NetCR: Knowledge Graph based Recommendation Framework for Manual Network Configuration

Zhenbei Guo, Fuliang Li, *Member, IEEE,* Jiaxing Shen, *Member, IEEE,* and Xingwei Wang, *Member, IEEE*

***Abstract*—Network configuration plays a vital role in quality assurance of network services, requiring considerable effort and time. Automatic network configuration approaches are promising due to their capacity to automatically generate and verify configurations. However, these methods suffer from drawbacks such as generated configuration content being largely unknown to network operators and inefficient for large-scale networks. Manual configuration is thus still the primary way of managing networks. To facilitate editing processes of manual configuration, a network-wide tool for recommending custom keywords is in urgent need. In this paper, we propose a keyword recommendation tool that recommends custom keywords across various network devices. We observe that network devices of the same type and role tend to have a unified template and similar configurations, which enables recommending custom configurations between them. However, the vision entails the following three challenges. First, configurations need to be modeled accurately. Second, a wide variety of network protocols need to be supported. Third, relationships between custom keywords might be implicit and difficult to find. To address the challenges, we first built a configuration knowledge graph that could accurately model configurations, extract latent relationships between keywords, and generate explainable recommendations. Then we applied a recommendation framework to the graph for appropriate keyword recommendations. Lastly, to validate the performance, we conduct recommendations on real configurations over $26,000$ times. Experiment results indicate that the overall coverage rate for matching expected configurations reaches 79.396%, and the redundancy rate is less than 20%.**

***Index Terms*—network management, configuration synthesis, manual network configuration, knowledge graph, configuration keyword recommendation**

I. Introduction

OPERATORS deploy a wide variety of networks in different application scenarios such as backbone network [1], [2], Internet of Things (IoT) [3], and datacenter [4]. However, it takes extensive effort and time to configure a network of various network devices such as routers [1], [2], [5], [6] and filters [7]–[9]. Automatic approaches thus become increasingly popular including parameter optimization [3], [10]–[13], configuration synthesis [1], [2], [6], [14]–[16], and network verification [17]–[25]. Configuration synthesis automatically generates configurations out of network policies. While network configuration verification inspects if the network will behave as expected.

Z. Guo, F. Li, and X. Wang are with Northeastern University, Shenyang 110819, China. E-mail: guozb777@163.com, lifuliang@cse.neu.edu.cn, wangxw@mail.neu.edu.cn.

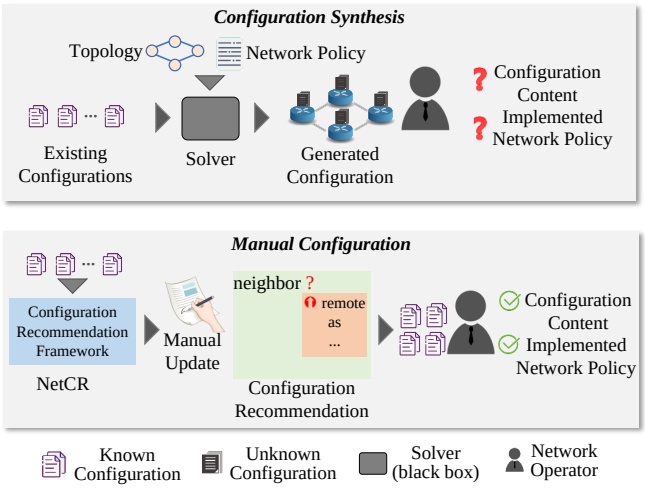
J. Shen is with Lingnan University, Tuen Mun, Hong Kong, E-mail: jiaxingshen@LN.edu.hk.

(Corresponding authors: Fuliang Li, Jiaxing Shen, and Xingwei Wang.)

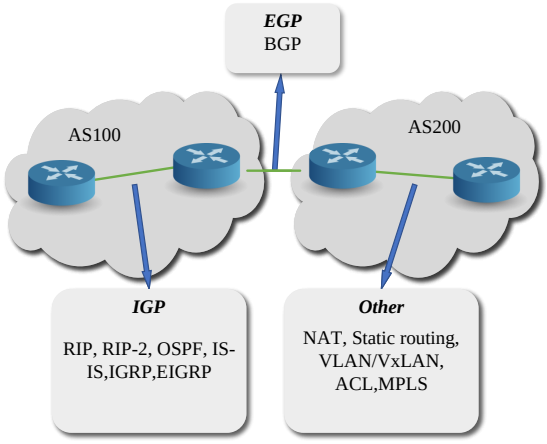
Promising as automatic configuration approaches are, the current drawbacks limit their usage to a narrow scope (such as configuring only router filters [16]) and impede their wide adoption by network operators [26]. First, due to the black-box solver, the generated configurations and implemented policies are largely unknown to operators, which poses a serious challenge for subsequent maintenance as illustrated in Figure 1a (*Configuration Synthesis*). Second, most synthesis methods are not scalable. For large-scale networks with more than a few hundred nodes, they are inefficient. For example, SyNET [6] spends more than 24 hours synthesizing configurations for a network with 64 nodes [2], [6]. More importantly, existing methods support limited routing protocols, such as OSPF and BGP [2], [6], [16]. However, there are usually multiple different protocols deployed in the network (Figure 1b). Operators have to manually configure other network protocols that are not supported by the synthesis approaches. Therefore, automatic configuration has not been widely adopted yet. Manual configuration still plays the primary role in most scenarios [10], [16], [27], [28].

Compared to automatic configuration, there are significantly fewer tools dedicated to manual configuration which requires extensive editing of manual templates and configuration files. One such tool is tab completion functionality, which is embedded in most networked devices [29]. It simplifies the manual editing process to a certain degree by automatically filling in partially typed internal configuration keywords (commands). However, internal keywords are predefined and only account for a small proportion of the whole configuration. The vast majority of configurations are customized by the operator during the editing of the configuration or template. In addition, custom configurations scattered across multiple devices are also affected by device type and role. For example, only switchers can configure VLAN/VxLAN. Recommendations that do not take this into account lead to invalid configurations or potential security issues. Therefore, we ask the question that *can we recommend network-wide configurations in manual network configuration?*

In this paper, we provide an affirmative answer to the question by proposing NetCR, a network-wide configuration recommendation tool. The process is depicted in Figure 1a (*Manual Configuration*). NetCR first extracts and analyzes existing configurations based on the existing configurations. Then, it recommends configuration keywords through a designed recommendation framework. Since operators implement the network policies manually, they avoid unknown configurations. In addition, we observe that operators tend to use a unified template and similar configurations across



(a) Network Configuration Approach



(b) Network Protocol and Technology

Fig. 1. Different configuration approaches and network protocols and technologies. Fig. 1a: illustration of differences between configuration synthesis (automated configuration) and manual configuration. Fig. 1b: the various network protocols and technologies.

network devices of the same type for management purposes [16], [30]. Such *configuration similarity* significantly reduces the difficulty of configuration recommendation and makes recommending configurations for devices of the same type more acceptable to operators. We could thus recommend appropriate configurations for devices with the same role.

Our vision, however, entails three challenges when applied to reality. First, we need to choose an intermediate approach for the model configuration that accurately captures the syntax and semantics of the configuration. Second, since there are usually multiple routing protocols in a network, the proposed approach should support different network protocols. Third, a keyword may exist in multiple configurations with different configuration branches. Given such a keyword, recommending keywords that do not differentiate between device types and different branches will result in invalid configurations or security issues.

To address the challenges, NetCR first builds a configuration knowledge graph based on existing configurations. Knowledge graphs are widely used in different recommendation areas to generate effective and explainable recommendations [31], [32]. NetCR parses configurations into the atomic form of *knowledge triple* to ensure the certainty and uniqueness of configuration, such as $\{ospf, ospf\_cost, 10\}$ that means the OSPF link cost values are 10. NetCR supports arbitrary configuration and protocol by building *knowledge triple*. Lastly, NetCR applies a recommendation framework based on *configuration similarity* to avoid invalid configurations.

To validate the performance of NetCR, we conduct extensive experiments on real configurations over $26,000$ times. The experimental results indicate that the overall coverage rate of NetCR reaches 79.396% and the redundancy rate is less than 20%. More than half of the configurations have a coverage rate of over 95%. High coverage rates guarantee the recommendation efficiency of NetCR during manual editing. In summary, our contributions are three folds:

TABLE I
NOTATIONS.

Symbol	Description
C	a configuration file
G	a knowledge graph
K	configuration knowledge $\{h, l, t\}$
h	head entity
l	relations between head and tail entities
t	tail entity
R, r	recommended keyword
F, f	parent keyword
S, s	children keyword of F
X	intersection set of all S
Y	union set of all S
IF	impact factor

- We propose efficient algorithms to analyze different configurations and model them as a unified configuration knowledge graph.
- We propose a configuration knowledge graph based recommendation framework to recommend configuration.
- We implement NetCR and validate its performance in real configurations.

We first show the overall design (§ II) and details of NetCR including the construction knowledge graph (§ III) and recommendation framework (§ IV), followed by the experiment results (§ V). Then we summarize related works (§ VI) and conclude this paper (§ VII) in the last two sections.

II. Design Overview

This section overviews the design of NetCR. Figure 2 presents the build process of the association configuration knowledge graph. Figure 3 shows the recommendation workflow of NetCR that consists of *Input*, *Recommendation Framework*, and *Output*. The main symbols used in this paper are shown in Table I.

Build Configuration Knowledge Graph Figure 2 illustrates the steps to build a configuration knowledge graph, the

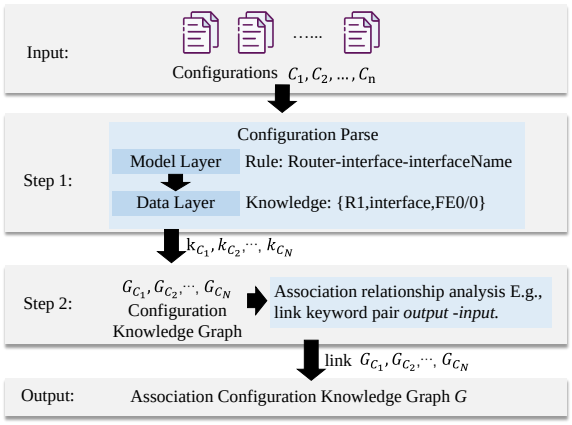


Fig. 2. The process of building an association configuration knowledge graph.

details of which are in Section III. First, NetCR takes existing configurations as input. Network topology can be the extra input, indicating the connection relationships between network devices and helping NetCR divide device roles. Secondly, the configuration of each device $(C_1, C_2, \cdots, C_n)$ is translated into configuration knowledge by pre-defined ontology (**Step 1**) described in Section III. NetCR then constructs configuration knowledge graphs $(G_{C_1}, G_{C_2}, \cdots, G_{C_n})$ for each device. Finally, NetCR analyzes the association relationships of keywords in these configuration knowledge graphs and links them to build the association configuration knowledge graph (**Step 2**). In addition, we provide the interpretability of recommended results based on the constructed knowledge graph, making them more convincing (Figure 9).

Recommendation workflow of NetCR Figure 3 shows the workflow NetCR, which is implemented in Section IV. NetCR takes incomplete configurations as the input (Figure 3a). Antecedent configuration keywords such as *filter* and *neighbor* belong to the parent keyword set $F = \{f_1, f_2, \cdots, f_n\}$. The input configuration fragment contains at least one parent keyword f, such as $filter$. NetCR sets the keywords requiring to be complemented as $R = \{r_1, r_2, \cdots, r_n\}$, which is represented by the symbol "?". Figure 3b shows the recommendation framework of NetCR, including *Secondary Classification Tag*, *Impact Factor Calculation*, and *Top-N Recommendation*. Based on the configuration knowledge graph G, NetCR first performs a keyword secondary classification tag to evaluate the configuration branches of all fork keywords, such as the *chassis* node shown in Figure 5b. NetCR then locates the F on the knowledge graph and calculates the impact factor IF of each child keyword based on the secondary classification tag. After the recommendation framework calculation, NetCR outputs R in order of IF from highest to lowest (Top-N recommendation).

NetCR takes the independent configuration keyword as the minimum recommended unit. Depending on the context of the parent node, NetCR outputs configuration fragments (e.g., *filter*) including multiple keywords or individual configuration keywords (e.g., *neighbor*) as shown in Figure 3c.

III. Build Association Configuration Knowledge Graph

The section describes the process of building an association configuration knowledge graph G in detail (Figure 2). A knowledge graph is denoted by G consisting of knowledge K. K is in the form of triples denoted by $\{h, l, t\}$, where h denotes the head entity, l denotes the relation between head and tail entities, and t denotes the tail entity. Depending on the context, any configuration keyword can be resolved to h, l, and t.

A. Configuration Ontology

The knowledge graph G is divided into *ontology* and *data* layer logically. An ontology defines how knowledge is exacted and stored in *data layer*. Since the configuration manuals define the function, format, parameters, view, and default of each keyword in detail, we directly define the configuration ontology described by OWL (Web Ontology Language) to parse configurations. Figure 4 shows the configuration ontology used by NetCR. We define classes to store head and tail entities, such as *Router* and *Interface*. *ObjectProperty* and *DataProperty* are relations, and their difference is that *ObjectProperty* links other classes, while *DataProperty* refers to its properties. *Range* denotes the type of tail entities. We show how ontology works and how data is stored through a simple configuration example, and the partial configurations of router $R1$ are as follows:

```
hostname R1
interface FE0/0
ip address 192.168.12.1 255.255.255.0
```

$R1$ and $FE0/0$ are stored in $Router$ and $Interface$ classes, respectively. Based on the ontology, configuration knowledge triples extracted from $R1$'s configurations are as follows:

```
{R1, interface, FE0/0}
{FE0/0, ipAddress, 192.168.12.1/24}
```

$R1$ and $FE0/0$ are connected by the $interface$ (*ObjectProperty*), and $FE0/0$ uses $ipAddress$ (*DataProperty*) to store its IP address. To implement quick classification and inference, these triples are stored ultimately by OWL language as follows:

```
:R1 rdf:type owl:NamedIndividual ,
:Router ;
:interface :FE0/0 .
:FE0/0 rdf:type owl:NamedIndividual ,
:Interface ;
:iPAddress :192.168.12.1/24 ;
```

It is important to note that knowledge triples and OWL codes can both build a configuration knowledge graph. We implement fast configuration queries and reasoning via OWL.

B. Configuration Parse

We now describe the configuration parsing process in **Step 1** (Figure 2). Configuration language belongs to DSL (Domain

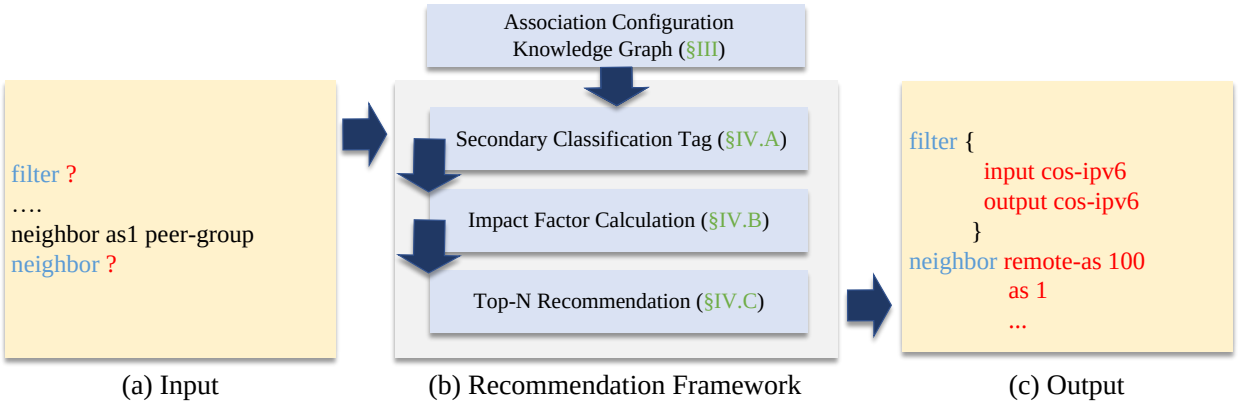


Fig. 3. The workflow of NetCR for recommending keywords.

owl:ObjectProperty	Class	Range
as_peer	Network	Network
interface	Router	Interface
connect	Interface	Interface
contain	Extra	Content
sibling	Extra	Content
.....	.....	.....

owl:DataProperty	Class	Range
bgpRouter	Router	xsd:string
ospf_cost	Interface	xsd:int
ipAddress	Interface	xsd:string
parameter	Extra	xsd:string
.....	.....	.....

Fig. 4. The configuration ontology used by NetCR.

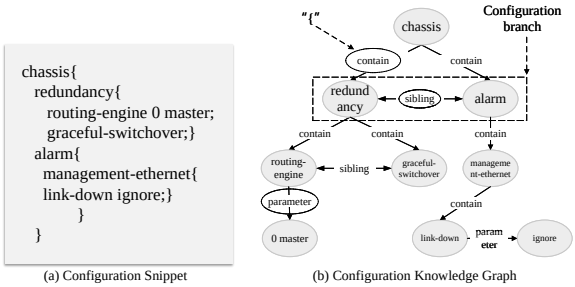


Fig. 5. An example of the configuration knowledge graph.

Specific Language), which has unique structural characteristics and semantic relations, such as the $R1$'s configurations. DSL's hierarchical structures and relations are implied in the fixed sentence pattern to realize the corresponding "function", which represents the execution target of configuration statements. A configuration statement consists of multiple keywords to implement "function". Therefore, NetCR takes the configuration keywords as the smallest units, decomposes these statements hierarchically, and converts them into knowledge triples based on the configuration ontology, such as the configuration knowledge of $R1$.

To handle more complex configurations (Figure 5a) and maintain the original semantics and syntax, NetCR defines extra relations to parse the strict hierarchical structure of configurations and indicates the position relationships of keywords, such as *contain*, *sibling*, and *parameter* shown in (Figure 5b). It is important to note that since the extra relations do not belong to any configuration file, they are just defined for better parsing configurations. Therefore, they will not be recommended to the operator as configuration keywords. An example of a configuration snippet and corresponding configuration knowledge graph is shown in Figure 5. The configurations in Figure 5a are converted into configuration knowledge as follows.

```
{chassis, contain, redundancy}
{chassis, contain, alarm}
{redundancy, sibling, alarm}
{alarm, sibling, redundancy}
{redundancy, contain, routing-engine}
{routing-engine, parameter, 0 master}
...
```

NetCR determines the configuration branch according to the symbol pair "{ }", and it defines extra relation *contain* to replace "{", such as {*chassis, contain, redundancy*}. *redundancy* and *alarm* are parsed into two separate configuration branches of parent node *chassis*, and extra relation *sibling* indicates their relationship (Figure 5b). In addition, extra relation *parameter* represents values and states of the parent keyword such as {*routing-engine, parameter,* 0 *master*}.

The configuration parsing algorithm is shown in Algorithm 1, which takes configuration files $C_1, C_2, \cdots, C_n$ as input. NetCR creates a root node for each configuration file, which serves as the starting node for the configuration knowledge graph (line 1 - line 2). NetCR parses configuration statements based on the current node and extracts configuration keywords and parameters to create new nodes (line 3 - line 5). NetCR updates nodes according to the end symbol (line 6 - line 12). Such as "*chassis* {" shown in Figure 5, the parent node of *redundancy* points to *chassis* instead of "{". Finally, NetCR returns configuration knowledge graph $G_1, G_2, \cdots, G_n$ for

Algorithm 1 Configuration Parse Algorithm.

Input: $C_1, C_2, \cdots, C_n$
Output: $G_1, G_2, \cdots, G_n$

1: **for** each configuration file $C_1, C_2, \cdots, C_n$ **do**
2: Create $g_i < C_i, root >$ as the root node of G_i;
3: **for all** configuration statement in C_i **do**
4: Parse keywords cmd and parameters p;
5: Create node $g_{i+1} < cmd, p >$ as the children of g_i;
6: **if** end symbol is "{" **then**
7: $g_i \leftarrow g_{i+1}$;
8: **else if** end symbol is "}" **then**
9: $g_{i+1} \leftarrow g_i$;
10: **end if**
11: **end for**
12: **end for**
13: **Return** $G_1, G_2, \cdots, G_n$;

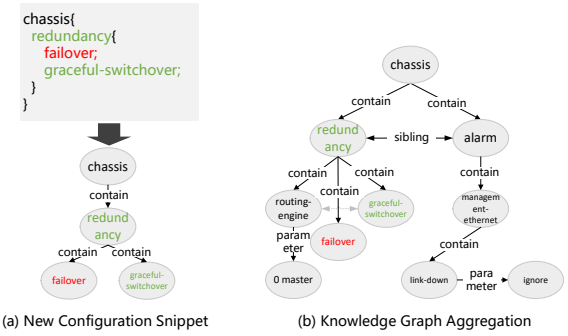
Algorithm 2 Mine keyword association relationship.

Input: $G_1, G_2, \cdots, G_n$
Output: library Lib

1: Initialize keyword cmd, parameter p, and frequency num;
2: Initialize the mapping triple $< p, cmd, num >$;
3: **for all** $G_1, G_2, \cdots, G_n$ **do**
4: Search all nodes in depth-first traversal;
5: **for** each non root node n_i **do**
6: Update $< p_{n_i}, cmd_{n_i}, num+ = 1 >$;
7: **end for**
8: **end for**
9: **for all** parameter p **do**
10: Add $< p, cmd, num >$ to library Lib when $num \geq num_{default}$;
11: **end for**
12: **Return** Lib;

(a) Configuration Snippet A (b) Configuration Snippet B

Fig. 6. An example of association keywords, and they appear on different configuration branches of the same parent keyword.



(a) New Configuration Snippet (b) Knowledge Graph Aggregation

Fig. 7. An example of the aggregating different knowledge graphs.

association keyword analysis (line 13).

C. Association keyword Analysis

We analyze the association relationships between keywords based on the G_1, G_2, $\cdots$, G_n (**Step 2** shown in Figure 2), which are aggregated to build the association configuration knowledge graph G.

Figure 6 shows two configuration snippets *A* and *B*. Configuration snippet *A* defines the used traffic policies. Three keywords *filter*, *input*, and *output* form associative relationships, such as <*filter, input*> and <*filter, output*>. Since *input* and *output* are usually configured in pairs and have the same parameter, they are always preferentially analyzed together when recommending. However, *filter* in configuration snippet *B* contains a new keyword *cos-ipv6*, which is a common parameter of *input* and *output*. The addition of configuration snippet *B* complicates the recommendation. When *filter* serves as recommendation input, we need to analyze which keyword is the desired keyword. Therefore, we should explore all hidden association keyword relations that exist in different configurations, linking them for unified analysis.

To find the association characteristics between keywords, NetCR uses Algorithm 2 to mine the association relationships between keywords. NetCR first initializes the mapping triple to record the frequency of keywords and parameters in all configuration knowledge graphs (line 1 - line 2). NetCR then traverses all nodes in $G_1, G_2, \cdots, G_n$, finds all keywords and parameters, and updates the mapping triple (line 3 - line 8). NetCR counts their frequency num, and it then decides whether to record keyword pairs according to the constant $num_{default}$ set by the operators (line 9 - line 11). NetCR uses the constant $num_{default}$ to filter low-frequency keywords, reducing the calculation and accelerating recommendation. The operator can reset $num_{default}$ based on their expert experiences and requirements, such as the number of devices. Finally, Algorithm 2 outputs the keyword library Lib that records high-frequency keyword pairs. NetCR links $G_1, G_2, \cdots, G_n$ based on Lib to build the association configuration knowledge graph G.

D. Association Configuration Knowledge Graph

NetCR aggregates $G_1, G_2, \cdots, G_n$ into the final association configuration knowledge graph G. An example of aggregating different knowledge graphs is shown in Figure 7. NetCR randomly selects a foundation graph G_i from $G_1, G_2, \cdots, G_n$, then aggregates other graphs based on G_i, such as NetCR takes Figure 5b as the foundation graph. A new configuration knowledge graph is shown in Figure 7a. NetCR retrieves the configuration structures and skips the common keywords between them, such as *redundancy*. There is a new keyword *failover* that does not exist on the foundation graph. NetCR checks its association pair in library Lib and attaches it to the foundation graph based on its context. If all $G_1, G_2, \cdots, G_n$ are processed, there are still keyword pairs that have not been

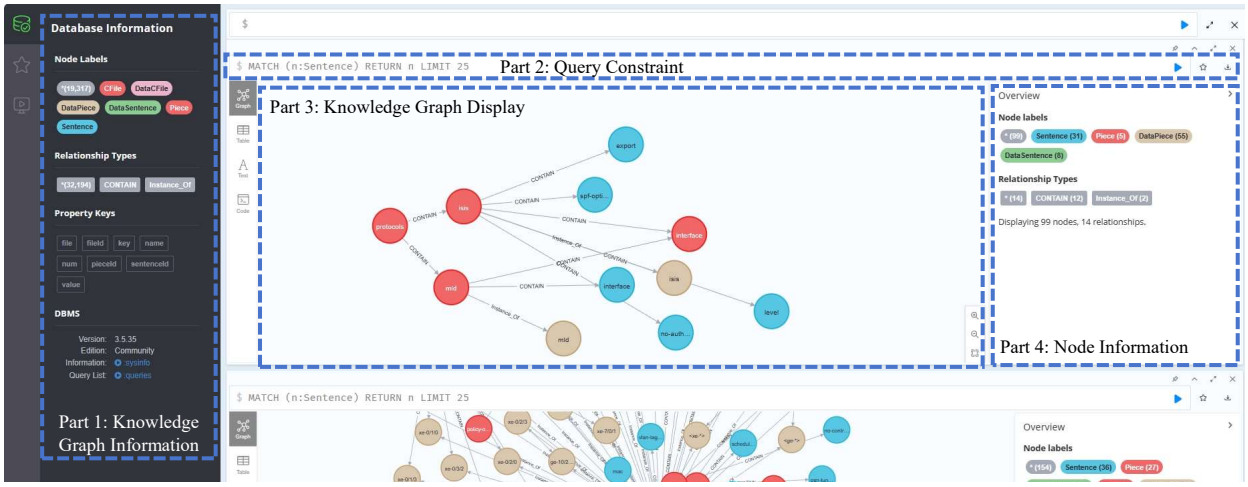


Fig. 8. The visual prototype system.

attached to the foundation graph. NetCR creates a new root node to connect them and the foundation graph.

E. Recommendation Interpretability

Each node and edge within the knowledge graph can serve as an explanation for query objects, thereby constituting their associated knowledge. Moreover, presenting the source of each recommended keyword adds a higher level of credibility compared to solely providing recommendation results. By analyzing the explanatory content provided, operators can assess the accuracy of current recommendation outcomes. Consequently, we elucidate specific recommended results by extracting context from the knowledge graph that pertains to them, encompassing their parent, sibling, and child nodes. We implement a visual prototype system by Neo4j [33] shown in Figure 8, which is divided into four parts. *Part* 1 and *Part* 4 provide the information about the knowledge graph and node, respectively. *Part* 2 provides an interactive query interface. *Part* 3 is a real-time visualization window that displays the configuration knowledge graph based on the query constraints. A practical explanation example is shown in Figure 9. NetCR explains *family inet* by showing its context. As an extra option, NetCR can explain its context through textual descriptions. For example, the graphical representation of knowledge triple $\{family, instance\_of, family\ inet\}$ is shown in Figure 9, and its textual description is that "*family inet is an instance of family class*". Interpretability allows the operators to understand why keywords are recommended, increasing the reliability of recommendation results.

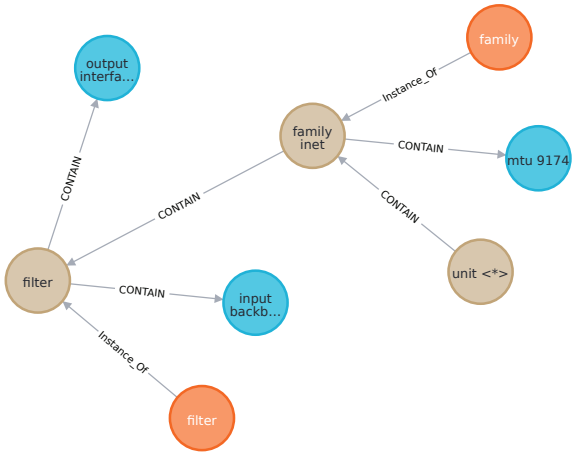


Fig. 9. The visual explanation of keyword *family inet*.

IV. Recommendation Framework

This section presents the recommendation framework of NetCR. Figure 3 shows a simple example of the configuration recommendation. NetCR takes different parent nodes *filter* and *neighbor* as input. NetCR outputs a configuration fragment and a single keyword based on their context, respectively.

However, there are more complex situations in practice. Figure 10 shows the node *family* with three different configuration snippets. There are multiple branch configurations of node *family*, and each sub-branch has a different configuration composition. In detail, *family* has three sub-branches including two *inet* and one *inet6*, and each *inet* has different configurations. Different configuration branches make it impossible to rely on library Lib alone to make accurate recommendations, such as the recommended result for *family* might be a configuration fragment or a single configuration keyword. We need to analyze all possible branches based on the input to find the expected keywords. Therefore, we design a recommendation framework to improve the recommendation coverage rate that includes the expected keywords, including *Secondary Classification Tag*, *Impact Factor Calculation*, and *Top-N Recommendation*. According to the composition of their descendants, NetCR uses the secondary classification tag to classify sub-keywords with the same parent node. These tags will be used as the basis for impact factor calculation. NetCR calculates the IF via a statistical algorithm and outputs N keywords according to the IF.

```
family inet{
    mtu 9174;
    filter{
     input backbone-in;
     output interface-out;
     }
}
```
Configuration Snippet A

```
family inet{
    mtu 9180;
}
```
Configuration Snippet B

```
family inet6{
    address 2001:da8:1:11::2/64 9180;
}
```
Configuration Snippet C

Fig. 10. keyword *family* with three different child node structures.

A. Secondary Classification Tag

NetCR implements the secondary classification tag to analyze the composition of configuration branches with the same parent node. Such as the configuration branch of parent node $family$ shown in Figure 10, NetCR labels the configuration content contained in $family\ inet$ and $family\ inet6$ with secondary classification tags. When $family$ is used as input, NetCR analyzes which configuration snippet is more suitable to be recommended based on these tags. A parent keyword is denoted by F. All children set of F in different configurations are denoted by $S_1, S_2, \cdots, S_n$. The child nodes of each S_i in $S_1, S_2, \cdots, S_n$ have the form $S_i = \{s_1^i, \cdots, s_j^i\}$. If the branch configurations of keyword F are different, they have the form:

$$S_1 \cap \cdots \cap S_n \Rightarrow \{s_1^1, \cdots, s_j^1\} \cap \cdots \cap \{s_1^n, \cdots, s_m^n\} = \emptyset$$

To further describe the configuration composition of keyword F, we set two additional sets X and Y. X is the intersection of S and has the form:

$$X = \{\{s_1^1, \cdots, s_j^1\} \cap \cdots \cap \{s_1^n, \cdots, s_m^n\}\}$$

where X denotes the keywords that must exist in the configuration branches of keyword F. Y is the union set of S and represents all possible children of F. The tag type set is defined by $Tag = \{Tag_1, Tag_2, \cdots, Tag_i\}$, and its priority decreases as the sequence number increases. Tag_1 represents that the current keyword must be the children of F, and it usually serves as the label for X. Tag_2 means that the current keyword may be the children of F. $\{Tag_3, \cdots, Tag_i\}$ represents the other constituent cases, whose purpose is to be able to completely label all children of F. Based on the above sets, the secondary classification algorithm is shown in Algorithm 3. NetCR first finds all branch nodes in G. For each branch node like $family$, NetCR records all its configuration branching cases and counts its total number in G (line 1 - line 4). NetCR then computes the rate for each S of F (line 5 - line 6). Finally, NetCR tags these keywords (line 7 - line 14).

B. Impact Factor Calculation

When NetCR finishes tagging keywords, NetCR utilizes these tags to calculate the IF. The greater the IF of the keyword, the higher the probability of becoming the recommended keyword r. The IF is calculated in the form:

$$IF = \frac{1}{(\vec{rate}_1^f, \cdots, \vec{rate}_i^f) \cdot (\vec{rate}_1^m, \cdots, \vec{rate}_j^m)}$$

$$\sum_1^n rate_i^f = 1, \sum_1^{n'} rate_j^m = 1$$

where f is the current input keyword, m is f's child node and sibling node of r, $rate = \frac{num}{sum}$ denotes the proportion shown in Algorithm 3, $\vec{rate}_1^f, \cdots, \vec{rate}_i^f$ is the rate that f appears in G alone, and $\vec{rate}_1^m, \cdots, \vec{rate}_j^m$ is the rate that f and m appear simultaneously. Since $\vec{rate}_1^f, \cdots, \vec{rate}_i^f$ is fixed, A higher value of $\vec{rate}_1^m, \cdots, \vec{rate}_j^m$ will increase the value of IF, meaning m is more valuable for recommendation.

Algorithm 3 Secondary classification tag.

Input: G
Output: G with tags

1: **for** each branch keyword F in G **do**
2: Record all F's children set $S = \{S_1, \cdots, S_n\}$ by $< F, \{S\} >$;
3: Count the number of F by Sum_F;
4: **end for**
5: Calculate each Num that the number of same children in $< F, \{S\} >$;
6: Update rate of S_n by $< F, S_n, \frac{Num_n}{Sum_F} >$;
7: **for** each F_i in $< F, \{S\} >$ **do**
8: Tag Tag_1 to $\{X\}$;
9: Tag Tag_2 to $\{Y\}$;
10: **for** $\frac{Num_i}{Sum_F}$ from high to low **do**
11: If the current node is not tagged, it is tagged Tag_i;
12: **end for**
13: **end for**
14: **Return** G;

C. Top-N Recommendation

NetCR supports taking a single parent node f, such as $filter$, or a configuration snippet containing multiple f, such as the configuration snippet shown in Figure 3.

NetCR achieves the configuration recommendations according to the following steps shown in Figure 11 :

- Begin: NetCR takes the incomplete configuration snippet C_s as input. C_s can include parent nodes, sibling nodes, and child nodes of r, and it contains at least one parent node f.
- Process 1: NetCR checks whether f has multiple children r (multiple configuration branches). If No, NetCR recommends r according to the keyword library Lib. Otherwise, NetCR checks the auxiliary condition. Auxiliary condition refers to all nodes except the f and its child node.
- Process 2: NetCR checks whether C_s can be used as the auxiliary conditions. If No, it means C_s only contains the parent node f and its child nodes, NetCR calculates

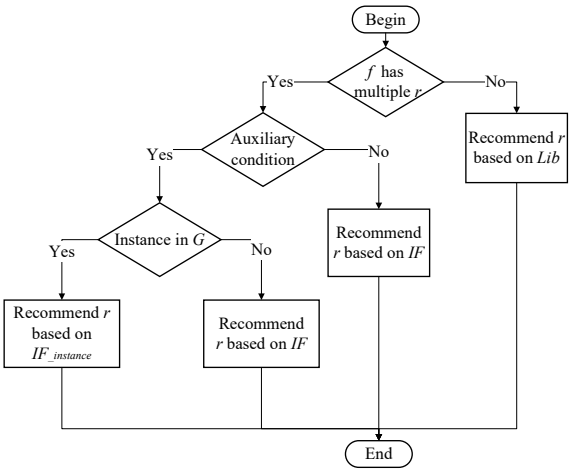


Fig. 11. The recommendation process of NetCR.

Algorithm 4 Recommendation Algorithm.

Input: G, C_s
Output: $C_{complment}$

1: Initialize $C_{complment}$ and locate the parent node F in C_s;
2: Build the configuration knowledge graph of C_s;
3: Record the sibling nodes $\{sn_1, \cdots, sn_n\}$ of r in C_s;
4: **for all** F in G **do**
5: Record r_i under the longest match $\{sn_1, \cdots, sn_n\}$;
6: Calculate and record $< r_i, IF_i, Tag_i >$;
7: **end for**
8: Take $< r_{max}, IF_{max}, Tag_{max} >$ with the maximum value IF_{max};
9: $C_{complment} \cap r_{max}$;
10: **for all** children S of r_{max} **do**
11: **if** r_{max} and S have Tag_{max} label **then**
12: $C_{complment} \cap S$;
13: **end if**
14: **end for**
15: **Return** $C_{complment}$;

the IF of each child node and recommends r based on IF. Otherwise, NetCR checks the instance of C_s in G.

- Process 3: NetCR matches all instances C_s in G, such as parsing C_s into a tree structure and matching it on G. If the match fails (No), NetCR calculates the IF of f's child nodes and recommends r based on IF. Otherwise, NetCR calculates the $IF_{instance}$ of each instance, and the keyword with a maximum $IF_{instance}$ is recommended.

NetCR adopts the configuration recommendation algorithm (Algorithm 4) to implement the above process. Operators take incomplete configurations C_s as input and output corresponding configurations $C_{complment}$ (Figure 3). NetCR first locates all parent nodes in C_s, parses configurations C_s, creates the corresponding configuration knowledge graph, and records the sibling nodes of r (line 1 - line 3). NetCR then finds out all possible r and records each IF and Tag, and it selects the r with the maximum impact factor (line 4 - line 9). Finally, NetCR relies on Tag to confirm sub-keywords of r such as the $input$'s sub-keyword shown in Figure 3c. If they exist, NetCR takes them as input (line 10 - line 15).

V. Evaluation

A. Experiment Setup

The configuration ontology of NetCR is created by Protégé [34], the knowledge graph is managed by Neo4j [33], and a snapshot of a visual prototype system is shown in Figure 8. NetCR is performed on a machine with a 2.90 GHz Intel Core i5 CPU and 8GB RAM.

Dataset We run extensive experiments on network configurations from *Internet2* [35]. We divide all configurations into *Small* and *Large* based on the configuration size.

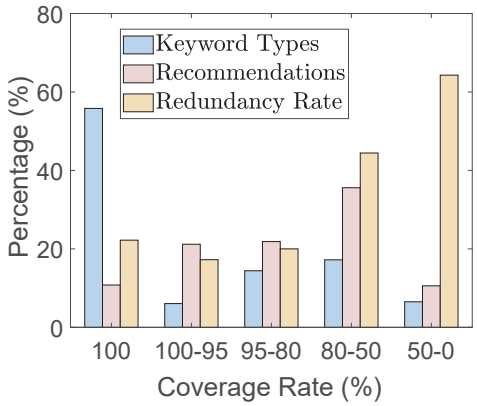
- *Small*: There are an average of 24 devices, and the average number of configuration lines for each device is 1260.
- *Large*: There are an average of 10 devices, and the average number of configuration lines for each device is 8236.

In addition, the constant $num_{default}$ in Algorithm 2 is set to 1, meaning that all configuration keywords are retained. NetCR constructs a highly complex knowledge graph based on these configurations and evaluates its performance against this knowledge graph.

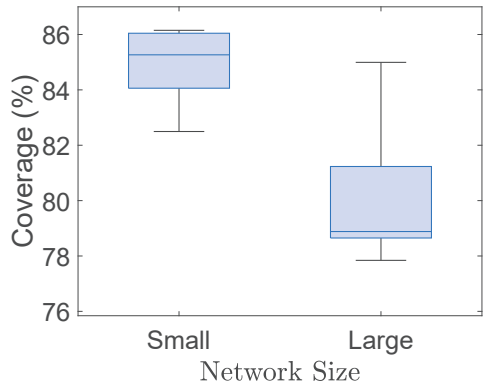
Recommendation For each recommendation, we randomly intercept configurations from the original configuration $C_{original}$ as input, such as a configuration keyword or a configuration fragment. We simulate the manual process of writing configurations by automatically inputting these configurations line by line into NetCR. Subsequently, NetCR computes and recommends configurations based on the provided input. Depending on the configuration sizes, *Small* recommends over $9,000$ times, and *Large* recommends over $17,000$ times.

Evaluation Metric We split the recommended configurations R (output) into two parts $R_{coverage}$ and $R_{redundancy}$. $R_{coverage}$ denotes the configurations that exist in the original configuration $C_{original}$, while $R_{redundancy}$ denotes the configurations that do not exist in the original configuration $C_{original}$. We evaluate and focus on the performance of NetCR from the following three aspects.

- *The coverage rate*. We calculate the coverage rate of NetCR by $\frac{R_{coverage}}{C_{original}}$. The coverage rate reflects the recommendation effectiveness of NetCR. The higher the coverage rate, the more accurate the recommendation. If the recommendation is a single configuration keyword rather than a configuration segment, then the coverage rate is equivalent to the accuracy rate.
- *The redundancy rate*. We calculate the redundancy rate of NetCR by $\frac{R_{redundancy}}{C_{original}}$. The redundancy rate is a significant metric that reflects the configuration composition with the highest IF based on the input. Consequently, redundant configurations can serve as valuable benchmarks for operators to comprehensively enhance their configurations.
- *Runtime*. We document the time dedicated to building knowledge graphs and providing configuration recommendations. The analysis of runtime can provide us with deeper insights into NetCR's performance.

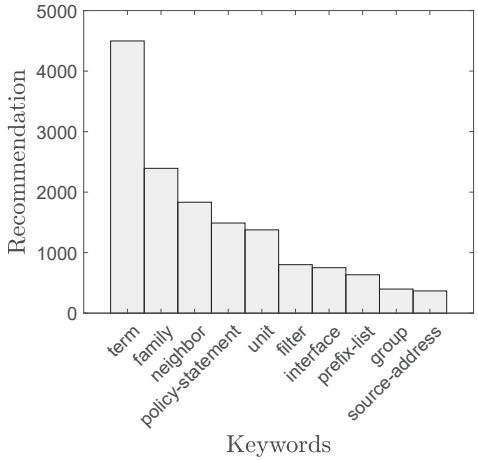


(a) Coverage and redundancy rate

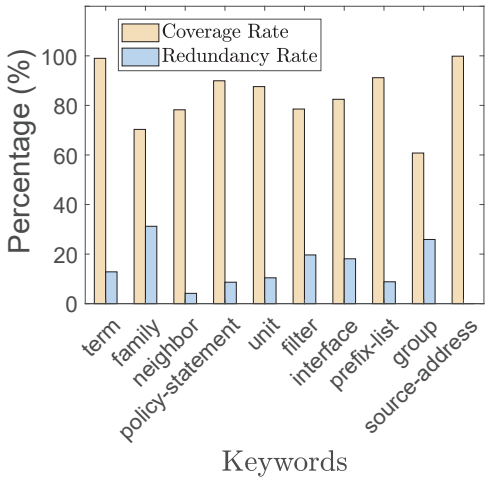


(b) Average coverage rate

Fig. 12. The coverage rates of NetCR. Fig. 12a shows the redundancy rates under different coverage rates. Fig. 12b shows the average coverage rate.



(a) Type



(b) Proportion

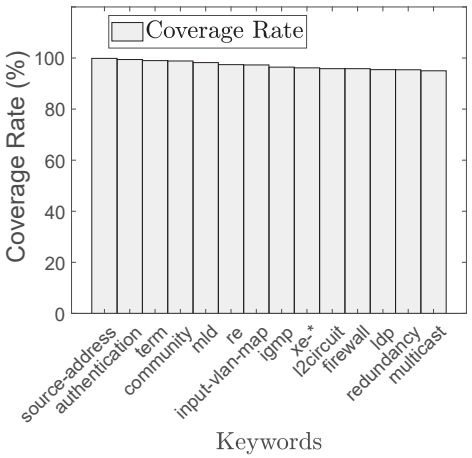
Fig. 13. The experimental results of top 10 most recommended keywords. Fig. 13a shows their maximum recommendations and Fig. 13b shows their coverage and redundancy rate.

B. Coverage and Redundancy Rate Analysis

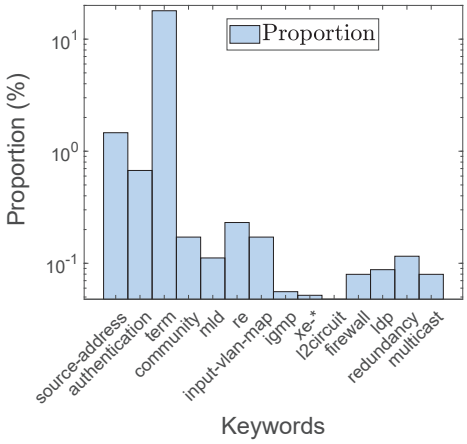
We first discuss the coverage and redundancy rate shown in Figure 12. *Keyword Types* denotes the keyword at the current coverage rate, such as parameter, and there are 215 keyword types in total. *Recommendations* represents the proportion of recommendations at the current coverage. Figure 12a shows 75% of the keywords achieve more than 80% coverage rate, they account for 61.46% of the total number of recommendations, and their average redundancy rate is below 20%. In particular, 54.35% of the keywords achieve 100% coverage rate, and the maximum redundancy rate does not exceed 22%. Keywords with coverage rates from 0 to 50% have the highest redundancy rate of 64.28%, but they only account for 8.85% of keyword types and 10.37% of total recommendations, respectively. The experimental results depicted in Figure 12a demonstrate that NetCR achieves a high coverage rate while simultaneously maintaining a low redundancy rate within the majority of recommendations. That is, NetCR can implement recommendations effectively. Figure 12b shows the average coverage rate of NetCR. The average coverage rate of NetCR in a small network size is higher than that of a large network size. The increase of configuration leads to the decrease of recommendation coverage rate, but NetCR still achieves an average coverage rate of at least 79.396% ($Large$) with a redundancy rate of less than 20%. It should be noted that redundant keywords also exist in the current network, and they do not mean that they are the wrong recommendations. Through redundant keywords, operators can further analyze the configuration of the existing network.

Figure 13 shows the top 10 most recommended keywords and their coverage and redundancy rates. *term* has the highest number of recommendations with 4695 times, and it reaches a 99% coverage rate and 12.7% redundancy rate. Among these frequently recommended keywords, the lowest coverage is *group*, but it also has a 68% coverage rate with a 17% redundancy rate. The experimental results in Figure 13a mean that they are widely distributed in different configuration branches. Moreover, the higher coverage rate and lower redundancy rate shown in Figure 13b indicate NetCR can effectively match them by calculating impact factors based on the secondary classification tag.

Figure 14 shows the partial keywords with more than 95% coverage rate and their proportion to the total number of recommendations. Due to configuration similarity, a large number of keywords appear repeatedly in configurations, resulting in



(a) Type



(b) Proportion

Fig. 14. The information of keywords with more than 95% coverage rate. Fig. 14a shows their types and Fig. 14b shows their proportion to the total number of recommendations.

their high coverage rate (Figure 14a). In Figure 14b, the total proportion of these keywords is as high as 22%. Among them, the keyword *term* has the highest recommendation ratio of 17%. Combined with the experimental results in Figure 12a, the total proportion of keywords with more than 50% coverage rate is 89.64%, meaning that it is feasible for NetCR to make recommendations based on configuration similarity.

C. Run Time

Figure 15 shows the total run time of NetCR, where *Formalization* presents the whole process of configurations from configuration parse to secondary classification tag, and *Recommendation* denotes the time of implementing configuration recommendations. On the small network size, which has 24 devices with an average of 1260 configuration lines. NetCR formalizes all configurations in 1.2 seconds, it makes 9,292 configuration recommendations in 1 second, and its total time is less than two seconds. Run time increases with the configurations. For the large network size, which has 10 devices with an average of 8236 configuration lines. NetCR takes nearly 4 seconds to formalize configurations

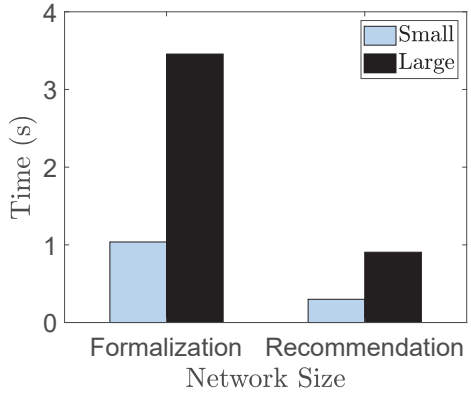


Fig. 15. The run time of NetCR.

and implements 17,545 configuration recommendations in 1 second. In summary, NetCR can parse and implement over 26,000 recommendations within 6 seconds, which satisfies the recommendation requirements of operators in real-time.

VI. Related Works

Configuration synthesis Multiple works [1], [2], [6], [7], [15], [16], [30] have shown how to automatically synthesize configurations out of network policies, avoiding misconfiguration caused by manual configuration. Partial synthesizers support only a single network protocol, such as Genesis [15] (static routing), Propane/PropaneAT [1], [30] (BGP), and Jinjin [7] (ACL). While other synthesizers like SyNet [6], NetComplete [2], and AED [16] support multiple routing protocols. In particular, NetComplete [2] and AED [16] can incrementally synthesize configurations.

Practical constraints of the above works, such as interpretability and generality, have prompted operators to opt for manual configurations instead of automatically generated ones. Our work focuses on aiding operators in manually modifying configurations and can serve as a practical complement to the aforementioned configuration works.

Configuration verification Verifiers are divided into control plane verification and data plane verification. Control plane verifiers have employed several approaches for simulating the configurations, such as Datalog [17], abstract interpretation [18], [19], [21], probability verification [22], explicit-state model checking [36], SMT coding [20], and difference calculation [24], [25]. Compared to control plane verification, data plane verifiers [14], [23], [37]–[42] directly check the data plane forwarding information, such as forwarding information base (FIB), to verify network invariants like blackhole-freedom and loop-freedom.

Our work focuses on parsing and recommending configurations rather than verifying them. Moreover, our approach can work seamlessly with configuration verifiers. We guarantee the correctness of configurations through them, which ultimately enables us to make recommendations based on error-free configurations.

VII. Conclusion

In this paper, we present the design of NetCR, the network configuration recommendation tool focusing on assisting network operators in editing configurations manually. NetCR models the configurations through the knowledge graph and performs keyword association analysis to aggregate all configuration knowledge graphs. NetCR then proposes a recommendation framework to recommend configurations. In addition, NetCR explains recommendation results based on the knowledge graph by providing context information such as the parent and sibling nodes. We implement a prototype system based on real network configurations. Experimental results show that NetCR implements configuration recommendations over $26,000$ times within 6 seconds and achieves a coverage rate of at least 79.396% with a redundancy rate of less than 20%. The keywords with more than 95% coverage rate account for more than half of the recommendation results, and the total proportion of keywords with more than 50% coverage rate is 89.64%. The experimental results confirm that NetCR can effectively recommend matched keywords based on the proposed recommendation method.

The demand for network intelligent management is steadily increasing. In future work, we will explore whether artificial intelligence models, such as the Large Language Model, can recommend and synthesize configurations more effectively than existing ones.

Acknowledgment

We would like to thank the Editors and Reviewers of the IEEE Internet of Things Journal in advance for their review efforts and helpful feedback. This work is supported by the National Natural Science Foundation of China under Grant Nos. 62072091, 62032013, U22B2005, and 92267206, and the financial support of Lingnan University (LU) (DB23A9) and Lam Woo Research Fund at LU (871236).

References

[1] R. Beckett, R. Mahajan, T. D. 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, Florianopolis, Brazil, August 22-26, 2016*, M. P. Barcellos, J. Crowcroft, A. Vahdat, and S. Katti, Eds. ACM, 2016, pp. 328–341. [Online]. Available: https://doi.org/10.1145/2934872.2934909

[2] A. El-Hassany, P. Tsankov, L. Vanbever, and M. T. Vechev, "Netcomplete: Practical network-wide configuration synthesis with autocompletion," in *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018*, S. Banerjee and S. Seshan, Eds. USENIX Association, 2018, pp. 579–594. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/el-hassany

[3] X. W. A. X. L. Haipeng Dai, Yikang Zhang and G. Chen, "Omnidirectional chargability with directional antennas," *IEEE Transactions on Mobile Computing*, 2023.

[4] 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 *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA: USENIX Association, Apr. 2023, pp. 629–644. [Online]. Available: https://www.usenix.org/conference/nsdi23/presentation/ramanathan

[5] Y. Xia, S. Chen, X. Gao, H. Dai, and G. Chen, "IGATA: an attraction-based online task recommendation framework in freemium-crowdsourcing platform," in *25th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2019, Tianjin, China, December 4-6, 2019*. IEEE, 2019, pp. 77–84. [Online]. Available: https://doi.org/10.1109/ICPADS47876.2019.00019

[6] A. El-Hassany, P. Tsankov, L. Vanbever, and M. T. Vechev, "Network-wide configuration synthesis," in *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, ser. Lecture Notes in Computer Science, R. Majumdar and V. Kuncak, Eds., vol. 10427. Springer, 2017, pp. 261–281. [Online]. Available: https://doi.org/10.1007/978-3-319-63390-9_14

[7] B. Tian, X. Zhang, E. Zhai, H. H. Liu, Q. Ye, C. Wang, X. Wu, Z. Ji, Y. Sang, M. Zhang, D. Yu, C. Tian, H. Zheng, and B. Y. Zhao, "Safely and automatically updating in-network ACL configurations with intent language," in *Proceedings of the ACM Special Interest Group on Data Communication, SIGCOMM 2019, Beijing, China, August 19-23, 2019*, J. Wu and W. Hall, Eds. ACM, 2019, pp. 214–226. [Online]. Available: https://doi.org/10.1145/3341302.3342088

[8] H. Dai, J. Yu, M. Li, W. Wang, A. X. Liu, J. Ma, L. Qi, and G. Chen, "Bloom filter with noisy coding framework for multi-set membership testing," *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[9] M. Li, D. Chen, H. Dai, R. Xie, S. Luo, R. Gu, T. Yang, and G. ChenFellow, "Seesaw counting filter: A dynamic filtering framework for vulnerable negative keys," *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[10] A. Mahimkar, A. Sivakumar, Z. Ge, S. Pathak, and K. Biswas, "Auric: using data-driven recommendation to automatically generate cellular configuration," in *ACM SIGCOMM 2021 Conference, Virtual Event, USA, August 23-27, 2021*, F. A. Kuipers and M. C. Caesar, Eds. ACM, 2021, pp. 807–820. [Online]. Available: https://doi.org/10.1145/3452296.3472906

[11] A. Mahimkar, Z. Ge, X. Liu, Y. Shaqalle, Y. Xiang, J. Yates, S. Pathak, and R. Reichel, "Aurora: conformity-based configuration recommendation to improve LTE/5G service," in *Proceedings of the 22nd ACM Internet Measurement Conference, IMC 2022, Nice, France, October 25-27, 2022*, C. Barakat, C. Pelsser, T. A. Benson, and D. R. Choffnes, Eds. ACM, 2022, pp. 83–97. [Online]. Available: https://doi.org/10.1145/3517745.3561455

[12] H. Dai, X. Wang, X. Lin, R. Gu, S. Shi, Y. Liu, W. Dou, and G. Chen, "Placing wireless chargers with limited mobility," *IEEE Trans. Mob. Comput.*, vol. 22, no. 6, pp. 3589–3603, 2023. [Online]. Available: https://doi.org/10.1109/TMC.2021.3136967

[13] H. Dai, Y. Xu, G. Chen, W. Dou, C. Tian, X. Wu, and T. He, "ROSE: robustly safe charging for wireless power transfer," *IEEE Trans. Mob. Comput.*, vol. 21, no. 6, pp. 2180–2197, 2022. [Online]. Available: https://doi.org/10.1109/TMC.2020.3032591

[14] 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 *SIGCOMM '22: ACM SIGCOMM 2022 Conference, Amsterdam, The Netherlands, August 22 - 26, 2022*, F. Kuipers and A. Orda, Eds. ACM, 2022, pp. 314–335. [Online]. Available: https://doi.org/10.1145/3544216.3544246

[15] K. Subramanian, L. D'Antoni, and A. Akella, "Genesis: synthesizing forwarding tables in multi-tenant networks," in *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, G. Castagna and A. D. Gordon, Eds. ACM, 2017, pp. 572–585. [Online]. Available: https://doi.org/10.1145/3009837.3009845

[16] A. Abhashkumar, A. Gember-Jacobson, and A. Akella, "AED: incrementally synthesizing policy-compliant and manageable configurations," in *CoNEXT '20: The 16th International Conference on emerging Networking EXperiments and Technologies, Barcelona, Spain, December, 2020*, D. Han and A. Feldmann, Eds. ACM, 2020, pp. 482–495. [Online]. Available: https://doi.org/10.1145/3386367.3431304

[17] A. Fogel, S. Fung, L. Pedrosa, M. Walraed-Sullivan, R. Govindan, R. Mahajan, and T. Millstein, "A general approach to network configuration analysis," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2015.

[18] S. K. Fayaz, T. Sharma, A. Fogel, R. Mahajan, T. D. Millstein, V. Sekar, and G. Varghese, "Efficient network reachability analysis using a succinct control plane representation," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

[19] A. Gember-Jacobson, R. Viswanathan, A. Akella, and R. Mahajan, "Fast control plane analysis using an abstract representation," in *SIGCOMM*, 2016.

[20] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, "A general approach to network configuration verification," in *SIGCOMM*, 2017.

[21] A. Abhashkumar, A. Gember-Jacobson, and A. Akella, "Tiramisu: Fast multilayer network verification," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2020.

[22] S. Steffen, T. Gehr, P. Tsankov, L. Vanbever, and M. T. Vechev, "Probabilistic verification of network configurations," in *SIGCOMM*, 2020.
[23] P. Zhang, X. Liu, H. Yang, N. Kang, Z. Gu, and H. Li, "Apkeep: Realtime verification for real networks," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2020.
[24] P. Zhang, Y. Huang, A. Gember-Jacobson, W. Shi, X. Liu, H. Yang, and Z. Zuo, "Incremental network configuration verification," in *HotNets*, 2020.
[25] P. Zhang, A. Gember-Jacobson, Y. Zuo, Y. Huang, X. Liu, and H. Li, "Differential network analysis," in *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4-6, 2022*, A. Phanishayee and V. Sekar, Eds. USENIX Association, 2022, pp. 601–615. [Online]. Available: https://www.usenix.org/conference/nsdi22/presentation/zhang-peng
[26] R. Birkner, D. Drachsler-Cohen, L. Vanbever, and M. T. Vechev, "Config2spec: Mining network specifications from network configurations." in *NSDI*, 2020, pp. 969–984.
[27] A. Gember-Jacobson, W. Wu, X. Li, A. Akella, and R. Mahajan, "Management plane analytics," in *Proceedings of the 2015 ACM Internet Measurement Conference, IMC 2015, Tokyo, Japan, October 28-30, 2015*, K. Cho, K. Fukuda, V. S. Pai, and N. Spring, Eds. ACM, 2015, pp. 395–408. [Online]. Available: https://doi.org/10.1145/2815675.2815684
[28] H. Chen, Y. Miao, L. Chen, H. Sun, H. Xu, L. Liu, G. Zhang, and W. Wang, "Software-defined network assimilation: bridging the last mile towards centralized network configuration management with nassim," in *SIGCOMM '22: ACM SIGCOMM 2022 Conference, Amsterdam, The Netherlands, August 22 - 26, 2022*, F. Kuipers and A. Orda, Eds. ACM, 2022, pp. 281–297. [Online]. Available: https://doi.org/10.1145/3544216.3544244
[29] Huawei, "Cli-based configuration guide of huawei." [Online]. Available: https://support.huawei.com/enterprise/en/doc/EDOC1100112346/51044c5f.
[30] R. Beckett, R. Mahajan, T. D. Millstein, J. Padhye, and D. Walker, "Network configuration synthesis with abstract topologies," in *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2017, Barcelona, Spain, June 18-23, 2017*, A. Cohen and M. T. Vechev, Eds. ACM, 2017, pp. 437–451. [Online]. Available: https://doi.org/10.1145/3062341.3062367
[31] F. Zhang, N. J. Yuan, D. Lian, X. Xie, and W. Ma, "Collaborative knowledge base embedding for recommender systems," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, and R. Rastogi, Eds. ACM, 2016, pp. 353–362. [Online]. Available: https://doi.org/10.1145/2939672.2939673
[32] S. Geng, Z. Fu, J. Tan, Y. Ge, G. de Melo, and Y. Zhang, "Path language modeling over knowledge graphsfor explainable recommendation," in *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*, F. Laforest, R. Troncy, E. Simperl, D. Agarwal, A. Gionis, I. Herman, and L. Médini, Eds. ACM, 2022, pp. 946–955. [Online]. Available: https://doi.org/10.1145/3485447.3511937
[33] Neo4j, "Graph database platform." [Online]. Available: https://neo4j.com.
[34] Protégé, "A free, open-source ontology editor and framework for building intelligent systems." [Online]. Available: https://protege.stanford.edu.
[35] "The internet2 observatory." [Online]. Available: http://www.internet2.edu/research-solutions/research-support/observatory.
[36] S. Prabhu, K.-Y. Chou, A. Kheradmand, B. Godfrey, and M. Caesar, "Plankton: Scalable network configuration verification through model checking," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2020.
[37] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2012.
[38] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P. B. Godfrey, and S. T. King, "Debugging the data plane with Anteater," in *SIGCOMM*, 2011.
[39] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte, "Real time network policy checking using header space analysis," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013.
[40] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey, "VeriFlow: Verifying network-wide invariants in real time," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013.
[41] N. P. Lopes, N. Bjørner, P. Godefroid, K. Jayaraman, and G. Varghese, "Checking beliefs in dynamic networks," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2015.
[42] A. Horn, A. Kheradmand, and M. R. Prasad, "Delta-net: Real-time network verification using atoms," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.

Zhenbei Guo received the BS degree in information and computational science from Northeastern University, Shenyang, China, in 2016 and the MS degree in 2019 from the School of Computer Science and Engineering, Northeastern University, where he is currently working toward the PhD degree. His research interests include network management and measurement, configuration synthesis, and network security.

Fuliang Li (Member, IEEE) received the BSc degree in computer science from Northeastern University, China in 2009, and the PhD degree in computer science from the Tsinghua University, China in 2015. He is currently an associate professor at the School of Computer Science and Engineering, Northeastern University, China. He has published more than 50 Journal/conference papers. His research interests include network management and measurement, cloud computing, and network security.

Jiaxing Shen (Member, IEEE) obtained B.E. in Software Engineering and Ph.D. in Computer Science from Jilin University in 2014 and The Hong Kong Polytechnic University in 2019, respectively. He was also a visiting scholar at Media Lab, Massachusetts Institute of Technology in 2017. His research interests include Mobile Computing, Data Mining, and IoT systems. He has published over 30 papers in top-tier journals and conferences. He has won Best Paper Award twice including one from INFOCOM 2020.

Xingwei Wang (Member, IEEE) received the BS, MS, and PhD degrees in computer science from Northeastern University, Shenyang, China, in 1989, 1992, and 1998, respectively. He is currently a professor with the College of Computer Science and Engineering, Northeastern University, Shenyang, China. He has authored or coauthored more than 100 journal articles, books and book chapters, and refereed conference papers. His research interests include cloud computing and future Internet. He was the recipient of several best paper awards.