topology generation

fabien.autrel

July 2024

1 Introduction

In the context of federated learning applied to the detection of intrusions, performance evaluation is generally done using datasets. However, current available datasets are not tailored specifically for federated approaches, and more globally not tailored for distributed approaches. Indeed the available public datasets are created on the basis of a single network topology, and using them to evaluate federated approaches require some transformations to emulate a cross-silo system. As a matter of fact, a cross-silo federation involves participants with different information systems implemented through different network topologies.

Creating multiple heterogeneous topologies by hand on which various attacks and life generators are then executed is a lengthy process.

To address this issue, we propose to automate the generation of network topologies to enable the creation of various network topologies which can then be instantiated on a virtualization infrastructure in order to create heterogeneous datasets.

1.1 Requirements

Such a topology generator should have some properties to enable the creation of datasets:

- The size of the generated topologies should be constrained in terms of the total number of nodes in the system and the total number of subnets
- The list of operating systems and services deployed in a topology should be constrained to reflect the specificity of some use cases
- The list of attacks which can be executed should be specified

Instead of generating topologies from scratch, which is a complex problem, we propose an intermediate approach where sub-topologies are specified by hand, then those sub-topologies are combined by an algorithm to create full topologies. Each sub-topology is composed of a router and several machines connected to it (figure 1). The sub-topologies are stored into a library and the

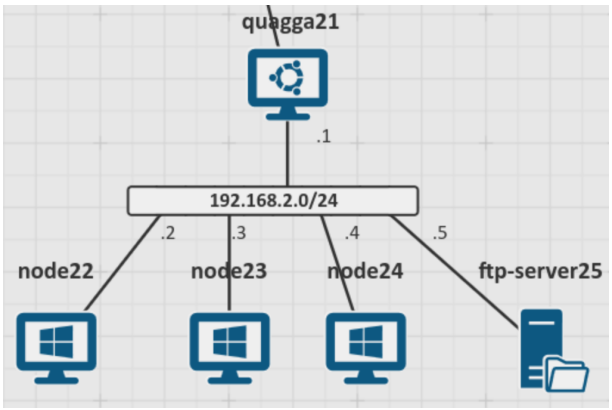

Figure 1: Example of a sub-topology instanciated on the Airbus Cyberrange. The machine named *quagga21* is the router

generation algorithm picks subsets of sub-topologies in the library to construct bigger topologies.

We choose to construct the topologies around a root sub-topology which contains a subnet with a fixed address, i.e. 192.168.241.0/24, which contains a DNS server. The generation process involves choosing a subset of the sub-topologies stored in the library to connect them either to the root sub-topology or to other sub-topologies in the choosen subset. More precisely, the generated topologies are trees which have the root topology as their roots. The DNS server in the root topology is configured to associate a name to each machine in the generated topology. figure 2 shows an example of a generated topology where three sub-topologies are connected to the root topology.

1.2 Implementation

As introduced in the previous section, we wish to generate topologies which satisfy some properties in terms of number of machines, set of services and attacks that can be executed. To do so, we use the CP-SAT solver from google to compute the subsets of sub-topologies, if any, which satisfy the specified constraints. We define the following constraints as inputs to our tool:

- Minimum number of nodes: the minimum number of machines in the generated topology
- Maximum number of nodes: the maximum number of machines in the generated topology
- Minimum number of sub-topologies: the minimum number of sub-topologies (i.e. subnets) from the library in the generated topology

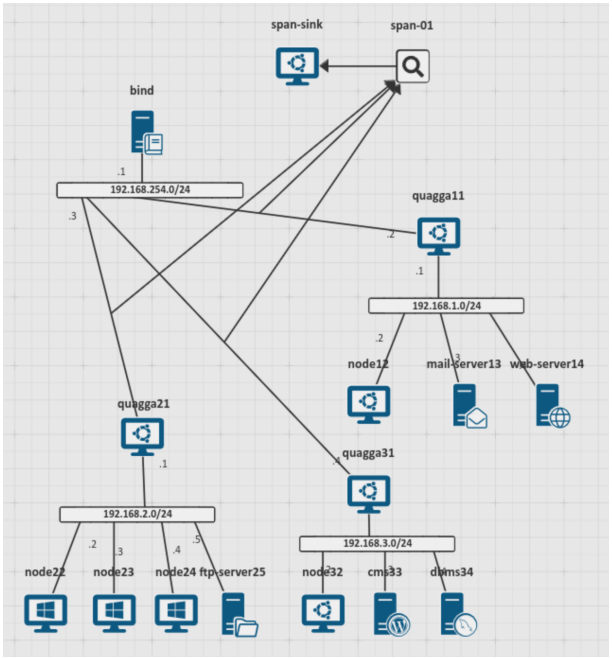

Figure 2: Example of a generated topology instanciated on the Airbus Cyber-range

- Maximum number of sub-topologies: the maximum number of sub-topologies (i.e. subnets) from the library in the generated topology
- Services list: list of services that must be present in the generated topologies
- Attacks list: list of attacks that must be able to be executed in the generated topologies
- Tree depth: the maximum depth of the trees generated

The CP-SAT solver is used to generate subsets of the sub-topologies library which satisfy the 6 first constraints, then the tree depth constraint is separately used to compute the possible trees from the generated subsets.

The tool is implemented in Python. Its output is a set of graphs whose nodes are the names of the sub-topologies in the library. Figure 3 shows three examples of such graphs.

1.3 Performance assessment

In this section we present some results regarding the performance of the tool, as well as the influence of some of the constraints on the performance. To evaluate

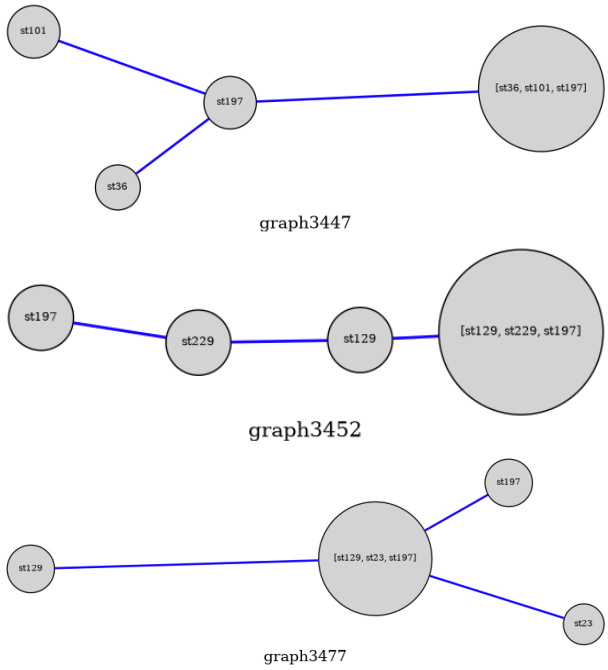

Figure 3: Examples of a generated trees with a maximum depth of 3 and a maximum number of 3 sub-topologies

the tool, we generated 253 sub-topologies which are stored in the library.

1.3.1 Influence of the sub-topologies library size

In this experiment, we vary the library size and set the values of the constraints to the following values:

- 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

The library size in our experiment varies from 1 to 29. For each value of the library size, 10 runs are made and the corresponding number of sub-topologies is randomly selected from the full library of 253 sub-topologies.

Figure 5 shows different metrics regarding this experiment. The first graph shows the evolution of the time taken to compute the topologies graphs when the library size increases. The Y axis uses a logarithmic scale, and the curve shows an exponential growth of the derivation time. This is expected due to the nature of the constraint solver and the tree generation algorithm. The number of sub-topology sets which satisfy the constraint show the same growth as well as the number of generated trees, although the number of trees is of course higher than the number of sub-topology sets since the maximum tree depth is set to 2.

1.3.2 Influence of the maximum number of nodes

In this experiment, we vary the maximum number of nodes and set the values of the other constraints to the following values:

- 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

The library size is set to 40.

Figure 5 shows different metrics regarding this experiment. The first graph shows the evolution of the time taken to compute the topologies graphs when the maximum number of nodes increases. The Y axis uses a logarithmic scale, and the curve shows an exponential growth of the derivation time. This is expected due to the nature of the constraint solver and the tree generation algorithm. The number of sub-topology sets which satisfy the constraint show the same growth as well as the number of generated trees, although the number of trees is of course higher than the number of sub-topology sets since the maximum tree depth is set to 2. We can also see that for a maximum number of nodes inferior to 4, no solution are found, which is linked to the fact that the sub-topologies in the library have at minimum 4 nodes.

1.3.3 Influence of the tree depth

In this experiment, we vary the maximum number of nodes and set the values of the other constraints to the following values:

- 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
- Tree depth: 2

The library size is set to 20.

Figure 6 shows different metrics regarding this experiment. The first graph shows the evolution of the time taken to compute the topologies graphs when the maximum tree depth increases. The Y axis uses a logarithmic scale, and the curve shows an exponential growth of the derivation time which then meets a plateau. This is expected due to the nature of the constraint solver and the tree generation algorithm. The plateau is linked to the fact that although the maximum tree depth increases, the number of possible solutions is constrained by the other constraints. The number of sub-topology sets which satisfy the constraint does not grow since the tree depth as no influence on the satisfaction problem. The variablity of the results is due to the fact that we randomly pick 20 sub-topologies from the full library. The number of generated trees is in contrast directly influenced by the maximum tree depth, although this value also reaches a plateau as the number of sub-topologies sets does not vary.

1.3.4 Influence of the number of services

In this experiment, we vary the number of requested services and set the values of the other constraints to the following values:

- 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

The library size is set to 40. The list of available services is the following: ldap, dbms, cms, dns, mail, syslog_server, web, ftp, proxy, cloud_storage.

Figure 7 shows different metrics regarding this experiment. The first graph shows the evolution of the time taken to compute the topologies graphs when the number of services constraint increases. We can see that the number of constraints on the services has no effect on the derivation time. In contrast, as the number of constraints on the services increase, the number of solutions

decreases to reach 0 because the sub-topologies don't contain all the services. Since the maximum of sub-topologies is limited to 10, we quickly reach a point where no combination of sub-topologies which satisfies this constraint.

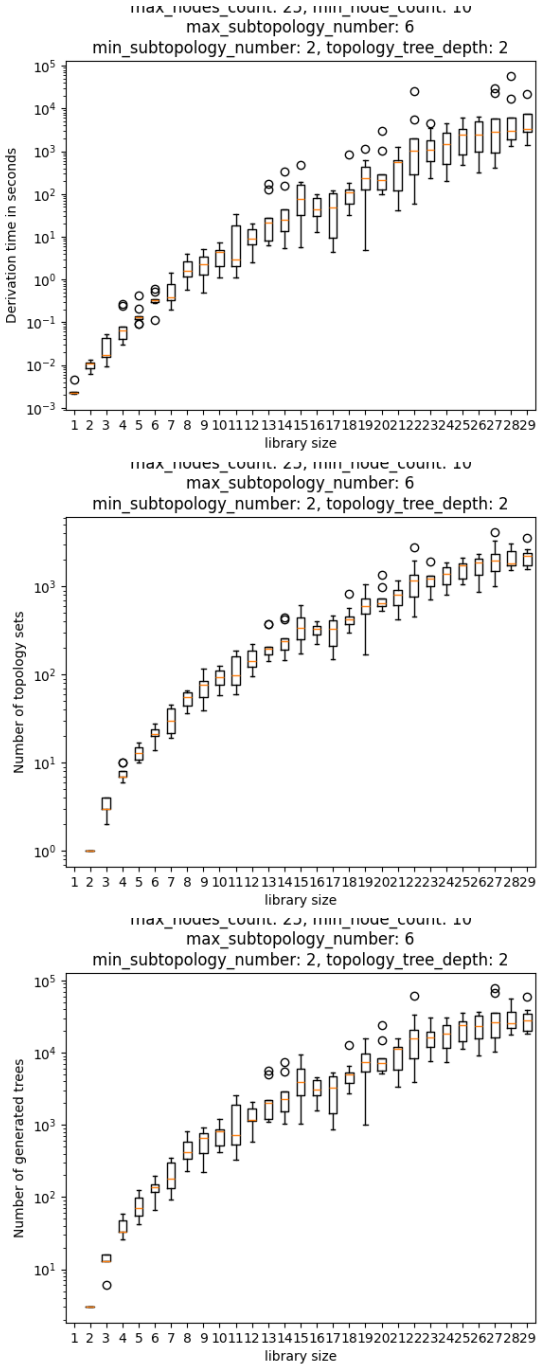

Figure 4: The Y axis for all images are logarithmic.

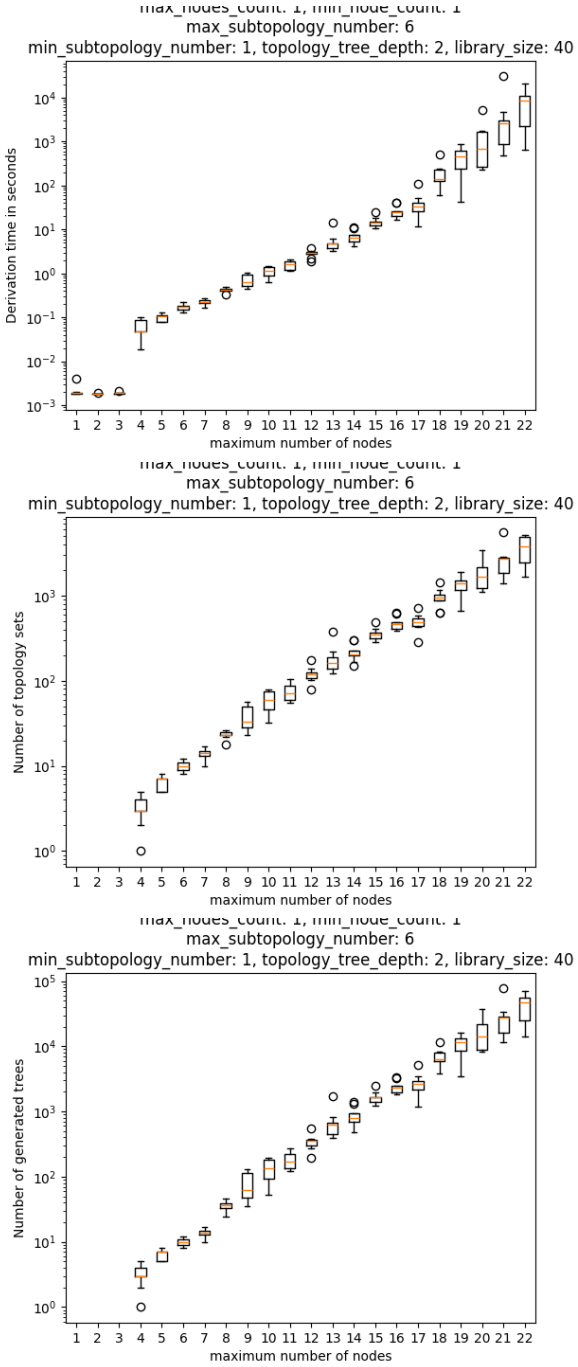

Figure 5: The Y axis for all images are logarithmic.

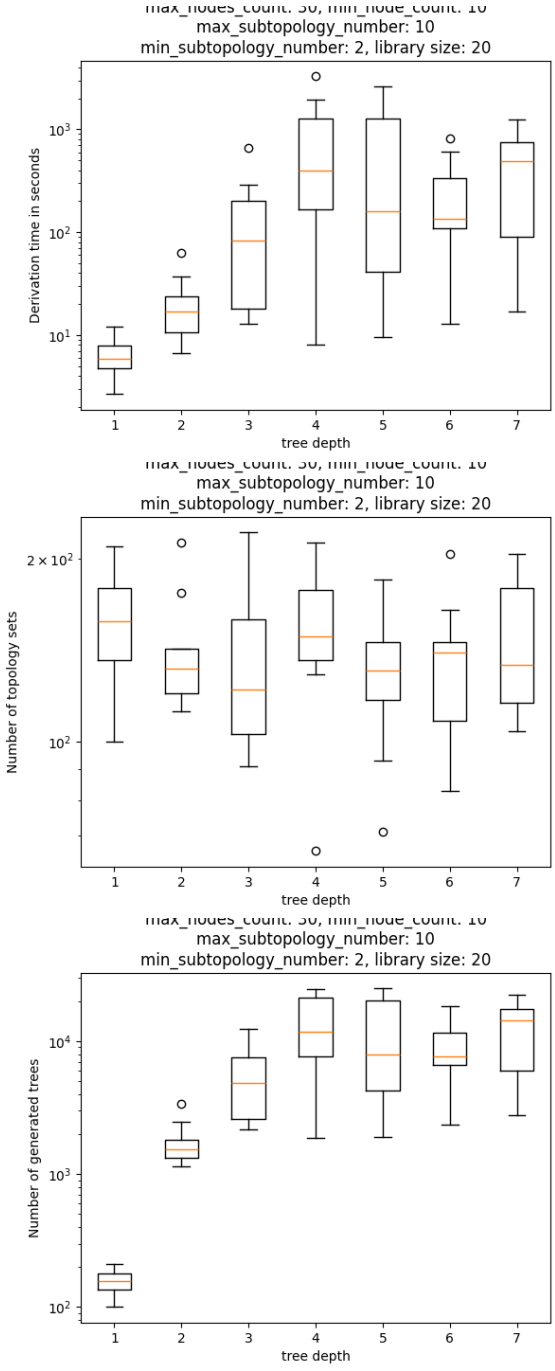

Figure 6: The Y axis for all images are logarithmic.

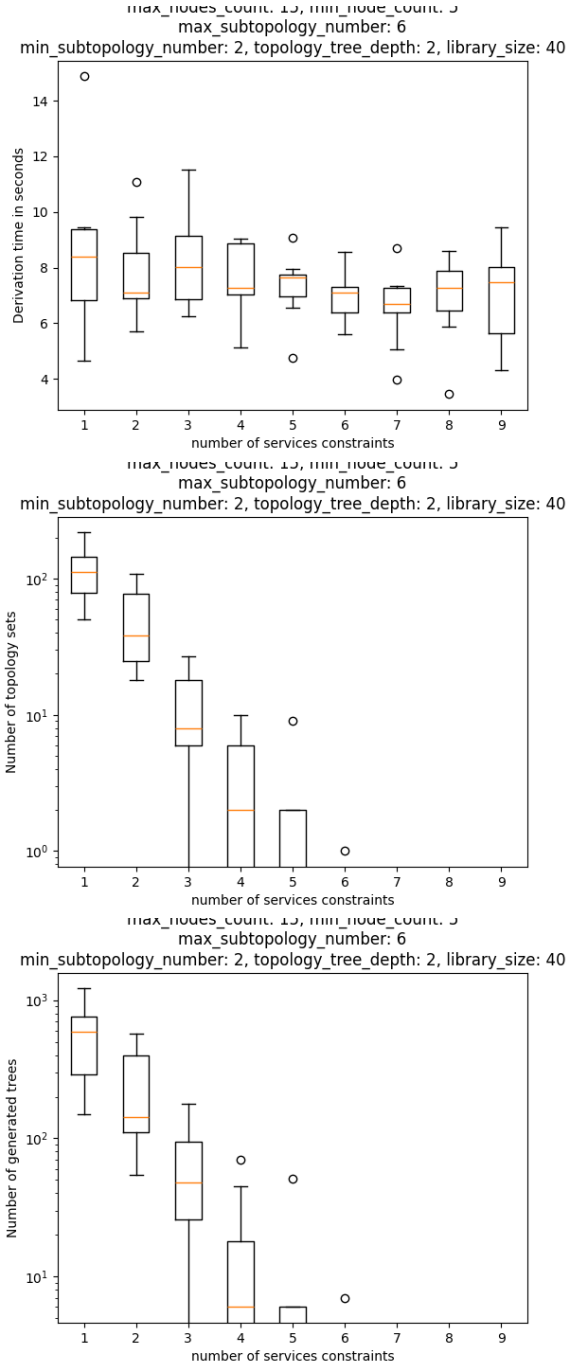

Figure 7: Influence of the number of requested services