Chapter 7

Topology Generation for Independent Distributed Datasets

Contents

7.1 Introduction 127
7.2 Requirements 128
7.3 Related Work 130
7.4 Topology Generator for Federated IT Networks 131
7.5 Performance Benchmark 135
7.6 Conclusion and Takeaways 140

7.1 Introduction

Over the course of the past chapters, we have seen how the performance of Federated Intrusion Detection Systems (FIDSs) can be impacted by the heterogeneity of the participants' data distributions. Yet, we have been limited in our experiments by the lack of datasets to thoroughly evaluate this aspect of FIDSs. Indeed, the existing public datasets in the literature are typically created using a single network topology, leaving researchers working on distributed approaches with two choices: (i) use the existing datasets and rely on partitioning strategies to simulate the heterogeneity of real-world data distributions; or (ii) apply standardized feature sets on existing public datasets to create training sets coming from independent, siloed infrastructures.

Unfortunately, both of these approaches have limitations. In the first case, the partitioning strategies cannot fully replicate the heterogeneity of real-world data distributions, as data will remain correlated to some extent. Moreover, the partitioning strategies are not always applicable to all datasets, and it requires a deep understanding of the way each dataset is generated to approach realistic data shards. In the second case, the number of clients is limited by the number of public datasets available, narrowing experiments to extremely small-scale federations. Additionally, because all datasets are independent, characterizing the heterogeneity of the data distributions is difficult, leaving little control over the experimental conditions. In Chapter 6, we leveraged a combination of both strategies to simulate *practical* NIID (Non Independent and Identically Distributed) settings, but the results remained limited in realism by the lack of control over the data

distributions.

To close the gap towards more realistic evaluations of FIDSs, we introduce a novel approach to generate heterogeneous network topologies that can be deployed in virtualized environments. Because creating a functional topology from scratch is particularly complex, we propose to compose topologies from a set of predefined building blocks that satisfy at set of user-defined constraints. By leveraging routing protocols and domain name resolution, we can dynamically a large number of topologies that can be used to evaluate the performance of FIDSs an heterogeneous, yet controlled, environment.

The content of this chapter originates from the preliminary work presented at the C&ESAR conference in late 2022 [Léo+22]. Its remainder is structured as follows. We start by laying out the requirements for topology generation in the context of FIDSs in Section 7.2, and review existing works in the field in Section 7.3. We then present our approach to generate topologies in Section 7.4, and evaluate the performance of our tool in Section 7.5. Finally, we discuss the perspectives of our work in ?? before concluding in Section 7.6.

Contributions of this chapter

- A novel approach to generate realistic network topologies for dataset generation by leveraging constraint-based topology composition.
- A benchmark of the number of topologies that can be generated using our approach.
- The foundations for the first truly distributed dataset in intrusion detection, enabling the evaluation of FIDSs in under-controlled conditions.

7.2 Requirements

The topic of topology generation has been extensively researched in the late 90s and early 2000s, notably to evaluate the performance of network protocols in large-scale networks [Med+01]. In a structuring survey on network topologies, Haddadi *et al.* [Had+08] synthesized the requirements for network topology generators, from which we extract the following requirements for our use case: representativeness, extensibility, and efficiency.

In addition to these requirements, since the goal of this work is to build topologies for dataset generation, we take inspiration from the literature on Network-based Intrusion Detection System (NIDS) datasets. In their work, Ring, Wunderlich, Scheuring, *et al.* [Rin+19] identify the qualities of a *perfect* dataset. It should: be up-to-date, correctly labeled, and publicly available, contain real network traffic with various attacks and normal

user behavior, and span a long time. Finally, because we strongly believe in the importance of reproducibility in experimental research, we follow the requirements laid out by Uetz *et al.* [Uet+21] for sound experiments. They need to be: valid (*i.e.*, well-defined and unrefutable), controllable (*e.g.*, parameterized), and reproducible (*i.e.*, the same results can be obtained by another group using the author's artefact). Based on these qualities, we derive the following requirements for the topology generator.

— *Representativeness:* the generated topologies must be representative of modern real-world networks and respect their statistical properties.

— *Extensibility:* the tool must allow users to extend its capabilities.

— *Efficiency:* the tools must be efficient to generate large-scale topologies without altering their properties.

— *Validity:* the generated topologies must be exempt of side effects or biases that could alter the results of the experiments.

— *Controllability:* the generator must allow precise control over the differences between the generated topologies.

— *Reproducibility:* the generator must be able to deterministically generate the same topology multiple times.

7.2.1 Controlling Heterogeneity

The major challenge in generating network topologies for FIDSs is to control the heterogeneity of the generated datasets. Notably, the generated topologies and the associated datasets must allow researcher to evaluate the impact of different data-distributions on FIDSs, what identify the aspects of heterogeneity that are the most impactful. Consequently, the generator must allow precise control over the differences between the generated topologies. To this end, we identify five main qualities that characterize the heterogeneity of network topologies: architecture, attack scenarios, hosted services, user behaviors, and maturity.

1. **Architecture.** The network architecture of the topology defines how services are interconnected, how the traffic is captured, and where data collection is performed. For instance, a topology with a single main gateway which captures the traffic of several services on the same network will produce a different dataset when compared with a star-shaped topology with multiple subnets. Appropriated metrics are required to characterize the impact of these differences, *e.g.*, size (number of hosts, of subnets), mean number of hops between a service and the last gateway, and so on.
2. **Attack scenarios.** The literature on intrusion detection is rich with different classes of attacks that generate different patterns of traffic. For instance, a Denial of Service

(DoS) or brute force attack will generate a lot of traffic which will vary depending on the targeted service, *e.g.*, an SSH server, a database over TCP, and so forth.

3. **Hosted services.** Different services can rely on different protocols, and therefore generate different kind of data. For example, a service using TCP will induce connection establishment, and therefore a lot of traffic back-and-forth, whereas something based on UDP will produce a more continuous stream of data. The Internet of Things (IoT) also introduce new kinds of network traffic patterns, with unusual protocols such as CoAP or MQTT. Therefore, different services (and protocols) might have different normal behaviors, causing heterogeneity among participants. The list of considered services must be adapted depending on the considered attack scenarios.
4. **User behaviors.** The behavior of users can also impact the network traffic depending on the considered use case and the type of organizations. For example, a university network will have a lot of students connecting to the internet, whereas a company will have a lot of internal traffic. Other factors like working hours, the sue of a Virtual Private Network (VPN), or the use of a Bring Your Own Device (BYOD) policy can also impact the network traffic.
5. **Maturity.** Security practices vary between organizations, depending on their threat model, previous expertise, and budget. For example, a company might have a dedicated security team, and therefore be able to implement a more mature security policy, whereas a small company might not have the resources to do so. This parameter is important to consider, as it can impact the quality of the dataset, *e.g.*, by having unseen attacks in the training data, supposed to be benign traffic.

User behaviors can be implemented directly in traffic generators and are not considered in the scope of this work. The maturity of the organization can be simulated by two approaches: either by generating different topologies with different security policies and relying on the service description to generate the topologies, or by altering data quality in the generated datasets afterward. Consequently, we focus on the architecture, attack scenarios, and hosted services to define the requirements of the topology generator.

7.3 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 [Tan+02]. Multiple tools have been developed to generate network topologies at the time, such as `GT-ITM` [CDZ97], `Tiers` [Doa96], or `BRITE` [Med+01]. Tangmunarunkit *et al.* [Tan+02] 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, notably the power-law distribution of the node degrees [FFF99]. 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.

Recent works on topology generation are rarer and focus on specific use cases. For instance, Laurito *et al.* [Lau+17] 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. Yet, their approach is limited to SDNs and does not allow automating data generation. Alrumaih and Alenazi [AA23] developed `GENIND`, a tool to generate industrial network topologies. 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 focus on generating topologies for industrial networks, and therefore generates topologies with specific constraints layer-by-layer, before connecting the different sub-topologies together in a multigraph. To the best of our knowledge, no tool has been developed to generate *deployable* network topologies for IT networks, and can be randomized to generate a large number of topologies with common characteristics.

7.4 Topology Generator for Federated IT Networks

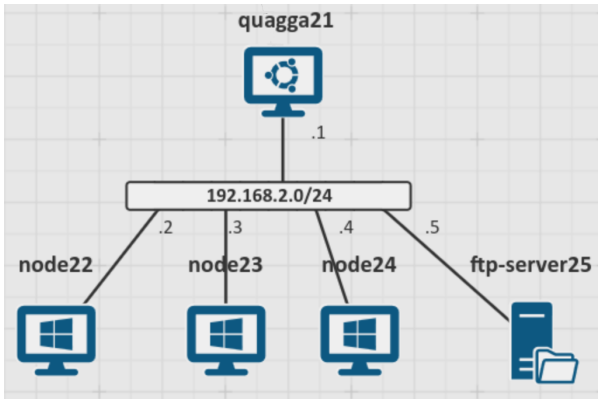
To solve the limitations of existing datasets in the literature, we introduce `FedITN_gen`, a topology generator for federated IT networks. In this section, we present the design and architecture of `FedITN_gen`, before detailing its implementation using Airbus' CyberRange platform [1].

7.4.1 Approach Overview

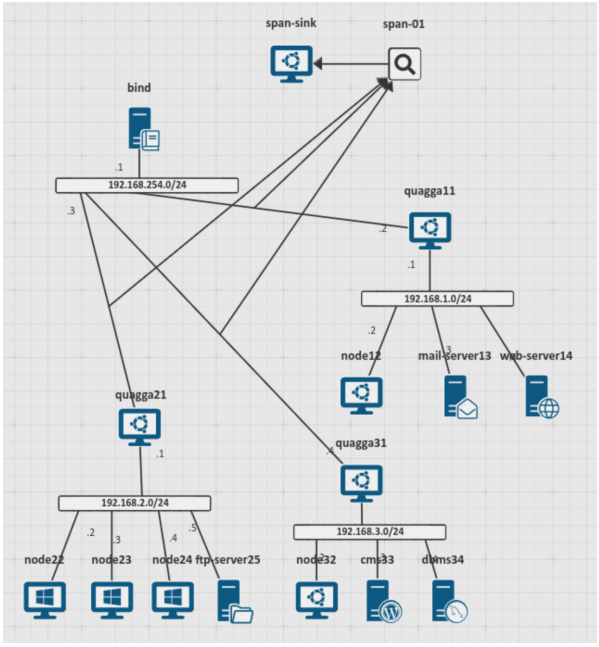
The core idea behind `FedITN_gen` is to compose network topologies by selecting and connecting a set of building blocks that satisfy user-supplied constraints from a library of predefined sub-topologies. While creating this library remains human operated, the composition of the topologies is fully automated, allowing the generation of different topologies with common characteristics. `FedITN_gen` is 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 tree-like structure starting from the *Master* sub-topology.

1. https://www.cyber.airbus.com/products/cyberrange/



(a) Example of a sub-topology instantiated on Airbus' CyberRange, containing 3 Windows clients and 1 FTP server. The node labeled `quagga21` is the gateway connecting the sub-topology to the rest of the network.



(b) Example of a complete topology composed of 3 sub-topologies. Each sub-topology is connected to the rest of the network through a gateway.

Figure 7.1 – Topology generation with `FedITN_gen`.

7.4.1.1 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 is composed of a subnet, a set of nodes (clients or servers), and a gateway that connects the subnet to the rest of the network. Figure 7.1a shows an example of a sub-topology with 4 nodes. Each sub-topology is formally defined as its `/24` subnet s, which also can serve as its identifier, and a set of clients H_s and services S_s that we generally refer to as nodes, *i.e.* $N_s = H_s \cup S_s$. The gateway g_s is a particular node (not comprised in N_s) that connects the subnet to the rest of the network. A sub-topology can therefore be represented as a tuple $t_s = (s, N_s, g_s)$. The topology selection is a Constraint Satisfaction Problem (CSP) where the goal is to find all sets of sub-topologies that satisfy the user-defined constraints. Definition 7.1 defines the concept of a CSP.

While dimensioning the output topologies can easily be translated using numerical 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`. An attack scenario is defined as another tuple $A_k = \langle \text{srcs} = \{N_1, N_2, \ldots\}, \text{targets} = \{N_{11}, N_{12}, \ldots\}\rangle$ where `srcs` and `targets` are sets of nodes available in the library that are compatible with the

Definition 7.1: Constraint Satisfaction Problem (CSP) [RN21]

A Constraint Satisfaction Problem (CSP) is a tuple $P = \langle X, D, C \rangle$ where:

- $X = \{X_1, X_2, \ldots, X_n\}$ is a set of variables.
- $D = \{D_1, D_2, \ldots, D_n\}$ is a set of non-empty domains, one for each variable.
- $C = \{C_1, C_2, \ldots, C_n\}$ is a set of constraints that specify allowable combinations of values in their domains.

Each constraint $C_j \in C$ is a ~~new~~ tuple $C_j = \langle \chi, R \rangle$ where R is a relation between the variables in $\chi \subseteq X$. Thus, solving a CSP is finding an assignment of values ~~for~~ from X in D that satisfies all constraints in C.

attack scenario. For instance, a Man-in-the-Middle (MitM) attack could be represented as $A_{\text{MITM}} = \langle \text{srcs} = \{N_{\text{attaker}}\}, \text{targets} = \{N_1, N_2\} \rangle$. Based on the aforementioned definitions, we can define our Topology Selection Problem (TSP) in Problem 7.1.

Problem 7.1: Topology Selection Problem (TSP)

Given a set of sub-topologies $T = \{t_1, t_2, \ldots, t_n\}$, a set of services $D_{service}$, and a set of attack scenarios $A = \{A_1, A_2, \ldots, A_m\}$, find all sets of sub-topologies $T' \subseteq T$ that satisfy the following:

1. *The number of sub-topologies in T' is between n_{min} and n_{max}.*
2. *The total number of nodes in T' is between h_{min} and h_{max}.*
3. *Each service in $D_{service}$ is available in at least one sub-topology in T'.*
4. *Each attack scenario in A is compatible with at least one sub-topology in T'.*

7.4.1.2 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 7.1b 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. Yet, many variations of the same tree can be created, as illustrated in Figure 7.2.

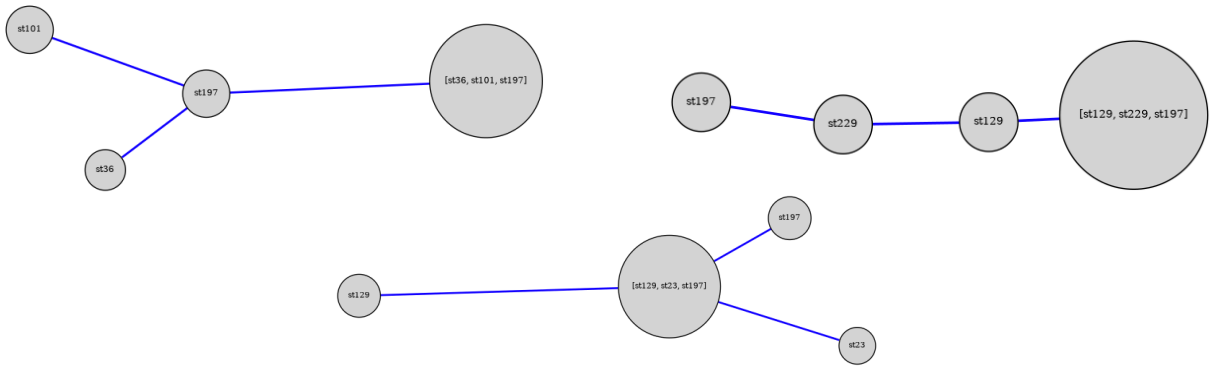


Figure 7.2 – Examples of tree-like topologies composed of 3 sub-topologies. The *Master* sub-topology is represented by the biggest node, and labeled with the IDs of the sub-topologies.

7.4.2 Implementation

In this section, we present parts of the implementation of `FedITN_gen` using Airbus' CyberRange platform. A CyberRange is a virtual environment that simulates a real-world network, allowing users to train and test their cybersecurity skills. Most importantly, such platforms come with a set of templates that can be used to create sub-topologies, and compatible attack scenarios and user traffic generation tools to generate realistic traffic. 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[2], and we develop a naive 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 operation to ensure deterministic results.

The *Master* sub-topology. The *Master* topology serves as the basis for the composition. It consists of a gateway connecting it to the Internet, a collection point for network logs, a DHCP server for distributing IP addresses, and a DNS server for resolving domain names across all topologies. This is the only topology that will be configured, in particular to allocate the correct domain names to the IPs of the machines hosting the services that need to be accessible. For example: web server (`webserver.local`), mail server (`mailserver.local`), file sharing (`fileserver.local`), etc. This approach makes it possible to define unique domain names for each service, and make them accessible from any topology.

2. https://developers.google.com/optimization/cp/cp_solver

Handling connectivity. Now that all services are available via their domain names, the next step is to ensure that the services are reachable from any sub-topology. To do so, each gateway g_s hosts a DHCP server with a dedicated range to allocate IP addresses 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 7.4.1, the topology selection is a CSP that can be solved using a constraint solver. In particular, we implement our cardinality-related constrains (*i.e.*, the number of sub-topologies and nodes) using the `CP-SAT` solver. This generates all possible combinations of sub-topologies that satisfy the constraints, and we then filter out the ones that do not contain the required services or are not compatible with the attack scenarios. The remaining sets of sub-topologies are then composed into complete topologies using a naive recursive algorithm that walks a tree, starting from the *Master* sub-topology, and randomly assigns children sub-topologies to the gateways. We implement a simple backtracking algorithm to roll back the composition when the tree-depth constraint is not satisfied, and try again from another branch.

7.5 Performance Benchmark

In this section we present some results regarding the performance of `FedITN_gen`, as well as the influence of some of the constraints on performance. To evaluate the tool, we generated 253 sub-topologies with various characteristics which are stored in the library. We then performed a series of experiments to evaluate the performance of `FedITN_gen` in various conditions. We notably measure the impact of the library size, the maximum number of nodes, the tree depth, and the number of service constraints on the performance of `FedITN_gen`. The performance is assessed via the execution time, the number of generated topology sets, and the number of generated final topologies (*i.e.*, tree compositions of the topology sets).

7.5.1 Influence of the library size

We first evaluate the influence of the library size on the performance of `FedITN_gen`. We randomly select a number (in the range 1–29) of sub-topologies from the full library of 253 sub-topologies. For each value of the library size, we perform ten experiments with the parameters fixed as in Table 7.4.

Table 7.1 – Fixed parameters for the library size benchmark.

Parameter	Value
Minimum number of nodes	`10`
Maximum number of nodes	`25`
Minimum number of sub-topologies	`2`
Maximum number of sub-topologies	`6`
Services list	`empty`
Attacks list	`empty`
Tree depth	`2`

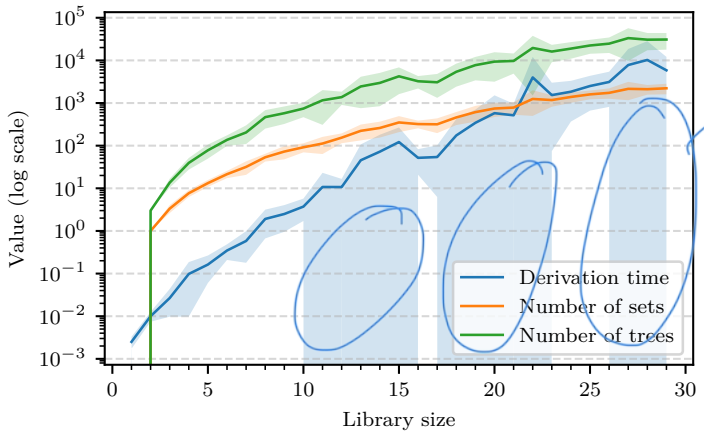


Figure 7.3 – Influence of the library size on the performance of `FedITN_gen`.

Table 7.2 – Fixed parameters for the maximum number of nodes benchmark.

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

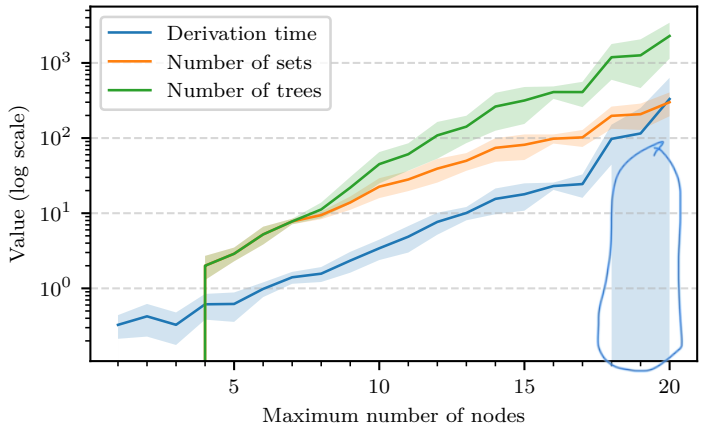


Figure 7.4 – Influence of the maximum number of nodes on the performance of `FedITN_gen`.

Figure 7.3 displays the different metrics on a log scale. We notably observe that the execution time increases exponentially 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.

7.5.2 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 as in Table 7.4. For each value of the maximum number of nodes, we likewise perform ten experiments.

Figure 7.4 displays the different metrics on a log scale. 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

Table 7.3 – Fixed parameters for the tree depth benchmark.

Parameter	Value
Minimum number of nodes	`10`
Maximum number of nodes	`30`
Minimum number of sub-topologies	`2`
Maximum number of sub-topologies	`10`
Services list	`empty`
Attacks list	`empty`
Library size	`20`

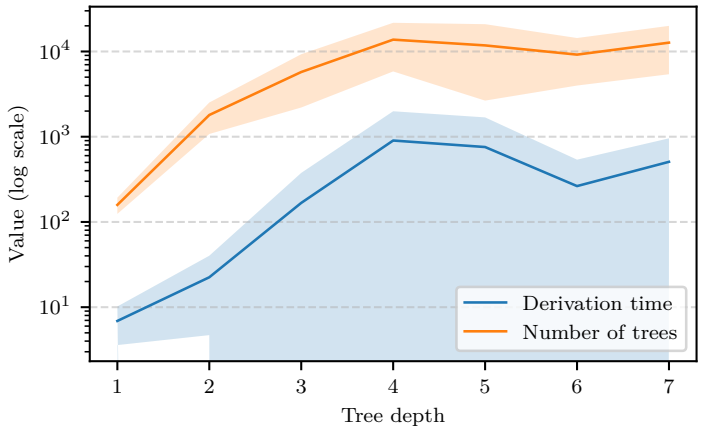


Figure 7.5 – Influence of the tree depth on the performance of `FedITN_gen`.

significantly higher than the number of sub-topology sets. Note that for a number of nodes inferior to 4, there are no solutions, are the topologies in our test library have at least 4 nodes.

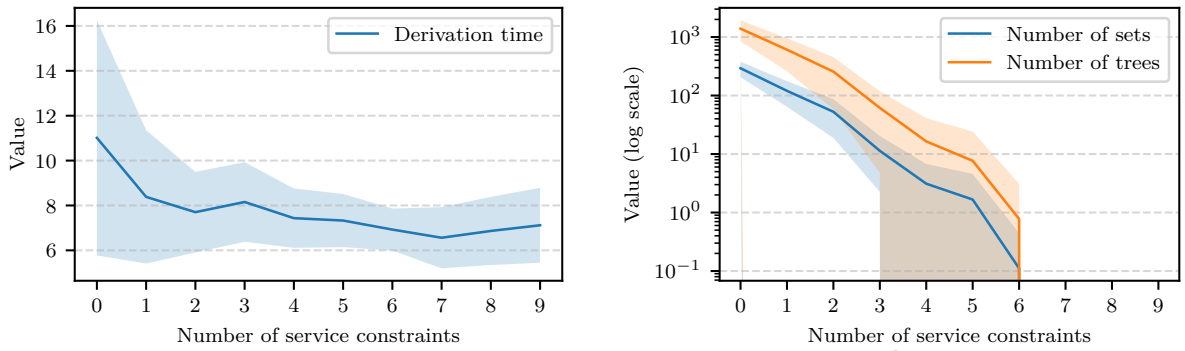
7.5.3 Influence of the tree depth

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 execution time. We vary the tree depth from 1 to 7, with the other parameters fixed as in Table 7.4.

Figure 7.5 displays the execution time and the number of tree compositions on a log scale. Both increase exponentially with the 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 variations in the results are only due to the random sampling of the sub-topologies in the library.

Table 7.4 – Fixed parameters for the number of service constraints benchmark.

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



(a) Execution time. **(b)** Number of tree compositions.

Figure 7.6 – Influence of the number of service constraints on the performance of `FedITN_gen`.

7.5.4 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 services constraints from 1 to 9, with the other parameters fixed as in Table 7.4.

Figures 7.6a and 7.6b display the execution time and the numbers of sets and tree generations, respectively. Unlike the other experiments, the execution time in almost constant here, due to the way these constraints are handled. While the other constraints are implemented as exploration problems, the service constraints are applied by pruning incompatible topology sets. This is why the number of sets and trees are progressively decreasing as the number of service constraints increases. By the 6th constraint, the problem 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.

7.6 Conclusion and Takeaways

In this chapter, we presented `FedITN_gen`, a tool to generate heterogeneous network topologies for generating 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 find 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. We believe that having independently generated datasets that are comparable with the state of the art, while allowing finer controls over the heterogeneity of the data, would enable addressing some of the current open challenges in the literature:

(a) *performance against heterogeneity*: the ability for the federation to maintain high performance with heterogeneous participants;

(b) *knowledge transfer between clients*: the ability for one client to recognize patterns that are absent from its local training data;

(c) *model adaptability*: the ability of a local model to evolve in time, and to adapt to new devices in the local network;

(d) *generation capability*: the ability for a local model to correctly characterize behavior for similar but different services.

Future Work `FedITN_gen` is 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. Another lead for improvement lies in the optimization of the constraint solver, which is currently the bottleneck of the tool. This is critical to introduce finer constraints and allow for more complex topologies. Finally, while we identified as a requirement the respect of the statistical properties of real-world IT networks, this currently remains an open challenge. Consequently, future works include the study and identification of said properties in real-world deployments, and the integration of these properties in the generation process.