2021 International Conference on Information and Communication Technology Convergence (ICTC) | 978-1-6654-2383-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICTC52510.2021.9620979

Automatic Data Model Mapper for Security Policy Translation in Interface to Network Security Functions Framework

Patrick Lingga*, Jeonghyeon Kim†, Jorge David Iranzo Bartolome*, and Jaehoon (Paul) Jeong†
* Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon, Republic of Korea
† Department of Computer Science and Engineering, Sungkyunkwan University, Suwon, Republic of Korea
Email: {patricklink, jeonghyeon12, jorgedavid, pauljeong}@skku.edu

***Abstract*—The Interface to Network Security Functions (I2NSF) Working Group in Internet Engineering Task Force (IETF) provides data models of interfaces to easily configure Network Security Functions (NSF). The Working Group presents a high-level data model and a low-level data model for configuring the NSFs. The high-level data model is used for the users to manipulate the NSFs configuration easily without any security expertise. But the NSFs cannot be configured using the high-level data model as it needs a low-level data model to properly deploy their security operation. For that reason, the I2NSF Framework needs a security policy translator to translate the high-level data model into the corresponding low-level data model. This paper improves the previously proposed Security Policy Translator by adding an Automatic Data Model Mapper. The proposed mapper focuses on the mapping between the elements in the high-level data model and the elements in low-level data model to automate the translation without the need for a security administrator to create a mapping table.**

***Index Terms*—Network Security, Policy Translation, Data Model Mapper, Automatic Mapper, I2NSF**

I. Introduction

The advent of the 5G network era has resulted in a limited operation of hardware-oriented networks and encouraged many researchers to conduct studies on software-oriented open networking technology. This research over software-oriented open networking technologies resulted in some technologies, such as SDN (Software-Defined Networking) and NFV (Network Functions Virtualization), which allowed the virtualization of network resources and services for the application of artificial intelligence to provide services dynamically depending on the condition of the network. However, with increasingly diverse and sophisticated networks, the likelihood of networking attacks targeting them is also increasing, and security administrators end up using a mix of several security solutions in order to mitigate attacks.

To manage and control various mixed security solutions and apply new security policies that change according to the situation, it is necessary to understand the network security functions (NSFs) developed by various manufacturers and environments, and to integrate them in order to manage security functions in an effective way. However, the distinctions of manufacturers and particularities of the environments make this task to be quite complex, which, in the end, results in less than optimal security systems due to integration problems.

In order to improve this, the Internet Engineering Task Force (IETF) Interface to Network Security Functions (I2NSF) Working Group (WG) established standard interfaces and standard YANG data models for the NSFs. The IETF I2NSF WG published RFC 8329 [1] to introduce the components and interfaces in I2NSF Framework as shown in Fig. 1. The components are:

- **I2NSF User:** The user of I2NSF Framework to control and manipulate the configuration of NSF. The user creates a high-level security policy and deliver it to the Security Controller.
- **Security Controller:** The instance that control the NSFs with the policy received from the I2NSF user. It translates the high-level security policy into the low-level security policy.
- **Developer's Management System (DMS):** The provider of the NSFs. It registers the capability of an NSF to the Security Controller
- **Network Security Function:** The network function that provides security services for the network.

The interfaces provided for the communication between I2NSF components are:

- **Consumer-Facing Interface:** The interface to deliver the user's high-level security policy to the Security Controller.
- **Registration Interface:** The interface to register an NSF with its capability from the DMS to the Security Controler.
- **NSF-Facing Interface:** The interface to deliver the translated low-level security policy from the Security Controller to the corresponding NSF.

The NSFs used in I2NSF aims to be freely used by users without any security expertise. To do this, I2NSF needs a security policy translator to translate a user's high-level policy into a low-level policy. However, even if only a minor translation mistake occurs when converting such a user's high-level policy into a low-level policy, a big security problem can threaten the entire network.

Intent Based Networking (IBN) emerged for solving that problem [2]. The main goal of IBN is to set a network and its security policies in an automatic way. This can minimize

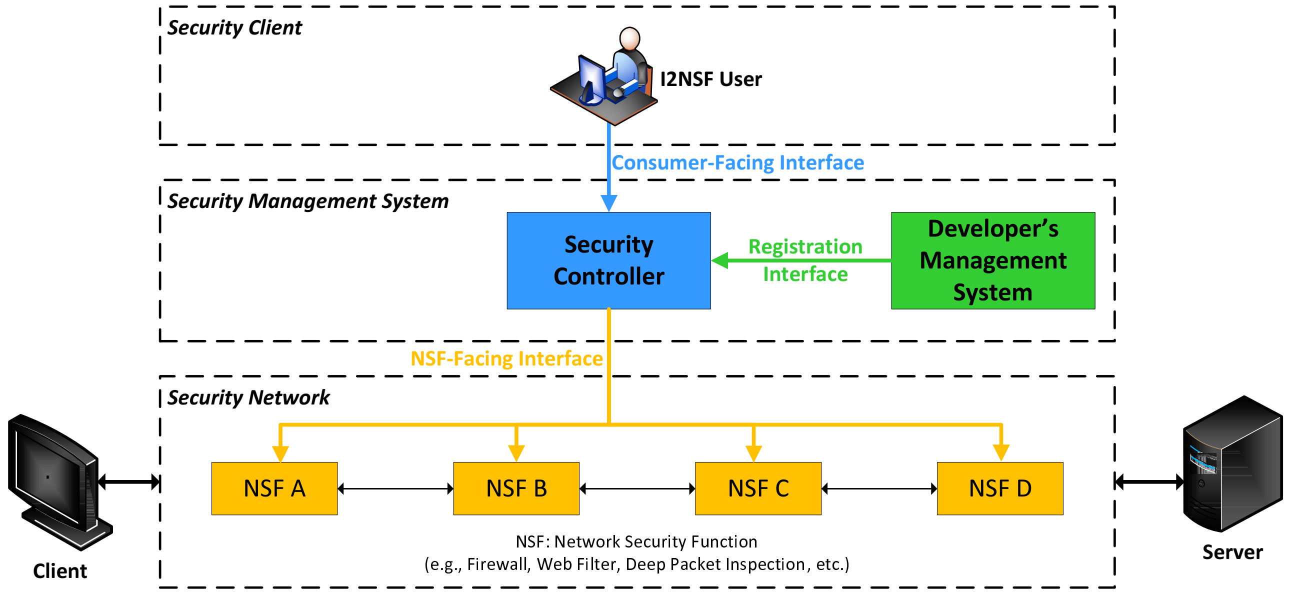
Security Client
I2NSF User
Consumer-Facing Interface
Security Management System
Security Controller
Registration Interface
Developer's Management System
NSF-Facing Interface
Security Network
NSF A
NSF B
NSF C
NSF D
Client
Server
NSF: Network Security Function
(e.g., Firewall, Web Filter, Deep Packet Inspection, etc.)

Fig. 1. I2NSF Framework

errors, especially human mistakes which can arise naturally by having human administrators manage the network manually. In addition, time and cost incurred by human administration might also be improved through automatic network management.

For accomplishing that, IBN automatically analyzes the intent in the user's high-level language, compensates for errors, and converts it into a low-level security policy, even if the network administrator does not describe all of the policies or services requirements that the network administrator wants to apply. However, in spite of the automation of the rest of the process, the conversion of the models into low-level policies is still performed manually by the security administrator. This manual method can cause damage due to a human error, and it acts as a bottleneck for the otherwise totally automated system, making it difficult to quickly cope with network environments that change in real time.

Therefore, we propose a policy mapping based on tree edit distance that is different from the existing mapping methods. In other words, a Data Model Mapper is proposed to automatically convert a high-level YANG data model to a low-level YANG data model. The contribution of this paper is as follows:

- **The improvement of the Security Policy Translator:** Automatic Data Model Mapper is used to automate the mapping between the high-level data model and the low-level data model.
- **Comparison of string matching algorithms:** This paper evaluates three string matching algorithms to find the best solution for our Automatic Data Model Mapper.

The remainder of this paper is composed as follows. Section II summarizes the related work of policy translation. Section III shows the architecture and implementation of our proposed solution. In Section IV, the performance evaluation for our Automatic Data Model Mapper is presented. Section V concludes this paper along with possible future work.

II. Related Work

In this section we introduce the related works in translation of security policy.

A. *Intent-Based Cloud Services for Security Applications*

J. Kim et al. [3] introduce an intent-based cloud service (IBCS) based on automatization and virtualization as a way of providing two assets at the same time: a virtualized security system for the security service providers, and an easy, quick and flexible cloud-based security service for security service consumers based on their intents. The IBCS-based security system consists of four steps: consumer's intent specification, security policy translation, the policy provisioning and, lastly, the policy enforcement. The article demonstrates the feasibility of the system by implementing a prototype. Our paper's goal is to improve the same type of system by fully automating the second step: security policy translation. Thus, improving the system flexibility and response against fast dynamic environmental changes and decreasing human error.

B. *Automata-based Security Policy Translation*

J. Yang et al. [4] use Automata to help users easily use NSF even without security knowledge. The paper constructed the policy translator by using Automata theory for a convenient security policy mapping management. They suggested Extractor for extracting data from the high-level policy by using Deterministic Finite Automaton (DFA) and attached NSF database to Data Converter for data mapping from abstract data to specified data which is required for NSF. Lastly, they constructed Generator for generating the low-level policies for each target NSF by using Context-Free Grammar (CFG). Using automata to extract data is an appropriate way to help translate high-level security policies into low-level security policies and support security administrators, but the proposed scheme has to map the data models manually to produce the translation.

C. Automatic Enforcement of Security Policies in NFV Networks

Basile et al. [5] they proposed a component called the Security Awareness Manager (SAM) to reflect users' intentions from high-level security policies to low-level security policies that respond to dynamically changing networks. The SAM is in charge of executing the policy refinement when the network is initialized and every time a change is detected into the network or into the security policy.

D. Reactive Configuration Updating for Intent-Based Networking

Tsuzaki et al. [6] introduces a procedure to update network configuration reactively in an extended NEMO (Intent-Based Network Modeling) language. The two use cases in [6] are automatic changing of the routing path depending on bandwidth usage, and automatic effective utilization of network bandwidth. It is, therefore, an example of trying to improve the Intent-based systems by adding automating them. In this particular case, the focus is automating the configuration of the system against environmental changes. Our paper, on the contrary, focuses on automating the high-level intent translation toward a low-level security policy. This would also, among other things, improve the system response speed for environmental changes, since it would take less time to fully apply the customer's ever-changing intents.

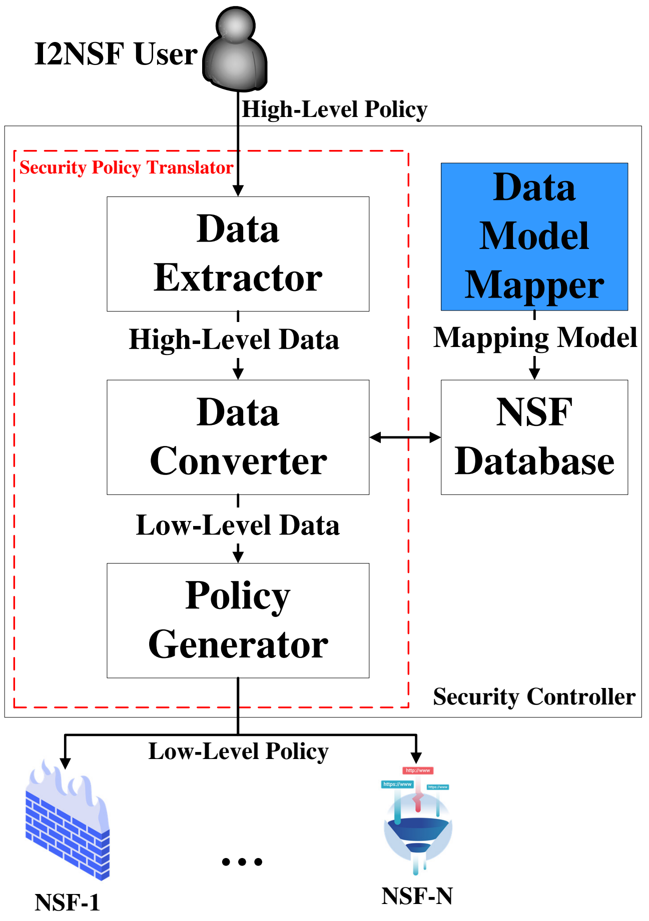

Fig. 2. Security Policy Translator Architecture

III. Policy Mapping Implementation

A. System Architecture

The overall architecture of our scheme is shown Fig. 2. The architecture consists of four components as proposed in [4], i.e., Data Extractor, Data Converter, NSF Database, and Policy Generator, with a newly added component, i.e., Data Model Mapper. The Data Model Mapper is used to automate the mapping between the high-level data model and the low-level data model.

When an I2NSF User sends a high-level Policy, it is encoded in an XML file that is based on a YANG data model to provide a standard model for the configuration. Each of the data in the high-level policy is contained in different XML elements to provide the meaning of the data. Each element of the XML has a different purpose for implementing the configuration. So does the low-level policy, where each data to be sent needs to be contained in the proper XML elements for the NSFs to understand.

When translating a policy, it is necessary to know which high-level XML element should be mapped into the low-level XML element. Previous work performs the mapping between elements manually. Hence, in this paper, we proposed a new component, i.e., Data Model Mapper, into the architecture to provide an automatic mapping between the high-level element and the low-level element.

Fig. 3 illustrates the proposed Automatic Data Model Mapper process. The complete algorithm can be seen in the Algorithm 1. To create a mapping model, we need the high-level YANG data model and the low-level YANG data model as the inputs. As a YANG data model can be represented automatically as a simplified graphical diagram of a tree with a software tool like "pyang" [7], we build both of the data models into tree graphs with the relationship of a parent and child in the tree graphs derived from the software tool. We start mapping the Data Model from the root node of the high-level graph as stated in Line 3 of the Algorithm 1. Then we calculate the Tree Edit Distance between the root node of the high-level graph with all nodes of the low-level graph. The minimum distance indicates the mapping for the high-level to the low-level. We start the looping for all children of the root and create a pair for calculating the distance with each child. The pair is obtained from the previous mapping of the parent. This is used to reduce the number of calculations for the distance and increases the accuracy of the mapping.

To calculate the minimum Tree Edit Distance, we use the algorithm proposed by Zhang-Shasha [8]. The Tree Edit Distance is obtained by calculating three operations on each

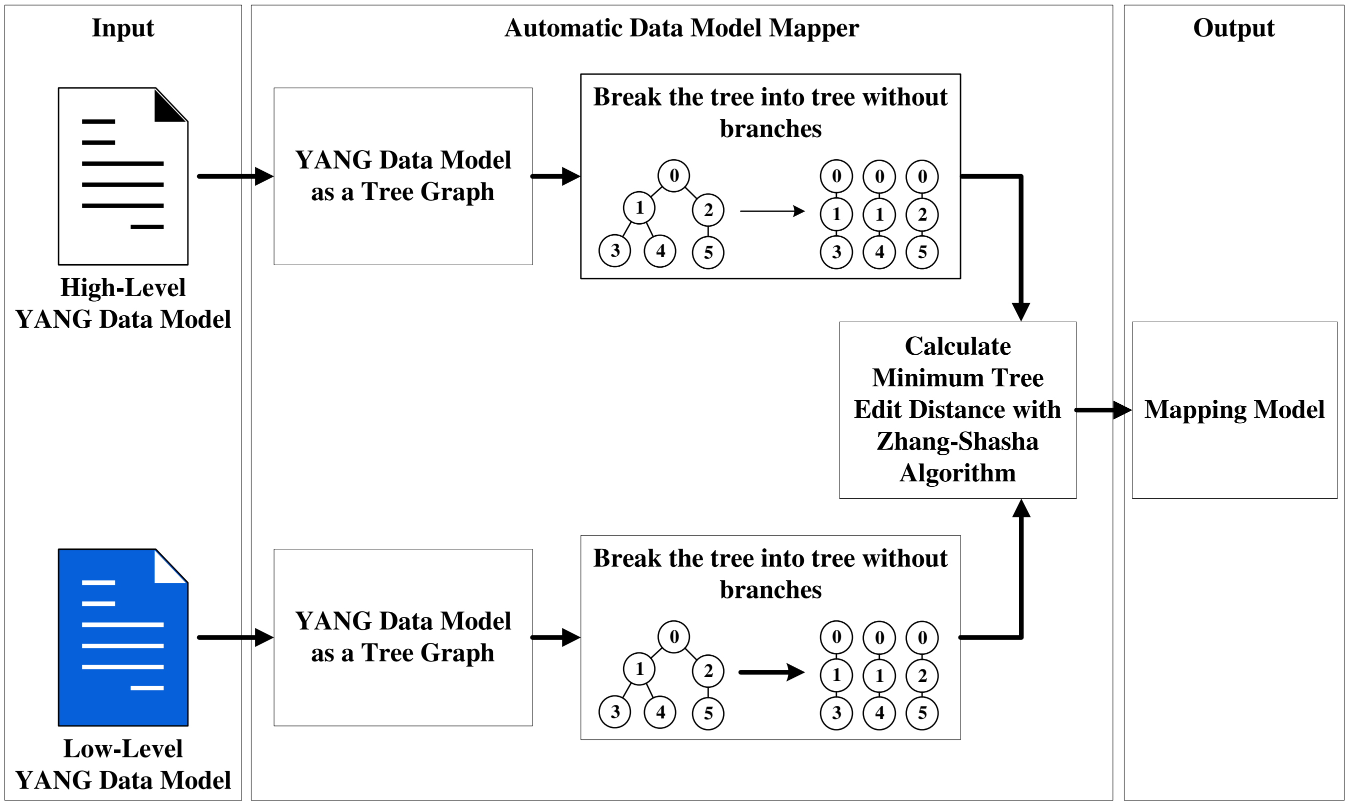
Input
Automatic Data Model Mapper
Output
High-Level YANG Data Model
Low-Level YANG Data Model
YANG Data Model as a Tree Graph
Break the tree into tree without branches
Calculate Minimum Tree Edit Distance with Zhang-Shasha Algorithm
Mapping Model

Fig. 3. Automatic Data Model Mapper Process

Algorithm 1 Automatic Data Model Mapper Algorithm

```
1: Inputs: High-Level Data Model ($H$), Low-Level Data Model ($L$)
2: Tree($H$), Tree($L$) // Create a high-level tree graph H and a low-level tree graph L
3: $current \leftarrow H[0]$
4: for $i = 1$ to $L.length$ do
5:     $treeDist[L[i]] \leftarrow ZSS(current, L[i])$ // Calculate Tree Edit Distance
6: end for
7: $map[current] \leftarrow$ Keys of $min(treeDist)$
8: for $i = 1$ to $H.length$ do
9:     $current \leftarrow H[i]$
10:    $pair \leftarrow map[current.parent].childs$
11:    for $j = 0$ to $pair.length$ do
12:        $treeDist[pair[j]] \leftarrow ZSS(current, pair[j])$
13:    end for
14:    $map[current] \leftarrow$ Keys of $min(treeDist)$ // The mapping uses the minimum value of Tree Edit Distances
15: end for
16: return $map$
```

step:

1) Insert: To insert a node.
2) Delete: To delete a node.
3) Change: To change the label of a node to another.

Each step will calculate the three operations and find the minimum number that can be achieved. The insert and delete operations can be calculated as the number of characters to be added or removed, respectively. For calculating the change operation, we use three string distance algorithms to find the best formula for our model, i.e., Levenshtein Distance [9], Cosine Similarity [10], and Sequence Matching [11].

B. Levenshtein Distance

Levenshtein Distance is a metric for measuring the edit distance between two strings [9]. The distance is calculated by obtaining the minimum number of single character edit operations (i.e., insert, delete, and substitute) required to change one string to another. The formula to calculate the Levenshtein Distance is shown in Equation (1).

$$dist_{a,b}(i,j) = \begin{cases} max(i,j), & \text{if } min(i,j) = 0, \\ min \begin{cases} dist_{a,b}(i-1,j)+1 \\ dist_{a,b}(i,j-1)+1 \\ dist_{a,b}(i-1,j-1)+1, \end{cases} & \text{otherwise,} \end{cases} \quad (1)$$

where

a : the first string to compare,
b : the second string to compare,
i : the character position of string a, and

j : the character position of string b.

To calculate the edit distance between the string a and b, we calculate $dist_{a,b}(|a|,|b|)$ where $|a|$ is the length of string a and $|b|$ is the length of string b.

C. Cosine Similarity

Cosine similarity is a measure between two non-zero vectors [10]. This technique can also be used to measure the similarity between strings. The cosine similarity for string is done to measure the word similarity between strings. The equation is shown in Equation (2).

$$\begin{aligned} similarity(a,b) &= \frac{a.b}{||a||.||b||} \\ &= \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2}}, \end{aligned} \tag{2}$$

where

a : the first string to compare, and
b : the second string to compare.

The result will be between 0 and 1 where 0 is no similarity and 1 is exactly the same. To calculate it as the distance between two strings, we derived Equation (3).

$$dist(a,b) = (1 - similarity(a,b)) \times |a|. \tag{3}$$

To calculate it as a distance, we reversed the result to define 0 as exactly the same and 1 as no similarity. This can be done using 1 subtracted by the similarity. And then we calculate the distance by multiplying it with $|a|$. For our model, we used the length of a to make all of the calculations are normalized with the same value for comparison of distances between nodes. By using Equation (3), we can calculate the distance between two strings for our Tree Edit Distance model.

D. Sequence Matching

Sequence Matching is a technique to compare a sequence of strings [11]. It measures the sequences' similarity between two strings. The formula used in Sequence Matching is shown in Equation (4).

$$similarity(a,b) = \frac{2 \times M(a,b)}{|a| + |b|}, \tag{4}$$

where

a : The first string to compare,
b : The second string to compare,
$M(a,b)$: The number of matches between A and B,
$|a|$: Length of a, and
$|b|$: Length of b.

Equation (4) results will be a ratio between 0 and 1. The formula needed to be converted to calculate the distance. We use a similar approach as in Equation 3 to calculate the distance between two strings. After converting the formula, we can calculate the distance using Sequence Matching.

IV. Performance Evaluation

The implementation of our model is carried out using the Python Programming Language. We use the high-level data model from the I2NSF Consumer-Facing Interface YANG data model [12] and the low-level data model from the I2NSF NSF-Facing Interface YANG data model [13]. We tested our implementation using 3 different techniques for distance calculation.

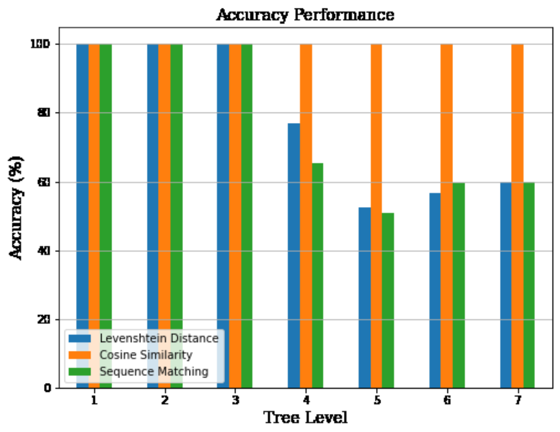

Fig. 4. Accuracy of Automatic Data Model Mapper

Fig. 4 shows the accuracy of our implementation. The y-axis shows the accuracy and the x-axis shows the Tree Level of the high-level data model. When the Tree Level is 1 to 3, i.e., the root of the Data Model, the mapping accuracy for all of the techniques is 100%. This happens because the tree is small and the comparison can be done easily. But when the tree level is 4, the Cosine Similarity maintains their 100% while the Levenshtein Distance and the Sequence Matching lose their accuracy. The accuracy for the Levenshtein Distance and Sequence Matching is maintained around 60% when the tree level is at the deepest for the high-level data model.

Cosine Similarity shows the best performance for our model to achieve 100% mapping accuracy even in the deepest tree level. This is caused by the structure of the Data Models. The Data Models contain similar words that can be translated. Cosine Similarity is comparing the words on the strings, while Levenshtein Distance and Sequence Matching are comparing the characters. We can see that for our Data Model, the best option is by using Cosine Similarity.

We also measure the time requirement for each technique to get the full mapping model as shown in Fig 5. The time to achieve the mapping model using the Levenshtein Distance is the fastest while the Cosine Similarity and Sequence Matching show a relatively similar processing time.

Despite the fact that the time process shows the worst performance, the Cosine Similarity is still the better way to map the data model as accuracy is more important in the network security area. A wrong configuration caused by wrong mapping can lead to a big problem in the systems.

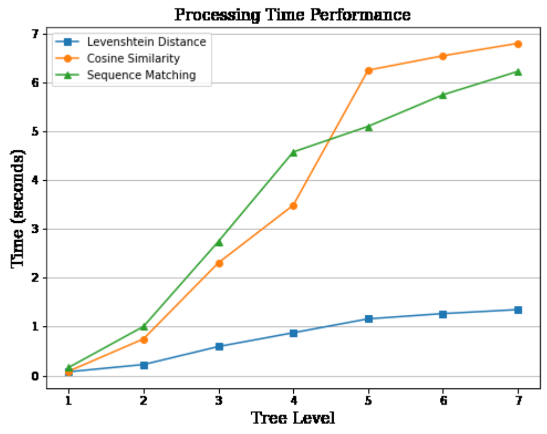

Fig. 5. Process Time of Automatic Data Model Mapper

The results in this performance evaluation show that the proposed method can work accurately for the I2NSF environment. Our proposed method will be able to solve the problem of manual mapping between a high-level data model and a low-level data model. Combined with the Automata-based Security Policy Translation in [4], the process of translating a high-level policy to a low-level policy can be fully automated.

V. Conclusion

In this paper, we provide an Automatic Data Model Mapper to improve the existing Security Policy Translation for I2NSF Framework. By adding our proposed component, we can automatically translate a high-level security policy to a low-level security policy without manually adding the Mapping Data Model. We proposed a mapper by calculating the Tree Edit Distance using the Zhang-Shasha algorithm. We implemented 3 different techniques to calculate the change operation in the Zhang-Shasha algorithm. In the performance evaluation, Cosine Similarity shows the best result for our Automatic Data Model mapper. As future work, we plan to add production rules for the Generator in the Security Policy Translator. [14]

Acknowledgments

This work was supported by Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Ministry of Science and ICT (MSIT), Korea (No. 2020-0-00395, Standard Development of Blockchain based Network Management Automation Technology). This work was supported in part by the MSIT, Korea, under the ITRC (Information Technology Research Center) support program (IITP-2021-2017-0-01633) supervised by the IITP. Note that Jaehoon (Paul) Jeong is the corresponding author.

References

[1] D. Lopez, E. Lopez, L. Dunbar, J. Strassner, and R. Kumar, "Framework for Interface to Network Security Functions," RFC 8329, Feb. 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8329.txt

[2] M. Beshley, A. Pryslupskyi, O. Panchenko, and H. Beshley, "Sdn/cloud solutions for intent-based networking," in *2019 3rd International Conference on Advanced Information and Communications Technologies (AICT)*, 2019, pp. 22–25.

[3] J. Kim, E. Kim, J. Yang, J. Jeong, H. Kim, S. Hyun, H. Yang, J. Oh, Y. Kim, S. Hares, and L. Dunbar, "Ibcs: Intent-based cloud services for security applications," *IEEE Communications Magazine*, vol. 58, no. 4, pp. 45–51, 2020.

[4] J. Yang and J. P. Jeong, "An automata-based security policy translation for network security functions," in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, 2018, pp. 268–272.

[5] C. Basile, F. Valenza, A. Lioy, D. R. Lopez, and A. Pastor Perales, "Adding support for automatic enforcement of security policies in nfv networks," *IEEE/ACM Transactions on Networking*, vol. 27, no. 2, pp. 707–720, 2019.

[6] Y. Tsuzaki and Y. Okabe, "Reactive configuration updating for intent-based networking," in *2017 International Conference on Information Networking (ICOIN)*, 2017, pp. 97–102.

[7] M. Bjorklund, "pyang," https://github.com/mbj4668/pyang, 2017.

[8] K. Zhang and D. Shasha, "Simple fast algorithms for the editing distance between trees and related problems," *SIAM J. Comput.*, vol. 18, pp. 1245–1262, 12 1989.

[9] V. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet physics. Doklady*, vol. 10, pp. 707–710, 1965.

[10] "Cosine distance, cosine similarity, angular cosine distance, angular cosine similarity," Mar 2017. [Online]. Available: https://www.itl.nist.gov/div898/software/dataplot/refman2/auxillar/cosdist.htm

[11] Python Software Foundation, "cpython," https://github.com/python/cpython/blob/main/Lib/difflib.py, Apr. 2020.

[12] J. P. Jeong, C. Chung, T.-J. Ahn, R. Kumar, and S. Hares, "I2NSF Consumer-Facing Interface YANG Data Model," Internet Engineering Task Force, Internet-Draft draft-ietf-i2nsf-consumer-facing-interface-dm-13, Mar. 2021, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-i2nsf-consumer-facing-interface-dm-13

[13] J. T. Kim, J. P. Jeong, P. Jung-Soo, S. Hares, and Q. Lin, "I2NSF Network Security Function-Facing Interface YANG Data Model," Internet Engineering Task Force, Internet-Draft draft-ietf-i2nsf-nsf-facing-interface-dm-12, Mar. 2021, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-i2nsf-nsf-facing-interface-dm-12

[14] J. P. Jeong, P. Lingga, S. Hares, L. Xia, and H. Birkholz, "I2NSF NSF Monitoring Interface YANG Data Model," Sep. 2021, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-i2nsf-nsf-monitoring-data-model-10