Constraint-based Network Topology Generation for Evaluating Federated Intrusion Detection Systems

Léo Lavaur[1,2,3][0000−0002−0379−7946], Fabien Autrel[2,3][0000−0002−8403−515X], and Yann Busnel[3,4][0000−0001−6908−719X]

[1] Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg, Luxembourg
[2] IMT Atlantique, Rennes, France
[3] IRISA (SOTERN Team), Rennes, France
[4] Institut Mines-Télécom, Palaiseau, France

Abstract. Federated Intrusion Detection Systems (FIDSs) emerged as a promising approach to collaborative cybersecurity, enabling organizations to train intrusion detection models without sharing sensitive data. However, evaluating such systems faces significant challenges due to the lack of available datasets that capture the heterogeneity of real-world distributed networks. Existing datasets are typically generated using single network topologies, forcing researchers to rely on artificial partitioning strategies that cannot replicate heterogeneous data distributions that exist in practice. To address this limitation, we propose a novel approach for generating heterogeneous network topologies specifically designed to evaluate distributed and federated intrusion detection systems. Because creating realistic topologies from scratch is particularly complex, we construct complex topologies from a library of predefined sub-topologies using constraint programming, and compose them into larger, realistic network structures. We implement a prototype and evaluate its performance across multiple parameters including library size, maximum number of nodes, tree depth, and service constraints. The results highlight that, while the derivation time scales rapidly with most parameters due to the combinatorial nature of the problem, the tool successfully generates large numbers of diverse topologies while maintaining control over their heterogeneity characteristics.

Keywords: Federated Learning · Intrusion Detection Systems · Network Topology Generation · Constraint Programming · Cybersecurity · Dataset Generation

1 Introduction

The advent of Federated Learning (FL) and analogous approaches has rekindled interest in collaboratively developing intrusion detection models [1]; whether it is to learn data representation from distributed endpoints, or to allow knowledge sharing between organizations operating different IT networks. However,

evaluating such systems has proven difficult, notably due to distribution biases when partitioning intrusion detection datasets. In reality, public datasets available in the Intrusion Detection System (IDS) literature are typically developed using a single network topology with a single collection point, thereby offering researchers focused on distributed methods two possible choices: (a) use an existing dataset and rely on partitioning strategies to simulate the heterogeneity of real-world data distributions; or (b) homogenize the features of multiple existing datasets to act as independent data silos.

Unfortunately, both approaches have limitations. In the first case, partitioning strategies cannot fully replicate the heterogeneity of real-world data distributions, as the data will remain correlated to some extent. Moreover, it requires a deep understanding of the way each dataset has been generated to approach realistic data shards. While it might suffice for distributed endpoints in one network, this makes such approaches inapplicable to simulate the data distribution of federated IT networks. In the second case, the number of clients is limited by the number of public datasets available, narrowing down experiments to extremely small-scale federations. Furthermore, because all datasets are independent, characterizing the heterogeneity of the data distributions is difficult, leaving little control over the experimental conditions. In the literature, both strategies have been used to simulate NIID (not Independent and Identically Distributed) settings [2, 3], but the results remained limited in realism by the lack of control over the data distributions.

To bridge the gap towards more realistic evaluations of cross-silo Federated Intrusion Detection Systems (FIDSs), we propose a novel approach to generate heterogeneous network topologies that can be deployed in virtualized environments. Independently generated datasets, which are comparable with the state of the art while allowing finer control over data heterogeneity, are essential for robust evaluation. Such datasets would provide the experimental conditions necessary for addressing several open challenges in the literature, such as effective knowledge transfer between clients to recognize patterns absent from local training data, adaptability of local models to evolve and accommodate new devices (data- and concept-drift), and improved generalization capability to similar but distinct services. Because creating a functional topology from scratch is particularly complex [4], we compose topologies from a set of predefined building blocks that satisfy user-defined constraints. Thus, we are able to dynamically create a wide variety of topologies that can be used to assess the performance of FIDSs in a heterogeneous but controlled and reproducible environment. Our contributions can be summarized as follows.

1. A novel approach to building realistic network topologies for dataset generation through constraint-based topology composition.
2. An implementation of a prototype with an extended performance evaluation of the topology generation process.
3. Foundations for the first truly distributed synthetic dataset in intrusion detection, enabling the evaluation of FIDSs under controlled conditions.

The remainder of this article is organized as follows. We start by reviewing FIDSs and the difficulties in generating appropriate datasets in Section 2, followed by a review of existing related works in Section 3. We then present our approach to generate topologies in Section 4, and evaluate the performance of our tool in Section 5. Finally, we discuss the perspectives of our work before concluding in Section 6.

2 Preliminaries

2.1 Federated Learning

FL is a distributed machine learning paradigm that trains a single model across multiple data sources, without sharing the raw data [5]. The algorithm is structured in rounds, where a server will select a subset of n clients p_i to train a model w_i^r on their local data. At the end of each round r, the server collects and aggregates local models to produce a global model W^r, which will be distributed to a new subset of clients in the next round. Depending on the use case, the fraction C of selected clients can vary, notably based on the number and availability of clients: cross-device *vs.* cross-silo federations [6]. Most importantly, each client p_i has a local dataset d_i that is not shared with the server or with other clients.

In general, clients minimize a loss function $\mathcal{L}(w_i^r, d_i)$. They do so by computing the gradient $g_i^r = \nabla\mathcal{L}(w_i^r, d_i)$, which can then be used to update the local model as $w_i^r = w_i^{r-1} - \eta g_i^r$, where η is the learning rate. The server then aggregates the uploaded parameters as a function $\mathcal{F}(\cdot)$ of the uploaded updates $\{w_i \mid i \in [\![1, n]\!]\}$, typically using a weighted average. Alternatively, gradients can be collected and aggregated on the server, which will update the global model directly as $W^r = W^{r-1} - \eta\mathcal{F}((g_i^r)_{i\in[\![1,n]\!]})$. In essence, FL is therefore a two-level Stochastic Gradient Descent (SGD), where the sampled clients compute local updates using sampled data. Because of it, FL is particularly sensitive to the data distribution among clients: Heterogeneous settings will locally produce biased estimates of the global distribution, impeding the convergence of the federation [7]. Various degrees of similarity exist between clients, from purely Independent and Identically Distributed (IID) partitioning to *pathological* non-IID (NIID) settings, where all clients have unique data distributions without class overlap [7].

2.2 Federated Intrusion Detection Systems

FIDSs [1] are a specific application of FL to the field of intrusion detection, where the goal is to collaboratively train a model that can detect malicious activities in a network. Intrusion detection datasets are typically made up of network traffic data, often represented as sequences of packets or flows, and possibly enriched with a label y_j indicating the nature of the traffic (*e.g.*, benign or malicious, attack type). In this case (*i.e.*, supervised learning), the problem can be reduced

to a classification task, the loss function being expressed as $\mathcal{L}(w_i^r, \boldsymbol{x}_j, y_j)_{j \in [\![1, |d_i|]\!]}$, with $\boldsymbol{x}_j$ and y_j being the j-th feature vector and its label, respectively.[5]

In this context, the distribution of data among clients can vary significantly depending on the types of devices, the behavior of users, the nature of the network traffic or the network topology. However, the original *pathological*-NIID settings used by McMahan *et al.* [5] are not really encountered in practice: a typical organization is likely to have multiple devices and services that generate different types of traffic on the same network.

Existing Datasets Representative and high-quality datasets are essential in intrusion detection, as they provide the ground truth for training and evaluating models. This is particularly true in the context of Machine Learning (ML) and Deep Learning (DL), as the performance of these models is highly dependent on the quality and representativeness of the training data. Until mid 2010s, most evaluations were performed on the KDD Cup 1999 dataset [8], which is now considered outdated and not representative of modern networks. Even NSL-KDD [9], one of the most widely used datasets of all time, uses the same data from the 1998 DARPA dataset. In the last decade, several new datasets have been proposed, such as UNSW-NB15 [10] or CICIDS2017 [11]. However, all these datasets are generated using a single network topology, which limits their applicability to FIDSs. Moreover, they are now known to suffer from significant statistical issues, some even having labeling and implementation errors [12, 13]. More recently, IDS datasets mentioning FL have been proposed, such as Edge-IIoTset [14], but they still suffer from the same limitations as the previous datasets, as they are generated using a single network topology. Table 1 summarizes the most widely used datasets in the IDS literature for reference.

Table 1: Most common feature-based datasets for Network-based Intrusion Detection Systems (NIDSs).

Dataset	Year	Use case	Feature extraction	Features	Records (train/test)[a]	Attack classes	Reference
KDD Cup 99	1999	Military IT Network	Bro-IDS	41	4,898,431/311,029	4	[8]
NSL-KDD	2009	Military IT Network	See KDD'99	41	125,973/22,544	4	[9]
UNSW-NB15	2015	Company IT Network	Bro, Argus, & Custom	49	2,540,044	10	[10]
CIDDS-001	2017	Small Business	NetFlow v9	10	31,959,175	5	[15]
CIDDS-002	2017	Small Business	NetFlow v9	10	16,161,183	5	[16]
CICIDS2017	2017	Company IT Network	CICFlowMeter	80	2,830,743	9	[11]
CICIDS2018	2018	Large-scale IT Network	CICFlowMeter	80	8,284,254	7	[11]
Bot-IoT	2019	Botnets and IoT	Argus & Custom	14	72,000,000+	4	[17]
ToN_IoT	2021	Cross-layer Infrastructure	Zeek & Custom	44	461,043	9	[18]
Edge-IIoTset	2022	Cross-layer Infrastructure	Zeek & TShark	61	181,156/30,440	15	[14]

[a] Some datasets do not have a recommended train/test split. In such cases, only the total number of records is provided.

[5] Other types of learning are also possible, but we focus on supervised learning here for sake of simplicity.

Heterogeneity Because there are not enough public datasets that are representative of the heterogeneity of distributed networks in the real world (see Section 1), the literature often relies on partitioning strategies to simulate the heterogeneity of data distributions. The way in which the data are partitioned is dictated by the use case considered. In Cross-Device Federated Learning (CD-FL), where the goal is to train a model across a large-scale fleet of devices (such as end-user devices or dedicated IDS appliances), the data distribution is likely to be heterogeneous, but significant overlap may exist if clients physically share the same network. This setting is frequently simulated by partitioning the data by IP address, although most available datasets were not produced with this consideration.

In a Cross-Silo Federated Learning (CS-FL) setting, where multiple organizations collaborate to train a model, such an overlap is unlikely to exist. The data distribution is expected to be highly heterogeneous, as each organization will have its own network topology, devices, and users. Therefore, most of the existing work in that category uses one of the following three approaches [19, 20]:

(a) *class agnostic*: the dataset is partitioned using a given distribution (often a Dirichlet one) without further consideration;
(b) *class drop*: the dataset is randomly partitioned into m shards before randomly dropping classes from each partition; or
(c) *class split*: the benign traffic is split into m shards, with each shard being assigned instances from a subset of attack classes, with or without overlap.

All three approaches have limitations, as they inherently result in correlated data distributions. Consequently, we argue that existing datasets are insufficient to properly evaluate cross-silo FIDSs and propose to address these challenges by generating independent network topologies, which will serve as a fundation for a new generation of distributed intrusion detection datasets.

3 Related Work

The motivation behind topology generation originates from the need to evaluate network protocols in simulations. In fact, while network topology should not influence the behavior of a protocol, it can significantly impact its performance [21]. At that time, several tools had been developed to generate network topologies, such as `GT-ITM` [22], `Tiers` [23], or `BRITE` [24]. Tangmunarunkit *et al.* [21] distinguish two main categories of topology generators: *structural*, which aim at reproducing the structural properties of the internet and particularly its hierarchical organization, and *degree-based*, which focus on the statistical properties of the network, particularly the power law distribution of the node degrees [25]. Most of these works are more than 20 years old and have been developed to generate topologies for Internet-scale networks, which are not directly applicable to the generation of FIDSs datasets, particularly in our siloed IT networks.

Recent works on topology generation are rarer and focus on specific use cases. For instance, Laurito *et al.* [26] developed `TopoGen`, a tool to generate network topologies using Software-Defined Networkings (SDNs). Their approach allows users to programmatically define the network topology using the Ruby programming language, and extract existing topologies from real-world networks. However, their approach is limited to SDNs and does not allow automating data generation. Alrumaih and Alenazi [27] developed `GENIND`, a tool to generate industrial networks. Similarly to us, the authors identified that most existing tools are too focused on Internet-inspired and Internet-scale topologies and do not allow generating topologies for specific use cases. Their tool focuses on generating topologies for industrial networks, and therefore generates topologies layer by layer, with specific constraints, before connecting the different sub-topologies together in a multigraph. To the best of our knowledge, no existing tool allows generating network topologies for IT networks, that can be randomized to obtain variations with common characteristics.

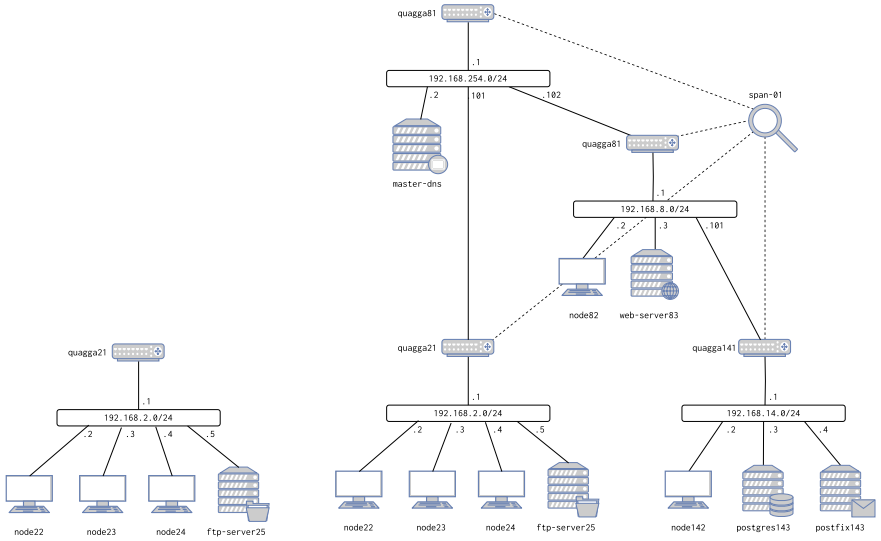
Another aspect of this work is the building of topologies that satisfy the requirements of specified attack scenarios and their deployment in a controlled environment. Recent works have focused on *cyber ranges*[6] for the latter. Both Venkatesan *et al.* [28] and Yamin and Katt [29] translate a scenario and a topology description into a deployable environment, but do not focus on the generation of the topologies themselves. More recently, Besson *et al.* [30] developed `URSID`, a tool to generate vulnerable network architectures that implement user-provided attack scenarios. Compared to previous work, `URSID` introduces variability in the scenarios and service deployments, but does not provide heterogeneous topology constructions. Similarly to us, Kaveh, Rohner, and Johnsson [31] study the impact of network topology on the generalization capabilities of introduction detection models Although their tests are limited to three variations and focused on depth, the results indicate significant difficulties in knowledge transfer, encouraging further studies on the topic. To our knowledge, however, no existing work provides a complete solution to generate heterogeneous FIDS datasets.

4 Topology Generator for Federated IT Networks

To solve the limitations of existing datasets in the literature, we introduce `FedITN_Gen`, a topology generator for federated IT networks. This contribution will serve as a foundational building block for creating a reliable, robust, and realistic dataset generator. In this section, we present the design and architecture of `FedITN_Gen`, before detailing its implementation using Airbus' CyberRange platform[7]. The code is available on request and will be released in open source in the future.

[6] A cyber range is a controlled environment simulating a real-world network, where users can deploy their own topologies and scenarios, offering a wide range of applications, from cybersecurity training to dataset generation [32].

[7] `https://www.cyber.airbus.com/products/cyberrange/`



(a) Example of a sub-topology containing 3 Windows clients and 1 FTP server.

(b) Example of a complete topology composed of 3 sub-topologies. The `span` captures the traffic of all gateways to simulate various probe positions afterward.

Fig. 1: Examples of topologies as instantiated on Airbus' CyberRange. Each node represents a machine, and the edges represent the connections between them. Nodes labeled `quagga*` are gateways connecting each sub-topology to the rest of the network.

4.1 Approach Overview

The core idea behind `FedITN_Gen` is to compose network topologies by selecting and connecting a set of predefined sub-topologies that satisfy user-supplied constraints. A sub-topology is a small network composed of a subnet, a set of nodes (clients or servers), and a gateway that connects the subnet to the rest of the network. We build a library of sub-topologies that represent common network configurations. While creating this library remains human-operated, the composition of the topologies is fully automated, allowing the generation of different topologies with common characteristics.

`FedITN_Gen` is thus a two-step algorithm:

1. *Topology selection:* Use constraint programming to find all sets of sub-topologies that satisfy the user-defined constraints, based on a library of predefined sub-topologies.
2. *Topology composition:* For each set, connect the sub-topologies in a random tree-like structure starting from the *Master* sub-topology.

Topology selection The topology selection is the most important part of the algorithm, as it requires finding all sets of sub-topologies that satisfy the user-defined constraints. A sub-topology τ_k is composed of a subnet, a set of nodes N_k (clients or servers), and a gateway that connects the subnet to the rest of the network. Figure 1a shows an example of a sub-topology with 4 nodes. The gateway h_k is a particular node (not belonging to N_k) that connects the subnet to the rest of the network. Therefore, a sub-topology can be represented as a tuple $\tau_k = (N_k, h_k)$. Selecting sub-topologies from the library is a Constraint Satisfaction Problem (CSP), which aims at finding all sets of sub-topologies that satisfy the user-defined constraints. Definition 1 defines the notion of CSP.

Definition 1 (Constraint Satisfaction Problem [33]). *A Constraint Satisfaction Problem (CSP) is a tuple $P = (X, D, C)$ where:*

- $X = (X_1, X_2, \ldots, X_n)$ *is a collection of variables.*
- $D = (D_1, D_2, \ldots, D_n)$ *is a collection of nonempty domains, one for each variable.*
- $C = (C_1, C_2, \ldots, C_n)$ *is a collection of constraints that specify allowable combinations of values in their domains.*

Each constraint $C_i \in C$ is a tuple $C_i = (\chi_i, R_i)$ where R_i is a relation between the variables in $\chi_i \subseteq X$. Thus, solving a CSP is finding an assignment of values from X in D that satisfies all constraints in C.

While dimensioning the output topologies can easily be translated using numeric boundaries, the constraints on the availability of services and attack scenarios in the sub-topologies are slightly more challenging. We first define the service domain D_{service} as the set of all services available in the library of sub-topologies, and tag each service node with its corresponding name, such as `ftp`, `postfix` (for email), or `ldap` for instance. An attack scenario is defined as another tuple ($\texttt{srcs} = \{N_1^{\texttt{s}}, N_2^{\texttt{s}}, \ldots\}, \texttt{targets} = \{N_1^{\texttt{t}}, N_2^{\texttt{t}}, \ldots\}$) where `srcs` and `targets` are sets of nodes available in the library that are compatible with the attack scenario. For instance, a Man-in-the-Middle (MitM) attack could be represented as $a_{\text{MitM}} = (\texttt{srcs} = \{N_{\text{attaker}}\}, \texttt{targets} = \{N_1, N_2\})$. Based on the aforementioned definitions, we can define our Sub-topology Selection Problem (STSP) in Problem 1.

Problem 1 (Sub-topology Selection Problem). Let $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a set of sub-topologies, D_{service} a set of services, $A = \{a_1, a_2, \ldots, a_m\}$ a set of attack scenarios, $k_{\min}$ and $k_{\max}$ the bounds of the number of subtopologies, and $\ell_{\min}$ and $\ell_{\max}$ the bounds for the total number of nodes. The Sub-topology Selection Problem (STSP) consists of finding all sets of sub-topologies $T \subseteq \mathcal{T}$ that satisfy the following:

1. $k_{\min} \leq |T| \leq k_{\max}$.
2. $\ell_{\min} \leq \sum_{\tau_k \in T} |N_k| \leq \ell_{\max}$.
3. $\forall s \in D_{\text{service}}, \exists \tau_k \in T,$ s.t. $s \in N_k$.
4. $\forall a \in A, \forall s \in a_{[\texttt{srcs}]}, \exists \tau_k \in T,$ s.t. s compatible with τ_k.
5. $\forall a \in A, \forall t \in a_{[\texttt{targets}]}, \exists \tau_k \in T,$ s.t. t compatible with τ_k.

Topology composition Once the sub-topologies are selected, the next step is to connect them to form a complete IT network. The composition of the topologies is done in a tree-like structure, starting from the *Master* sub-topology. The *Master* sub-topology is a special sub-topology that acts as the root of the tree and contains the necessary services to route traffic between the sub-topologies. Figure 1b shows an example of a complete topology composed of 3 sub-topologies. At this point of the algorithm, the sub-topologies are already selected, and the composition is a simple matter of connecting the gateways while respecting the last constraint of tree depth. However, many structural arrangements of the same set of nodes can be created, as illustrated by the trees in Figure 2. The composition process is further detailed in Algorithm 1.

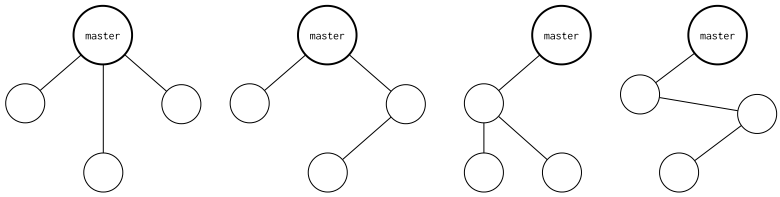


Fig. 2: Possible unlabeled tree structures for a topology composed of 3 sub-topologies. The root node represents the *Master* sub-topology.

4.2 Implementation

In this section, we present parts of the implementation of `FedITN_Gen` using the Airbus CyberRange platform. Most importantly, such platforms come with a set of templates that can be used to create sub-topologies, compatible attack scenarios, and user traffic generation tools to generate realistic traffic. Although `FedITN_Gen` currenlty only supports the latter, its extensible architecture would make it easy to implement any other interface. We implement our topology generator as a Python script that interacts with the CyberRange platform through its API to query the available sub-topologies, services, and attack scenarios. The topology selection algorithm is implemented using Google's `CP-SAT` solver[8], and we develop a simple recursive algorithm to compose the topologies. The constraints on the availability of services and attack scenarios are implemented as a simple filtering algorithm that removes sub-topologies that do not contain the required services or are not compatible with the attack scenarios. We seed all random operations to ensure the reproducibility of the results.

The Master *sub-topology.* The latter serves as the basis for the composition. It consists of a gateway connecting to the Internet, a DHCP server to distribute IP

[8] `https://developers.google.com/optimization/cp/cp_solver`

Algorithm 1 Recursive Tree Generation for Sub-topology Composition

Require: Set of sub-topologies $S \subseteq \mathcal{T}$, maximum depth $\delta_{\max} \in \mathbb{N}$, current depth $\delta \in \mathbb{N}$, tree $\mathbb{T} = (V, E)$, current node $v \in V$
Ensure: Collection $\mathcal{G}$ updated with all valid tree compositions

▷ ***Base Cases:*** ◁
if $S = \emptyset$ **then**
  $\mathcal{G} \leftarrow \mathcal{G} \cup \{\text{copy}(\mathbb{T})\}$ **if** $\mathbb{T} \notin \mathcal{G}$
  return true
if $\delta + 1 > \delta_{\max}$ **then**
  return false ▷ *Depth limit exceeded*
if $|S| = 1$ **then**
  $s \leftarrow$ sole element of S
  Create leaf: $\mathbb{T} \leftarrow \mathbb{T} \cup \{v \to s\}$ ▷ *Add edge $v \to s$*
  $\mathcal{G} \leftarrow \mathcal{G} \cup \{\text{copy}(\mathbb{T})\}$ **if** $\mathbb{T} \notin \mathcal{G}$
  Remove leaf: $\mathbb{T} \leftarrow \mathbb{T} \setminus \{v \to s\}$ ▷ *Backtrack*
  return true

▷ ***Recursive Case:*** ◁
for $k = 1$ **to** $|S|$ **do** ▷ *Try all possible numbers of children*
  for each $C \in \binom{S}{k}$ **do** ▷ *Try all k-subsets as children*
    $S' \leftarrow S \setminus C$ ▷ *Remaining sub-topologies*
    ▷ ***Expand tree*** ◁
    for each $s \in C$ **do**
      $\mathbb{T} \leftarrow \mathbb{T} \cup \{v \to s\}$ ▷ *Add child s to v*
    ▷ ***Recurse on each new child*** ◁
    for each $s \in C$ **do**
      RecursiveTreeGeneration$(S', \delta_{\max}, \delta + 1, \mathbb{T}, s, \mathcal{G})$
    ▷ ***Backtrack*** ◁
    for each $s \in C$ **do**
      $\mathbb{T} \leftarrow \mathbb{T} \setminus \{v \to s\}$ ▷ *Remove child s from v*
return false

addresses, and a DNS server to resolve domain names in all topologies. This is the only topology that receives manual configuration. We set up the DNS server to associate domain names to the IPs of the services hosted in the sub-topologies: *e.g.*, web server (`webserver.local`), mail server (`mailserver.local`), file sharing (`fileserver.local`). This approach makes it possible to define unique domain names for each service, and make them accessible from any sub-topology.

Handling connectivity. Now that all services are available using their domain names, the next step is to ensure that the services are reachable from any sub-topology. To do so, each gateway g_k hosts a DHCP server with a dedicated range to allocate IP addresses for its child sub-topologies. The DNS configuration of the *Master* topology is propagated to the DHCP server of each gateway, so that the domain names are resolved correctly. To route traffic between the sub-topologies, the gateways also run an OSPF daemon to exchange routing information, announcing their own subnet to the rest of the network. This setup allows to dynamically configure the routing tables of the sub-topologies upon deployment and ensures that all machines are reachable.

Constraint satisfaction. As noted in Section 4.1, the topology selection is a CSP that can be solved using a constraint solver. In particular, we implement our cardinality-related constraints (*i.e.*, the number of sub-topologies and nodes) using Google's `CP-SAT` solver. This generates all possible combinations of sub-topologies that satisfy the numeric constraints. We then prune the sets that do not contain the required services or are not compatible with the attack scenarios. For each set, we compose the complete topology using a simple recursive algorithm that connects the gateways of the sub-topologies in a tree-like structure, starting from the *Master* sub-topology. The algorithm includes a backtracking mechanism to roll back the composition when the tree-depth constraint is not satisfied, and tries again from another branch.

5 Performance Benchmark

In this section, we present some preliminary results on the performance of `FedITN_Gen`, as well as the influence of some constraints on the latter. Note that these results are obtained from a prototype implementation and are more indicative of what the approach could achieve rather than its current performance. To evaluate our prototype, we build a synthetic library L of 253 unique sub-topologies with various characteristics, and perform a series of experiments to evaluate the performance of `FedITN_Gen` in various conditions.

We review the influence of library size (*i.e.*, $|T|, T \subseteq L$), maximum number of nodes, tree depth, and number of service constraints on the performance of `FedITN_Gen`. To this end, we measure the total generation time (*i.e.*, selection and composition), the number of generated sub-topology sets, and the number of generated topology instances (*i.e.*, tree compositions of the sub-topology sets). The fixed parameters used in each of the experiments are available as appendices (Tables 2a to 2d).

5.1 Influence of the library size

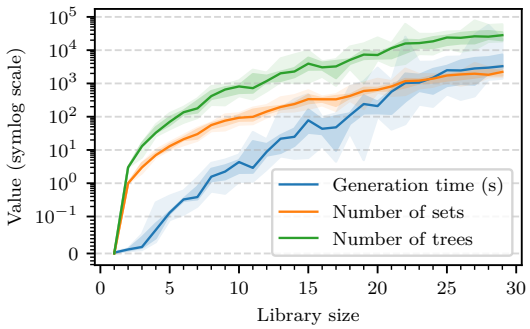


Fig. 3: Influence of the library size on the performance of `FedITN_Gen`.

The size of the library of sub-topologies has a direct impact on the number of combinations to explore. To evaluate the influence of library size on the performance of `FedITN_Gen`, we vary the library size l from 1 to 29, with the other parameters fixed (*cf.*, Table 2a). For each run, we build a smaller library T such that $T \subseteq L$, and $|T| = s$. We randomly select s sub-topologies from the full library to constitute the library set. To smooth the results and avoid artifacts, we conduct ten experiments for each value of library size, thereby increasing the robustness of the results.

Figure 3 displays the different metrics using a symmetric logarithmic scale[9]. We notably observe that the execution time increases dramatically with the library size. This is expected due to the nature of the constraint solver and the tree composition algorithm. The number of sub-topology sets and tree compositions also increase with the library size, but with a lower slope. Note that even with a tree depth of 2, the number of tree compositions is superior to the number of sub-topology sets by a factor of 10. In fact, there is a combinatorial explosion in the number of possible compositions.

5.2 Influence of the maximum number of nodes

We then evaluate the influence of the maximum number of nodes on the performance of `FedITN_Gen`. We vary the maximum number of nodes from 1 to 20, with the other parameters fixed (*cf.* Table 2b). For each value of the maximum number of nodes, we likewise perform ten experiments.

[9] As per its definition, a logarithmic scale is strictly positive. To accommodate values at $y = 0$, we use a symmetric logarithmic scale, which applies logarithmic scaling to positive and negative values while using linear scaling in the interval $[-0.1, 0.1]$ to provide a smooth transition through zero.

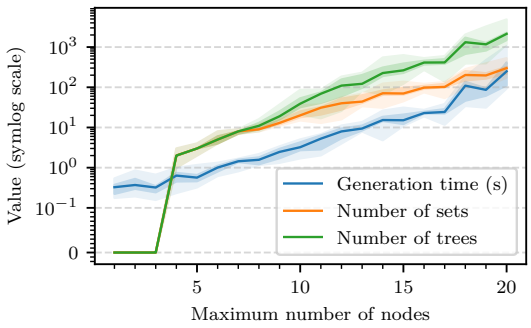


Fig. 4: Influence of the maximum number of nodes on the performance of `FedITN_Gen`.

Figure 4 displays the different metrics. Again, all metrics increase exponentially with the maximum number of nodes. Indeed, this parameter indirectly influences the cardinality of the sub-topology sets generated, and thus the number of tree compositions. Since we kept a tree depth of 2, the number of tree compositions is still significantly higher than the number of sub-topology sets. Note that for a number of nodes lower than 4, there do not exist any solutions, as the topologies in our test library host at least 4 nodes.

5.3 Influence of the tree depth

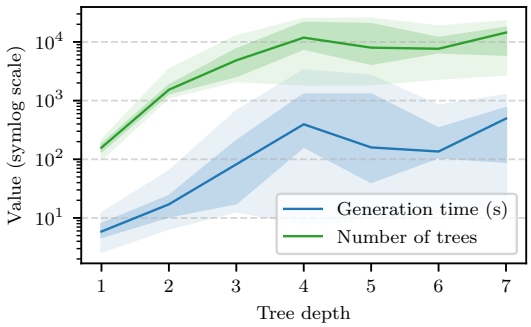


Fig. 5: Influence of the tree depth on the performance of `FedITN_Gen`.

The next experiment evaluates the influence of the tree depth on `FedITN_Gen`'s performance. This parameter has no impact on the number of sub-topology sets, so we only measure the number of tree compositions and the generation time. We vary the tree depth from 1 to 7, with the other parameters fixed (*cf*. Table 2c).

Figure 5 displays the execution time and the number of tree compositions using a log scale. Both increase rapidly with tree depth, as expected, before reaching a plateau with a tree depth of 4. This is due to the fact that the number of tree compositions is limited by the number of sub-topology sets, which in turn depends on the other constraints. Consequently, the high variations in the results are due to the random sampling of the sub-topologies in the library.

5.4 Influence of the number of service constraints

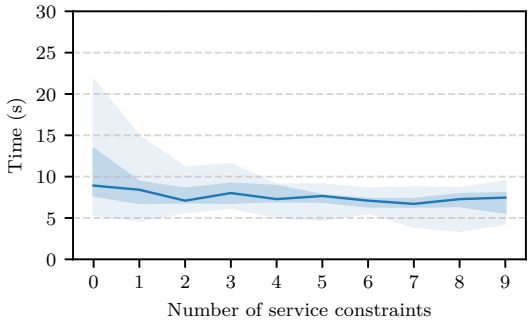
This last experiment evaluates the influence of the number of service constraints on performance. The library of sub-topologies is generated with a fixed number of available services, namely: `ldap`, `dbms`, `cms`, `dns`, `mail`, `syslog_server`, `web`, `ftp`, `proxy`, and `cloud_storage`. We vary the number of service constraints from 1 to 9, with the other parameters fixed (*cf.* Table 2d).

Figures 6a and 6b display the generation time and the number of sets and tree generations, respectively. Unlike the other experiments, the execution time remains nearly constant in this scenario due to the handling of these constraints. While the other constraints are implemented as exploration problems, the service and attack constraints are applied by pruning incompatible sub-topology sets. Therefore, the number of sets and trees is progressively decreasing as the number of service constraints increases. By the 6^{th} constraint, the resolution becomes infeasible, as there are no sub-topologies that satisfy all the constraints. Increasing the maximum number of nodes and sub-topologies would allow for more solutions, but would also increase the execution time (*cf.* previous sections).

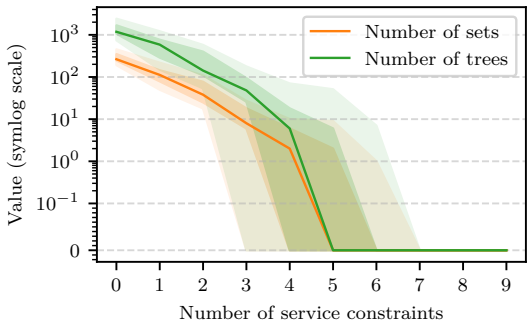
6 Conclusion and Takeaways

In this paper, we presented `FedITN_Gen`, a tool for generating heterogeneous network topologies to construct FIDSs datasets. Because generating such topologies from scratch is impractical, we propose to build a library of predefined sub-topologies that can be combined to form larger topologies according to a set of constraints. We implement a first prototype of `FedITN_Gen` to validate the feasibility of the approach and to evaluate its performance. Due to the combinatorial nature of the problem, we rely on a constraint solver to retrieve valid combinations of sub-topologies. Yet, the derivation time of our tool currently scales exponentially with most of the parameters. Meanwhile, the number of generated topologies is also exponential, making `FedITN_Gen` a powerful tool to generate a large number of topologies while controlling their heterogeneity.

Perspectives The current implementation of `FedITN_Gen` is a first step towards a more complete tool that would support data generation. However, this task requires solving multiple open challenges, such as characterizing the impact of collection points' placement on data distribution, or ensuring the statistical properties of the generated topologies match those of real-world networks. Likewise,



(a) Execution time.



(b) Number of generated sets and tree compositions.

Fig. 6: Influence of the number of service constraints on the performance of `FedITN_Gen`.

data labelling can quickly become complex in distributed environments, especially when various environments for the same attack scenario are considered. Existing frameworks can help with classification, such as MITRE ATT&CK [34], but they remain high-level and labelling might still require extensive manual labor.

Future Works Apart from the aforementioned challenges, `FedITN_Gen` remains currently a preliminary prototype that we plan to improve in several ways. The first and obvious improvement is to pursue the implementation of the tool to support the automated execution of scenarios (both attacks and legitimate traffic generation) to generate datasets, relaying on existing traffic generators of the literature. Another lead for improvement lies in the optimization of the constraint solver, which is currently the bottleneck of the tool. Multiple strategies can be considered, such as using a more efficient modeling of the problem, or using other solving techniques such as column generation [35]. This is critical to introduce finer constraints and allow for more complex topologies, without falling in the curse of dimensionality.

Acknowledgments. This research was partly funded by the chair CyberCNI.fr with the support of the European Union through the European Regional Development Fund (ERDF) of the Brittany Region (EU001043). The CyberRange platform has been provided by Airbus Cyber as part of their framework agreement with IMT Atlantique. Léo Lavaur was originally part of IMT Atlantique and IRISA, but has since joined the SnT at the University of Luxembourg.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Lavaur, L., Pahl, M.-O., Busnel, Y., Autrel, F.: The Evolution of Federated Learning-based Intrusion Detection and Mitigation: A Survey. TNSM (2022)
2. Campos, E.M., Saura, P.F., González-Vidal, A., Hernández-Ramos, J.L., Bernabé, J.B., Baldini, G., Skarmeta, A.: Evaluating Federated Learning for Intrusion Detection in Internet of Things: Review and Challenges. Computer Networks **203**, 108661 (2022). https://doi.org/10.1016/j.comnet.2021.108661. https://www.sciencedirect.com/science/article/pii/S1389128621005405 (visited on 04/25/2024)
3. Popoola, S.I., Gui, G., Adebisi, B., Hammoudeh, M., Gacanin, H.: Federated Deep Learning for Collaborative Intrusion Detection in Heterogeneous Networks. In: 2021 IEEE 94th Vehicular Technology Conference (VTC2021-Fall), pp. 1–6 (2021). https://doi.org/10.1109/VTC2021-Fall52928.2021.9625505
4. Li, Z., Wang, X., Pan, L., Zhu, L., Wang, Z., Feng, J., Deng, C., Huang, L.: Network Topology Optimization via Deep Reinforcement Learning. IEEE Transactions on Communications **71**(5), 2847–2859 (2023). https://doi.org/10.1109/TCOMM.2023.3244239. (Visited on 06/20/2025)

5. McMahan, B., Moore, E., Ramage, D., Hampson, S., Arcas, B.A.y.: Communication-Efficient Learning of Deep Networks from Decentralized Data. In: Singh, A., Zhu, J. (eds.) Proceedings of the 20th International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, pp. 1273–1282. PMLR (2017). https://proceedings.mlr.press/v54/mcmahan17a.html
6. Kairouz, P., McMahan, H.B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A.N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., D'Oliveira, R.G.L., Eichner, H., Rouayheb, S.E., Evans, D., Gardner, J., Garrett, Z., Gascón, A., Ghazi, B., Gibbons, P.B., Gruteser, M., Harchaoui, Z., He, C., He, L., Huo, Z., Hutchinson, B., Hsu, J., Jaggi, M., Javidi, T., Joshi, G., Khodak, M., Konečný, J., Korolova, A., Koushanfar, F., Koyejo, S., Lepoint, T., Liu, Y., Mittal, P., Mohri, M., Nock, R., Özgür, A., Pagh, R., Raykova, M., Qi, H., Ramage, D., Raskar, R., Song, D., Song, W., Stich, S.U., Sun, Z., Suresh, A.T., Tramèr, F., Vepakomma, P., Wang, J., Xiong, L., Xu, Z., Yang, Q., Yu, F.X., Yu, H., Zhao, S.: "Advances and Open Problems in Federated Learning".
7. Huang, Y., Chu, L., Zhou, Z., Wang, L., Liu, J., Pei, J., Zhang, Y.: Personalized Cross-Silo Federated Learning on Non-IID Data. AAAI **35**(9), 7865–7873 (2021). https://doi.org/10.1609/aaai.v35i9.16960. https://ojs.aaai.org/index.php/AAAI/article/view/16960 (visited on 09/26/2022)
8. SigKDD, KDD Cup 1999 Dataset, (1999). https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html (visited on 04/26/2021)
9. Tavallaee, M., Bagheri, E., Lu, W., Ghorbani, A.A.: A Detailed Analysis of the KDD CUP 99 Data Set. In: 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, pp. 1–6. IEEE (2009). https://doi.org/10.1109/CISDA.2009.5356528. http://ieeexplore.ieee.org/document/5356528/
10. Moustafa, N., Slay, J.: UNSW-NB15: A Comprehensive Data Set for Network Intrusion Detection Systems (UNSW-NB15 Network Data Set). In: 2015 Military Communications and Information Systems Conference (MilCIS), pp. 1–6 (2015). https://doi.org/10.1109/MilCIS.2015.7348942. https://ieeexplore.ieee.org/abstract/document/7348942 (visited on 10/09/2023)
11. Sharafaldin, I., Habibi Lashkari, A., Ghorbani, A.A.: Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In: Proceedings of the 4th International Conference on Information Systems Security and Privacy, pp. 108–116. SCITEPRESS - Science and Technology Publications, Funchal, Madeira, Portugal (2018). https://doi.org/10.5220/0006639801080116. http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0006639801080116 (visited on 10/14/2021)
12. Lanvin, M., Gimenez, P.-F., Han, Y., Majorczyk, F., Mé, L., Totel, E.: Errors in the CICIDS2017 Dataset and the Significant Differences in Detection Performances It Makes. In: CRiSIS 2022 - International Conference on Risks and Security of Internet and Systems, pp. 1–16, Sousse, Tunisia (2022). https://hal.archives-ouvertes.fr/hal-03775466 (visited on 09/29/2022)
13. Engelen, G., Rimmer, V., Joosen, W.: Troubleshooting an Intrusion Detection Dataset: The CICIDS2017 Case Study. In: 2021 IEEE Security and Privacy Workshops (SPW), pp. 7–12 (2021). https://doi.org/10.1109/SPW53761.2021.00009. https://ieeexplore.ieee.org/document/9474286 (visited on 06/14/2024)
14. Ferrag, M.A., Friha, O., Hamouda, D., Maglaras, L., Janicke, H.: Edge-IIoTset: A New Comprehensive Realistic Cyber Security Dataset of IoT and IIoT Applications for Centralized and Federated Learning. IEEE Access **10**, 40281–40306 (2022).

https://doi.org/10.1109/ACCESS.2022.3165809. https://ieeexplore.ieee.org/document/9751703 (visited on 04/12/2024)
15. Ring, M., Wunderlich, S., Grüdl, D., Landes, D., Hotho, A.: Creation of Flow-Based Data Sets for Intrusion Detection. Journal of Information Warfare **16**(4), 41–54 (2017). JSTOR: 26504117. https://www.jstor.org/stable/26504117
16. Ring, M., Wunderlich, S., Grüdl, D., Landes, D., Hotho, A.: Flow-Based Benchmark Data Sets for Intrusion Detection. Proceedings of the 16th European Conference on Cyber Warfare and Security (ECCWS) (2017)
17. Koroniotis, N., Moustafa, N., Sitnikova, E., Turnbull, B.: Towards the Development of Realistic Botnet Dataset in the Internet of Things for Network Forensic Analytics: Bot-IoT Dataset. Future Generation Computer Systems **100**, 779–796 (2019). https://doi.org/10.1016/j.future.2019.05.041. https://linkinghub.elsevier.com/retrieve/pii/S0167739X18327687 (visited on 10/23/2021)
18. Moustafa, N.: A New Distributed Architecture for Evaluating AI-based Security Systems at the Edge: Network TON_IoT Datasets. Sustainable Cities and Society **72**, 102994 (2021). https://doi.org/10.1016/j.scs.2021.102994. https://www.sciencedirect.com/science/article/pii/S2210670721002808 (visited on 06/21/2024)
19. Lavaur, L., Busnel, Y., Autrel, F.: Demo: Highlighting the Limits of Federated Learning in Intrusion Detection. In: Proceedings of the 44th International Conference on Distributed Computing Systems (ICDCS), Jersey City, NJ, USA (2024)
20. Hsu, H., Qi, H., Brown, M.: Measuring the Effects of Non-Identical Data Distribution for Federated Visual Classification. In: NeurIPS Workshop on Federated Learning (2019)
21. Tangmunarunkit, H., Govindan, R., Jamin, S., Shenker, S., Willinger, W.: Network Topology Generators: Degree-Based vs. Structural. SIGCOMM Comput. Commun. Rev. **32**(4), 147–159 (2002). https://doi.org/10.1145/964725.633040
22. Calvert, K., Doar, M., Zegura, E.: Modeling Internet Topology. IEEE Communications Magazine **35**(6), 160–163 (1997). https://doi.org/10.1109/35.587723. https://ieeexplore.ieee.org/document/587723 (visited on 07/07/2024)
23. Doar, M.: A Better Model for Generating Test Networks. In: Proceedings of GLOBECOM'96. 1996 IEEE Global Telecommunications Conference, pp. 86–93 (1996). https://doi.org/10.1109/GLOCOM.1996.586131. https://ieeexplore.ieee.org/document/586131 (visited on 07/07/2024)
24. Medina, A., Lakhina, A., Matta, I., Byers, J.: BRITE: An Approach to Universal Topology Generation. In: MASCOTS 2001, Proceedings Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pp. 346–353 (2001). https://doi.org/10.1109/MASCOT.2001.948886
25. Faloutsos, M., Faloutsos, P., Faloutsos, C.: On Power-Law Relationships of the Internet Topology. SIGCOMM Comput. Commun. Rev. **29**(4), 251–262 (1999). https://doi.org/10.1145/316194.316229
26. Laurito, A., Bonaventura, M., Pozo Astigarraga, M.E., Castro, R.: TopoGen: A Network Topology Generation Architecture with Application to Automating Simulations of Software Defined Networks. In: 2017 Winter Simulation Conference (WSC), pp. 1049–1060 (2017). https://doi.org/10.1109/WSC.2017.8247854. https://ieeexplore.ieee.org/abstract/document/8247854 (visited on 07/07/2024)
27. Alrumaih, T.N.I., Alenazi, M.J.F.: GENIND: An Industrial Network Topology Generator. Alexandria Engineering Journal **79**, 56–71 (2023). https://doi.org/10.1016/j.aej.2023.07.062. https://www.sciencedirect.com/science/article/pii/S111001682300649X (visited on 07/08/2024)

28. Venkatesan, S., Youzwak, J.A., Sugrim, S., Chiang, C.-Y.J., Poylisher, A., Witkowski, M., Walther, G., Wolberg, M., Chadha, R., Newcomb, E.A., Hoffman, B., Buchler, N.: VulnerVAN: A Vulnerable Network Generation Tool. In: MILCOM 2019 - 2019 IEEE Military Communications Conference (MILCOM), pp. 1–6 (2019). https://doi.org/10.1109/MILCOM47813.2019.9021013. https://ieeexplore.ieee.org/document/9021013 (visited on 07/12/2024)
29. Yamin, M.M., Katt, B.: Modeling and Executing Cyber Security Exercise Scenarios in Cyber Ranges. Computers & Security **116**, 102635 (2022). https://doi.org/10.1016/j.cose.2022.102635. https://www.sciencedirect.com/science/article/pii/S0167404822000347 (visited on 07/12/2024)
30. Besson, P.-V., Tong, V.V.T., Guette, G., Piolle, G., Abgrall, E.: URSID: Automatically Refining a Single Attack Scenario into Multiple Cyber Range Architectures. In: Mosbah, M., Sèdes, F., Tawbi, N., Ahmed, T., Boulahia-Cuppens, N., Garcia-Alfaro, J. (eds.) Foundations and Practice of Security, pp. 123–138. Springer Nature Switzerland, Cham (2024). https://doi.org/10.1007/978-3-031-57537-2_8
31. Kaveh, A., Rohner, C., Johnsson, A.: Impact of Attack Variations and Topology on IoT Intrusion Detection Model Generalizability. In: 2024 IEEE 21st International Conference on Mobile Ad-Hoc and Smart Systems (MASS), pp. 364–370 (2024). https://doi.org/10.1109/MASS62177.2024.00055. https://ieeexplore.ieee.org/document/10723576 (visited on 10/29/2024)
32. National Initiative for Cybersecurity Education, The Cyber Range: A Guide, (2023)
33. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach. Pearson, Hoboken, NJ (2021)
34. Strom, B.E., Applebaum, A., Miller, D.P., Nickels, K.C., Pennington, A.G., Thomas, C.B.: MITRE ATT&CK®: Design and Philosophy. 10AOH08A-JC, (2020). https://attack.mitre.org/docs/ATTACK_Design_and_Philosophy_March_2020.pdf (visited on 07/07/2022)
35. Jia, Z., Wu, Q., Dong, C., Yuen, C., Han, Z.: Column Generation for Optimization Problems in Communication Networks. IEEE Network **37**(3), 86–92 (2023). https://doi.org/10.1109/MNET.108.2100634. https://ieeexplore.ieee.org/document/9961136/ (visited on 06/20/2025)

Appendices

Table 2: Fixed experiment parameters for the different benchmarks.

(a) Library size. (5.1)

Parameter	Value
Min. number of nodes	10
Max. number of nodes	25
Min. number of sub-topologies	2
Max. number of sub-topologies	6
Services list	empty
Attacks list	empty
Tree depth	2

(b) Maximum number of nodes. (5.2)

Parameter	Value
Min. number of nodes	1
Min. number of sub-topologies	1
Max. number of sub-topologies	6
Services list	empty
Attacks list	empty
Tree depth	2
Library size	40

(c) Maximum tree depth. (5.3)

Parameter	Value
Min. number of nodes	10
Max. number of nodes	30
Min. number of sub-topologies	2
Max. number of sub-topologies	10
Services list	empty
Attacks list	empty
Library size	20

(d) Number of service constraints. (5.4)

Parameter	Value
Min. number of nodes	5
Max. number of nodes	15
Min. number of sub-topologies	2
Max. number of sub-topologies	6
Attacks list	empty
Tree depth	2
Library size	40