because it also involves changing a configuration of its software by using a different implementation although it does not make any change to the network topology.

The pseudocode for SDA is presented in Algorithm 1 and that for Random-A is described in Algorithm 7 of the appendix file. In our experiment, we compare the performance of No-A, Random-A, Random-Graph-C, and three variants of SDA with three different thresholds ρ in terms of the 4 performance metrics discussed in Section V-A. We treat Random-A, Random-Graph-C, and the SDA schemes as adaptation schemes while No-A is treated as a baseline scheme without adaptation.

TABLE II
KEY DESIGN PARAMETERS, THEIR MEANINGS, AND THEIR DEFAULT VALUES.

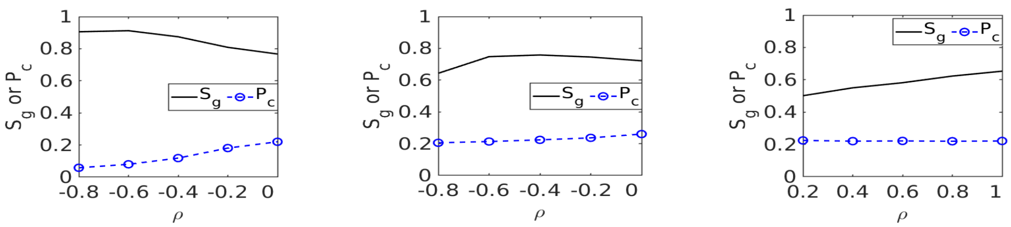
Param.	Meaning	Value
N	Total number of nodes in a network	1000
p	Connection probability between pairs of nodes in a ER network	0.025
γ	Intrusion detection probability	0.95
k	The upper bound of hops considered in calculating software diversity $SD^i_{k,l}$	[1,2]
l	The upper bound of # of paths considered in calculating software diversity $SD^i_{k,l}$	1
n_r	Number of simulation runs	100
N_s	Number of software packages available	[3,7]
P_a	Percentage of attackers in a network	[10,30]
ρ	Threshold of fraction of edges adapted	$[-1, 1]$
SV	A vector of vulnerabilities associated with software packages which are selected based on the uniform distribution with the range in $(0, 0.5]$ (i.e., $U(0, 0.5]$). For the maximum 7 different software packages, the $\mathbf{SV}$ of the corresponding vulnerabilities are used.	$\begin{bmatrix} 0.41 \\ 0.35 \\ 0.48 \\ 0.22 \\ 0.16 \\ 0.19 \\ 0.12 \end{bmatrix}^T$

C. Environment Setup

1) Parameters and Data Collection: Table II summarizes the key parameters, their meanings, and their default values used in this work. We use the average of the performance measures collected based on 100 simulation runs. In the experiment, we examine the effect of the following key design parameters on performance: (1) attack density (i.e., percentage of attackers); and (2) the number of software packages available. For the ER network, we also study the effect of the network connection probability on performance in Section C.3 of the appendix file.

2) Network Topology Datasets: We setup 4 different undirected networks to evaluate the proposed work: (1) a sparse network from an observation of the Internet at the autonomous systems level [36]; (2) a medium dense network derived from an Enron email network [36]; (3) a dense Facebook ego network [36]; and (4) an Erdös-Rényi (ER) random network [46]. The network topologies and their degree distributions are shown in Figs. 1 and 2 of the appendix file. Except for the medium dense network, we use the original network topologies. For the medium dense network, in order to derive a network of comparable size with the other networks (the Enron email network has 36,692 nodes and 183,831 edges) we generate the medium dense network with 985 nodes and 7,994 edges using the following procedure: (i) Rank all nodes in the Enron email network by degree in descending order; (ii) identify the medium dense network as the largest connected component of the induced subgraph consisting of nodes with ranks from 501 to 1500.

3) Optimal Parameter Settings Used for SDA: **Fraction of edges to be adapted** (ρ): We have conducted a sensitivity analysis of ρ for the SDA scheme in terms of maximizing the size of the giant component (S_g) for network resilience without overly increasing the fraction of compromised nodes (P_c) for network security. As shown in Fig. 2, the optimal ρ for the SDA scheme with respect to S_g and P_c in dense, medium dense, and sparse networks have been identified as $\rho = -0.6, \rho = -0.4$ and $\rho = 1$, respectively. Due to space



(a) Dense Network (b) Medium Network (c) Sparse Network

Fig. 2. Effect of ρ (fraction of edges to be adapted) on performance of SDA in terms of the size of the giant component (S_g) and the fraction of compromised nodes (P_c). The optimal ρ for the SDA scheme with respect to S_g and P_c in dense, medium dense, and sparse networks are identified as $\rho = -0.6, \rho = -0.4$ and $\rho = 1$, respectively.

constraints, we have conducted the sensitivity analysis of ρ for the ER random network in Appendix C.2 of the appendix file, from which we have observed the optimal ρ with respect to S_g and P_c for the ER random network is -0.6. In summary, the optimal values of ρ are observed at -0.6, -0.4, 1, and -0.6 for dense, medium dense, sparse, and ER random networks, respectively.

The number of maximum attack paths (l) and the maximum hop distance in each attack path (k): The network type (i.e., dense, medium dense, sparse, or ER random) affects node density which in turn can affect the optimal setting of l and k under which SDA can best achieve both security (i.e., a low fraction of compromised nodes) and network resilience (i.e., a large size of the giant component). We have conducted a sensitivity analysis of l or k on the performance of the SDA scheme in all four types of networks. Due to space constraints, we put the sensitivity analysis of l and k on performance of SDA in Sections D and E of the appendix file. In summary, for dense, medium dense, and ER random networks, we have selected $k = 1$ and $l = 1$ to calculate software diversity $sd_i(k, l)$ for each node in the network because we have observed no significant performance improvement with $k > 1$ and $l > 1$. For the sparse network, we have not observed high sensitivity when $l > 1$. However, for k, we have observed that SDA performs the best when $k = 2$ with $\rho = 1$. Thus, we have selected $k = 2$ and $l = 1$ for the sparse network.

Although network connectivity during the edge adaptation process is highly sensitive to the network density (i.e., the number of edges), a network can be either more or less vulnerable to the epidemic attacks than another network even when both have the same density due to differences in the topology. Since our proposed SDA algorithm has both the removing-only phase and the recovery phase that can adapt edges based on software diversity, it can meet the goals of maximizing network connectivity and minimizing security vulnerability in the network.

VI. Experimental Results and Analysis

In this section, we present the experimental results for a comparative performance analysis of the proposed SDA scheme against the counterpart baseline schemes and provide physical interpretations of the results. In our experiment, we compare 6 schemes: (1) Non-adaptation (No-A); (2) Random adaptation (Random-A); (3) Random graph coloring (Random-Graph-C); (4) SDA with $\rho = 0$; (5) SDA with $\rho = 1$; and (6) SDA with optimal ρ. See Section V-B for more detail on how each scheme is implemented. The 6th scheme "SDA with optimal ρ" is network-type dependent. As discussed earlier in Section V-C3, the optimal values of ρ are observed at -0.6, -0.4, 1, and -0.6 for dense, medium dense, sparse, and ER random networks, respectively.

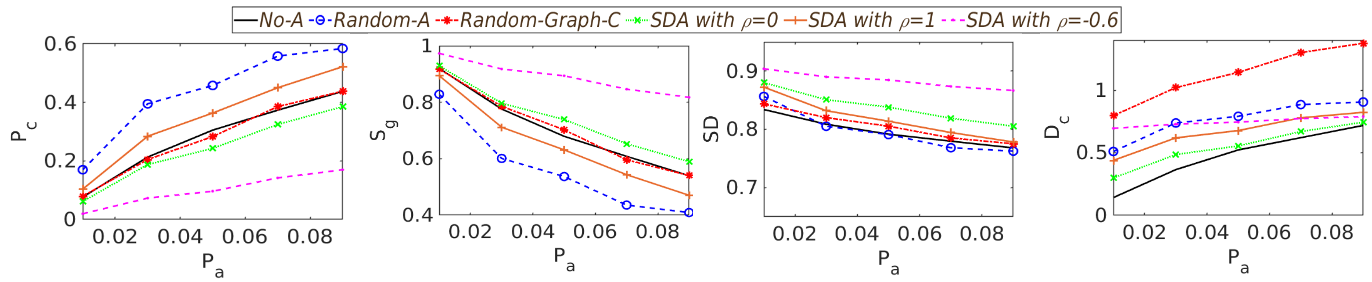
Initially a set of attackers is randomly and uniformly distributed to the network based on the percentage of attackers parameter P_a and all such attackers perform epidemic attacks as described in Section III-C. See Algorithm 8 of the appendix file for detail on how the attackers perform epidemic attacks. Below we only report the experimental results under dense, medium dense, and sparse networks. The experimental results under the ER random network are reported in Section C.3 of the appendix file due to space constraints.

A. Comparative Performance Analysis under a Dense Network

1) Effect of Varying the Fraction of Initial Attacks (P_a): Fig. 3 shows the effect of varying the attack density (P_a) on the performance of the six schemes in terms of the four metrics in Section V-A under the dense network, whose network topology and degree distribution are shown in Fig. 1 (a) of the appendix file. We observe that increasing the percentage of attackers (P_a) decreases software diversity (SD) and the size of the giant component (S_g) while increasing the percentage of compromised nodes (P_c) and the defense cost (D_c). We note that when more nodes are compromised, the defense cost would also increase since it requires more site percolation based adaptations to be performed when compromised nodes are detected by the IDS (i.e., for disconnecting all edges of a detected, compromised node).

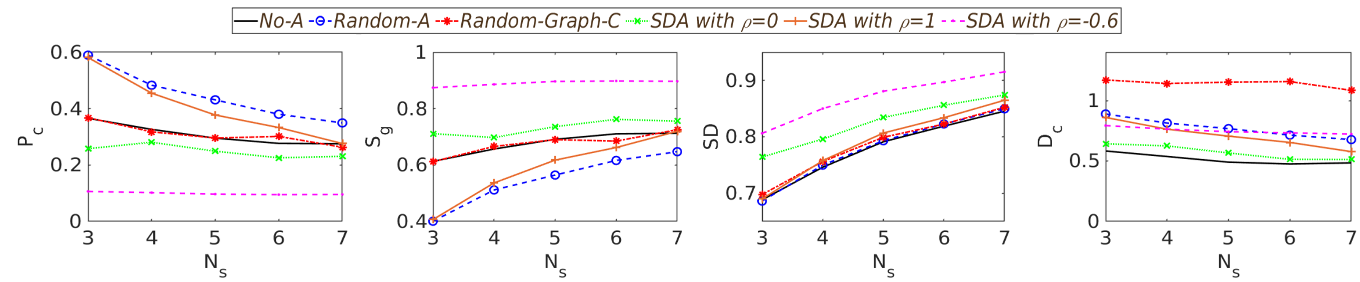
The overall performance order with respect to P_c (representing network security) and S_g (representing network connectivity and resilience) is observed as: SDA with optimal ρ (set at -0.6) $\geq$ SDA with $\rho = 0 \geq$ Random-Graph-C $\approx$ No-A $\geq$ SDA with $\rho = 1 \geq$ Random-A. It is apparent that the network density of a given network significantly affects both security and performance since SDA with $\rho = -0.6$ and SDA with $\rho = 0$ have relatively fewer edges after adaptation and perform better than the other schemes in terms of P_c, S_g and SD (i.e., the average software diversity level), as shown in Figs. 3 (a)-(c).

In Fig. 3 (d), SDA with optimal ρ (set at -0.6) also shows significant resilience with relatively low defense cost (D_c) as



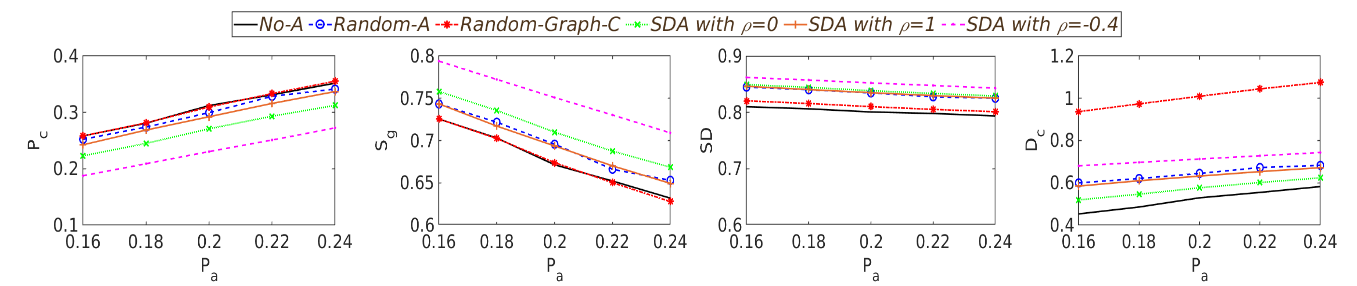
(a) Fraction of compromised nodes (P_c) (b) Size of the giant component (S_g) (c) Software diversity (SD) (d) Defense cost (D_c)

Fig. 3. Effect of varying the fraction of attackers (P_a) under a dense network.



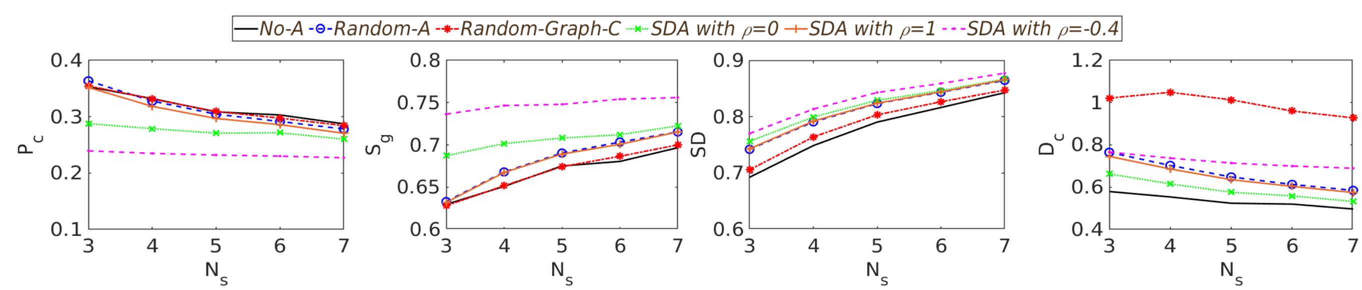
(a) Fraction of compromised nodes (P_c) (b) Size of the giant component (S_g) (c) Software diversity (SD) (d) Defense cost (D_c)

Fig. 4. Effect of the number of software packages (N_s) under a dense network.



(a) Fraction of compromised nodes (P_c) (b) Size of the giant component (S_g) (c) Software diversity (SD) (d) Defense cost (D_c)

Fig. 5. Effect of varying the fraction of attackers (P_a) under a medium network.



(a) Fraction of compromised nodes (P_c) (b) Size of the giant component (S_g) (c) Software diversity (SD) (d) Defense cost (D_c)

Fig. 6. Effect of the number of software packages (N_s) under a medium network.

P_a increases. The overall performance order for the other five schemes in D_c is: Random-Graph-C $\geq$ Random-A $\geq$ SDA with $\rho = 1 \geq$ SDA with $\rho = 0 \geq$ No-A. Not only do the SDA schemes outperform the counterpart baseline schemes in P_c, S_g, and SD, but also the defense cost of SDA schemes are significantly lower than that of Random-Graph-C and are comparable with Random-A (e.g., compared to SDA with optimal $\rho = -0.6$) and No-A (e.g., compared to SDA with $\rho = 0$). This is a significant merit as SDA-based schemes outperform the counterpart baseline schemes with relatively low defense cost.

2) Effect of Varying the Number of Software Packages (N_s): Fig. 4 shows the effect of varying the number of software packages available (N_s) on the performance of the six schemes with respect to the metrics defined in Section V-A under the dense network. We observe that increasing the number of software packages available (N_s) increases software diversity (SD) and the size of the giant component (S_g) while decreasing the percentage of compromised nodes (P_c) and the defense cost (D_c). Note that based on the concept of N-version programming, the number of software packages (N_s) here refers to the number of versions being implemented for the same piece of software. Hence, as N_s increases, the software diversity strength increases, resulting in a decrease of the percentage of nodes being compromised due to attacks, an increase of the network connectivity, and a decrease of the defense cost because less nodes are being compromised.

The overall performance order in P_c, S_g, and SD is very similar to what we observed in Fig. 3, with SDA with optimal $\rho = -0.6$ outperforming all other schemes. For D_c, SDA with optimal $\rho = -0.6$ generates a defense cost comparable to that generated by Random-A and in-between those generated by No-A (lowest cost) and Random-Graph-C (highest cost).

B. Comparative Performance Analysis Under a Medium Network

1) Effect of Varying the Fraction of Initial Attacks (P_a): Fig. 5 demonstrates the effect of varying the percentage of initial attacks on metrics defined in Section V-A under the medium network, whose network topology and degree distribution are shown in Fig.1 (b) of the appendix file. Similar to Fig. 3, Fig. 5 also shows that increasing the percentage of attackers (P_a) decreases software diversity (SD) and the size of the giant component (S_g) while increasing the percentage of compromised nodes (P_c) and the defense cost (D_c).

The overall performance order in terms of P_c (representing network security) and S_g (representing network connectivity and resilience) is: SDA with optimal $\rho = -0.4 \geq$ SDA with $\rho = 0 \geq$ SDA with $\rho = 1 \approx$ Random-A $\geq$ Random-Graph-C $\approx$ No-A. In terms of SD (software diversity), a similar performance order is observed except that Random-Graph-C has a higher SD than No-A. These results demonstrate that SDA schemes clearly are more effective than traditional software shuffling schemes that do not change the network topology (e.g., Random-Graph-C). For the defense cost (D_c), the overall performance order (the lower cost the better) is: Random-Graph-C $\geq$ SDA with optimal $\rho = -0.4 \geq$ Random-A $\approx$ SDA with $\rho = 1 \geq$ SDA with $\rho = 0 \geq$ No-A. Again these results support the claim that SDA-based schemes incur relatively low cost, while outperforming all counterpart baseline schemes in P_c, S_g, and SD.

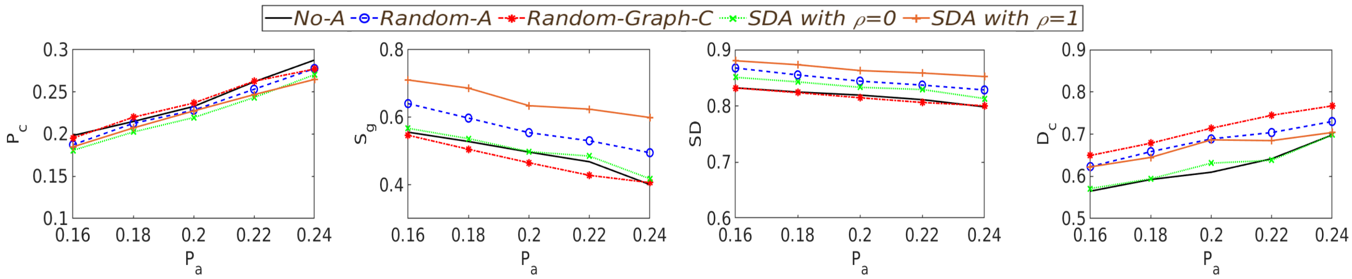
2) Effect of Varying the Number of Software Packages (N_s): Fig. 6 shows the effect of N_s on performance under the medium network. We again observe that increasing the number of software packages available (N_s) increases software diversity (SD) and the size of the giant component (S_g) while decreasing the percentage of compromised nodes (P_c) and the defense cost (D_c). The overall performance order is the same as that in Figs. 5, with SDA with optimal $\rho = -0.4$ outperforming all other schemes in terms of SD, S_g, and P_c and performing comparably to Random-A in terms of D_c. By comparing Fig. 6 (for the medium dense network) with Fig. 4 (for the dense network), we also observe that SDA with optimal ρ is more effective in the dense network. We attribute this to node density. That is, SDA is more effective when there are many connections between nodes in the network allowing SDA to effectively decide which edges to add or remove to effectively maximize software diversity (SD) and the size of the giant component (S_g) thereby minimizing the percentage of compromised nodes (P_c).

C. Comparative Performance Analysis Under a Sparse Network

1) Effect of Varying the Fraction of Initial Attacks (P_a): Fig. 7 shows the effect of varying the initial attack density (P_a) on the performance of the five schemes with respect to the 4 performance metrics discussed in Section V-A under the sparse network, whose network topology and degree distribution are shown in Fig. 1 (c) of the appendix file. Unlike in the cases of the medium and dense networks, the SDA with optimal ρ scheme in the sparse network is the same as the SDA with $\rho = 1$ scheme which restores all edges from the lost edges in Step 1 (i.e., $\rho = 1$). Therefore, we only show comparative experimental results of the five schemes.

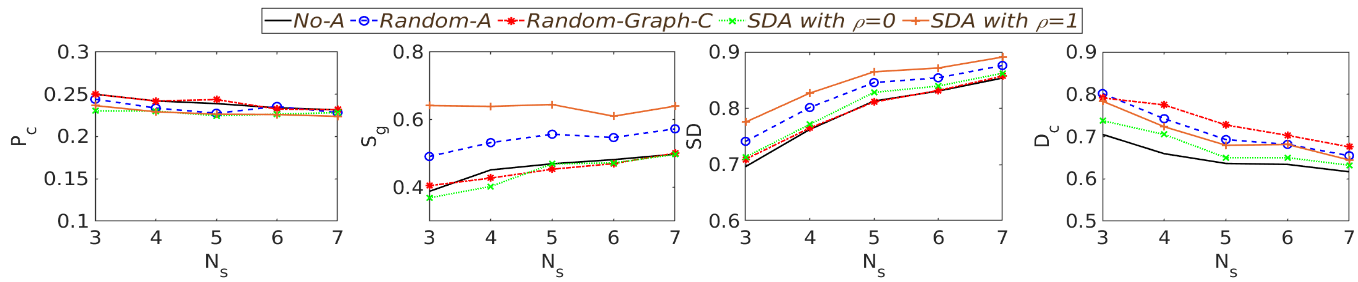
In the sparse network, the degrees of most nodes are very small, implying that nodes are minimally connected where most nodes only have 1-3 neighbors at most. This means that the network itself is relatively much less vulnerable to epidemic attacks because the attackers inherently cannot reach many nodes to compromise due to network sparsity. On the other hand, this means that when there is a higher percentage of attackers, the damage upon an attack success (i.e., failing or compromising a node) is more detrimental by resulting in a much smaller size of the giant component representing a significantly lower network resilience (or availability), which introduces a great hindrance to providing continuous services due to a lack of paths available from a source to a destination. This trend can be clearly observed with the sharp decrease in the size of the giant component (S_g) under high attack density (i.e., $P_a = 0.24$), when compared to the corresponding results under the medium network (i.e., Fig. 5 (b)). A more interesting result is that the overall performance trend does not follow the previous results shown under the dense network (i.e., Fig. 3) and medium network (i.e., Fig. 5) which have a sufficiently larger number of edges than the sparse network. The performance order in S_g is: SDA with optimal $\rho = 1 \geq$ Random-A $\geq$ Random-Graph-C $\approx$ No-A $\geq$ SDA with $\rho = 0$. Since the original network itself is sparsely connected, SDA with $\rho = 0$ is not as effective as shown in our previous results for S_g under the dense network (see Fig. 3) and medium network (see Fig. 5). SDA with optimal $\rho = 1$ with all edges restored from the lost edges in Step 1 performs the best in S_g. This result is reasonable because the sparse network does not need to disconnect more edges because it is already sparse enough and significantly less vulnerable to epidemic attacks.

The overall performance with respect to P_c is very similar among all five schemes, with slightly better results in the two SDA schemes. Similarly to the result shown for the



(a) Fraction of compromised nodes (P_c) (b) Size of the giant component (S_g) (c) Software diversity (SD) (d) Defense cost (D_c)

Fig. 7. Effect of varying the fraction of attackers (P_a) under a sparse network.



(a) Fraction of compromised nodes (P_c) (b) Size of the giant component (S_g) (c) Software diversity (SD) (d) Defense cost (D_c)

Fig. 8. Effect of the number of software packages (N_s) under a sparse network.

dense network, Random-Graph-C exhibits the same level of performance as No-A in P_c and S_g, but with a higher software diversity (SD). This indicates the advantage of topology-aware adaptation in a sparse network. For the defense cost (D_c) the performance order is: Random-Graph-C $\geq$ Random-A $\geq$ SDA with optimal $\rho = 1 \geq$ SDA with $\rho = 0 \geq$ No-A. It is interesting to observe that all SDA-based schemes incur a lower defense cost than Random-A and Random-Graph-C possibly due to fewer compromised nodes in the system and thus less frequent IDS interventions.

2) Effect of Varying the Number of Software Packages (N_s): Fig. 8 shows the effect of varying the number of software packages available (N_s) on the performance of the five schemes under the sparse network. As expected, as N_s increases, SD (software diversity) increases and D_c (defense cost) decreases. As N_s increases, S_g (size of the giant component) also increases for all schemes except for the SDA with optimal $\rho = 1$ scheme. The reason is that when $\rho = 1$, SDA will restore all edges removed in Step 1 (see Step 1 in Algorithm 1). When N_s is higher, fewer edges will be removed in Step 1 because of a smaller probability that two neighbor nodes will have the same software package. Consequently, when N_s is higher, the very same smaller number of edges will be added back in Step 2 (see Step 2 in Algorithm 1), thus resulting in the size of the giant component in the shuffled topology not necessarily larger than the one when N_s is lower.

By comparing Fig. 8 (for the sparse network) with Fig. 6 (for the medium dense network) and Fig. 4 (for the dense network), we notice that SDA with optimal ρ is most effective in the dense network. We conclude that our proposed SDA algorithm is most effective in a dense network under which SDA can effectively decide which edges among many to add or remove to effectively maximize software diversity (SD) and the size of the giant component (S_g) as well as minimizing the percentage of compromised nodes (P_c).

VII. Conclusions

A. Summary

In this section, we summarize the contributions of this work:

- We proposed a software diversity metric based on vulnerabilities of attack paths reachable to each node. We called this scheme 'software diversity-based adaptation' (SDA) and used it to adapt edges to generate a resilient network topology that can minimize security vulnerability while maximizing network resilience (or connectivity) to provide seamless services under epidemic attacks.
- We conducted extensive simulation experiments in order to demonstrate the performance of the proposed SDA scheme compared against other existing counterpart baseline schemes (i.e., random adaptation, random graph coloring, and no adaptation). Via extensive simulation experiments, we found our proposed SDA scheme outperforms counterpart baseline schemes in terms of the fraction of compromised nodes by epidemic attacks, the size of the giant component, and the level of software diversity. In addition, we analyzed the defense cost associated with each scheme and proved the proposed SDA scheme incurs comparable defense cost over existing counterparts.
- We also identified the optimal setting for executing SDA to meet the imposed performance goals. This allows each node to efficiently compute its software diversity value and use it for adapting edges to maximize its software diversity, leading to minimizing security vulnerability while maximizing network connectivity.

- We conducted an extensive simulation study with four different real networks in order to investigate the effect of network density on the optimal setting of SDA under which it can best achieve the dual goals of security (i.e., minimum vulnerability) and performance (i.e., maximum network connectivity).
- We effectively incorporated the techniques of percolation theory in the network science domain into software diversity-based security analysis in the computer science domain. To be specific, in terms of the computer science perspective, the proposed software diversity metric used attack path vulnerabilities, which are derived based on software vulnerabilities of the intermediate nodes on the attack paths. On the other hand, in terms of the network science perspective, this work also adopted percolation theory to examine the effect of software diversity-based edge adaptation on network resilience measured by the size of the giant component. Based on the rationale that network interconnectivity can increase both network vulnerability and network connectivity [5], this work addressed the tradeoff relationship in the context of cybersecurity, which has not been addressed in the literature.

B. Key Findings

From our extensive simulation experiments, we obtained the following key findings:

- Overall under epidemic attacks, more interconnectivity between nodes in a network introduces higher security vulnerability while bringing a larger size of the giant component, implying higher network connectivity. In addition, when two nodes use the same software package where the vulnerability of the software package is known to an attacker, it provides a high advantage to the attacker. How nodes are connected to each other is highly critical in determining the network's vulnerability to epidemic attacks.
- Even if two network topologies have the same network density (i.e., the same number of edges), how nodes are connected to each other can vastly change the extent of the security vulnerability to epidemic attacks. It is even possible that a sparser network may introduce more security vulnerability than a denser network depending on how the nodes are connected to each other.
- It is not necessary to consider the entire network topology for each node to make effective edge adaptation decisions to minimize security vulnerability while maximizing network connectivity. Our SDA algorithm allows each node to make effective decisions on edge adaptation in a lightweight manner. This is because edge adaptation decisions are determined based on ranking of node software vulnerability values, which is more flexible than using a threshold, to achieve the dual goals of security and performance.
- Under medium dense and dense networks, our SDA scheme significantly outperforms existing counterpart baseline schemes. However, under the sparse network, although our SDA scheme still outperforms other schemes, the difference was less significant. We conclude that our SDA scheme is most effective in a dense network under which SDA can effectively decide which edges among many existing connections to add or remove to effectively maximize software diversity and the size of the giant component as well as minimizing the percentage of compromised nodes.
- Our proposed SDA scheme is extremely resilient to harsh environments. The performance gain relative to counterpart baseline schemes increases as the environment is harsher, i.e., as the percentage of attackers increases or as the number of the software packages decreases. This proves the high resilience of the proposed SDA scheme under a highly disadvantageous environment.

References

[1] "Common Vulnerability Scoring System (CVSS)." [Online]. Available: https://www.first.org/cvss/

[2] R. Albert, H. Jeong, and A. Barabási, "Error and attack tolerance of complex networks," *nature*, vol. 406, no. 6794, pp. 378–382, 2000.

[3] A. Avizienis, "On the implementation of N-version programming for software fault tolerance during execution," *Proc. COMPSAC*, pp. 149–155, 1977.

[4] ——, "The N-version approach to fault-tolerant software," *IEEE Transactions on Software Engineering*, vol. 11, no. 12, pp. 1491–1501, Dec. 1985.

[5] A.-L. Barabási, *Network Science*, 1st ed. Cambridge University Press, 2016.

[6] J. L. Cardy and P. Grassberger, "Epidemic models and percolation," *Journal of Physics A: Mathematical and General*, vol. 18, no. 6, p. L267, 1985.

[7] L. Chen and J. H. R. May, "A diversity model based on failure distribution and its application in safety cases," *IEEE Transactions on Reliability*, vol. 65, no. 3, pp. 1149–1162, Sept. 2016.

[8] Y. Chen, B. Boehm, and L. Sheppard, "Value driven security threat modeling based on attack path analysis," in *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*, 2007, pp. 280a–280a.

[9] J.-H. Cho and T. J. Moore, "Software diversity for cyber resilience: percolation theoretic approach," *Nature-Inspired Cyber Security and Resiliency: Fundamentals, techniques and applications*, p. 313, 2019.

[10] J.-H. Cho, I.-R. Chen, and D.-C. Wang, "Performance optimization of region-based group key management in mobile ad hoc networks," *Performance Evaluation*, vol. 65, no. 5, pp. 319–344, 2008.

[11] C. Colbourn, "Network Resilience," *SIAM Journal on Algebraic Discrete Methods*, vol. 8, no. 3, pp. 404–409, 1987.

[12] D. Das, "A fuzzy multiobjective approach for network reconfiguration of distribution systems," *IEEE transactions on power delivery*, vol. 21, no. 1, pp. 202–209, 2005.

[13] A. Desai and S. Milner, "Autonomous reconfiguration in free-space optical sensor networks," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 8, pp. 1556–1563, 2005.

[14] Z. Dezsö and A.-L. Barabási, "Halting viruses in scale-free networks," *Physical Review E*, vol. 65, May 2002.

[15] Q. Do, B. Martini, and K.-K. R. Choo, "Exfiltrating data from Android devices," *Computers & Security*, vol. 48, pp. 74–91, 2015.
[16] N. Fernández, R. J. D. Barroso, D. Siracusa, A. Francescon, I. de Miguel, E. Salvadori, J. C. Aguado, and R. M. Lorenzo, "Virtual topology reconfiguration in optical networks by means of cognition: Evaluation and experimental validation [invited]," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 7, no. 1, pp. A162–A173, 2015.
[17] "Common Vulnerabilities and Exposures (CVE)," Forum of Incident Response and Security Teams. [Online]. Available: https://cve.mitre.org/
[18] M. Franz, "E unibus pluram: Massive-scale software diversity as a defense mechanism," in *Proceedings of the 2010 New Security Paradigms Workshop*, ser. NSPW '10. New York, NY, USA: ACM, 2010, pp. 7–16.
[19] G. Grimmett, "Percolation and disordered systems," in *Lectures on Probability and Statistics*. Springer Berlin Heidelberg, 1997, pp. 153–300.
[20] K. J. Hole, "Diversity reduces the impact of malware," *IEEE Security Privacy*, vol. 13, no. 3, pp. 48–54, May 2015.
[21] ——, "Toward anti-fragility: A malware-halting technique," *IEEE Security Privacy*, vol. 13, no. 4, pp. 40–46, July 2015.
[22] ——, "Diversity reduces the impact of malware," *IEEE Security & Privacy*, vol. 13, no. 3, pp. 48–54, 2013.
[23] A. Homescu, T. Jackson, S. Crane, S. Brunthaler, P. Larsen, and M. Franz, "Large-scale automated software diversity–program evolution redux," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 2, pp. 158–171, March 2017.
[24] J. B. Hong and D. S. Kim, "Assessing the effectiveness of moving target defenses using security models," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 2, pp. 163–177, Mar. 2016.
[25] J. B. Hong, S. Yoon, H. Lim, and D. S. Kim, "Optimal network reconfiguration for software defined networks using shuffle-based online mtd," in *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*, 2017, pp. 234–243.
[26] S. Hosseini and M. A. Azgomi, "The dynamics of an SEIRS-QV malware propagation model in heterogeneous networks," *Physica A: Statistical Mechanics and its Applications*, vol. 512, pp. 803–817, 2018.
[27] C. Huang, S. Zhu, and R. Erbacher, *Toward Software Diversity in Heterogeneous Networked Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, July 2014, pp. 114–129.
[28] C. Huang, S. Zhu, Q. Guan, and Y. He, "A software assignment algorithm for minimizing worm damage in networked systems," *Journal of Information Security and Applications*, vol. 35, no. Supplement C, pp. 55–67, 2017.
[29] T. Kaur and J. Baek, "A strategic deployment and cluster-header selection for wireless sensor networks," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 4, pp. 1890–1897, 2009.
[30] M. Keramati, A. Akbari, and M. Keramati, "CVSS-based security metrics for quantitative analysis of attack graphs," in *ICCKE 2013*, 2013, pp. 178–183.
[31] J. Knight, J. Davidson, A. Nguyen-Tuong, J. Hiser, and M. Co, "Diversity in cybersecurity," *Computer*, vol. 49, no. 4, pp. 94–98, Apr. 2016.
[32] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.
[33] P. Larsen, S. Brunthaler, and M. Franz, "Security through diversity: Are we there yet?" *IEEE Security & Privacy*, vol. 12, no. 2, pp. 28–35, Mar. 2014.
[34] ——, "Automatic software diversity," *IEEE Security & Privacy*, vol. 13, no. 2, pp. 30–37, Mar. 2015.
[35] A. S. Leong, D. E. Quevedo, A. Ahlén, and K. H. Johansson, "On network topology reconfiguration for remote state estimation," *IEEE Transactions on Automatic Control*, vol. 61, no. 12, pp. 3842–3856, 2016.
[36] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, June 2014.
[37] D. Li, Q. Zhang, E. Zio, S. Havlin, and R. Kang, "Network reliability analysis based on percolation theory," *Reliability Engineering & System Safety*, vol. 142, pp. 556–562, 2015.
[38] H. Li, K. Ota, and M. Dong, "Learning IoT in edge: Deep learning for the internet of things with edge computing," *IEEE Network*, vol. 32, no. 1, pp. 96–101, Jan 2018.
[39] T. Li, C. Feng, and C. Hankin, "Improving ICS cyber resilience through optimal diversification of network resources," *CoRR*, vol. abs/1811.00142, 2018. [Online]. Available: http://arxiv.org/abs/1811.00142
[40] A. X. Liu and M. G. Gouda, "Diverse firewall design," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 9, pp. 1237–1251, 2008.
[41] M. R. Lyu and L. K. Lau, "Firewall security: Policies, testing and performance evaluation," in *Proceedings 24th Annual International Computer Software and Applications Conference. COMPSAC2000*. IEEE, 2000, pp. 116–121.
[42] P. K. Manadhata and J. M. Wing, *A Formal Model for a System's Attack Surface*. New York, NY: Springer New York, 2011, pp. 1–28.
[43] S. Mavoungou, G. Kaddoum, M. Taha, and G. Matar, "Survey on threats and attacks on mobile networks," *IEEE Access*, vol. 4, pp. 4543–4572, 2016.
[44] C. Moore and M. E. Newman, "Epidemics and percolation in small-world networks," *Physical Review E*, vol. 61, no. 5, p. 5678, 2000.
[45] W. Najjar and J. L. Gaudiot, "Network resilience: a measure of network fault tolerance," *IEEE Transactions on Computers*, vol. 39, no. 2, pp. 174–181, Feb. 1990.
[46] M. E. J. Newman, *Networks: An Introduction*, 1st ed. Oxford University Press, 2010.
[47] A. J. O'Donnell and H. Sethu, "On achieving software diversity for improved networksecurity using distributed

coloring algorithms," in *Proceedings of the 11th ACM Conference on Computer and Communications Security*, ser. CCS '04, 2004, pp. 121–131.

[48] Y. Prieto, N. Boettcher, S. E. Restrepo, and J. E. Pezoa, "Optimal multiculture network design for maximizing resilience in the face of multiple correlated failures," *Applied Sciences*, vol. 9, no. 11, p. 2256, 2019.

[49] C. Pu, A. P. Black, C. Cowan, J. Walpole, and C. Consel, "A specialization toolkit to increase the diversity of operating systems," 1996.

[50] R. S. Rao, K. Ravindra, K. Satish, and S. Narasimham, "Power loss minimization in distribution system using network reconfiguration in the presence of distributed generation," *IEEE Transactions on Power Systems*, vol. 28, no. 1, pp. 317–325, 2012.

[51] J. P. Sterbenz, D. Hutchison, E. K. Cetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith, "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines," *Computer Networks*, vol. 54, no. 8, pp. 1245–1265, 2010.

[52] E. Totel, F. Majorczyk, and L. Mé, "Cots diversity based intrusion detection and application to web servers," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2005, pp. 43–62.

[53] B. Walker, A. Kinzig, and J. Langridge, "Original articles: plant attribute diversity, resilience, and ecosystem function: the nature and significance of dominant and minor species," *Ecosystems*, vol. 2, no. 2, pp. 95–113, March 1999.

[54] A. D. Wood and J. A. Stankovic, "Denial of service in sensor networks," *Computer*, vol. 35, no. 10, pp. 54–62, Oct. 2002.

[55] Y. Yang, S. Zhu, and G. Cao, "Improving sensor network immunity under worm attacks: A software diversity approach," in *Proceedings of the 9th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, ser. MobiHoc '08, 2008, pp. 149–158.

[56] ——, "Improving sensor network immunity under worm attacks: A software diversity approach," *Ad Hoc Networks*, vol. 47, no. Supplement C, pp. 26–40, 2016.

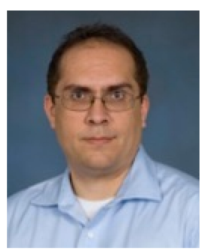
[57] M. Zhang, L. Wang, S. Jajodia, A. Singhal, and M. Albanese, "Network diversity: A security metric forevaluating the resilience of networks againstzero-day attacks," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 5, pp. 1071–1086, May 2016.

[58] Y. Zhang, H. Vin, L. Alvisi, W. Lee, and S. K. Dao, "Heterogeneous networking: A new survivability paradigm," in *Proceedings of Network Security Paradigms Workshop (NSPW'01)*. Cloudcroft, New Mexico, USA: ACM, Sep. 2001.

[59] Y. Zhang, M. Murata, H. Takagi, and Yusheng Ji, "Traffic-based reconfiguration for logical topologies in large-scale wdm optical networks," *Journal of Lightwave Technology*, vol. 23, no. 10, pp. 2854–2867, 2005.

Qisheng Zhang is currently a Ph.D. student in the Department Computer Science at Virginia Tech. He received the B.S. degree in mathematics from Shandong University in 2017 and the M.S. degree in mathematics from the University of Warwick in 2018. His research interests include network security and network science.

Jin-Hee Cho is currently an associate professor in the department of computer science at Virginia Tech since 2018. Prior to joining the Virginia Tech, she worked as a computer scientist at the U.S. Army Research Laboratory (USARL), Adelphi, Maryland, since 2009. Dr. Cho has published over 140 peer-reviewed technical papers in leading journals and conferences in the areas of trust management, cybersecurity, metrics and measurements, network performance analysis, resource allocation, agent-based modeling, uncertainty reasoning and analysis, information fusion / credibility, and social network analysis. She received the best paper awards in IEEE TrustCom'2009, BRIMS'2013, IEEE GLOBECOM'2017, 2017 ARL's publication award, and IEEE CogSima 2018. She is a winner of the 2015 IEEE Communications Society William R. Bennett Prize in the Field of Communications Networking. In 2016, Dr. Cho was selected for the 2013 Presidential Early Career Award for Scientists and Engineers (PECASE). Dr. Cho received MS and PhD degrees in computer science from the Virginia Tech in 2004 and 2008, respectively. She is a senior member of the IEEE and a member of the ACM.



Terrence J. Moore received the B.S. and M.A. degrees in mathematics from American University in 1998 and 2000, respectively, and the Ph.D. degree in mathematics from the University of Maryland, College Park in 2010. He is currently a researcher in the Network Science Division of the U.S. Army Research Laboratory. His research interests include sampling theory, constrained statistical inference, stochastic optimization, network security, geometric and topological applications in networks, and network science.

Ing-Ray Chen (M'90) received the B.S. degree from National Taiwan University, and the M.S. and Ph.D. degrees in computer science from the University of Houston. He is currently a Professor with the Department of Computer Science, Virginia Tech. His research interests include trust and security, network and service management, and reliability and performance analysis of mobile wireless networks and cyber physical systems. He was a recipient of the IEEE Communications Society William R. Bennett Prize in Communications Networking and of the U.S. Army Research Laboratory (ARL) Publication Award. Dr. Chen currently serves as an associate editor for IEEE TRANSACTIONS ON SERVICES COMPUTING, IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, and The Computer Journal.