SPT: Security Policy Translator for Network Security Functions in Cloud-Based Security Services

Patrick Lingga, *Student Member, IEEE*, Jaehoon Jeong, *Member, IEEE*, Jinhyuk Yang, *Student Member, IEEE*, and Jeonghyeon Kim, *Student Member, IEEE*

***Abstract*—Interface to Network Security Functions (I2NSF) Working Group within Internet Engineering Task Force (IETF) has developed a framework and its interfaces with YANG data models for configuring Network Security Functions (NSF). These models include a high-level security policy (i.e., an overview of configuration) and a low-level security policy (i.e., a detailed and specific configuration) to facilitate the configuration of NSFs. In this paper, a Security Policy Translator (SPT) is proposed to translate high-level security policies created by users into the corresponding low-level security policies. It leverages the design of I2NSF YANG data models to accurately translate security policies. The SPT performs a translation by extracting the high-level security principles using Deterministic Finite Automaton (DFA) construction from the high-level YANG data model. It converts the extracted information to a low-level form by utilizing a mapping model created by comparing the two YANG data models, such as the Consumer-Facing Interface (CFI) and NSF-Facing Interface (NFI) YANG data models. It selects the optimal NSFs based on the security policies to provide maximum security performance. It generates low-level security policies for the NSFs to deploy the security services. The proposed approach allows security policy translation for the I2NSF framework with high accuracy and speed.**

***Index Terms*—Network management, network security, network automation, I2NSF, policy translator.**

I. Introduction

THE rapid development of computer networks in recent years has changed the way people live, work, and communicate. With the introduction of the 5G networks, the availability and quality of the Internet have reached a new level. Sharing information over the Internet is a normal practice for most of the world's population. There are many types of services available on the Internet, such as e-commerce, social media, online entertainment, and messaging. The great convenience of the Internet makes more and more users take advantage of those types of services. According to DataReportal [1], the number of Internet users has recently reached 5 billion people, which is equivalent to 63% of the entire world population. The number of users has grown by almost 200 million over the last year and continues to grow at a rate of 4% per year.

The increase in online activity has also led to a sprout of new businesses and industries, many of which operate exclusively or primarily online. This increase creates significant opportunities for entrepreneurs and organizations, but also brings new risks and challenges. Most companies start to focus on their core business plans without considering the security aspects of their systems. A growing number of online businesses are vulnerable to risks and dangers on the Internet as more and more users access their services. Unfortunately, many smaller businesses may not have the resources to adequately protect themselves against these security threats. In fact, a recent report found that only 8% of small-sized businesses and 14% of medium-sized businesses have a dedicated cybersecurity budget to focus on handling security risks [2]. This presents a major challenge, as these companies are often targeted by cyberattacks due to their vulnerabilities.

Moreover, with the development of technology, the methods used by cybercriminals to breach network security are changing. This means that companies need to stay up-to-date with the latest threats and invest in new security measures accordingly. However, meeting this requirement is often easier said than done. In many cases, companies may not have the necessary knowledge or resources to keep pace with the rapidly changing landscape of network security management. Even with the proper skills and resources, implementing and managing Network Security Functions (NSF) can be a complex and daunting task, because a number of NSFs' features and functions are supported by multiple security vendors and open-source technologies. Hence there is a need for a framework that integrates and translates business requirements without the need for a deeper understanding of network security. Interface to Network Security Functions (I2NSF) Working Group (WG) of the Internet Engineering Task Force (IETF) proposed a framework for users to control and manage network security services that are enforced by multiple security functions from different vendors or open-source technologies [3]. It provides businesses with an opportunity to manage network security in a more user-friendly and cost-effective way.

The I2NSF WG aims to provide a set of software interfaces and relevant data models based on YANG [4] to manage NSFs' aspects in an accessible manner for the users. To manage the NSFs, the user has the ability to specify rules, query, and monitor

Manuscript received 11 April 2023; revised 21 February 2024; accepted 27 February 2024. Date of publication 29 February 2024; date of current version 13 November 2024. This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) under Grant 2022-0-01015 and Grant 2022-0-01199. *(Corresponding author: Jaehoon Jeong.)*

Patrick Lingga and Jinhyuk Yang are with the Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon 16419, South Korea (e-mail: patricklink@skku.edu; jin.hyuk@skku.edu).

Jaehoon Jeong and Jeonghyeon Kim are with the Department of Computer Science and Engineering, Sungkyunkwan University, Suwon 16419, South Korea (e-mail: pauljeong@skku.edu; jeonghyeon12@skku.edu).

Digital Object Identifier 10.1109/TDSC.2024.3371788

NSFs. In order to simplify the rule-set specification for the users, I2NSF provides two configuration interfaces and data models, which are used to construct and deliver a high-level security policy and a low-level security policy. A high-level security policy is a comprehensible and less-detailed set of rules expressed directly by an I2NSF User. A low-level security policy is a detailed rule set with specific configuration information that is used by the NSFs to provide network security services.

But in the practical aspect, the I2NSF Framework needs an automated translator that translates the user's high-level security policy accurately and consistently, eliminating the possibility of misinterpretation or oversight. In real-world scenarios, networks are becoming increasingly complex, especially in large-scale networks where numerous security policies are in place. An automated translator plays an important role in simplifying this complexity and enhancing the efficiency of network security management. It helps the maintenance of a robust security posture and ensures that the NSFs work as intended, adhering to the specified high-level security policy.

First, automation in finding mappings between high-level security policies and low-level security policies significantly reduces the burden on users. In practical terms, this means that network administrators do not need to delve into the intricate details of every security rule. Instead, they can express their security requirements at a higher level of abstraction, making the entire process more user-friendly. This simplification is especially relevant in large-scale networks where managing numerous security policies can be overwhelming.

Second, optimizing algorithms for NSF provisioning through automation ensures the efficient utilization of network resources. In real-world scenarios, networks often operate under constraints such as limited bandwidth and computational power. The optimizing algorithms help the allocation of these resources fairly based on the translated security policies. Thus, this optimization is essential for ensuring that security services are delivered promptly without compromising the overall network performance.

Therefore, in this paper, we proposed a Security Policy Translator (SPT) to simplify the management of NSFs to offer tangible benefits for maintaining secure and efficient networks. The main contributions of this paper for the Security Policy Translator (SPT) from a high-level security policy to a low-level security policy are as follows:

- *An automatic mapper between the high-level and low-level YANG data models:* SPT handles the automatic mapping of the two YANG data models, providing model mapping suggestions. Its implementation leverages the design similarity of the high-level and low-level YANG data models to find the accurate mapping between the two models based on the Zhang-Shasha algorithm [5]. The proposed mapper significantly reduces human involvement in the translation process through dynamic mapping of the elements (see Section IV-A).
- *A Deterministic Finite Automaton (DFA)-based security policy extraction:* SPT constructs a DFA from a high-level YANG data model to precisely extract and validate the user security policies. In real-world scenarios, the data models often need to be adjusted or extended to respond to changing threats and compliance requirements. This DFA allows the flexibility to manipulate the YANG data model without the need for manual interruption (see Section IV-B).
- *An optimized NSFs selection:* SPT selects a set of NSFs that can automatically deliver security services requested by the I2NSF User without any knowledge of the network architecture. The optimization of security service implementations is highly valuable in real-world networks that are very dynamic and diverse (see Section IV-C).
- *An evaluation of the proposed approaches:* SPT is evaluated by measuring different performance indicators for each proposed component in the SPT. It is evaluated against a number of parameters to find out the potential of the proposed approach for the translation of security policies. SPT's evaluation process ensures that it can be continuously improved to meet the evolving challenges in the future (see Section V).

The rest of this article is organized as follows. In Section II, related work is summarized along with analysis. Section III discusses the framework and a target scenario used to formulate the problem. Section IV describes in detail the proposed approach for translating a high-level security policy to the corresponding low-level security policy. In Section V, we evaluate our proposed SPT on its components using a variety of evaluation methods. Finally, Section VI concludes this paper along with the discussion of future research.

II. Related Work

The number of devices and networks utilizing cloud technology increases, making traditional manual methods of managing and configuring policies on networks gradually more difficult. As a result, Intent-Based Networking (IBN) technology is gaining prominence as a way to easily manage networks based on a user's intent.

However, natural language expressions of the intent cannot be used directly to configure networks, hence intent translation is required. Therefore, the abstract intent must be translated into low-level network policies that network devices can understand, and this technology is essential in IBN [6]. Research is actively conducted on translating an intent expressed in a natural language into network policies based on Natural Language Processing (NLP) to reflect an abstract user intent in the network.

In addition, demand for IBN services in the field of cloud network security is increasing, and research in this area is also actively conducted. The IETF proposed NETCONF to manage various heterogeneous network devices based on user intent, using YANG [4] for data modeling. In [3], a standard has been proposed for managing and controlling various network security functions (called NSF) created by different vendors in a Software-Defined Network (SDN) cloud environment. To manage heterogeneous IBN-based network devices in the I2NSF framework, it is essential to translate a high-level security policy (i.e., a user's intent) into the corresponding low-level security policy.

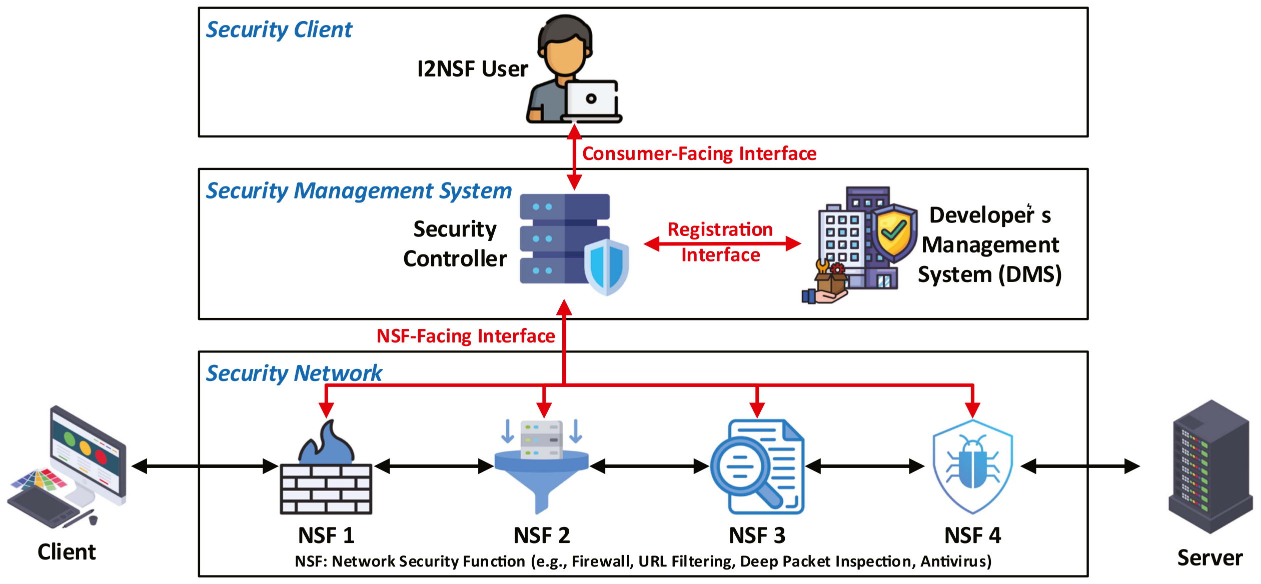

Security Client
I2NSF User
Consumer-Facing Interface
Security Management System
Security Controller
Registration Interface
Developer's Management System (DMS)
NSF-Facing Interface
Security Network
Client
NSF 1
NSF 2
NSF 3
NSF 4
Server
NSF: Network Security Function (e.g., Firewall, URL Filtering, Deep Packet Inspection, Antivirus)

Fig. 1. I2NSF framework.

The IBCS framework [7] has been proposed for the efficient management of heterogeneous NSFs used to mitigate various security issues that arise in intent-based cloud service environments and to process a user security intent. The authors translate high-level security policies received from users into low-level security policies that NSFs with the corresponding capabilities can understand to provision the corresponding security services by using a security translator within a security controller. In [7], the authors used a semi-automatic mapping method. Also, the NSFs selection is done without considering the optimal selection.

In [8], network slicing was implemented using IBN in a 5G environment. The user's intent related to network slicing was an input in Graphical User Interface (GUI) as a template, and the input high-level policy related to network slicing was mapped into the required format for configuration using a knowledge-based Policy Store. This method also went through a policy translation process that involves a one-to-one matching of, rather than a separately calculated algorithm.

However, in our paper, we have added a data model mapper with Zhang-Shasha algorithm to calculate the Tree Edit Distance between the high-level security policy YANG data model and the low-level security policy YANG data model. In addition, we have implemented an optimal policy provisioning which can select the appropriate NSFs that can provide the required security services. In this way, we can achieve the conversion of a high-level security policy to a low-level security policy and automatically select the optimal NSF(s).

III. Problem Formulation

Interface to Network Security Functions (I2NSF) Working Group (WG) of Internet Engineering Task Force (IETF) proposed a framework for users to control and manage NSFs by specifying rule sets. Fig. 1 illustrates the architecture of the I2NSF Framework [3]. It consists of four components and three interfaces. The components of the I2NSF Framework are:

- *I2NSF User:* A user of the I2NSF Framework that specifies the rule-sets for the NSFs to configure the security services. The rule sets are served as high-level security policies, i.e., configuration information that is easy for the non-security expert to understand.
- *Security Controller:* An application that controls and manages the NSFs through specific rule-sets created by I2NSF Users. It is also responsible for translating the high-level security policies to the corresponding low-level security policies to activate the appropriate NSFs.
- *Developer's Management System (DMS):* A vendor's system that provides NSFs for virtualized security services. It registers the available NSFs and their capabilities with the Security Controller.
- *Network Security Functions:* Virtualized network instances that detect and mitigate threats to ensure integrity, confidentiality, or safe operation of network communications. It accepts low-level security policies from the Security Controller to provide security services.

I2NSF WG connects the components in the I2NSF Framework with standardized interfaces. The interfaces are designed with various YANG [4] data models and implemented using either NETCONF [9] or RESTCONF [10]. The I2NSF interfaces are as follows:

- *Consumer-Facing Interface (CFI):* An interface for delivering high-level security policies from the I2NSF User to the Security Controller. The CFI YANG data model [11] is designed to resemble a human natural language as closely as possible.
- *Registration Interface (RI):* An interface for registering capabilities of NSFs with the Security Controller. The RI YANG data model [12] is designed so that the DMS can register the capabilities of the NSFs with the Security Controller and also the Security Controller can query a new

NSF with specific capabilities that can provide security services.

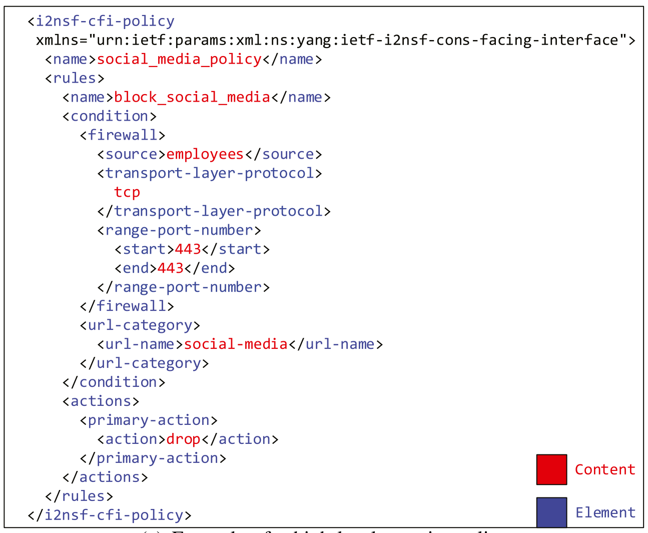
- *NSF-Facing Interface (NFI):* An interface that provides NSFs with translated low-level security policies. The NFI YANG data model [13] is intended to provide security policy configuration for the NSFs that can be used to deploy the requested security services, e.g., access control lists.

The I2NSF Framework provides an automated system to enable NSFs for high-level instructions from a user. In order to enable the NSFs to follow the high-level instructions, a security policy translator (called SPT) in the Security Controller is required. It must provide an accurate translation for the target NSFs in order to protect a target network efficiently, assuming that a user properly provides the correct configuration. An inaccurate translation will cause gaps in the protection for the network.

A target scenario is the translation of high-level security policies into the corresponding low-level security policies in a company. Since the main purpose is to provide users with an easy and understandable way to configure NSFs, the translator must automatically and accurately translate the given high-level security policy into the corresponding low-level security policy. The translator must also determine the correct NSF(s) that can provide the requested security service in accordance with the given security policy.

Fig. 2 illustrates an example scenario, in which a high-level security policy is translated to the corresponding low-level security policies based on the available NSFs. In Fig. 2(a), the I2NSF User (e.g., a network administrator) provides a high-level security policy without any knowledge of the network and available NSFs. This security policy is equivalent to "`All employees are prohibited from accessing unauthorized social-media websites on company devices`" in English.

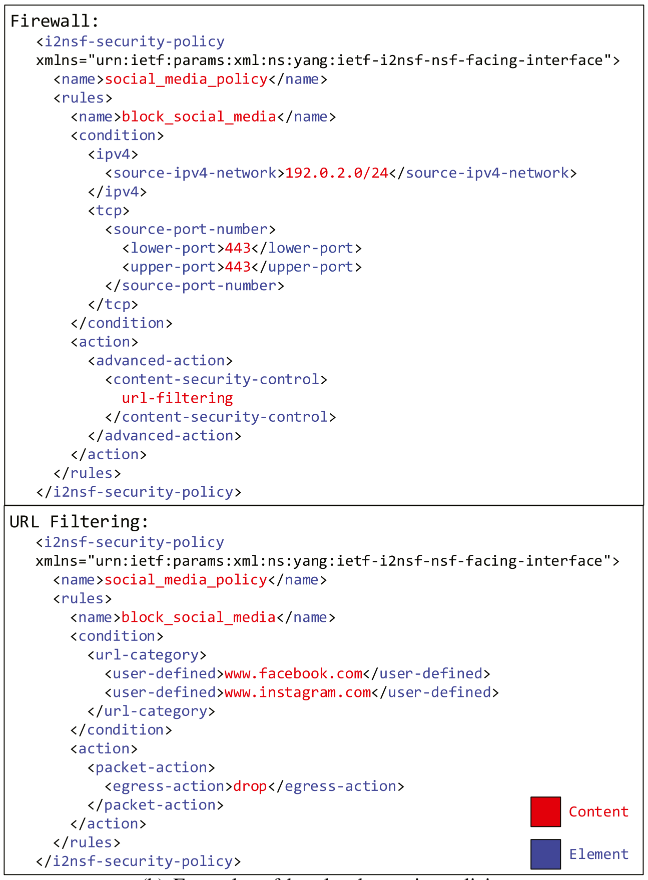
Fig. 2(b) shows the results of the translation. Since the NSFs are unable to process `employees` and `social-media`, they must be translated to the subnet address (e.g., `192.0.2.0/24`) and the hostname URL (e.g., `www.facebook.com` and `www.instagram.com`), respectively. The elements in the XML file of Fig. 2(a) must also be converted into a structure that is understandable to the NSFs. The Security Controller must also ensure that the policies are provisioned to the appropriate NSFs that can actually perform the security policy. In this particular scenario, a Firewall is used to verify the packets that have the IP addresses of the `employees` with TCP packets through a standard HTTPS port. If a packet meets this condition, the Firewall will forward it to the URL Filtering to block it, which tries to access `social-media`. Next section explains the proposed Security Policy Translator to handle this scenario.



```
<i2nsf-cfi-policy
 xmlns="urn:ietf:params:xml:ns:yang:ietf-i2nsf-cons-facing-interface">
  <name>social_media_policy</name>
  <rules>
    <name>block_social_media</name>
    <condition>
      <firewall>
        <source>employees</source>
        <transport-layer-protocol>
          tcp
        </transport-layer-protocol>
        <range-port-number>
          <start>443</start>
          <end>443</end>
        </range-port-number>
      </firewall>
      <url-category>
        <url-name>social-media</url-name>
      </url-category>
    </condition>
    <actions>
      <primary-action>
        <action>drop</action>
      </primary-action>
    </actions>
  </rules>
</i2nsf-cfi-policy>
```

🟥 Content
🟦 Element

(a) Example of a high-level security policy



```
Firewall:
   <i2nsf-security-policy
   xmlns="urn:ietf:params:xml:ns:yang:ietf-i2nsf-nsf-facing-interface">
     <name>social_media_policy</name>
     <rules>
       <name>block_social_media</name>
       <condition>
         <ipv4>
           <source-ipv4-network>192.0.2.0/24</source-ipv4-network>
         </ipv4>
         <tcp>
           <source-port-number>
             <lower-port>443</lower-port>
             <upper-port>443</upper-port>
           </source-port-number>
         </tcp>
       </condition>
       <action>
         <advanced-action>
           <content-security-control>
             url-filtering
           </content-security-control>
         </advanced-action>
       </action>
     </rules>
   </i2nsf-security-policy>
```

```
URL Filtering:
   <i2nsf-security-policy
   xmlns="urn:ietf:params:xml:ns:yang:ietf-i2nsf-nsf-facing-interface">
     <name>social_media_policy</name>
     <rules>
       <name>block_social_media</name>
       <condition>
         <url-category>
           <user-defined>www.facebook.com</user-defined>
           <user-defined>www.instagram.com</user-defined>
         </url-category>
       </condition>
       <action>
         <packet-action>
           <egress-action>drop</egress-action>
         </packet-action>
       </action>
     </rules>
   </i2nsf-security-policy>
```

🟥 Content
🟦 Element

(b) Examples of low-level security policies

Fig. 2. Example of a translated security policy in XML.

IV. Security Policy Translator

In this section, the proposed Security Policy Translator (SPT) for the I2NSF framework is explained. Fig. 3 shows the proposed SPT architecture. It consists of four main components and one supporting component. A brief explanation of the components is as follows:

1) *Data Model Mapper:* Its task is to map the high-level elements to the corresponding low-level elements. The mapping is performed automatically by the comparison of the elements in the high-level and low-level YANG data models using the Zhang-Shasha Algorithm [5]. This component is used when initializing SPT and when updating one or two of the YANG data models. The results

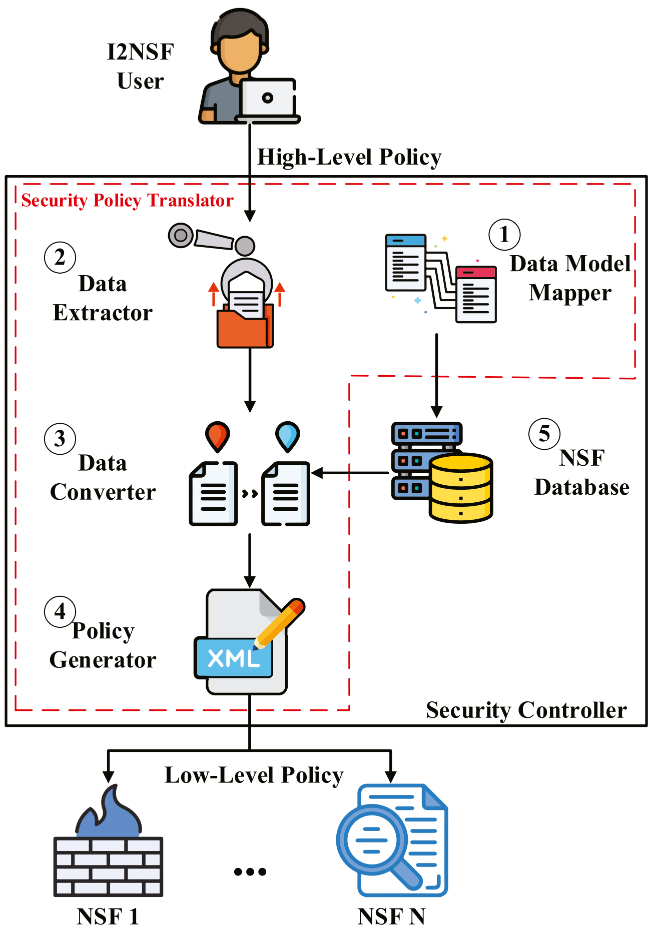


Fig. 3. Architecture of the security policy translator.

are saved in the NSF database for use in the conversion process.

2) *Data Extractor:* It verifies the high-level security policy provided by the I2NSF User and extracts the high-level data paired with the high-level elements. This component is built in accordance with the concept of the Deterministic Finite Automaton (DFA).
3) *Data Converter:* It is responsible for converting the extracted high-level contents/elements to the corresponding low-level contents/elements. It also provides policy provisioning, i.e., selects the NSF(s) that can provide the requested security service according to their capabilities. The conversion is performed based on the information saved in the NSF database.
4) *Policy Generator:* It is in charge of creating the low-level security policy in an XML/JSON form to be delivered to the selected NSF(s) for the high-level security policy.
5) *NSF Database:* It supports the SPT to collect and distribute the necessary information to the Data Converter. It holds the endpoint data (e.g., user identification and end devices' IP addresses), NSFs' capabilities, and a mapping model from the Data Model Mapper. The NSF Database must be secured and well encrypted as it contains private information and network details that may reveal the vulnerabilities of the network. The security and implementation of the NSF Database are out of the scope of this paper as the primary emphasis is put on the translation of security policies.

Algorithm 1: Data Model Mapper Algorithm.

```
1: function Map_Data_Model$H, L$ ▷ $H$ is the CFI YANG data model and $L$ is the NFI YANG data model.
2:   $List_H$ ←Linear_List($H$)   ▷ Break $H$ into a Linear List.
3:   $List_L$ ←Linear_List($L$) ▷ Break $L$ into a Linear List.
4:   for $high$ in $List_H$ do
5:     for $low$ in $List_L$ do   ▷ Loop through all combinations.
6:       $treeDist[low] \leftarrow ZSS(high, low)$   ▷ Calculate tree edit distance with Zhang-Shasha (ZSS) algorithm.
7:     end for
8:     $result[high] \leftarrow$ Keys of $min(treeDist)$   ▷ The mapping results are the lowest tree edit distance.
9:   end for
10:  return$result$
11: end Function
```

A. Data Model Mapper

In the I2NSF framework, an I2NSF User delivers the high-level security policy encoded in either XML or JSON format with the NETCONF or RESTCONF protocol. The high-level security policy follows the data model defined in the Consumer-Facing Interface YANG data model [11]. Each of the elements in the data model is used to provide different services. In order to correctly translate the high-level security policy into the corresponding low-level security policy, it is crucial to map each high-level element to at least one corresponding low-level element.

The Data Model Mapper is used to automatically generate data model mapping information, i.e., the mapping of each element between the high-level and low-level YANG data models. With the data model mapping information, this component is used in the initial phase of the SPT. It is also used when at least one of the YANG data models is updated or extended. Next, the result is saved in the NSF Database for the actual translation process.

The proposed Data Model Mapper shown in Algorithm 1 is specifically designed to work for the I2NSF Framework, with the process illustrated in Fig. 4. In the I2NSF Framework, the high-level and low-level YANG data models are designed to have similar labels for the elements with similar semantics, e.g., `source` in the high-level YANG data model and `source-ipv4-network` in the low-level YANG data model. In the case where an element of the high-level YANG data model and another element of the low-level YANG data model have similar labels, but have different semantic meanings, the context of these elements should be considered. But in the case of the I2NSF Framework, the I2NSF data models are designed such that each pair of two labels with similar names on both YANG data models

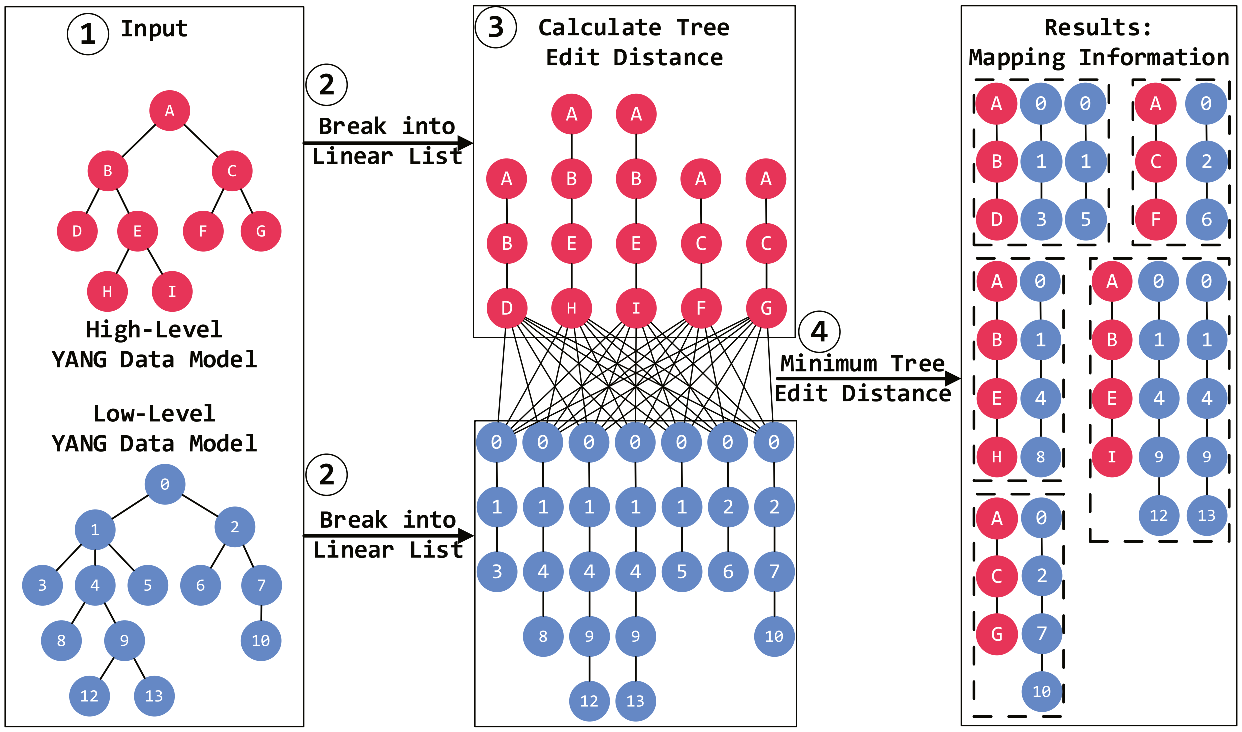

1 Input
High-Level YANG Data Model
Low-Level YANG Data Model
2 Break into Linear List
2 Break into Linear List
3 Calculate Tree Edit Distance
4 Minimum Tree Edit Distance
Results: Mapping Information

Fig. 4. Process of data model mapper.

has the same semantics properly. Hence, a text-based similarity can work well without considering the semantics within the I2NSF environment.

Moreover, it is worth noting that both the high-level and low-level YANG data models are engineered with the consideration of low cardinality. In the realm of data modeling, cardinality refers to the number of elements in a set. Considering the cardinality of these models enables a solution that is inherently efficient and frugal. During the process of data model mapping, the mapper can focus on matching semantics based on labels without being overly burdened by a vast array of elements. Hence, by leveraging these design principles, the Data Model Mapper is designed by taking advantage of the nomenclature similarity between the high-level and low-level YANG data models to automatically generate the data model mapping information.

The algorithm works with the high-level and low-level YANG data models as inputs. The YANG data models can be expressed as tree data structures. Then both the data models are split into separate linear lists, of which each consists of one leaf and its parent until the root of the tree. Next, the tree edit distance from each linear list of the high-level YANG data model to each linear list of the low-level YANG data model is computed. The tree edit distance can be calculated with the Zhang-Shasha (ZSS) algorithm, which calculates the minimum number of operations that can be done to make two trees similar. In this operation, the labels (e.g., `condition` and `source-ipv4-network` in Fig. 2) of the elements are used to determine the distance. The operations are as follows:

- *Insert:* When there are missing elements, it adds the elements.
- *Delete:* When there are excessive elements, it removes the elements.
- *Change:* When the elements are equal, it modifies the labels of the elements.

The insert and delete operations are calculated by the number of characters of the labels to be added or removed, respectively. To calculate the change operation, a distance algorithm is needed as follows. Considering the close similarity of words for the exact mapping in the design of the high-level and low-level YANG data models, the proposed approach is the *Cosine Similarity* algorithm [14].

Cosine Similarity measures two non-zero vectors which are used to measure the similarity between strings. To obtain the vector value of the elements, count-based vectorization is used [15] as the vocabularies are limited to the CFI and NFI YANG data models. The equation to calculate the string similarity is as follows:

$$similarity(h,l) = \frac{h.l}{||h||.||l||} = \frac{\sum_{i=1}^{n} h_i l_i}{\sqrt{\sum_{i=1}^{n} h_i^2}\sqrt{\sum_{i=1}^{n} l_i^2}}, \qquad (1)$$

where

- h: the label of the high-level element, and
- l: the label of the low-level element.

The equation's result is a similarity index of the two compared elements from 0 to 1, where 0 indicates no similarity and 1 indicates the exact same words. To modify it as a distance, the following equation is used:

$$dist(h,l) = (1 - similarity(h,l)) \times |h|. \qquad (2)$$

(2) returns the result as a distance value by inversing the $similarity(h, l)$ and multiplying it by $|h|$, i.e., the length of the high-level element's label. The length of h is used to normalize the values of all distances to find the lowest distance. Next, the results of the tree edit distance calculation are used to find the pair for each of the high-level elements. The result can have multiple low-level elements as the pair's partner candidates for one high-level element. Selection of the correct element is done by the Data Converter (see Section IV-C) as the value of the element needs to be considered (e.g., according to the version of IP, that is IPv4 or IPv6).

The proposed Data Model Mapper significantly reduces the need for human involvement through its dynamic mapping that can adapt to changes in the data models. This flexibility ensures that even as data models evolve, the system can autonomously adjust its data model mapping for security policy translation, while minimizing disruptions and maintaining efficiency. While the Data Model Mapper operates with minimal human intervention, there are specific instances where human expertise remains invaluable. For example, updating complex data models or re-initializing the system might still need human involvement. From the point of view of network security, this limited human intervention ensures that the system functions with precision, effectively aligning with the evolving demands of network security. Thus, the balance between automated adaptability and human expertise guarantees the system's accuracy and relevance to network security.

B. Data Extractor

Data Extractor is an SPT component built on the concept of DFA. Data Extractor's purpose is to extract the data from the high-level security policy and relay it to the Data Converter that converts it into the data for the corresponding low-level security policy. The high-level security policy is based on the CFI YANG data model [11]. To extract the data, DFA can generally be constructed as in (3). DFA follows the hierarchy of the CFI YANG data model completely and is convenient for automatic construction.

$$M = (Q, \Sigma, \delta, q_0, F), \tag{3}$$

where

- Q: $\{Accepting, Middle, Extracting\}$,
- Σ: A set of all clause names in the YANG data model,
- δ: A transition for each edge for each label in the YANG data model,
- q_0: The initial state, i.e., $Accepting$, and
- F: The final states, i.e., $Accepting$.

There are three types of internal states in DFA Data Extractor: $Accepting$, $Middle$, and $Extracting$. $Accepting$ state is the initial and final state. If the DFA finishes reading the entire policy and enters the $Accepting$ state, the high-level security policy is accepted and the extracted data is relayed to Data Converter. The $Middle$ state is used to interact with the elements that constitute the CFI YANG data model hierarchy. The $Extracting$ state is located at each leaf node position in the CFI YANG data model and the data corresponding to the leaf node field is extracted.

Algorithm 2: Data Extractor Algorithm.

```
1: function Extract_Data$DM, P$ ▷ $DM$ is the CFI YANG data model, and $P$ is the high-level security policy.
2:   $S \leftarrow Accepting$                    ▷ $S$ is for the current state.
3:   $Result \leftarrow \emptyset$
4:   while $True$ do
5:     $T \leftarrow$Read_Next_Element_XPath($P$)
6:     if $T == empty$ then
7:       return$GrammarError$          ▷ Return a Grammar Error.
8:     end if
9:     $S \leftarrow$Get_State($DM, T$)
10:    if $S == Extracting$ then
11:      $Result[T] \leftarrow Data$
12:    else if $S == Accepting$ then
13:      break                          ▷ Extraction finish
14:    else if $S == \emptyset$ then         ▷ T mismatch with DM
15:      return$GrammarError$          ▷ Return a Grammar Error.
16:    end if
17:  end while
18:  return$Result$                        ▷ Return a result.
19: end Function
```

Algorithm 2 shows the scheme of the Data Extractor. The objective of the algorithm is to extract every data from the leaf and make sure that the grammar is correct. To accomplish this, the input must include the CFI YANG data model and the high-level security policy. To properly extract every data, the algorithm reads each element individually with the Read_Next_Element_XPath(P) function. It then uses the Get_State(DM, T) function to obtain the next state, which is set to S as the current state, by matching the XPath of the XML element with the CFI YANG data model element. This function also compares items in the CFI YANG data model to the elements in the high-level security policy. If there is a mismatch, it will return a *GrammarError* because the policy does not properly correspond to the CFI YANG data model. If the element is properly matched with the data model, then it will return either the $Accepting$, $Middle$, or $Extracting$ state.

Fig. 5 illustrates the DFA transition graph constructed on the basis of the XML example given in Fig. 2(a). When the state is $Extracting$, it will save the data as a key-value pair where the element is the key and the content is the value. If the Data Extractor is in the $Middle$ state, then it will continue to process the next label. When the state returns to $Accepting$, then the Data Extractor has finished extracting all data and returns the mapping values. The result will be relayed to the Data Converter for the next process.

With this method, I2NSF developers can conveniently manage the Data Extractor component because DFA can be built automatically even if the CFI YANG data model is modified, because the connection of each DFA node follows the hierarchy of the YANG data model. As the data model is constantly updated due to both the characteristics of the standardization

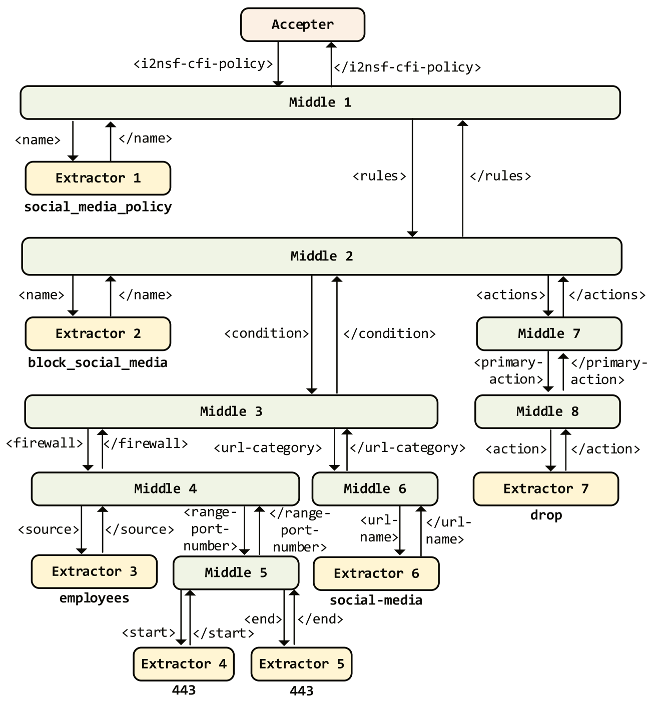


Fig. 5. Data extractor based on deterministic finite automaton (DFA).

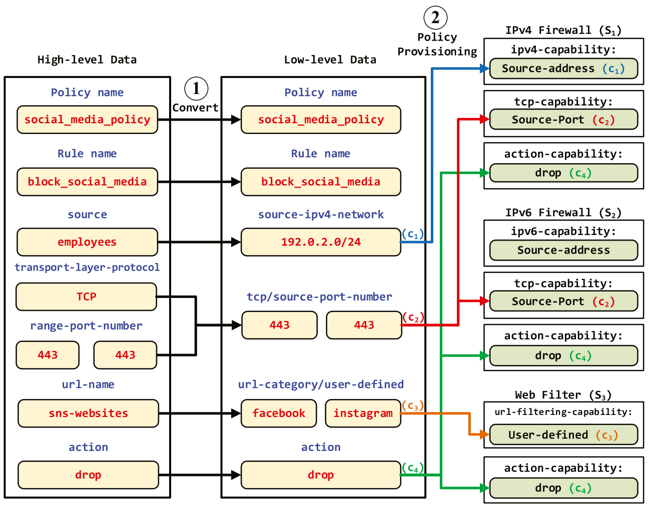


Fig. 6. Data converter with policy provisioning.

work and the changes in customer security requirements, this proposed component design can grant data model flexibility for the I2NSF developers through the reconstruction of the data model mapping.

C. Data Converter

The Data Converter converts the high-level data into the corresponding low-level data for the NSFs. The data is the key-value pairs extracted by the Data Extractor where each key represents an element and its corresponding value is content (e.g., `source` and `employees` are a pair of key and value). It is also responsible for policy provisioning, which eliminates the need for an I2NSF User to explicitly specify the target NSFs for a high-level security policy. Thus, this component provides the convenience of the translation.

To perform these tasks, the Data Converter must be well-connected to the NSF Database where the necessary information, i.e., mapping information and NSFs' capabilities, is stored. The information must be saved initially before the actual translation process. The mapping information consists of data model mapping information and endpoint data to convert the corresponding element and content, respectively. The information is obtained from the Data Model Mapper in the initial stage of the SPT, while the endpoint data can be registered by the I2NSF User with the NSF Database in the Security Controller via the CFI.

Fig. 6 illustrates converting and policy provisioning processes that are done by the Data Converter for the example given in Fig. 2. The first converting process is the conversion of elements and contents of the high-level data. The contents are converted into their corresponding contents based on the information from the NSF database, e.g., `employees` $\rightarrow$ `192.0.2.0/24`. The values of the converted content will affect the selection of the element from the data model mapping information. For example, the converted content of the element `source` can either be an IPv4 address or an IPv6 address. If the content is an IPv4 address, then the selected element will be the `source-ipv4-network` as it will hold an IPv4 address.

The second policy provisioning process is the provisioning to select the appropriate NSFs for the security policy. The selection of NSFs is based on the capabilities of the NSFs registered with the NSF Database. The capabilities of NSFs are defined in the I2NSF Capability YANG data model document [16] and registered with the NSF Database via the Registration Interface. Each low-level element is correlated to an NSF capability. The example presented in Fig. 6 shows that three different NSFs are registered with the NSF Database, i.e., IPv4 Firewall, IPv6 Firewall, and Web Filter. Each NSF has unique capabilities to handle different network traffic. The IPv4 and IPv6 Firewalls are used to filter network traffic based on the packet's IP address (e.g., IPv4 or IPv6 address, respectively), while the Web Filter is used to restrict the websites that a client can use. Therefore, the policy provisioning function is designed to satisfy the following requirements:

1) *Each element must be regulated by an NSF:* The most important requirement is to fully comply with the I2NSF User's request. If any of the elements cannot be supported by the registered NSFs, the Security Controller must request a new NSF with the missing capabilities to the DMS. If the DMS is unable to provide the NSFs for any reason, then the Data Converter will generate an error message for the I2NSF User.
2) *The selected NSFs should be optimal:* To optimize resource and network usage, the number of NSFs involved in security policy regulation should be minimized. The NSFs that are able to cover more elements should be selected.
3) *The selection time should be short:* Having a fast translator for the swift deployment of NSF as the security of the

Algorithm 3: Policy Provisioning Algorithm.

```
1: function Policy_Provisioning($convertedData, S$)  ▷ $convertedData$ contains the results of the conversion. ▷ $S$ is the set of all NSFs that have at least one capability, which constructs $U$.
2:   $U$ ←Find_Universe($convertedData$)  ▷ $U$ is the Universe, which contains the capabilities required.
3:   $nsf$ ←Find_NSFs($U, S$)  ▷ Find_NSFs($U, S$) finds a set of NSFs (denoted as $nsf$) to encompass the elements (i.e., capabilities) of $U$.
4:   $Results \leftarrow \emptyset$
5:   for $key, val$ in $convertedData$ do
6:     for $i$ in $len(nsf)$ do
7:       if $nsf[i]$ has the capability for $key$ then
8:         if $(''action''$ in $key)$ and $(i < len(nsf) - 1)$ then  ▷ The condition to activate Service Function Chaining (SFC).
9:           $Results[nsf[i]][key] \leftarrow nsf[i+1]$
10:        else
11:          $Results[nsf[i]][key] \leftarrow val$
12:        end if
13:      end if
14:    end for
15:  end for
16:  return$Results$
17: end Function
```

network is important. A slow translator may cause serious losses for the users.

With these requirements, the policy provisioning problem can be defined as a Set-Cover Problem [17] to find a minimum number of NSFs that can include all of the required elements. The following definitions are used for policy provisioning:

Definition IV.1: Let U be **Universe** set (i.e., a policy-enforcing capability set) having essential capabilities ($U = \{c_1, c_2, \ldots, c_n\}$) required for a security policy. For example, in Fig. 6, the required capabilities for a given high-level security policy are IPv4 source-address (c_1), TCP source-port (c_2), the URL (c_3) verification, and packet denial capability (c_4).

Definition IV.2: Let S_i be a **Subset** (i.e., an NSF Capability set) having the capabilities of an NSF from the NSF database. For example, Fig. 6 shows the NSF S_1 with IPv4 source-address (c_1), TCP source-port (c_2), and packet denial capability (c_4), i.e., $S_1 = \{c_1, c_2, c_4\}$. Where NSF with a corresponding subset (S_i) has the coverage of capabilities for the Universe set U.

For this optimization, we propose a Policy Provisioning algorithm as shown in Algorithm 3. The algorithm takes inputs of $convertedData$ and S, where $ConvertedData$ is the result of the previous conversion process and S is the set of all NSFs that have at least one capability that is an element in the set U. In line 2, the Find_Universe($convertedData$) function takes $convertedData$ to find the necessary capabilities as the Universe based on the capabilities defined in the I2NSF Capabilities YANG data model [16].

Algorithm 4: Set-Cover Algorithm.

```
1: function Set_Cover$U, S$  ▷ $U$ is the Universe, $S$ is the set of subsets.
2:   $X \leftarrow U$  ▷ $X$ stores the uncovered elements.
3:   $C \leftarrow \emptyset$▷ $C$ stores the subsets and their elements of the cover.
4:   while $X \neq \emptyset$ do
5:     Select $S_i$ from $S$ such that $S_i$ covers the most elements in $X$
6:     $X \leftarrow X - S_i[elements]$
7:     $C[S_i] \leftarrow S_i[elements]$
8:     $S \leftarrow S - S_i$
9:   end while
10:  return$C$
11: end Function
```

In line 3, a subfunction called Find_nsfs(U, S) is called, taking both the universe set U and the set of subsets S_i as input to find an optimal set of NSF(s) that can completely operate the requested security policy. In this paper, two approaches are used and compared to discover the best method that can find an optimal set of NSFs, i.e., *Greedy Algorithm* and *Linear Programming*.

1) *Greedy Algorithm:* As shown in Algorithm 4, the algorithm iteratively selects subsets (S_i) from S that cover elements in U [18]. Each loop selects a subset S_i that covers the most elements in U until every element in U has been covered.
2) *Linear Programming:* It maximizes or minimizes a linear objective function with linear constraints [18]. The equation to find an optimal solution is as follows:

$$minimize \quad \sum_{j=1}^{n} c_j x_j, \tag{4}$$

$$subject\ to \quad \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \qquad \text{for } i = 1, \ldots, m, \tag{5}$$

$$x_j \geq 0, \qquad \text{for } j = 1, \ldots, m.$$

In lines 6 - 16 of Algorithm 3, the $convertedData$ is looped in order to properly deliver the appropriate security policies to each NSF. In line 7, the function checks whether the current NSF has the capability for the current key. If it does, then in line 8, the function ensures that the security policies are able to activate the Service Function Chaining (SFC) [19] by checking whether the key includes the word ”action” and whether the current index is the last NSF. If so, the value for the key in the results dictionary is set to the next NSF in the set. But if one or more conditions are false, the value is set to the original value from the $convertedData$ input. When all key/value pairs are processed, the results dictionary is returned. This result will be used in the next process, i.e., Policy Generator, which is explained in Section IV-D.

The choice of NSFs is critical to timing the security of the network. The time complexity of Algorithm 3 can be expressed as $\mathcal{O}(|convertedData||S| + T(Find\_nsfs(U, S))$,

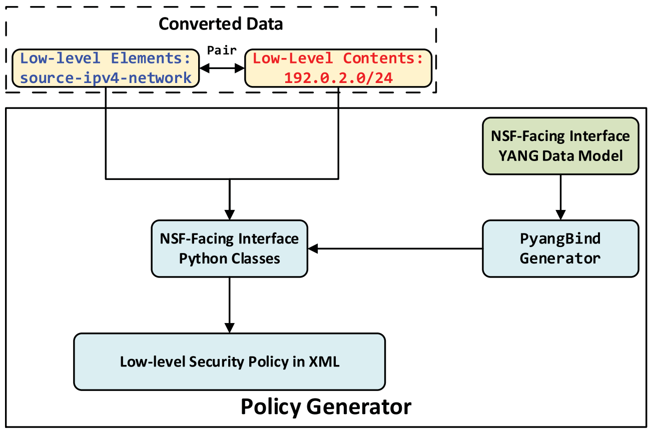

Converted Data
Low-level Elements: source-ipv4-network
Pair
Low-Level Contents: 192.0.2.0/24
NSF-Facing Interface YANG Data Model
PyangBind Generator
NSF-Facing Interface Python Classes
Low-level Security Policy in XML
Policy Generator

Fig. 7. Policy generator.

where $T(Find\_nsfs(U,S))$ is the time complexity of the function Find_nsfs(U,S). At worst, the Find_nsfs(U,S) function, which uses either *Greedy Algorithm* or *Linear Programming* approach, will have the time complexity of $\mathcal{O}((|convertedData||S|)^3)$. Overall, the time complexity for Policy Provisioning algorithm is $\mathcal{O}(|convertedData||S| + (|convertedData||S|)^3)$. However, as the number of elements and NSFs should remain within a reasonable bound, the duration of selecting the most suitable NSFS should also remain reasonable.

D. Policy Generator

The Policy Generator is the final process to completely translated the user's security policy for the NSFs. The objective of this component is to generate the corresponding low-level security policies for the NSFs based on the converted data received from the Data Converter. The low-level security policies must be separated for each NSFs in order to implement it. The low-level security policies are delivered in XML or JSON format with either NETCONF or RESTCONF protocol, and the low-level security policies conform to the NFI YANG data model [13].

Fig. 7 shows how the Policy Generator produces the low-level security policy. PyangBind [20] is used to automatically generate low-level security policies and ensure that they comply with the YANG data model. Pyangbind allows YANG data models to be used as a basis for defining Python classes, with each YANG data element being represented as a class attribute. This allows developers to use the familiar syntax and structure of Python classes to interact with data modeled in YANG, making it easier to write software that works with a YANG data model. Pyangbind is built on top of the Pyang [21] tool, which is a Python library for validating and manipulating YANG data models. Pyangbind uses Pyang to parse and validate YANG data models, and also generate Python classes based on the data model. It also includes a number of additional features and utilities for working with the generated Python classes, such as the ability to serialize and deserialize data between different formats (e.g., XML and JSON). With Pyangbind, any changes made to the YANG data model can be easily applied to the translator to generate the low-level security policy.

Algorithm 5: Policy Generator Algorithm.

```
 1: function Generator$P$          ▷ $P$ is the result of the
    Policy_Provisioning($convertedData$).
 2:   $nfi \leftarrow ietf\_i2nsf\_nsf\_facing\_interface()$   ▷
      $ietf\_i2nsf\_nsf\_facing\_interface$ is the Python
      Class Generated by PyangBind.
 3:   for $nsf, data$ in $P$ do
 4:     if $nsf$ is $null$ then
 5:       return"Error, NSF not found."
 6:     end if
 7:     for $k, v$ in $data$ do
 8:       $attr$ ←Get_Attr($nfi, k$)
 9:       Set_Attr($attr, v$)
10:     end for
11:     $result[nsf] \leftarrow pybind(nfi)$
12:   end for
13:   return$result$
14: end Function
```

Algorithm 5 shows the generation of security policies for each provisioned NSFs by utilizing PyangBind. The function takes a single argument, P, which is the result of the function Policy_Provisioning($convertedData$) in Algorithm 3. In line 2, a Python class called $ietf\_i2nsf\_nsf\_facing\_interface$ is instantiated, which is generated by PyangBind, and is assigned as an object called nfi. The nfi object is used to manipulate data that is expressed in the NFI Yang data model. Then, the function enters a loop that iterates over each item in P. For each item, nsf is the key and $data$ is the value. In line 4, the function checks if nsf is null or not. If so, the function returns a string indicating that the NSF was not found. If nsf is not null, the function enters another loop in line 7 that iterates over each key-value pair in the $data$. For each key-value pair, the function calls the Get_Attr(nfi, k) function to retrieve the Python Class attribute of k from the nfi, and saves it as $attr$. In line 9, the function calls the Set_Attr($attr, v$) function to set the Python Class attribute of k with v as the value. After processing all of the key-value pairs, the function calls the pybind(nfi) function to produce the low-level security policy in an XML or JSON format [20] and stores the result as a dictionary, using nsf as the key.

V. Performance Evaluation

In this section, the performance of SPT is evaluated to assess the effectiveness and efficiency of the translator. A set of indicators and parameters is established to measure and compare the performance of SPT against its stated goals and objectives. We analyzed every proposed component using a variety of methods, gathering a wealth of information about the program's performance and identifying areas for improvement. Table I shows the specification and environment used to evaluate the performance of SPT.

TABLE I
TEST CONFIGURATION FOR SPT

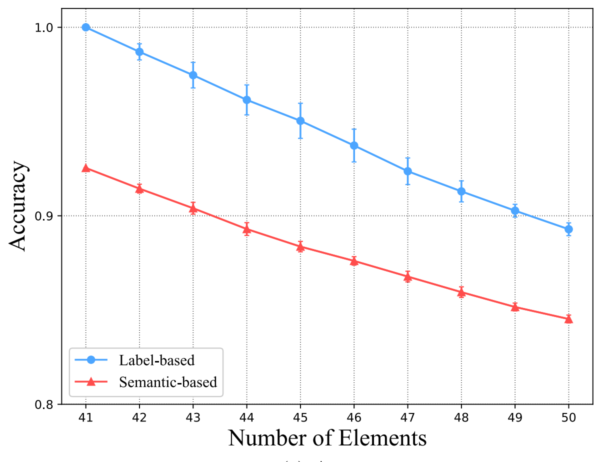
Description	Specification
CPU	Intel(R) Xeon(R) Gold 6230R CPU @ 2.10GHz
Memory	251 GiB
OS	Ubuntu 18.04.5 LTS
Python	version 3.7.4
Number of NSFs	5 ∼ 20 NSFs
Number of Elements	4 ∼ 14 elements (100 different security policies for each number of elements)
Extended Elements	41 ∼ 50 elements

A. Data Model Mapper

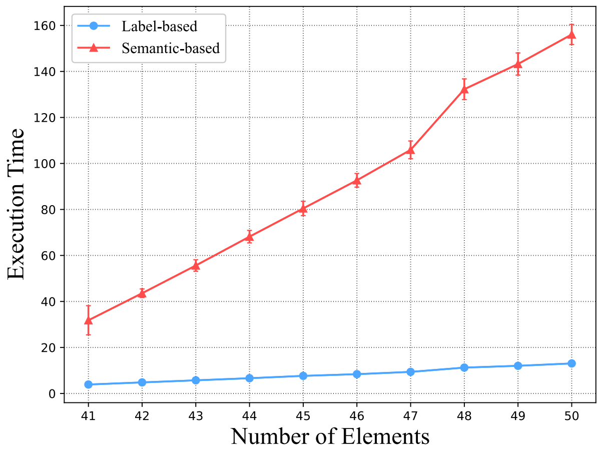
The Data Model Mapper proposed in this paper is a specific method that can be used to map the elements between the CFI YANG data model and the NFI YANG data model. The proposed dynamic method leverages the design similarity between the two YANG data models to accurately suggest a mapping model between each pair of elements in the data models. Two methods are compared to find the effectiveness of the Data Model Mapper, i.e., label-based mapping and semantic-based mapping. Label-based mapping only utilizes the labels of the elements and is calculated with Cosine Similarity explained in Section IV-A. Semantic-based mapping considers the semantics of the elements to find the mapping information utilizing Natural Language Processing (NLP) to find the semantic similarity. In the experiment, Universal Sentence Encoder [22] is used within the spaCy library [23] which allows Docs, Spans, and Tokens to be embedded directly from the Universal Sentence Encoder family.

We prepare extensions of the CFI YANG data model to find the effect of the extension on the accuracy of the proposed mapper. The extensions are created based on missing details in CFI elements compared to the NFI elements, i.e., detailed information of a packet header condition such as packet length and TTL information.

Fig. 8 presents the performance of the Data Model Mapper, emphasizing the distinction between the label-based mapping and the semantic-based mapping in mapping the elements between the CFI YANG data model and the NFI YANG data model. As shown in Fig. 8(a), the label-based mapping can map the elements with 100% accuracy for 41 elements but can map the elements with 89% accuracy for 50 elements. Thus, as the number of elements increases, the accuracy of the label-based mapping decreases gradually. On the other hand, the semantic-based mapping has lower accuracy from 94% to 83% over the number of elements from 41 to 50 than the label-based mapping. From these results, it is seen that the label-based mapping approach outperforms the semantics-based mapping. These results come from the nature of the design of the CFI and NFI YANG data models where the labels of the elements have high similarity. Furthermore, this outcome can be attributed to several factors rooted in both semantic analysis and complexity associated with mapping intricate data models.



(a) Accuracy



(b) Execution Time

Fig. 8. Impact of YANG data model extension to data model mapper.

Another issue with the semantic-based mapping with NLP is the execution time. In the field of network security, time is very critical. As presented in Fig. 8(b), the semantic-based mapping takes up to around 156 seconds with 50 extended elements in the data model, while the label-based mapping takes up to 13 seconds with 50 extended elements. This shows that the semantic-based mapping may cause a problem for the prompt defense of the network since the longer it takes to update the system, the longer the vulnerability window for the network is. This is because the semantic-based mapping delves deeply into the meaning and context of words, phrases, and sentences which will take a longer time to process each element.

Therefore, it is shown that the proposed Data Model Mapper using the label-based mapping approach can effectively perform the mapping between the CFI and NFI with better accuracy and shorter execution time in network security services.

B. Data Extractor

The proposed extraction method is based on the concept of DFA. It constructs the semantics that must be followed based on

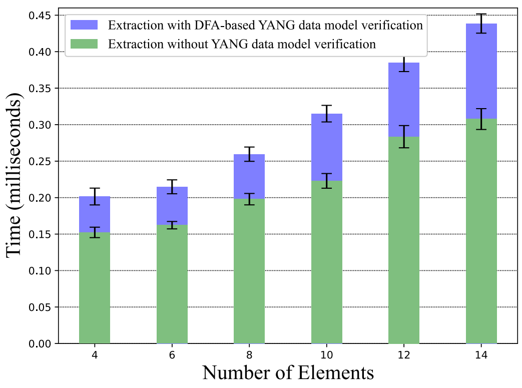


Fig. 9. Data Extractor Performance with and without YANG data model verification.

the YANG data model. This process verifies whether a high-level policy is acceptable or not according to the given CFI YANG data model. The proposed component guarantees complete extraction with 100% accuracy. To assess the performance, we compare the time it takes to extract the high-level security policy with and without YANG data model verification. A YANG data model verification validates the given high-level security policy to check whether it conforms to the CFI YANG data model or not. We also examine the impact of the number of elements to be extracted.

To evaluate the Data Extractor, we compile 100 high-level security policies for each element quantity (from 4 elements to 12 elements). Fig. 9 displays the performance of the proposed Data Extractor. It shows that the YANG data model verification increases the time of extraction by around 30% of the extraction process without YANG data model verification. However, since YANG data model verification is necessary to guarantee that the security policy is correctly deployed, it is unavoidable to spend more time on the extraction process. The number of elements also increases the operational time of the Data Extractor as more elements of the high-level security policy are used. The duration slowly increases from 0.2 milliseconds for 4 elements and increases to 0.4 milliseconds for 12 elements. The increasing number of elements does not fully impact the time duration of extraction, and the time duration is negligible to humans.

This performance evaluation shows that the Data Extractor can be used to perform the extraction with negligible time duration for the I2NSF system.

C. Data Converter

The Data Converter involves two processes, i.e., conversion and provisioning. The conversion process is handled by exchanging information with the NSF database. We omit the performance of conversion as it shows the performance of the NSF database rather than the performance of the translation. For the provisioning process, the evaluation setting is as follows:

- *Performance Metrics:* (i) *Average Time*, (ii) *Solution Quality*, and (iii) *Memory Usage* are used as metrics for the performance.
- *Approaches:* To find the best approach for our optimization, we tested (i) *Linear Programming* and (ii) *Greedy Algorithm* as optimization techniques. We use *Combination* to find the guaranteed optimal solution as the baseline.
- *Parameters:* For the performance, the impact of the *Number of NSFs* is investigated. The NSFs have different sets of capabilities to verify the integrity of the solution.

Fig. 10 displays the performance of the proposed Policy Provisioning. Fig. 10(a) shows the average time needed to find the solution. All three approaches present a longer time duration when the number of NSFs involved increases. The *Greedy Algorithm* and *Linear Programming* approaches show a slow increment when the number of NSFs increases. The *Combination* method shows exponential growth as it has to calculate every possible combination and find the best solution that can provide the security service. The *Greedy Algorithm* exhibits the shortest average execution time compared to the other approaches, while the *Linear Programming* is slower than the *Combination* if the number of NSFs is lower than 14.

Fig. 10(b) presents the solution quality generated by the three approaches. The *Combination* approach will always provide the best optimal solution as it calculates every possibility that can be used to find the solution. Similarly, the *Linear Programming* approach is also able to find an optimal solution for 100% over time for any security policy even with the increasing number of NSFs. The *Greedy Algorithm* approach shows a lower solution quality than the *Linear Programming* and the *Combination*. The *Greedy Algorithm* approach cannot perfectly provide the most optimal solution for every security policy. When there are 5 NSFs, it can provide an optimal solution for 95% over time, but the percentage decreases as the number of NSFs increases.

Fig. 10(c) shows the memory usage of each approach. The *Combination* approach exhibits a high memory usage when there are more than 10 NSFs involved. Whereas the memory usages of the *Greedy Algorithm* and *Linear Programming* approaches show parallel results which are lower than the *Combination* approach. The *Linear Programming* shows slightly higher memory usage than the *Greedy Algorithm* approach. Both of the approaches increase at the same rate with the increasing number of NSFs.

Overall, to find an optimal solution for Policy Provisioning, the best approach is to use the *Linear Programming* as it can provide the most balanced solution with negligible operational time and low memory usage even with a larger number of NSFs, even though the *Greedy Algorithm* approach is able to quickly find a solution.

D. Policy Generator

The Policy Generator utilizes PyangBind to generate the low-level security policies either in an XML or JSON format. To evaluate the performance difference between the generated security policies in an XML and JSON format, we use the converted high-level security policies and measure the average time it takes to generate the low-level security in XML and JSON format. We also measure the impact of elements' quantity on the performance of the Policy Generator.

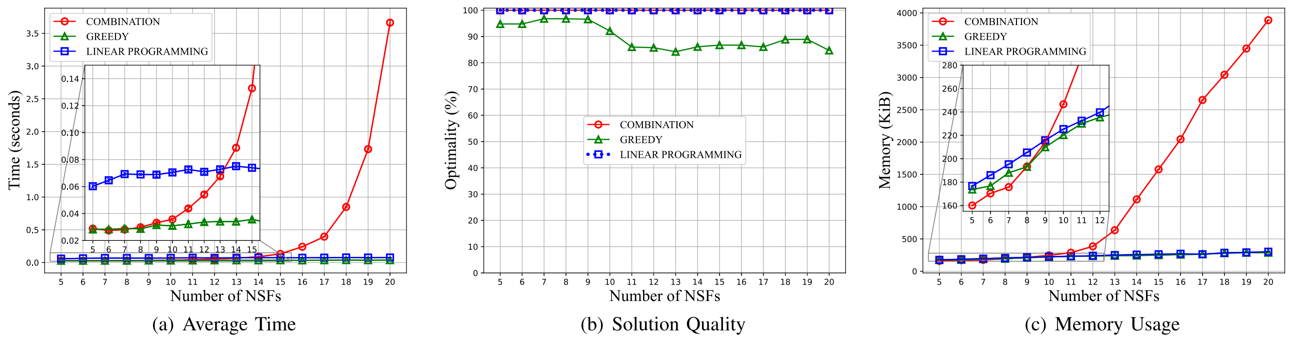


Fig. 10. Impact of NSFs' quantity on policy provisioning performance.

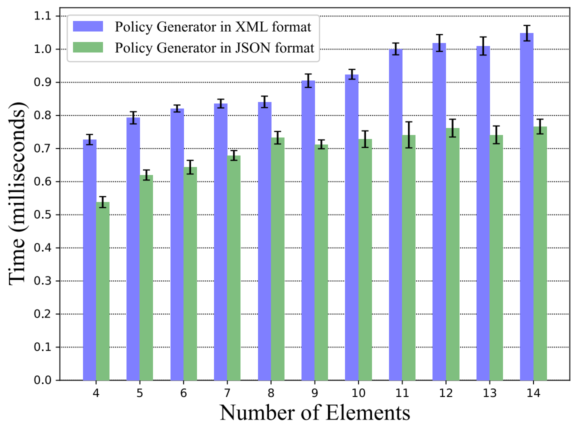


Fig. 11. Policy generator performance.

Fig. 11 displays the average time for generating the low-level security policies in an XML and JSON format. Based on the result, both of these formats are affected by the addition of elements, with the average time increasing steadily with a larger number of elements. From the figure, it is clear that the Policy Generator can perform better when generating security policies in a JSON format. The average time for the JSON format starts from around 0.5 milliseconds to 0.7 milliseconds for 4 elements to 14 elements, while the average time for the XML format is around 0.7 milliseconds to 1 ms. Overall, the performance of the Policy Generator in an XML and JSON format is acceptable to generate low-level security policies as it is negligible.

E. Discussion

The proposed Security Policy Translator is used to translate a high-level security policy to the corresponding lower-level security policy in an I2NSF Framework. The proposed method is a specific approach that is designed to work specifically for the I2NSF Framework to achieve accurate automatic translation. This approach effectively bridges the gap between abstract high-level policy requirements and the complexities associated with low-level device configurations. Automated translations minimize the risk of misconfigurations, which are often a significant source of network vulnerabilities. Consequently, the network becomes more robust and less susceptible to security breaches, thereby bolstering overall system reliability.

In terms of the quality and accuracy of translated results, the proposed methods designed specifically for the I2NSF Framework play a pivotal role. The specificity of the method ensures that the translations are not generic but precisely aligned with the requirements of the I2NSF architecture. This approach enhances the accuracy of translations, as it takes into account the unique features and intricacies of the framework. As a result, the translated security policies align perfectly with the intended configurations, which suggests that the proposed scheme can be utilized in real-world scenarios.

Overall, the proposed Security Policy Translator is a significant upgrade designed for the I2NSF Framework. It embodies a paradigm shift towards Network Management Automation, which seamlessly translates abstract high-level security policies into accurate and reliable low-level device configurations. This transformation not only ensures the network's security but also significantly improves its overall reliability, making it a foundation for secure, efficient, and dependable network infrastructures.

VI. Conclusion

In this article, we propose a Security Policy Translator (SPT) for Network Security Functions (NSFs) in cloud-based security services. We use a standardized framework developed by the Interface to Network Security Function (I2NSF) Working Group to control and manage network security services. The proposed translator allows I2NSF User to protect their networks without the need for network security knowledge by providing a high-level security policy. The proposed translator is able to extract the high-level security policy by constructing Deterministic Finite Automaton (DFA) based on the standardized Consumer-Facing Interface YANG data model. It then converts the extracted data to a lower-level form which is done by utilizing a data model mapper between the high-level YANG data model and the low-level YANG data model. It also selects the optimal NSFs that can realize the requested security policies. Finally, it generates the low-level security policies in either XML or JSON format to be deployed to the selected NSFs. We have evaluated SPT and showed that it is able to perform well as an automated translator.

As future work, we will extend our SPT to support natural language processing (NLP) to allow I2NSF User to request security services in a more convenient way. We also want to extend the SPT to perform in a less specific networking area, e.g., routing devices (e.g., BGP gateways) and 5G core networks for better networking management.

ACKNOWLEDGMENT

Figs. 1 and 3 have been designed using images from Flaticon.com.

REFERENCES

[1] S. Kemp, "Digital 2022: April global statshot report," 2022. [Online]. Available: https://datareportal.com/reports/digital-2022-april-global-statshot
[2] Corvus, "Survey findings: SMB cyber readiness," 2022. [Online]. Available: https://insights.corvusinsurance.com/cyber-risk-insight-index-q1--2022/survey-findings-smb-cyber-readiness#
[3] 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
[4] M. Björklund, "The YANG 1.1 data modeling language," RFC 7950, Aug. 2016. [Online]. Available: https://www.rfc-editor.org/info/rfc7950
[5] 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, Dec. 1989.
[6] A. Leivadeas and M. Falkner, "A survey on intent based networking," *IEEE Commun. Surv. Tut.*, vol. 25, no. 1, pp. 625–655, First Quarter 2023.
[7] J. Kim et al., "IBCS: Intent-based cloud services for security applications," *IEEE Commun. Mag.*, vol. 58, no. 4, pp. 45–51, Apr. 2020.
[8] K. Abbas, T. A. Khan, M. Afaq, and W.-C. Song, "Network slice lifecycle management for 5G mobile networks: An intent-based networking approach," *IEEE Access*, vol. 9, pp. 80128–80146, 2021.
[9] R. Enns, M. Björklund, A. Bierman, and J. Schönwälder, "Network Configuration Protocol (NETCONF)," RFC 6241, Jun. 2011. [Online]. Available: https://www.rfc-editor.org/info/rfc6241
[10] A. Bierman, M. Björklund, and K. Watsen, "RESTCONF protocol," RFC 8040, Jan. 2017. [Online]. Available: https://www.rfc-editor.org/info/rfc8040
[11] 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-31, May 2023. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-i2nsf-consumer-facing-interface-dm/
[12] S. Hyun, J. P. Jeong, T. Roh, S. Wi, and P. Jung-Soo, "I2NSF registration interface YANG data model for NSF capability registration," Internet engineering task force, internet-draft draft-ietf-i2nsf-registration-interface-dm-26, May 2023. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-i2nsf-registration-interface-dm/
[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-29, Jun. 2022. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-i2nsf-nsf-facing-interface-dm/
[14] F. Rahutomo, T. Kitasuka, and M. Aritsugi, "Semantic cosine similarity," in *Proc. 7th Int. Student Conf. Adv. Sci. Technol.*, vol. 4, no. 1, 2012, Art. no. 1.
[15] A. Kedia and M. Rasu, Hands-on python natural language processing: explore tools and techniques to analyze and process text with a view to building real-world NLP applications. Packt publishing, 2020. [Online]. Available: https://books.google.co.kr/books?id=_tmbzQEACAAJ
[16] S. Hares, J. P. Jeong, J. T. Kim, R. Moskowitz, and Q. Lin, "I2NSF capability YANG data model," Internet engineering task force, Internet-draft draft-ietf-i2nsf-capability-data-model-32, May 2022. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-i2nsf-capability-data-model/
[17] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms, ser. Mit Electrical Engineering and Computer Science*. Cambridge, MA, USA: MIT Press, 2001. [Online]. Available: https://books.google.co.id/books?id=NLngYyWFl_YC
[18] V. V. Vazirani, *Approximation Algorithms*. Berlin, Germany: Springer, 2010.
[19] J. M. Halpern and C. Pignataro, "Service function chaining (SFC) architecture," RFC 7665, Oct. 2015. [Online]. Available: https://www.rfc-editor.org/info/rfc7665
[20] R. Shakir, "pyangbind," 2018. [Online]. Available: https://github.com/robshakir/pyangbind
[21] M. Bjorklund, "pyang," 2017. [Online]. Available: https://github.com/mbj4668/pyang
[22] D. Cer et al., "Universal sentence encoder," 2018, *arXiv: 1803.11175*.
[23] M. Mensio, "Spacy - universal sentence encoder," 2023. [Online]. Available: https://github.com/MartinoMensio/spacy-universal-sentence-encoder

Patrick Lingga (Student Member, IEEE) received the BS degree from Bandung Institute of Technology, Indonesia, in 2019. He is currently working toward the PhD degree in the Department of Electrical and Computer Engineering with Sungkyunkwan University since Fall, in 2019. His PhD degree advisor is professor Jaehoon (Paul) Jeong. His major was Telecommunication Engineering in the Department of Electrical Engineering and Informatics. His research interests include Software-Defined Networking (SDN), Network Functions Virtualization (NFV), Intent-Based Networking (IBN), and 5 G Networks.

Jaehoon (Paul) Jeong (Member, IEEE) received the BS degree from the Department of Information Engineering, Sungkyunkwan University, the MS degree from the School of Computer Science and Engineering, Seoul National University, in Korea, in 1999 and 2001, respectively, and the PhD degree in the Department of Computer Science and Engineering, the University of Minnesota, in 2009. He is an associate professor in the Department of Computer Science and Engineering with Sungkyunkwan University, in South Korea. His research areas include Internet of Things (IoT), Network Security, Software-Defined Networking (SDN), Network Functions Virtualization (NFV), Intent-Based Networking (IBN), 5 G Networks, and Indoor Localization. He is a member of ACM and the IEEE Computer Society.

Jinhyuk Yang (Student Member, IEEE) received the MS degree from the Department of Electrical and Computer Engineering of Sungkyunkwan University, in August 2019. His advisor was professor Jaehoon (Paul) Jeong. His research interests include Network Functions Virtualization (NFV), Intent-Based Networking (IBN), and Natural Language Processing (NLP).

Jeonghyeon Kim (Student Member, IEEE) received the BS degree from Pusan National University. He is currently working toward the PhD degree in the Department of Computer Science and Engineering with Sungkyunkwan University, South Korea since spring 2021. His PhD degree advisor is professor Jaehoon (Paul) Jeong. His research interests include Software-Defined Networking (SDN), Network Functions Virtualization (NFV), Intent-Based Networking (IBN), 5G Networks, Cloud Native Computing, and Indoor Localization.