Topology-driven Configuration of Emulation Networks with Deterministic Templating

Satoru Kobayashi, Ryusei Shiiba, Shinsuke Miwa, Toshiyuki Miyachi, and Kensuke Fukuda

***Abstract*—Network emulation is an important component of a digital twin for verifying network behavior without impacting on the service systems. Although we need to repeatedly change network topologies and configuration settings as a part of trial and error for verification, it is not easy to reflect the change without failures because the change affects multiple devices, even if it is as simple as adding a device. We present topology-driven configuration, an idea to separate network topology and generalized configuration to make it easy to change them. Based on this idea, we aim to realize a scalable, simple, and effective configuration platform for emulation networks. We design a configuration generation method using simple and deterministic config templates with a new network parameter data model, and implement it as dot2net. We evaluate three perspectives, scalability, simplicity, and efficacy, of the proposed method using dot2net through measurement and user experiments on existing test network scenarios.**

***Index Terms*—Configuration management, Template, Emulation network, Topology graph**

I. Introduction

Digital twins have attracted attention in recent years as an approach to improve the reliability of information services. They are digital duplicates of real-world objects available for digital applications such as analysis and verification [1]. In network management, emulation network is a suitable choice to achieve the digital twins [2]. It allows the operators to verify network configurations more dynamically, such as changing configurations and injecting faults [3], [4].

In order to effectively use the emulation networks for these purposes, it is necessary to perform trial and error on configurations by changing the network topology. However, it is time-consuming for operators to repeatedly change emulation network configurations [5]. For example, if we want to add a router to a network, we usually need to change not only the configuration settings of the added router, but also the settings of the adjacent or logically connected devices. This process is inefficient and sometimes results in configuration errors.

An effective approach to solve this problem is to separate network configurations into topology and general configurations. Network device settings usually have some common parts with those of the other devices. By generalizing these settings, we can separate the configuration settings into those for topology structures and those for device roles and behaviors. With this separation, we can change the network topology only by modifying the topology parts, and change the detailed configurations only by modifying the behavior parts. We call this idea as "topology-driven configuration," which is a major goal of this paper.

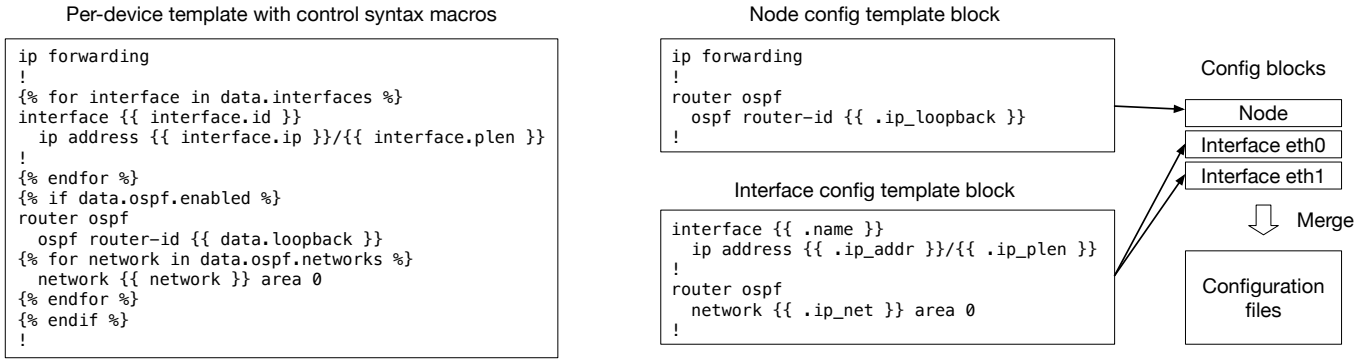
One possible approach to generalize configurations for the topology-driven configuration is to use config templates [6]. A config template is an incomplete configuration description that includes some variable specifications. By specifying variable parameters for templates, we can obtain configuration settings for multiple devices with similar roles. The advantage of the config template is its generality; It simply generates configuration strings with variables, so the approach is independent of network protocols or configuration formats.

However, existing frameworks based on config templates are not reasonable for network configuration. Network configuration depends on various objects that interact with each other, such as interfaces and connections, and it usually needs to refer to parameters on the other objects. For this reason, network configuration requires dynamic templating using several control syntax macros such as "for" and "if" as shown in Figure 1a [7], [8]. Due to the complicated components of networks, the control syntax can be nested multiple times, which is likely to cause descriptive errors in the templates. To overcome this problem, we need a more reasonable config templating mechanism for emulation networks with simple and deterministic (i.e., no control syntax macros) config template blocks like Figure 1b.

Our goal is to provide a scalable, simple, and effective way to configure emulation networks on the basis of topology-driven configuration using a template-based approach. To this end, we propose a method for generating network configuration files from network topology and generalized configurations including templates. It separates the input into a labeled graph to describe network topologies and the label definitions to define the roles and behaviors corresponding to the labels, and generates configuration files from them. The method relies on a new data model to describe network configuration parameters, which is available with a simplified deterministic templates.

We implement dot2net, a network configuration platform based on the proposed template-based approach. It can generate multiple types of configuration files, such as emulation network configuration of the entire architecture or application configuration for each device, consistently. For example, dot2net can automate the deployment of Docker-based emulation networks with modified configurations by combining with some existing emulation network platforms such as Containerlab [9].

S. Kobayashi is with Okayama University.
R. Shiiba is with Sokendai.
S. Miwa and T. Miyachi are with National Institute of Information and Communications Technology.
K. Fukuda is with National Institute of Informatics, and Sokendai.



(a) Dynamic templating (Conventional approach)

(b) Deterministic templating (Proposed approach)

Fig. 1: Difference of templating methods in conventional and proposed approaches.

We evaluate the scalability, simplicity, and efficacy of the proposed method using dot2net. For scalability, we demonstrate the processing time and memory consumption to generate configuration files for large-scale networks. For simplicity, we compare the differential file size to modify configuration files for topology changes with and without dot2net. For efficacy, we present user experiment results and demonstrate that dot2net allows us to change configuration with high speed and few configuration errors.

The contribution of this paper is as follows[1]. (1) We propose a topology-driven configuration method that generates network configuration files from a labeled graph of the network topology and a generalized configuration including config templates. (2) We propose a new data model for network parameters that allows us to use simple and deterministic config templates in topology-driven configuration. (3) We implemented dot2net, a configuration platform for emulation networks, based on the proposed method. We evaluated dot2net in the terms of scalability, simplicity, and efficacy.

II. Related work

A. Network emulation

Network emulation is one of the effective methods to verify network behavior without affecting the service network itself. There are several types of such methods, including theoretical approach, network simulation, and network emulation. Theoretical approach abstracts the behavior and properties of the network systems to the level of mathematical formulas for efficient computation and modeling. This approach is highly exhaustive and useful for comprehensive verification of configuration (i.e., control plane verification [11]) and packet forwarding behavior (i.e., data plane verification [12]). Network simulation reproduces the network behavior with partially simplified functions and properties of the network. This approach is effective for analyses that can be performed efficiently by simplification, such as the quantitative nature of the data [13]. Network emulation deploys virtual networks, imitating the real networks, in a virtual environment using virtual machines or containers. In this approach, we can use the same software as the real network except the infrastructure, such as router applications, so it is effective for debugging application software, verifying configurations [14], collecting behavioral data [15], [16], or education [17], [18]. The objects of network emulation differ depending on their purpose, and both inter-AS (Internet) emulation [17]–[19] and intra-AS emulation [14] have been used in past literature.

There are multiple platforms that support the deployment of emulation networks, such as Netkit [20], Mininet [21], Kathará [22], TiNET [23], and Containerlab [9]. These platforms automate the configuration of bridges or Linux namespaces used in virtual networks, and build emulation networks using virtual machines and containers. However, these platforms do not provide support for efficiently provisioning large numbers of devices or changing configurations. As a result, it currently takes nearly as much effort to configure emulation networks as it does to configure real networks.

B. Configuration synthesis

In the context of configuration automation, configuration synthesis has been gaining attention in recent years. Configuration synthesis generates configuration settings from some high-level representation of operators' intent (so called specifications, requirements, policies, etc. for each paper), which is more intuitive for operators. There are various approaches, such as domain-specific languages [24], [25], formal verification [26]–[29], and machine learning [30], [31].

The idea of topology-driven configuration, the goal of this study, is actually different from and independent of configuration synthesis. Configuration synthesis generates configuration settings from high-level representations of operators' intent. In contrast, topology-driven configuration provides generalized description of configuration settings by abstracting topology-dependent parameters. With this generalized description, operators can modify the network topology without changing the fundamental behavior of the network. Thus, configuration synthesis and topology-driven configuration are independent actions. It might also be possible to generate generalized settings for topology-driven configuration from high-level repre-

[1]This paper is an extended version of the work published in Ref. [10]. The main difference is (2) the proposal of a new data model for deterministic templating. We also extend (3) the evaluation especially in terms of scalability and efficacy.

sentations of operators' intent through configuration synthesis (out of scope in this study).

C. Configuration generalization

The configuration generalization, providing generalized representation of configuration settings, has mainly been developed as a part of configuration automation technology. Templating is the most common approach for generalization. AutoNetkit [7], [8] uses config templates to generate configuration files for network emulation. Figure 1a is based on the example config template for AutoNetkit, and we can see that it requires a lot of "for" and "if" macros as described in Section I. PRESTO [6] generates configuration files through a mechanism called configlet, which is an extension of config template. A configlet is essentially a template in which SQL is embedded as macros, which is the opposite of our approach in this paper, simple and deterministic config templates.

The approach using config templates has the advantage of enabling a wide range of configuration generation consistently, not just network configuration, because it is simple string generation. Still, these existing techniques do not achieve topology-driven configuration in their generalization. The major challenge is that the data model does not fit the purpose of changing the topology. This leads to the complexity of templates using macros.

In this paper, we use the config template approach in the configuration generation step for generality to the supported configuration formats. In addition, we also intend to solve the problem of complicated config templates with control syntax macros that exists in existing approaches, by a method based on a new data model for generating configuration files with simple and deterministic templates.

III. Proposed Method

A. Key challenges and ideas

We have two key challenges in designing the proposed method.

The first challenge is to separate the network configuration into network topology graph and generalized configuration, as explained in Section I. This separation is intended for easier changes of network topology or configuration. With proper separation, we can change network topologies by modifying the network topology using only topology data, and change detailed settings common to multiple devices by modifying the generalized configuration. Our key idea for the first challenge is to use config templates. Configuration templates have an advantage over existing approaches in terms of versatility; They can be used to generate a wide range of configuration settings without relying on network protocols or applications.

The other challenge is to separate the config templates and their parameter data models. As explained in Figure 1, we conventionally need to use control syntax macros to template network configurations in existing platforms. This means that the config templates and the data models are not sufficiently separated to avoid embedding procedures that refer to the architecture of the data models in the config templates. In other words, the config templates depend on the dynamic parameter reference with the embedded procedure. To make the parameter reference deterministic, we need to rethink the design of the data model architecture for templating network configurations.

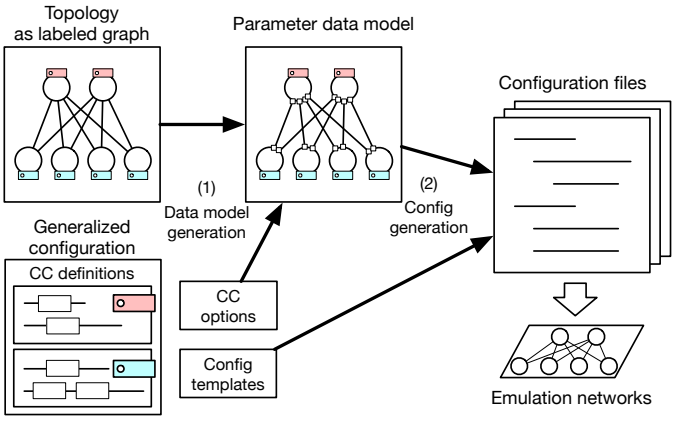


Fig. 2: Overview of the proposed method.

In summary, our key idea is twofold: (1) Template-based approach for separating network topology and configuration. (2) New parameter data model architecture to separate config templates and data models.

B. System overview

Figure 2 shows an overview of the proposed method. There are two inputs for the proposed method: network topology graph and generalized configuration. The network topology graph represents the relationships between network components (e.g., an interface of node A and an interface of node B are connected by a link). The graph basically describes Layer 2 level connection links, but it can also include logical relationships with different levels of abstraction, such as tunneling and overlay networks.

In addition, a network topology graph is a labeled graph to represent the role of each component in the network configuration with labels. The definitions of the labels are described as the generalized configuration, the other input. Generalized configurations include settings and config templates, which are required for generating parameter data model and configuration files.

There are two steps to automatically generate configuration files: data model generation and config generation as shown in Figure 2. The data model generation part constructs a parameter data model containing parameters to be embedded in the config templates. The config generation part generates config blocks with the config template blocks and the parameter data model, and output configuration files by merging the config blocks.

C. Data model

1) Requirements: Before explaining the steps to generate configuration files, we explain the design of the proposed data models.

There are two requirements for the parameter data model that allows deterministic parameter references. First, the data

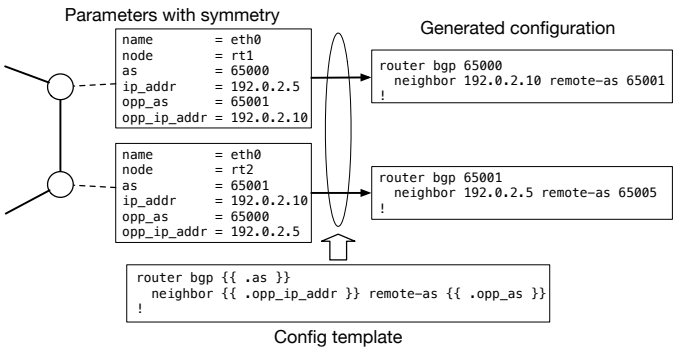


Fig. 3: Parameter symmetry required for deterministic reference in templating.

model needs to satisfy the symmetry of the parameters. Figure 3 shows an example of config generation with a template and two interfaces. The config generation is achieved without control syntax macros because the parameters satisfy a symmetry (i.e., same locations with different parameters for each interface. The conventional data model has a limitation in satisfying this symmetry; It has a top-down structure starting from the entire network or a device, and the network components are organized based on the ownership. In the model, the granularity of symmetry is limited to large objects such as entire networks or devices.

To satisfy this parameter symmetry, the config templates should not be prepared per device as in the conventional tools, but on a per object basis, such as the interfaces in Figure 3 as the unit of parameter symmetry. We call the config templates with finer granularity as config template blocks, and the configuration settings generated by the config blocks.

Second, the data model needs to completely cover the parameters required for templating. In network configuration, devices often need to use parameters from different devices, for example, to specify network connections and routes. The parameters must always be included in the symmetric data model for each network component.

We generate a parameter namespace for each network component to satisfy both the symmetry and completeness.

2) Design: We start with the design of components in the network data model. We call the network components ”objects”, and abstract them using a two-level object-oriented paradigm. One level is Architecture Class (AC), which represents the feature of components on the network topology. For example, ACs include entire networks, nodes (i.e., devices), interfaces, and connections (links). They have role-independent ownerships, and they form a conventional top-down data model as shown in Figure 4. The other level is Configuration Class (CC), which defines the role and behavior of network components. Each object belonging to the same CC generates a config block based on the same config templates. This means that the object needs to have a parameter namespace that contains the required parameters for templating.

The ACs are defined in the platform (or additional modules), and the CCs can be defined by the users manually. An object belongs to only one AC, and it can belong to multiple CCs. AC and CC correspond to the two inputs of the system: the network topology graph represents at the AC level, and the generalized configurations as the definitions of graph labels correspond to the CC definition by the users.

The parameter data model for templating consists of the objects represented by some ACs and CCs, as shown in Figure 4. The basic architecture of the data model is the same as the conventional top-down model based on the AC-level ownerships. This is because the top-down model is useful for aggregating configurations or calculating the relationships of the objects. Each object in the data model has a parameter namespace. The parameter namespace is based on a CC-level model, which is a bottom-up model for representing object relationships. The CC-level model is converted to a namespace so that it satisfies the symmetry of the parameters.

3) Iteration: Since the proposed method does not use control syntax macros, we need an alternative method for them. As for the “if” syntax, it can basically be replaced by giving options to the CCs to control whether the corresponding config template block is generated or not. On the other hand, for the “for” syntax, the proposed method needs to provide an alternative configuration iteration method. In the above model, we can aggregate multiple config blocks based on the top-down AC model of ownership. However, we cannot describe iterative configuration with the model alone when the target objects are based on relative relationships such as Layer 3 adjacency.

As an alternative method, the proposed method has a feature of referential child objects. It generates child objects for the number of targets when generating config blocks for multiple targets, excluding those based on the ownership relationships already including in the top-down model. The namespaces of these generated child objects are based on the parent object, and additionally have the parameters of the existing corresponding objects. To avoid namespace bloat, the proposed method generates the child objects only for the parent objects that specify as such in their CCs. The config blocks of these parent objects will include the config blocks of their child objects. With this feature, we can generate iterative configurations of multiple relative objects without control syntax macros.

In this paper, we refer to the ACs of objects specified in the input topology graphs as substantial ACs, and those generated as child objects for iterative config block generation as referential ACs. The examples of these ACs are demonstrated in Subsection IV-C.

4) Layering: To describe complicated network structures, we introduce the idea of protocol layers into the proposed method. The layers are not specified by separating the input graphs, but by indirectly specifying the graph labels (i.e., CCs) corresponding to the layers. This design is effective in reducing the number of graphs that the users need to edit for topology changes. If there are multiple input graphs per layer, the users need to edit multiple or all the graphs when changing the topology, leading to problems of unintentionally omitting changes (Figure 5a). In comparison, if the input is a labeled graph, the users can change the topology by modifying only one labeled graph (Figure 5b). For example, the users

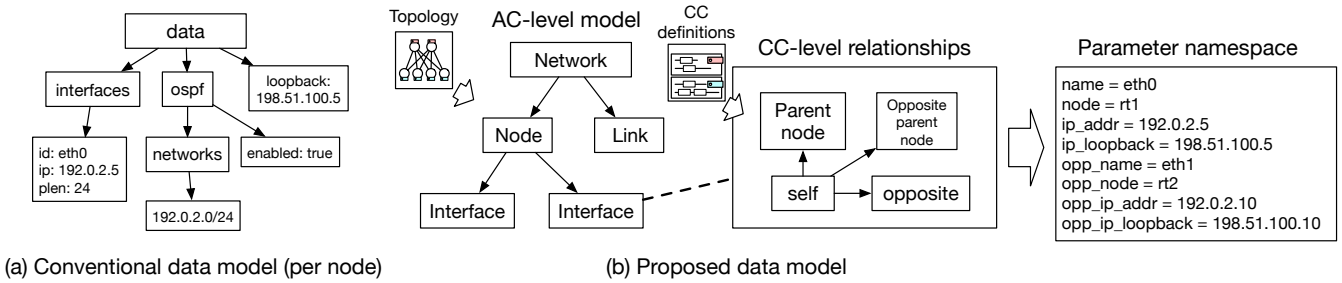


Fig. 4: Comparison of parameter data models.

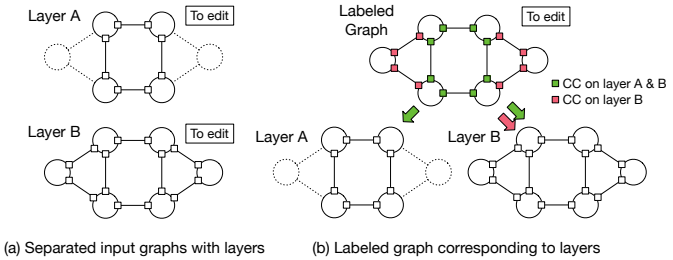


Fig. 5: Comparison of layering approaches for the network topology graphs.

can easily add a device that has roles on multiple layers by imitating the descriptions for similar devices with the same label in the graph input.

The layers are used as a flags and used in the definition of CCs. They separate the calculation of object relationships at the CC level. This feature allows us to describe complicated networks such as IPv4-IPv6 dualstack or overlay protocols (e.g., Enable multiplexed IP address assignment by separately calculating the Layer 3 connection relationship for IPv4 and IPv6).

D. Generation process

1) Data model generation: Here, we explain the procedure to generate the parameter data model based on the input, the network topology graph and generalized configurations. The system first creates a set of objects at the AC level of abstraction from the network topology graph and constructs a top-down model of the objects. For example, we first create the root object of the entire network. It then adds node objects corresponding to graph nodes, and connections between node objects corresponding to graph edges. For each connection, it also adds interface objects to the both end nodes of the corresponding connection. It also performs automatic naming for nodes and interfaces that do not have names specified in the graph.

Next, it calculates the relationships between the objects. The calculation is based on the top-down model and its CC-level behavior. An example is the Layer 3 level subnet separation. It can be simulated based on the Layer 2 connectivity and IP awareness of interfaces (i.e., an interface is a bridge or a router interface with an IP address), where the process is explained in the previous work [10]. If the CCs have layer flags, the calculation is separated for each layer. The calculated relationships are distinguished from the ownership relationships that comprise the top-down model, and are kept as the references to other objects or additional objects of referential ACs in the model.

After the calculation, the system constructs parameter namespaces for the objects. A parameter namespace is initialized with the parameters in the object itself. It then adds a set of relative parameters (i.e., parameters in related objects) from the other initialized namespaces. The relative parameters are based on the object relationships obtained from the above processes, such as parent-child relationships, opposing interfaces, or Layer 3 adjacency, to satisfy the coverage of parameters. The relative parameters are also intended to satisfy the symmetry of parameters in namespaces using the relative names starting from the object. Therefore, the namespaces in the parameter data models will satisfy both the symmetry and coverage of parameters.

2) Configuration generation: The proposed system finally generates configuration files based on the config templates specified in the CCs and the parameter data models. This step uses existing template engines, such as Jinja [32], and it uses only parameter embedding features without control syntax macros (However, this does not preclude the use of macros in the future). The config blocks corresponding to the objects are generated in the order from bottom to top in the top-down model. This is because of the nested templates: config templates that embed config blocks of child objects as parameters. The config blocks should have inclusion relationships corresponding to object ownership, such as the interface-related configuration included in the node configuration.

There are multiple types of configuration files with different setting scope: configuration files for the while network such as network emulation configuration files, and configuration files for each node such as application configuration files. By manipulating the inclusion relationships of config blocks, the system can generate any of these configuration files.

IV. Implementation

A. Design principles

Based on the proposed method, we implemented dot2net, a configuration platform for emulation networks. Dot2net is implemented in the Go language and is publicly available as an open source software [33]. At present, dot2net is implemented for use in conjunction with existing emulation network

platforms such as Containerlab [9] and TiNET [23]. We have established the following design principles on dot2net.

Declarative description style: Dot2net uses a declarative style of description rather than a procedural style because it focuses on what to configure rather than how to configure [34].

Minimal manual parameter assignment: As explained in Subsection III-D1, dot2net can automatically assign parameters to the objects during the parameter model generation process. It supports automatic assignment of parameters such as IP addresses, numeric values or names containing them. In addition, we can also specify parameters manually by describing them as labels in the input graph. The automatic parameter assignment of dot2net can take the manually specified parameters into account by avoiding parameter duplication and competition. With this mechanism, dot2net can generate configuration files with minimal manual parameter assignments.

Modular extension of data models: We will make dot2net modular design in terms of extending data models for advanced network protocols or configurations. These modules will extend AC definitions (including relationships to other ACs), the calculation process of object relationships and automatic parameter assignments, and the definitions of object relationships to relative namespaces. Currently, dot2net does not satisfy this principle, but it will be supported in the near future.

B. Graph labels

There are several types of graph labels. The labels can be specified in combination in multiple numbers and types for a single object. The most common is the class label, which specifies the CCs that correspond to the objects (Subsection III-C2). Dot2net uses the names of the CCs as class labels.

In addition, several label formats are provided for manual parameter assignment and relative parameter referencing. For manual parameter assignment, we can use the value label. It specifies the parameter of the target object, described such as `[ip_addr=192.0.2.5]`. In addition to manually specifying the parameters to be automatically assigned in advance, it can also extend the namespace by specifying unique parameter names. This label allows us to intuitively manage the manual parameters on the topology graph.

For relative parameter referencing, there are two types of labels, place label and meta value label. Place label defines object names that can be relatively referenced from any object, which expands the namespace from the perspective of the relative relationships of objects. For example, by assigning a label of `[@n2]` to a node, the node's `ip_addr` parameter of the node can be referenced by any device on the network using a placeholder such as {{`.n2_ip_addr`}}. Meta value label is a kind of alias for the place labels. For example, if there is a placeholder {{`.target_name`}} in a config template block, it refers to the parameter corresponding to {{`.n2_name`}} in the node specified by `[@target=n2]` and {{`.n3_name`}} in the node specified by `[@target=n3]`. Since the extension of the relative relationships of objects usually depends on the network topology, the meta value label is useful for controlling the extension in the topology graph rather than in the config templates.

TABLE I: Dot2net standard architecture classes (ACs)

Name	Configuration	Owner	Category
Network	Network	-	Substantial
Node	Node	Network	Substantial
Interface	Interface	Node	Substantial
Connection	Interface	Network	Substantial
Group	Group	Network	Substantial
Neighbor	Neighbor	Interface	Referential
Member	Member	any[2]	Referential

C. Architecture classes

Table I shows the list of standard ACs in dot2net. As explained in Subsection III-C3, there are two types of ACs: substantial ACs and referential ACs. Substantial ACs describe the behavior of objects corresponding to the components of the input graph. These substantial ACs form a top-down model with Network, corresponding to the entire network, at the top (Figure 4). Group is represented as clusters in the topology graphs and is used, for example, for areas or ASes. Referential ACs have no corresponding components in the graph, but are generated by CCs to avoid using "for" macros to generate config blocks for multiple target objects. Neighbor generates config blocks for multiple Layer 3 adjacent interfaces, and Member generates config blocks for all objects belonging to the same specified CC.

Note that Connection is a special AC. In network configuration, we usually describe config blocks not for connection links, but for the both ends of interfaces. In dot2net, Connection objects are used as a link during the calculation of relationships, but the parameter namespaces and config templates are used to generate config blocks of the both ends of interfaces.

V. Evaluation

We demonstrate that dot2net based on the proposed method is scalable, simple, and efficient for emulation network configurations.

Scalability: We measure the processing time and memory consumption for generating configuration files of emulation networks. We compare them by changing the size of two target network topologies in order to demonstrate the effectiveness in generating configuration files for large-scale emulation networks.

Simplicity: We compare the input file size for emulation networks with and without dot2net. The target emulation networks are from multiple test networks for FRRouting (FRR) [35], an open source router software.

Efficacy: We confirm the efficacy of dot2net in changing emulation network topologies through user experiments. The experiments are intended to reveal the work time and number of builds by the users in correctly modifying the configuration files.

TABLE II: Topologies of large-scale clos networks.

Name	Nodes				Links
	Tier 1	Tier 2	Tier 3	Total	
Clos100	64	32	4	100	2,176
Clos200	128	64	8	200	8,704
Clos300	192	96	12	300	19,584
Clos500	320	160	20	500	54,400
Clos1000	640	320	40	1,000	217,600
Clos2000	1,280	640	80	2,000	870,400
Clos3000	1,920	960	120	3,000	1,958,400
Clos5000	3,200	1,600	200	5,000	5,440,000

TABLE III: Network topology scenarios for evaluation.

Network scenario		Base topology	
Source	Name	Nodes	Links
FRR topotests	rip_topo1	9 (+1) [3]	8 (+1) [3]
FRR topotests	ospf_topo1	10	9
FRR topotests	ospf6_topo1	10 (+3) [3]	9 (+4) [3]
FRR topotests	bgp_features	15	15
FRR topotests	bgp_evpn_vxlan_topo1	5	9
TiNET examples	basic_clos	14	16

A. Scalability evaluation

We use the following two basic network topologies to evaluate the scalability of dot2net. For each basic topology, we change the number of routers (n) to scale the topology. (1) Ring topology (Ring): We connect n routers in a circle. The number of connection links is equal to n. We performed the measurements while changing n from 100 to 1,000,000. (2) 3-tier clos topology (Clos): We divide the n routers into three stages. Each pair of adjacent stages (e.g., Tier 1 and Tier 2) forms a complete bipartite graph with the connection links. We used eight clos topologies with different n as shown in Table II. The scenarios are described to deploy the emulation networks with Containerlab.

These two topologies differ mainly in the number of contained components; The Ring topology has relatively more nodes, while the Clos topology has more connection links. By using both of them in the scalability evaluation, we can confirm the bottlenecks in scaling the target topologies. For each topology, we generate a set of configuration files for BGP-based clos networks using IPv6 and 4-byte AS numbers for scaling.

In the following experiments, we use an Ubuntu 22.04 server (x86_64) equipped with an AMD Ryzen 7 7700X and 64 GB of memory.

Figure 6 shows the measurement results. The x-axes show the number of nodes for Ring and the number of connections for Clos, where the number of nodes for Clos is also shown next to the measurement point. In the Ring topologies, we can see that the processing time (Figure 6a) and memory consumption (Figure 6b) are linearly dependent on the number of nodes. The change is gradual when the number of nodes is small because the general processing, which is independent of the size of the topology, has a relatively large impact. In Clos topologies, the processing time (Figure 6c) and memory consumption (Figure 6d) depend linearly on the number of connections. This is because dot2net includes operations that depend on the number of connections, such as searching for subnets and generating a config block for each interface.

The processing time of dot2net is small enough for the use in automating emulation network configuration. For example, Containerlab (version 0.55.1) needs 30 seconds to build the Ring topology network with $n = 100$ and 18 seconds to delete it on the same measurement environment. In the case of the Clos topology network with $n = 100$ (Clos100), dot2net needs 383 seconds to build it and 21 seconds to delete it. In contrast, the processing time to generate the configuration files is less than 0.1 seconds for both of them. When we combine dot2net and Containerlab for automating emulation network deployment, the processing time of dot2net is sufficiently small compared to that for the build.

In a general environment for network emulations, the bottleneck would be memory consumption. In both basic topologies, the memory consumption approached the limit of the measurement environment's memory at a processing time of about 100 seconds. The increase in memory consumption is mainly due to the expansion of the target configuration files and the increase in the number of namespaces associated with the increase in the number of objects (i.e., interfaces).

In summary, dot2net can generate configuration files for emulation networks with millions of devices or links in a small processing time and with a reasonable memory consumption. By comparison, existing frameworks used smaller networks for evaluation. For example, AutoNetKit [7] uses up to 100 routers, and Propane [24] evaluates configuration generation using several thousand routers. This demonstrates that dot2net has sufficient scalability, surpassing existing methods, for the large-scale network emulation.

B. Simplicity evaluation

We use six network topology scenarios in dot2net for the simplicity evaluation. Table III lists the scenarios and their scale. Five of them are from FRR topotests [4], a suite of topology tests on mininet. The tests are described in Python scripts, so we generate equivalent configuration files for Containerlab and TiNET using dot2net. There is another scenario "basic_clos" from TiNET examples [5], which is the basic scenario used in the scalability evaluation.

We use dot2net version v0.3.5 for the following evaluation. The scenarios are available in public as a part of examples in dot2net [33].

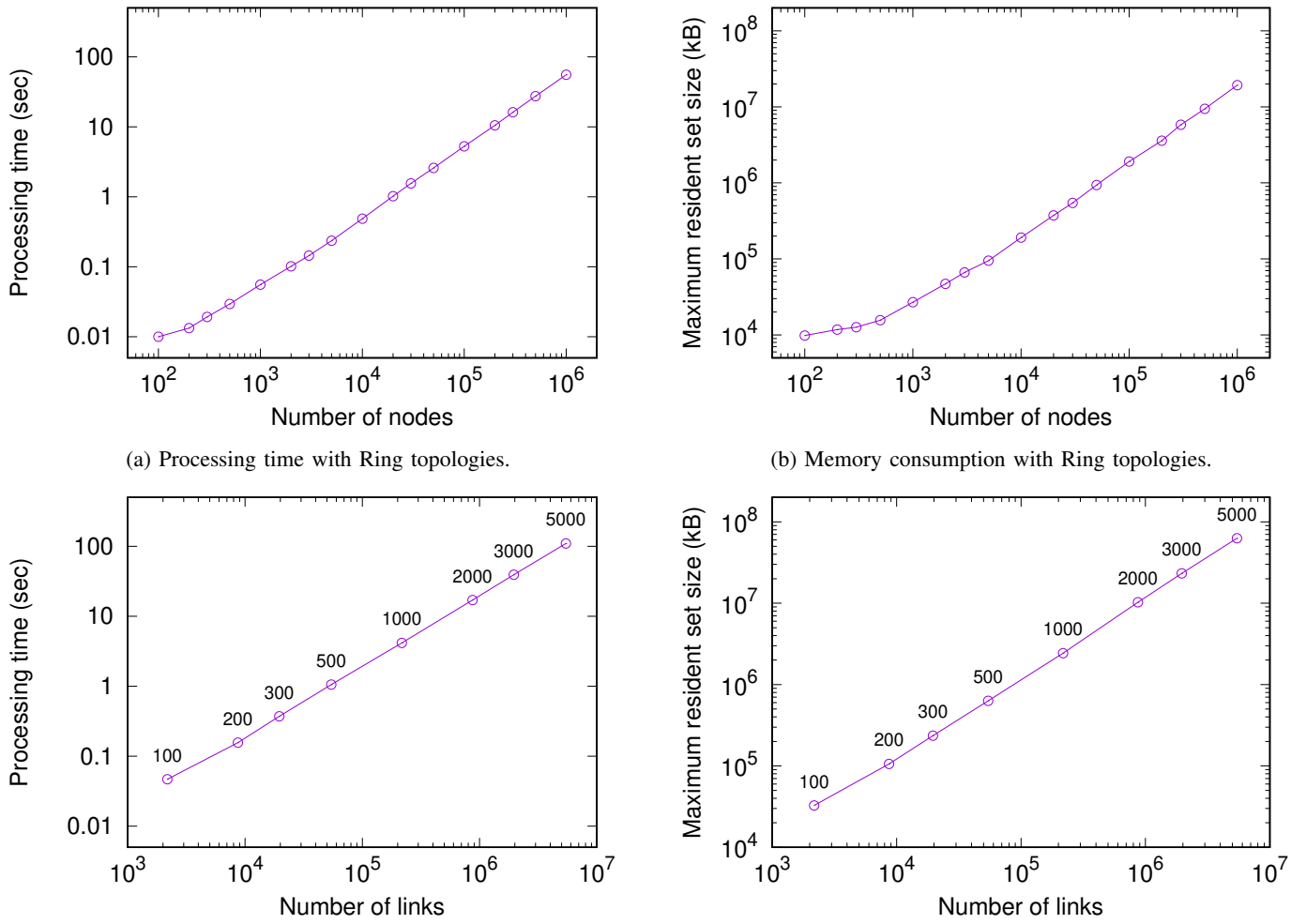
Table IV lists the comparison of input file size to configure the six scenarios with or without dot2net. In this table, we generate config files for Containerlab and TiNET with dot2net and compare their total file size (bytes) with the dot2net input. Dot2net reduces the file size by half to a third in every scenario. This is due to the reduction of duplicate descriptions by templates and the simplified template description without macros.

[2]Member can be a child of any objects.

[3]The number in parentheses indicates the number of virtual objects that do not generate configuration settings but are used only for parameter assignment and calculation for relationships.

[4]https://github.com/FRRouting/frr/tree/master/tests/topotests

[5]https://github.com/tinynetwork/tinet/tree/master/examples



(a) Processing time with Ring topologies.

(b) Memory consumption with Ring topologies.

(c) Processing time with Clos topologies.

(d) Memory consumption with Clos topologies.

Fig. 6: Scalability evaluation of dot2net.

In addition, we extend the network topologies (shown in Table III) by adding one router and one connection ("+1" in Table IV) to the simplicity of dot2net in extending network topologies. In dot2net, expanding the topology only increases the size of the topology (DOT). The difference in config file size is less than 10% in every scenario. Therefore, we can confirm that dot2net effectively shrinks the required changes for emulation network configurations.

C. Efficacy evaluation

We conducted user experiments to evaluate the efficacy in changing the topologies of emulation networks. For the experiment, the users need to understand how to describe the router configurations, test the network, and debug the behavior, which prevents us from collecting a sufficient number of target users locally. Therefore, we conducted online experiments with students as the users.

The target users are undergraduate and graduate students, ranging from 4th year undergraduates to 3rd year Ph.D. students, with basic knowledge of network configurations [6]. The target users joined the experiments remotely on demand, used their own computers to work on the given tasks, and reported the results via email. We have obtained informed consent from the target users to participate in this experiment and to use the data obtained. This experiment was reviewed and approved by the Research Ethics Committee of the Graduate School of Environmental, Life, Natural Science and Technology, Okayama University (No. 2023-10).

The experiment consists of four assignments to try configuration changes of emulation networks. The participants are divided into two groups, Group A and Group B, in advance. They first read a tutorial on configuration changes and emulation network deployment using dot2net and Containerlab. After that, they try out the assignments using the method specified for each group, Containerlab only (Clab) or dot2net with Containerlab (dot2net). In order to reduce the impact of learning effects during the assignments, the method specified for each group is alternated for each assignment, with a total of two questions for each method. Each assignment has a set of

[6]Students from Okayama Univ., Hiroshima Univ., Sokendai, Grenoble INP, and Sorbonne Univ.

TABLE IV: Comparison of configuration file size (bytes).

Scenario	(Expansion)	dot2net Topology	(diff)	dot2net Config	(diff)	dot2net Total	(diff)	Containerlab	(diff)	TiNET	(diff)
rip_topo1		536		3,221		3,757		6,650		7,786	
rip_topo1	(+1)	591	(+55)	3,221	(±0)	3,812	(+55)	8,249	(+1,599)	9,665	(+1,879)
ospf_topo1		539		3,097		3,636		10,611		12,218	
ospf_topo1	(+1)	582	(+43)	3,097	(±0)	3,679	(+43)	12,538	(+1,927)	14,475	(+2,257)
ospf6_topo1		982		2,933		3,915		9,392		10,853	
ospf6_topo1	(+1)	1,026	(+44)	2,933	(±0)	3,959	(+44)	11,082	(+1,690)	12,833	(+1,980)
bgp_features		1,037		6,472		7,509		17,889		20,205	
bgp_features	(+1)	1,122	(+85)	6,472	(±0)	7,594	(+85)	20,523	(+2,634)	23,181	(+2,976)
bgp_evpn_vxlan_topo1		777		4,220		4,997		11,662		13,513	
bgp_evpn_vxlan_topo1	(+1)	862	(+85)	4,220	(±0)	5,082	(+85)	12,605	(+943)	14,848	(+1,335)
basic_clos		858		1,396		2,254		9,300		10,275	
basic_clos	(+1)	909	(+51)	1,396	(±0)	2,305	(+51)	9,932	(+632)	10,970	(+695)

TABLE V: Average working time and number of builds in the user experiments (Group A in purple and Group B in green).

Assignments	Containerlab only Time	Containerlab only #Builds	dot2net + Containerlab Time	dot2net + Containerlab #Builds
Assignment 1	25.9	2.8	8.6	1.7
Assignment 2	28.6	3.0	6.8	1.1
Assignment 3	42.1	4.1	21.9	2.4
Assignment 4	27.7	2.3	14.6	1.6

basic network configuration files corresponding to the original configuration. They make specified changes (e.g., add a router and a switch to the specified locations) to the basic network. They finally try to build the network with Containerlab and confirm that the tests specified in the assignment pass. During the assignments, users are not allowed to consult with other users, except for tutorials. The content of the assignments is published on GitHub [7].

For each assignment, users report the work time and the number of builds before completing the assignment. The number of builds is equivalent to the number of failures plus one to describe correct settings, which allows us to assess the likelihood of configuration errors. In addition, to reduce the burden on the participants, they can leave the task before completing it if it takes more than 60 minutes (we will count this as 60 minutes in the compiled results).

Table V shows the average work time and number of builds for each assignment. Note that the specified method is different for each group in order to reduce the impact of learning effects, so the populations of each item are not consistent (instead, the items are colored according to the groups). Based on the comparison of the two methods, it can be confirmed that the working time and the number of builds have been greatly reduced to about 1/3 to 1/2 by using dot2net in all of the assignments. From this, we can say that dot2net greatly improves the efficiency of topology changes in emulation network configurations.

Figure 7 shows the distribution of work time and number of builds for each participant. In all assignments, the work time for Clab varies greatly depending on the skill of the participant. On the other hand, for dot2net, the variation between participants is small, which means that even users with less ability can efficiently change the configuration files. There are outliers in all of the assignments (especially on Assignment 3), regardless of the method used, which is because the work time includes not only the configuration change, but also the build and test, and there are cases where the assignments cannot be completed due to problems unrelated to the configuration changes (e.g., Some students had difficulty appropriately testing Assignment 3).

Looking at the individual assignment, there is more variation in Assignment 1 (Figure 7a) for both methods than for the other tasks. This is because the participants approach the tutorial differently, and there are differences in the amount of time to learn the usage of the required tools or to download the container image. In contrast, for the other assignments, the processing time and the number of builds for dot2net are relatively stable.

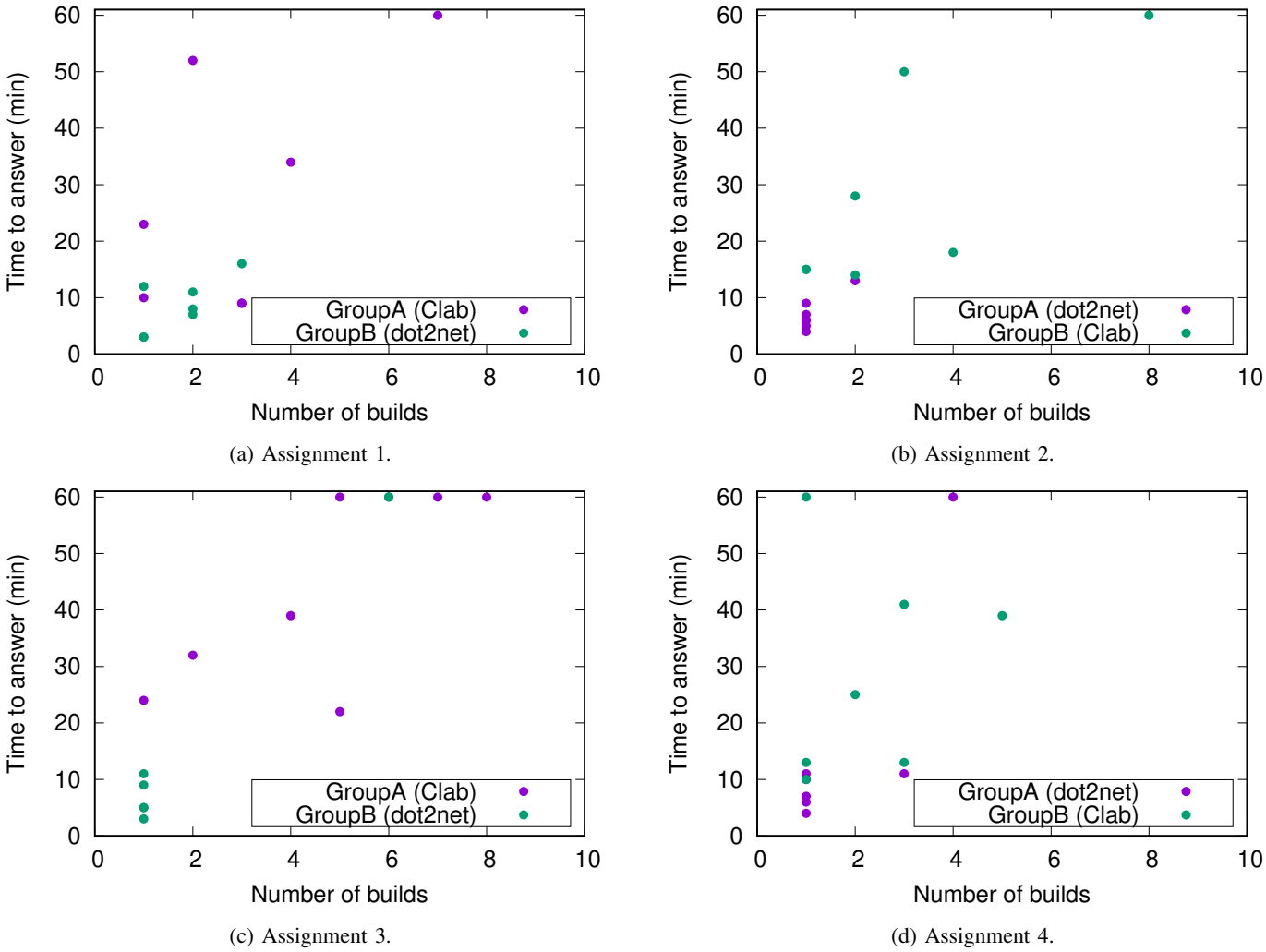
[7] https://github.com/cpflat/dot2net-evaluation/

VI. Discussion

A. Robustness of description

We demonstrate the robustness of the proposed method in describing network configurations through the description of multiple test networks performed to evaluate the simplicity of dot2net (Subsection V-B). In terms of network protocols, the CC layering feature of dot2net (Subsection III-D1) allows us to successfully describe some advanced network protocols. For example, the "ospf_topo1" scenario describes a network routed with OSPFv2 for IPv4 and OSPFv3 for IPv6 in parallel. The IP dualstack for this scenario is enabled with the CC layering explained in Subsection III-D1. The "bgp_evpn_vxlan_topo1" scenario relies on VXLAN, one of the overlay networking technologies. It can be described with two layers, one for the provider network (with tunneling) and one for the customer network (bridged with VXLAN).

In addition, dot2net's parameter assignment is flexible and efficient, with the automated parameter assignment feature compatible with the standard ACs (Subsection IV-A). The "bgp_features" scenario is a BGP-based network that includes non-BGP routers (routed with OSPF) and L2 switches. The BGP neighbor router parameters are automatically assigned with the Neighbor objects explained in Subsection IV-C (i.e., add BGP neighbor settings for each Layer 3 neighbor router with the corresponding label). The "rip_topo1" and "ospf6_topo1" scenarios involve mixed routing policies: dynamic routing and static routing. The static routing policies can be described and modified in the topology graph using place labels and meta value labels to relatively specify the



(a) Assignment 1.

(b) Assignment 2.

(c) Assignment 3.

(d) Assignment 4.

Fig. 7: Distribution of response status of participants in the user experiment.

destination nodes. If there are out-of-emulation (stub) nodes and subnets in the static routing policies, they can be described using manual parameter assignment with value labels or using virtual objects (objects that do not generate configuration files). By distinguishing and reconciling these static routing policies with dynamic routing policies using CCs, we can describe these scenarios in a consistent and natural way.

B. Application

Although this paper focuses on the emulation network configurations, the proposed method can be useful in other network applications because it is independent of the nature of emulation networks. Since it also does not depend on any specific protocol, the proposed method can generate wide variety of output file formats. It is expected to contribute to the automation of network configuration in conjunction with remote configuration (e.g., netconf [36]) or configuration distribution methods (e.g., ansible [37]).

We believe that network configuration automation based on the proposed method will increase the efficiency of network configurations, especially when configuration changes are made frequently.

C. Limitation

Limitations in proposed method: The proposed method does not efficiently describe any configurations that can be expressed in the conventional approach using control syntax macros. Instead of “for” syntax, the proposed method needs to generate multiple objects of referential ACs. In a scenario where the “for” syntax is nested in the conventional templates, the referential AC also becomes multi-layered with the proposed method, and the class structure becomes more complex instead of a template. In this case, even if the configuration change is easy, the initial class design is not easy for the users.

Also, instead of “if” syntax, the proposed method depends on the flags in CC definitions. The flag items are built in the platform or additional modules, so users cannot freely define conditions for items that are not provided in them.

Limitations in implementation: Dot2net depends on DOT as the input topology graph. This means that it is difficult to efficiently represent information that cannot be expressed by DOT. An example is source routing. Since DOT cannot represent paths in the graphs, the entire routing paths cannot be managed on the topology graphs and must be configured as

the behavior of individual routers. For such issues, we need to provide extension modules in the future, e.g., using external files of individual format.

Limitations in practical use: The proposed method is effective at changing the network topologies, as demonstrated in Subsection V-B and Subsection V-C, but it requires the same or more effort than normal configuration when creating label definitions during initial network construction. As explained in Subsection II-B, template generation may be automated in the future using configuration synthesis technology. For instance, Liu et al. [38] use machine learning-based configuration synthesis to generate config templates. Combining such technology with topology-driven configuration could facilitate the construction and operation of emulation networks in the future.

VII. Conclusion

In this paper, we proposed a scalable, simple, and effective way to configure emulation networks using config templates. The proposed method is based on Topology-driven configuration, the idea of separating network configuration into topology and generalized configuration. In the method, we proposed a new data model of network parameters to achieve a simple config template description with template blocks of appropriate granularity and without using control syntax macros. Based on the proposed method, we developed dot2net, a configuration platform for emulation networks. Dot2net allows us to automate the process from configuration to deployment of Docker-based emulation networks in conjunction with network emulation platforms such as Containerlab.

We evaluated the proposed method using dot2net from three perspectives, scalability, simplicity, and efficacy. First, for scalability, we measured the processing time and memory consumption for large-scale emulation networks and showed that dot2net can generate configuration files in a short processing time compared to the build step with a generally acceptable memory consumption. Next, for simplicity, we confirmed that dot2net can change the network topologies of six test network scenarios with a change of less than 10% of the file size than without dot2net. Finally, for efficacy, we demonstrated that users can change emulation network topologies in about 1/2 to 1/3 of the work time and with fewer configuration errors than without dot2net.

In future work, we will investigate the potential applications of dot2net, including those other than emulation networks. We will also continue to enhance the required functionality for the applications as dot2net modules.

Acknowledgements

This work is partially supported by JSPS KAKENHI Grant Number JP25K15079 and JP22K17886.

References

[1] A. Fuller, Z. Fan, C. Day, and C. Barlow, "Digital twin: Enabling technologies, challenges and open research," *IEEE Access*, vol. 8, pp. 108 952–108 971, 2020.

[2] H. X. Nguyen, R. Trestian, D. To, and M. Tatipamula, "Digital twin for 5G and beyond," *IEEE Communications Magazine*, vol. 59, no. 2, pp. 10–15, 2021.

[3] R. Alimi, Y. Wang, and Y. R. Yang, "Shadow configuration as a network management primitive," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication (SIGCOMM'08)*, 2008, pp. 111–122.

[4] H. H. Liu, Y. Zhu, J. Padhye, J. Cao, S. Tallapragada, N. P. Lopes, A. Rybalchenko, G. Lu, and L. Yuan, "CrystalNet: Faithfully Emulating Large Production Networks," in *Proceedings of the 26th ACM Symposium on Operating Systems Principles (SOSP'17)*, 2017, pp. 599–613.

[5] R. Emiliano and M. Antunes, "Automatic network configuration in virtualized environment using gns3," in *Proceedings of the 10th International Conference on Computer Science & Education (ICCSE'15)*, 2015, pp. 25–30.

[6] W. Enck, T. Moyer, P. McDaniel, S. Sen, P. Sebos, S. Spoerel, A. Greenberg, Y.-W. E. Sung, S. Rao, and W. Aiello, "Configuration management at massive scale: system design and experience," *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 3, pp. 323–335, 2009.

[7] H. Nguyen, M. Roughan, S. Knight, N. Falkner, O. Maennel, and R. Bush, "How to Build Complex, Large-Scale Emulated Networks," in *Proceedings of the International Conference on Testbeds and Research Infrastructures (TridentCom'11)*, vol. 46, 2011, pp. 1–16.

[8] S. Knight, H. Nguyen, O. Maennel, I. Phillips, N. Falkner, R. Bush, and M. Roughan, "An automated system for emulated network experimentation," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies (CoNEXT'13)*, 2013, pp. 235–246.

[9] Nokia, "Containerlab," https://containerlab.dev/, 2021.

[10] S. Kobayashi, R. Shiiba, R. Miura, S. Miwa, T. Miyachi, and K. Fukuda, "dot2net: A labeled graph approach for template-based configuration of emulation networks," in *Proceedings of the 19th International Conference on Network and Service Management (CNSM'23)*, 2023, pp. 319–327.

[11] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, "A general approach to network configuration verification," in *Proceedings of the ACM SIGCOMM 2017 Conference (SIGCOMM'17)*, 2017, pp. 155–168.

[12] D. Guo, S. Chen, K. Gao, Q. Xiang, Y. Zhang, and Y. R. Yang, "Flash: fast, consistent data plane verification for large-scale network settings," in *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM'21)*, 2022, pp. 314–335.

[13] Q. Zhang, K. K. W. Ng, C. W. Kazer, S. Yan, J. Sedoc, and V. Liu, "MimicNet: Fast Performance Estimates for Data Center Networks with Machine Learning," in *Proceedings of the ACM SIGCOMM 2021 Conference (SIGCOMM'21)*, 2021, pp. 287–304.

[14] H. H. Liu, Y. Zhu, J. Padhye, J. Cao, S. Tallapragada, N. P. Lopes, A. Rybalchenko, G. Lu, and L. Yuan, "CrystalNet: Faithfully Emulating Large Production Networks," in *Proceedings of the 26th ACM Symposium on Operating Systems Principles (SOSP'17)*, 2017, pp. 599–613.

[15] M. Landauer, F. Skopik, M. Wurzenberger, W. Hotwagner, and A. Rauber, "Have it your way: Generating customized log datasets with a model-driven simulation testbed," *IEEE Transactions on Reliability*, vol. 70, no. 1, pp. 402–415, 2021.

[16] C. Regal-Mezin, S. Kobayashi, and T. Yamauchi, "netroub: Towards an emulation platform for network trouble scenarios," in *Proceedings of the CoNEXT Student Workshop 2023 (CoNEXT-SW'23)*, 2023, p. 17–18.

[17] T. Holterbach, T. Bü, T. Rellstab, and L. Vanbever, "An Open Platform to Teach How the Internet Practically Works," *ACM SIGCOMM Computer Communication Review*, vol. 50, no. 2, pp. 45–52, May 2020.

[18] W. Du, H. Zeng, and K. Won, "SEED Emulator: An Internet Emulator for Research and Education," in *Proceedings of the 21th ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets'22)*, 2022, p. 7.

[19] M. Großmann and S. J. A. Schuberth, "Auto-Mininet : Assessing the Internet Topology Zoo in a Software-Defined Network Emulator," in *Proceedings of the GIITG-Workshop MMBNet 2013*, 2013, pp. 1–10.

[20] M. Pizzonia and M. Rimondini, "Netkit: network emulation for education," *Software: Practice and Experience*, vol. 46, no. 2, pp. 133–165, 2016.

[21] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets'10)*, 2010, pp. 1–6.

[22] G. Bonofiglio, V. Iovinella, G. Lospoto, and G. Di Battista, "Kathará: A container-based framework for implementing network function virtualization and software defined networks," in *Proceedings of IEEE/IFIP Network Operations and Management Symposium (NOMS'18)*, 2018, pp. 1–9.

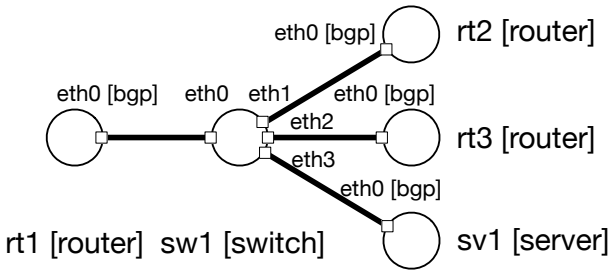


Fig. 8: A tiny example network topology that requires control syntax macros in existing approaches.

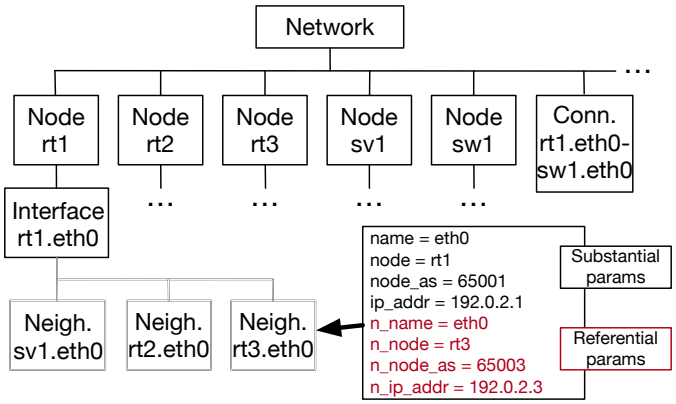


Fig. 9: A parameter data model for example network topology.

[23] tinynetwork, "tinet," https://github.com/tinynetwork/tinet/, 2019.

[24] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker, "Don't mind the gap: Bridging network-wide objectives and device-level configurations," in *Proceedings of the ACM SIGCOMM 2016 Conference (SIGCOMM'16)*, 2016, pp. 328–341.

[25] S. Ramanathan, Y. Zhang, M. Gawish, Y. Mundada, Z. Wang, S. Yun, E. Lippert, W. Taha, M. Yu, and J. Mirkovic, "Practical Intent-driven Routing Configuration Synthesis," in *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI'23)*, 2023, pp. 629–644.

[26] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker, "Network configuration synthesis with abstract topologies," *ACM SIGPLAN Notices*, vol. 52, no. 6, pp. 437–451, 2017.

[27] A. El-Hassany, P. Tsankov, L. Vanbever, and M. Vechev, "NetComplete: Practical Network-Wide configuration synthesis with autocompletion," in *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18)*, 2018, pp. 579–594.

[28] A. Abhashkumar, A. Gember-Jacobson, and A. Akella, "AED: Incrementally synthesizing policy-compliant and manageable configurations," in *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT'20)*, 2020, pp. 483–495.

[29] T. Schneider and L. Vanbever, "Snowcap: Synthesizing Network-Wide Configuration Updates," in *Proceedings of the ACM SIGCOMM 2021 Conference (SIGCOMM'21)*, 2021, pp. 33–49.

[30] Y. Dai, H. Zhang, J. Wang, J. Liao, and P. Zhang, "INCS: Intent-driven network-wide configuration synthesis based on deep reinforcement learning," *Computer Networks*, vol. 251, p. 110640, 2024.

[31] Z. Guo, F. Li, J. Shen, T. Xie, S. Jiang, and X. Wang, "Configreco: Network configuration recommendation with graph neural networks," *IEEE Network: The Magazine of Global Internetworking*, vol. 38, no. 1, p. 7–14, 2024.

[32] Pallets, "Jinja," https://palletsprojects.com/p/jinja/.

[33] S. Kobayashi, "dot2net," https://github.com/cpflat/dot2net/, 2023.

[34] T. Xu and Y. Zhou, "Systems approaches to tackling configuration errors: A survey," *ACM Computing Surveys*, vol. 47, no. 4, 2015.

[35] FRRouting Project, "Frrouting," https://frrouting.org/, 2017.

[36] T. Anderson, "Local-Use IPv4/IPv6 Translation Prefix," RFC 8215, Aug. 2017. [Online]. Available: https://rfc-editor.org/rfc/rfc8215.txt

[37] Ansible Inc., "Ansible," https://www.ansible.com/, 2012.

[38] J. Liu, L. Chen, D. Li, and Y. Miao, "CEGS: Configuration Example Generalizing Synthesizer," in *Proceedings of the 22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI'25)*, 2025, pp. 1327–1347.

Appendix A
Example to replace control syntax macros

We demonstrate how the proposed method can describe network configurations, where existing approaches require control syntax macros, such as "for" and "if", with referential ACs explained in Subsection III-C3. We use an example network topology shown in Figure 8. This network has three routers (rt1, rt2, and rt3) and a server (sv1), connected to a Layer 2 switch (sw1). In this network, the interface eth0 of router rt1 (henceforce rt1.eth0) has three Layer 3 neighboring interfaces: rt2.eth0, rt3.eth0, and sv1.eth0. When we want to describe a config block about the BGP neighbors (Layer 3 adjacent interfaces of BGP routers) of rt1.eth0, we need to generate a config block using the parameters of each BGP neighbor (e.g., AS numbers) with the given template. Conventional models require control syntax macros, "for" to iterate Layer-3 neighboring interfaces and "if" to check whether the interface is of a BGP router or not.

The proposed method can generate this config block using Neighbor objects belonging to one of the referential ACs. Figure 9 shows an abstracted parameter data model generated by the proposed method. In the model, the substantial ACs (Network, Node, Interface, and Connection) form a top-down model based on the ownership described in the input graph (Figure 8, excluding automatically named interface names). There are also Neighbor objects under each Interface object, corresponding to the Layer 3 neighbor interfaces. The namespaces of these Neighbor objects are based on the parameters of the parent object (substantial parameters) and additionally contain the parameters of the corresponding objects (referential parameters). Therefore, the template block used in a Neighbor object can use both the parameters of the parent object and the corresponding object.

The template of an object has several options, one of which is to output the config block only if the corresponding object (in this case the Interface corresponding to the Neighbor object) belongs to a specified CC. With this option, the proposed method can generate a set of config blocks only for the interfaces of BGP routers (equivalent to "if"). Also, by merging the generated config blocks of Neighbor objects on the parent Interface object, the proposed method can generate a config block including settings for all the BGP neighbors (equivalent to "for"). In this way, the proposed method realizes conditional branching of config block generation using the CC options and iteration of config blocks using the objects of Referential ACs.

Appendix B
Example of dot2net input

There are two dot2net input files for generating configuration files: a topology graph (DOT) and a label definition file (YAML). Figure 10 and Figure 11 demonstrate examples

```
digraph {
  r1[class="router"];
  r2[class="router"];
  r3[class="router"];

  r1->r2[dir=none];
  r2->r3[dir=none];
}
```

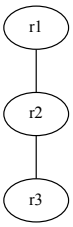


Fig. 10: Example topology graph (a DOT file and its visualization).

```
layer:
  - name: ip
    default_connect: true
    policy:
      - name: ip
        range: 10.0.0.0/16
        prefix: 24
      - name: lo
        type: loopback
        range: 10.0.255.0/24

nodeclass:
  - name: router
    primary: true
    tinet:
      image: quay.io/frrouting/frr:8.5.0
    clab:
      kind: linux
      image: quay.io/frrouting/frr:8.5.0
    policy: [lo]
    config:
      - file: daemons
        sourcefile: ./daemons
      - file: vtysh.conf
        sourcefile: ./vtysh.conf
      - file: frr.conf
        template:
          - "ip forwarding"
          - "!"
          - "router ospf"
          - " ospf router-id {{.ip_loopback}}"
          - "!"

interfaceclass:
  - name: default
    primary: true
    policy: [ip]
    config:
      - file: frr.conf
        template:
          - "interface {{.name}}"
          - " ip address {{.ip_addr}}/{{.ip_plen}}"
          - "!"
          - "router ospf"
          - " network {{.ip_net}} area 0"
          - "!"
```

Fig. 11: An excerpt of example label definitions (YAML).

```
r1 {{ .ip_loopback }} = 10.0.255.1
r1 {{ .name }} = r1
r1.net0 {{ .ip_addr }} = 10.0.0.1
r1.net0 {{ .ip_net }} = 10.0.0.0/24
r1.net0 {{ .ip_plen }} = 24
r1.net0 {{ .name }} = net0
r1.net0 {{ .node_ip_loopback }} = 10.0.255.1
r1.net0 {{ .node_name }} = r1
r1.net0 {{ .opp_ip_addr }} = 10.0.0.2
r1.net0 {{ .opp_ip_net }} = 10.0.0.0/24
r1.net0 {{ .opp_ip_plen }} = 24
r1.net0 {{ .opp_name }} = net0
r1.net0 {{ .opp_node_ip_loopback }} = 10.0.255.2
r1.net0 {{ .opp_node_name }} = r2
r2 {{ .ip_loopback }} = 10.0.255.2
r2 {{ .name }} = r2
r2.net0 {{ .ip_addr }} = 10.0.0.2
r2.net0 {{ .ip_net }} = 10.0.0.0/24
r2.net0 {{ .ip_plen }} = 24
r2.net0 {{ .name }} = net0
r2.net0 {{ .node_ip_loopback }} = 10.0.255.2
r2.net0 {{ .node_name }} = r2
r2.net0 {{ .opp_ip_addr }} = 10.0.0.1
r2.net0 {{ .opp_ip_net }} = 10.0.0.0/24
r2.net0 {{ .opp_ip_plen }} = 24
r2.net0 {{ .opp_name }} = net0
r2.net0 {{ .opp_node_ip_loopback }} = 10.0.255.1
r2.net0 {{ .opp_node_name }} = r1
...
```

Fig. 12: An excerpt of parameters in example namespaces. A line consists of target object, parameter name (replacer), and the corresponding parameter value.

of each file (dot2net version v0.3.5). Note that the network topology and configurations assumed here differ from those used in Appendix A.

Figure 10 shows three defined nodes, r1, r2, and r3, with the class attributes “router”, meaning these nodes belong to the Node CC named “router”. If required, we can also specify multiple labels by separating them with “;” in the class attribute strings. Also, the two connections do not have any class label specification. If defined, the interfaces belong to “default” classes. The two classes, “router” Node CC and “default” Interface CC are included in the label definitions in Figure 11.

This label definition also defines one layer and two IP address auto-assignment policies. Each assignment policy is specified for each CC, which means that IP addresses are assigned to objects with those labels according to their respective policies.

Figure 12 shows a part of the namespaces generated by dot2net based on the input files. The namespace includes automatically assigned parameters and relative names for referencing them from different objects. Using these names in the template allows us to embed parameters of different objects into the configuration settings of an object (although this is not included in the simple example template in Figure 11). In this way, the final configuration settings are generated by embedding the namespace parameters into the templates in the label definitions.

Satoru Kobayashi is an Assistant Professor at Faculty of Environmental, Life, Natural Science and Technology, Okayama University. He received his Ph.D. in Information Science and Technology from the University of Tokyo in 2018. His research interests are network management and data mining.

Ryusei Shiiba is a Ph.D. candidate at Department of Informatics, School of Multidisciplinary Sciences, The Graduate University for Advanced Studies (SOKENDAI). His research interests are network verification and testing.

Shinsuke Miwa is a Chief Senior Researcher at StarBED Technology Center, Testbed Research, Development and Operations Laboratory, National Institute of Information and Communications Technology (NICT). He received his Ph.D in 1999 from Japan Advanced Institute of Science and Technology (JAIST). His research interests are network emulation and evaluation.

Toshiyuki Miyachi is a Chief Planning Manager at the Strategic Planning Office, Strategic Planning Department, National Institute of Information and Communications Technology (NICT). He received his Ph.D. in Information Science from the Japan Advanced Institute of Science and Technology (JAIST) in 2007. His research interests include network testbeds and large-scale network technology validation.

Kensuke Fukuda is a Professor at Information Systems Architecture Research Division, National Institute of Informatics and Department of Informatics, School of Multidisciplinary Sciences, The Graduate University for Advanced Studies (SOKENDAI). He received his Ph.D. in 1999 from Keio University. His research interests span Internet traffic analyses, anomaly detection, modeling networks and QoS over the Internet.