Computer Networks 214 (2022) 109109

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Review article

A deterministic approach for extracting network security intents

Rafael Hengen Ribeiro [a,*], Arthur Selle Jacobs [b], Luciano Zembruzki [b], Ricardo Parizotto [b], Eder John Scheid [a], Alberto Egon Schaeffer-Filho [b], Lisandro Zambenedetti Granville [b], Burkhard Stiller [a]

[a] Communication Systems Group CSG, Department of Informatics IfI, Universität Zürich UZH, Switzerland
[b] Computer Networks Group, Institute of Informatics, Federal University of Rio Grande do Sul, Brazil

ARTICLE INFO

Keywords:
Network management
Intent-based networking
Intents
Programming languages

ABSTRACT

Intents brought significant improvements in network management by the use of intent-level languages. Despite these improvements, intents are not yet fully integrated and deployed in most large-scale networks. As a result, network operators may still experience problems when deploying new intents, for instance, learning a vendor-specific language to understand previously deployed configurations of a network device. Additionally, traditional configurations are distributed across multiple devices, each configured using low-level, vendor-specific languages. As a result, inferring intents from these low-level configurations is a time-consuming process. Furthermore, current solutions for deriving high-level representations from bottom-up configuration analysis do not provide results as intents or have a very limited scope, missing essential details that enhance the representation.

In the solution to these shortcomings, a deterministic bottom-up approach was developed to extract intents from network configuration files, which translates them into a high-level intent-defined language. By parsing security configurations from various network devices and translating them into an extended version of the Nile (Jacobs et al. 2018) language, an intent-defined language, the prototype demonstrates the concept of this approach. While three case studies illustrate the effectiveness of the approach proposed in real-world scenarios, additional evaluations exploit dumps of real-world firewall and Network Address Translator (NAT) configurations consisting of rules from different servers and institutions. These evaluations demonstrate that the proposed solution can represent configurations at an intent-level language, maintaining high accuracy while representing key details of low-level configurations.

1. Introduction

The tasks of implementation, configuration, and troubleshooting require within traditional networks technically skilled network operators to provision and manage multivendor networks [1]. This need for a constant manual intervention leads to high operational costs, *e.g.*, ZK Research [2] states that 78% of IT budgets are spent on maintaining current environments. Novel technologies, such as Software-Defined Networking (SDN), emerged as viable alternatives, increasing network flexibility and simplifying network management by allowing operators to program the control plane and manage multiple devices in a centralized manner. However, programmable networks still require a highly skilled person to maintain the network's operational level [3], unless higher-level abstractions are provided.

High-level abstractions for network management can be provided by using intents, which determine abstract declarations written by network operators to specify the desired network behavior [4,5] inside an Intent-Based Networking (IBN) management system. IBN is a networking approach in which network configuration and maintenance are based on users' and applications' needs, not specific device-level configurations [6]. With IBN, there is no need to set up each router and switch individually, *e.g.*, manually configuring routing protocols or an ACL on a group of routers. Rather than that, configurations are applied automatically to all network device groups based on the user-provided intent for how the network should operate. For instance, in IBN, a user can write a high-level intent such as "block video streaming" instead of configuring several devices in order to perform this action. An IBN tool, such as Nile [7] or Jinjing [8], then is responsible for refining the intent and transforming it into actual configurations that are deployed into network devices.

* Corresponding author.
E-mail addresses: ribeiro@ifi.uzh.ch (R.H. Ribeiro), asjacobs@inf.ufrgs.br (A.S. Jacobs), lzembruzki@inf.ufrgs.br (L. Zembruzki), rparizotto@inf.ufrgs.br (R. Parizotto), scheid@ifi.uzh.ch (E.J. Scheid), alberto@inf.ufrgs.br (A.E. Schaeffer-Filho), granville@inf.ufrgs.br (L.Z. Granville), stiller@ifi.uzh.ch (B. Stiller).

https://doi.org/10.1016/j.comnet.2022.109109
Received 3 December 2021; Received in revised form 9 June 2022; Accepted 11 June 2022
Available online 17 June 2022

The employment of specification languages closer to natural languages has been exploited in IBN environments to define and deploy network intents, *e.g.*, Nile [7] and Jinjing [8], thereby improving network management and enabling the rapid deployment of new features. However, current IBN solutions focus on top-down approaches, specifying network-wide configurations as intents and refining the generation of corresponding low-level configurations, thus, indicating that they are only applicable to IBN-enabled networks.

The effort required to integrate traditional networks' configurations into IBN environments is not only challenging, but often leads operators to reject the use of modern intent solutions. Some efforts were made to interpret low-level configuration directives and express them in higher levels of abstractions, *e.g.*, in the context of security, where solutions are capable of interpreting low-level firewall configurations and representing them using higher-level abstractions, such as tables [9], symbolic models [10], and graphs [11]. These representations can assist operators in comprehending firewall rules by bottom-up parsing low-level configuration and representing firewall rules in a vendor-independent fashion. However, such abstractions are not yet "intent-level" representations and require skilled operators to know very specific firewall configuration details. Only a few solutions address the problem of helping operators migrate already deployed network rules into an IBN environment, such as Anime [12], which shows a non-deterministic behavior. Generating an intent-level representation of previously deployed network configurations helps operators reduce necessary effort to understand current network rules and enable reasoning about previously deployed intents, thus, facilitating the migration to modern intent-based environments.

As of today, only a few strategies can express the network behavior of already deployed configurations into an intent-level language, and they are limited to the forwarding behavior, for instance, not capable of representing security intents. Given this lack of solutions, this work develops a deterministic bottom-up approach for parsing and representing network security configurations at an intent-level language. The solution proposed can (1) interpret rules from low-level network configuration files, (2) create an intermediate model, and (3) extract high-level intents that are translated to the Nile language. A prototype implemented on top of SCRIBE validates the concept, and a respective tool developed interacts with third-party tools to parse vendor-specific configurations and combine their output with complementary information to display network rules in an intent-level language. As input, the system reads configuration files exported from various systems, processes Access Control List (ACL) and Network Address Translation (NAT) rules, and refines them by grouping related characteristics, such as Internet Protocol (IP) source and destination pairs as well as network services. Finally, these intents are supplemented with additional information in order to translate them to corresponding Nile intents and present them to operators.

Three case studies demonstrate the feasibility of the proposed approach: (*i*) a small business's typical infrastructure; (*ii*) a high-speed campus network with a science Demilitarized Zone (DMZ) design pattern; and (*iii*) a network of IP security cameras. Each case study demonstrates the solution's ability to describe network behavior using intents, which closely resemble natural language. Its evaluation is performed using datasets from a public repository [10] containing firewall rules from multiple servers and institutions demonstrates the translation's accuracy and functionality, as results demonstrate that network configurations can be displayed with high fidelity at the intent level, capturing low-level details present in input configuration files.

The remainder of this paper is structured as follows. While Section 2 describes related work, Section 3 details the end-to-end methodology to extract intents from existing network configurations. Based on the essential details of the implementation within Section 4, Section 5 studies the three case in the context of real-world scenarios, followed by the evaluation to demonstrate accuracy and functionality in reality. Finally, Section 6 summarizes the work and outlines future steps possible.

2. Related work

This section presents the concepts of IBN and the state-of-the-art in bottom-up approaches to derive network configurations

2.1. Intent-based networking (IBN)

IBN has emerged as cutting-edge technology for network automation and orchestration that has the potential to alter existing network architectures and technologies fundamentally. It is a networking approach capable of automatically converting, deploying, and configuring network resources according to user requirements to the operator's intentions. This strategy is capable of overcoming and controlling unexpected events or failures. Additionally, it can continuously monitor network resources and collect statistics for life cycle management [5]. Numerous industrial organizations have developed IBN-based systems, with the Internet Engineering Task Force (IETF) being responsible for the standardization of intelligent IBN systems.

Intents enable network applications or practitioners to specify what the network should do at the policy level rather than how the network should achieve the desired configuration [5]. Because the low-level implementations of intents may react and change in response to network changes or practitioner requests, intents have a lifecycle [7]. They can be succinctly modeled as intent state machines. Intent frameworks, such as ONOS Intent Framework, can react to any changes at the low level that affect high-level intents, and the coordination of intents is accomplished by monitoring any relevant state changes induced by apps or the network. For example, the intent manager receives a topology change event and determines whether any installed intents are affected; in that case, the intents will be recompiled and reinstalled. Additionally, intent managers check whether state changes enable previously failed intents to be installed successfully. Finally, intent managers can install and remove policies derived from intents as practitioners or applications require. While IBN efforts such as Lumi [13] enable natural language representation of high-level intents, they are not able to integrate existing configurations from traditional networks into the IBN environment.

2.2. Bottom-up approaches and IBN languages

Numerous configuration synthesis solutions can precisely generate a logical model from given input configurations, reproducing their characteristics and interactions [14,15]. However, they still cannot generate configurations in high-level representations, such as intents. These efforts enable the efficient reproduction of low-level configuration characteristics in higher-level platform-independent representations such as abstract topologies [15], abstract models [14], or a tabular representation [9]. Batfish [14], for instance, can reproduce data and plane models. This approach takes configuration files as input and generates the control plane model followed by the generation of the data plane using an environment and the previously generated control plane model, which consists of the up/down status of each link in the network and a set of route announcements for Border Gateway Protocol (BGP) synthesis.

Abstract models are required because debugging raw configurations requires separate frameworks for each configuration, which becomes impractical and costly. Operators can detect errors and ensure the accuracy of planned or current network configurations for traditional routers using Batfish's model (*e.g.*, Cisco IOS or Juniper JunOS). Additionally, by simulating the data and control plane behavior, Batfish enables the analysis of forwarding decisions and error checkings, such as the absence of black holes or loops.

In the context of network security, approaches focusing on firewalls and NAT exist to derive high-level and platform-independent models from low-level configurations. For instance, previous work such as [11,16] provides a high-level representation of filtering behavior in

firewalls representing its policies using graphs and vendor-independent languages. Recently, a work called FireWall Synthesizer (FWS) [9] outlined a formal characterization of firewall and NAT configurations. FWS parses firewall rules from various firewall utilities (*e.g.*, `iptables`) and converts them to a tabular format. FWS employs first-order logic predicates to determine which packets are accepted by the firewall and supports NAT redirection behavior. The resulting table contains a *flattened* representation of firewall and NAT rules, *i.e.*, an unordered representation of firewall rules. Additionally, FWS includes a language that enables the user to import vendor-specific rules from various firewalls and perform queries for specific IP addresses, ports, and networks. By displaying all configurations in a vendor-independent manner, the tabular representation helps users in comprehending firewall rules. This representation, however, still requires expert knowledge of firewall tables and NAT. Additionally, for distributed firewall configurations, this solution separates each device configuration into a separate table, with each row representing a single device's rules. Due to a large number of tables, operators may have difficulty reasoning about rules deployed in a network-wide context.

Bottom-up approaches are seen rarely to expresses the network's behavior using intent-level languages. For instance, Net2Text [17] and Anime [12] provide summaries of the forwarding behavior. Net2Text follows a bottom-up solution to summarize the forwarding behavior and represent it in natural language, which allows users to interact with a "chatbot", allowing them to query for a specific forwarding behavior for which the system replies with the forwarding state of the query specified. However, while this summarizes network-wide forwarding state, producing succinct summaries in natural language, it is limited to forwarding behavior for Wide Area Networks (WAN), not supporting, for instance, ACL and NAT. Anime determines a framework to infer high-level intents by mining commonalities among the network's forwarding behavior and can infer high-level intents from a provided configuration. Anime's behavior is non-deterministic and the context is limited to forwarding behavior of the WAN.

More recently, the bottom-up approach SCRIBE [18,19] was proposed to infer high-level intents from network configurations. The prototype takes raw configuration files as input, creates an intermediate model, and derives high-level intents that are translated to the Nile language. Although SCRIBE parses network configurations and represents them using intents, it is limited to very simple allow rules for firewalls and elementary NAT operations, not representing complex and realistic firewall and NAT configurations at all. Furthermore, the Policy Intent Inference (PII) system [20] was proposed to bridge the semantic gap between underlying networks and policy management solutions. PII's intent inference scope is also applicable to network-wide functions and services, aiming to onboard legacy and non-intent native networks onto IBN platforms.

Research efforts on intents being focused on creating top-down approaches, such as Nile [7], Lumi [13], INSpIRE[21], or Jinjing [8] lead to operators dictating the behavior of the network through intents. However, they do not provide a mechanism being capable of translating already deployed low-level configurations into an intent-level language.

Currently, the effort required to integrate traditional network configurations into IBN environments is not only difficult, but currently discourages operators from implementing modern intent solutions. As in production grade networks operators need to update their security configurations and detect possible errors in already deployed ones [8], additional research is required to close this gap and integrate intent-based solutions into established traditional networks. Thus, the focus of this paper is precisely on the absence of solutions for presenting existing configurations in an intent-level representation.

3. Solution overview

The proposed solution consists of a bottom-up approach to synthesize network configurations, create the abstract model, and represent configurations into an intent-level language, which built on top of existing work parsing low-level configurations and creating an intermediate representation. SCRIBE [18] is used as a base for the newly developed solution to represent complex firewall rules of "allowing" and "blocking" instead of the SCRIBE's default-block policy, which only foresees "allow" rules. This approach does represent complex and realistic configurations, including ranges and exceptions, which are displayed in an intent-level language that extends Nile [7].

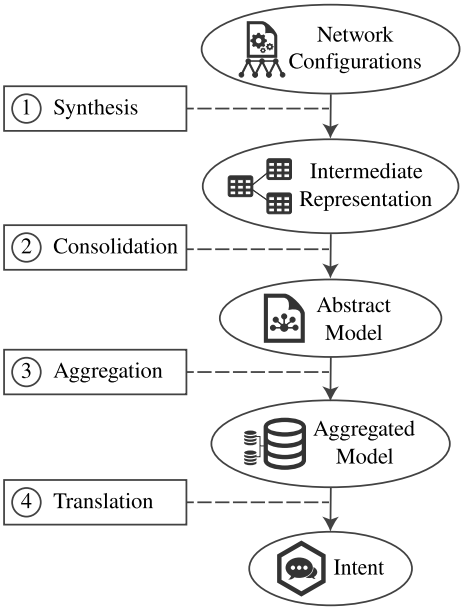


Fig. 1. Process of intent extraction.

3.1. Process

The process of intent extraction is divided into four steps (*cf.* Fig. 1): First, the process expects configuration files exported from networked devices, specifically firewalls and NAT boxes, as input. Based on these configuration files as of step ①they are stored in a centralized database and, posteriorly, synthesized. During step ②entities representing parts of the configuration are exported and gathered to generate a platform-independent abstract representation, called "meta-intent". Step ③enriches meta-intents with complementary information provided by various sources. The final result here reaches an enriched representation of network configurations called "aggregated meta-intents". Finally step ④translates these aggregated meta-intents into the extended version of Nile.

Configuration synthesis

The synthesis step ①collects low-level configuration directives from various configuration files, resulting from configurations exported off multiple devices at different levels of the network hierarchy. This step parses the output of configurations exported from these devices and gathers these along with the network topology and aliases. Each configuration file consists of various rules and parameters. For instance, a rule may be composed of source and destination IP addresses, network ports, forwarding rules, and stateful elements. Finally, given that network devices can be distributed in multiple levels of the network topology, the acquisition of rules is performed, maintaining all configurations centralized in the same database for future exportation.

Step ①is performed by a Synthesizer and an Exporter (*cf.* Fig. 2). The Synthesizer is responsible for collecting and gathering configuration files and synthesize configurations into an intermediate representation. The synthesis must be low-level and detailed in order

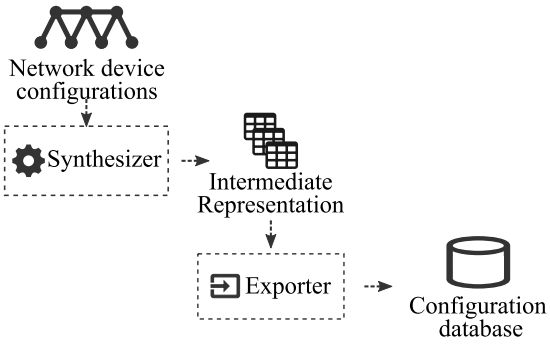


Fig. 2. Configuration synthesis.

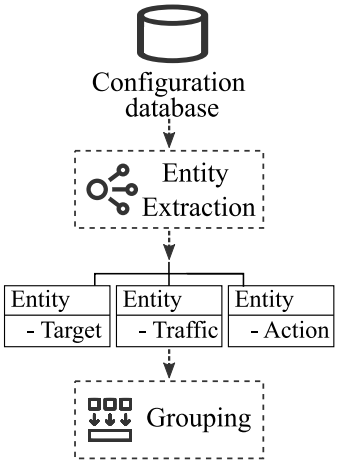


Fig. 3. Consolidation.

to create a trustworthy representation that can reproduce all aspects of configurations and their interactions. The information is leveraged from multiple sources in the network and exploits existing literature solutions to extract such information in a structured format. Third-party synthesizers are used as a part of the solution to avoid the construction of a parser for each vendor-specific language, which is subject to future optimizations. However, since there is no globally accepted single solution, the solution developed parses and aggregates configurations from multiple input sources.

The last step of the synthesis involves collecting and putting configurations generated by synthesizers into a centralized database. This process is run by the Exporter component that receives configurations as outputs from the Synthesizer and exports them into a centralized configuration database.

Consolidation

Consolidation raises the network model's abstraction level with the representation into an intermediate format (*i*) to extract from an intermediate representation of higher-level entities that represent parts of the configuration, (*ii*) to group these entities by similar characteristics, and (*iii*) to generate the meta-intent representation. Fig. 3 indicates the pipeline of substeps, within which the Entity Extraction is responsible for loading configurations from the database and generating network entities. The Grouping step compacts the set of rules. The meta-intent Exporter generates the output in a meta-intent format.

Within entity extraction, a configuration is defined as a composition of entities and is divided into three parts. Each part of the configuration is represented by a distinct entity that can be designated as Target, Traffic, or Action. Formally, an entity is defined as $E = \{Target,\ Traffic,\ Action\}$. Entities represent a set of objects belonging to different categories. An entity can be described as a class, representing a set of attributes that a Firewall contains. A property, on the other hand, defines an individual object represented by the pair <key, value>. An example of property can be a *source* of a firewall rule.

The $Target$ represents a set of objects composed of sources and destinations of a determined rule. A Target entity is specialized to represent network target objects in different scenarios and composed out of a set of devices in a traditional network, where the rule will be applied. For example, a common firewall rule for blocking traffic contains the source and destination of the traffic that will contain parameters belonging to a Target. For instance, source and destination prefixes can be modeled by $Target := \langle src.prefix,\ dst.prefix \rangle$.

The $Traffic$ entity represents the traffic type in a network. The traffic type differentiates between network services. For instance, service types can be distinguished by Transmission Control Protocols (TCP) and User Datagram Protocol (UDP) and the port used by each service. For example, in firewall and NAT contexts, source and destination ports and the protocol (TCP or UDP) compose the Traffic entity for traffic filtering, differentiating between traffic types in the network.

The $Action$ represents an operation to be executed for a pair $\langle Target,\ Traffic \rangle$. Operations consist of a behavior belonging to a network function, such as a forwarding in a NAT box or a block rule in a firewall. Each action contains the parameters needed to run the desired operation. The set $P = (p_1,\ p_2,\ ...,\ p_n)$ represents parameters of a given Action. For example, a NAT forward operation can be modeled as an action $Action$ = `{forward}` and includes a set of parameters, such as $p_1 = \text{DNAT}_{\text{IP}}$ and $p_2 = \text{DNAT}_{\text{Port}}$, respectively, indicating the destination NAT IP and port for a specified traffic.

Finally, to group entities that produce the same intent *grouping* is performed to search through configurations extracted and to match pairs of entities by similar characteristics through grouping functions. The grouping is defined as a function composed of entities $g : E \times E \mapsto E$. The grouping is performed as an one-level operation and does not consider the type of entities involved in this process.

The grouping process is defined by set operations. Formally, S represents a set of rules containing network properties, each of them represented by p_i. Given a set S, a property p_i is selected as a varying parameter v_i and a subset G containing the $length(S) - 1$ continues as remaining properties. For a given set $S = \{p_1, p_2,\ ...,\ p_n\}$, where n is the set size, to each v_i a property of set S is attributed, as instance $v_1 = p_1$, $v_2 = p_2$, ..., and $v_n = p_n$. For each v_i, a superset V is created, containing the property v_i and the subset $G = \{S \setminus v_i\}$. In this way, G derives the groups $V_1 = \{v_1, S \setminus v_1\}$, $V_2 = \{v_2, S \setminus v_2\}$, ..., $V_n = \{v_n, S \setminus v_n\}$.

The grouping process consists of searching for all sets generated of V, grouping entities V with similar G subsets, and gathering them by evidencing the varying parameter v_i. These sets G for each V set are compared and grouped, if they are equal. Take V_i and V_j as an instance. If the subset $G_i \in V_i = G_j \in V_j$, they are grouped in a new set I and include in that set the varying parameter, so the final grouped set is $I = \{G_i, v_i, v_j\}$.

The last step of the consolidation generates meta-intents being composed of pairs of information, such as origin, destination, services, and the default action for a given rule in a well-structured format. Meta-intents show high-level expression, much closer to an intent. They can easily be represented by common data-interchange formats, such as JSON (JavaScript Object Notation). After grouping sets, the Meta-intent Extractor traverses them and maps values of V_i sets to the intermediate representation meta-intent. This high-level representation maps all values from sets in the form `{''Entity.Property'' : ''value''}`. For each $p_i \in E_i$, where p_i is a property and E_i an entity, the meta-intent extractor will map to an entry {"$E_i.p_i$":"value"}.

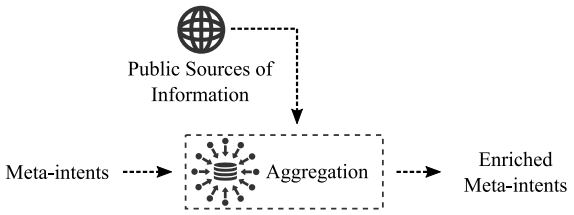


Fig. 4. Aggregation.

Aggregation

While meta-intents were generated as an intermediate structured representation, they contains low-level information, such as IP addresses. Thus, all meta-intents generated in the consolidation step are gathered and enriched with complementary information from various sources. This step displays the information needed at a user-friendly level.

The aggregation (*cf.* Fig. 4) collects hostnames and topology data as a piece of additional information from the network to enrich meta-intents, featuring an option to input "friendly" names for hosts that help users identify network devices. With all complementary information gathered, IP addresses are replaced with names provided and a query to the Service Name and Transport Protocol Port Number Registry [22] is performed to infer the service or protocol name according to ports provided.

Intent translation

After enriching meta-intents with complementary information, the next step translates resulting intents to Nile, the target language. Thus, it turns easy for users to understand the most common operations currently supported by networks. The Nile grammar's symbols allows for the representation of network security rules. Most of these enriched intents can be directly mapped to Nile symbols. Initially, the *Target* entity is mapped to the Nile ⟨from_to⟩ symbol. The Nile symbol ⟨from_to⟩ expects two *endpoints* as parameters. The arguments of the *Target* entity are mapped to generate two Nile symbols ⟨endpoints⟩, representing *source* and *destination*.

The second phase of the Nile translation process involves mapping *Traffic* entities to Nile ⟨matches⟩ symbols, exploiting service, traffic, protocol, and group traffic types. This work extended Nile to allow for the representation of addresses utilized to express the entity *Traffic* in different levels of abstraction. The last phase of this translation processes the representation of *Action* entities by Nile ⟨rules⟩. The grammar symbol ⟨rules⟩ permits the representation of actions, such as allow and block, and ⟨middleboxes⟩ allows for the representation of several network functionality supported by different types of middleboxes. In addition, the Nile grammar was extended to support a forward operation, allowing the representation of NAT forwarding actions (*cf.* Section 3.3).

3.2. Specializations

Different specializations were proposed to apply the solution in specific contexts, *e.g.*, specializations for firewalls and NATs.

Firewalls

This "Firewall" specialization specifies how the approach can be applied to the firewall context, expecting to receive input configurations in a tabular form. This tabular representation of rules must contain the fields *source* and *destination* for a Target entity, *port* and *protocol* (TCP or UDP) for a Traffic entity and an Action, which contains a function to *allow* or *deny* a packet. Given a tuple (*Target.source*, *Target.destination*, *Traffic.protocol*, *Traffic.port*) the firewall applies an operation to *allow* or *deny* a packet based on the Target and Traffic fields.

Table 1
Traffic classification by source and destination.

Source	Destination	Classification
external	internal	incoming traffic
internal	external	outgoing traffic
internal	internal	internal traffic

The synthesis outputs these fields in the tabular format. While this paper does not generate a firewall model for each vendor, a third-party solution was used to generate the tabular representation. For firewall specializations a firewall synthesizer [9] is utilized to parse vendor-specific rules and transforms them into a tabular representation. The firewall specialization follows the steps (*i*) to transform tabular intermediate representation into the intent-defined language. and (*ii*) proceeds with the consolidation step, described above.

For the aggregation rules are divided into three categories: internal, incoming traffic, and outgoing traffic (*cf.* Table 1). The traffic is considered internal, when source and destination addresses are inside the network prefix; for instance, source and destination addresses are included in a prefix (*e.g.*, 10.0.0.0/8). Incoming traffic is a classification for the traffic in which source addresses are not part of the internal network and destination addresses belong to the internal network, *i.e.*, the destination belongs to the network prefix. Lastly, outgoing traffic refers to situations, where the source address is in the internal network and the destination address is not. This classification does not consider packets created inside the network with a (possible) external source address.

Firstly, the specialization for a firewall names the origin and destination of the target: IP addresses and prefixes are changed to a hostname or other name provided by the user. These targets denote, where the traffic is input, output, or internal. The wildcard "*" denotes any address, which in firewall rules means that this rule that matches all addresses. When this wildcard appears, it is replaced by "all'. Otherwise, IP addresses are renamed according to the user-friendly names provided, if they are available. The function `rename` attributes friendly names to an IP address or a prefix. The rename function receives as a parameter an IP address or prefix and searches in the database for aliases to the address. For instance, if a firewall rule shows the source address any ('*') and the destination 192.168.0.2, which is internally known as a "Web Server", the algorithm will change the original wildcard "*" to "All" and the address 192.168.0.2 to "Web Server".

Network address translation NAT

This specialization receives input configurations in a tabular form, while NATs must report the fields *source* and *destination* for a Target entity, *port* and *protocol* (TCP or UDP) for the Traffic entity and an Action, which contains a function to *forward* a packet and a parameter of *sourceNAT* or *destinationNAT*. Given a tuple (*Target.source*, *Target.destination*, *Traffic.protocol*, *Traffic.port*) the NAT applies a *forward* operation based on these characteristics, considering the parameter *sourceNAT* or *destinationNAT*.

The synthesis step generates a tabular representation for the NAT. In this case, the synthesizer [9] is used to extract vendor-specific rules and convert them to a tabular format. The categorization of traffic is then carried out using this tabular representation. Similar to the aforementioned firewall specialization (*cf.* Table 1), the NAT traffic classification follows a traffic classification algorithm that determines whether the traffic belongs to incoming or outgoing based on the source and destination pair. However, for a NAT's incoming traffic, the final destination is inaccessible directly to the origin. Thus, the traffic destination is the NAT. Inside the NAT, the destination is changed to the correct destination, an internal address. The algorithm that defines the origin and destination for incoming traffic verifies the source IP and renames it to the defined friendly name. It also changes the

`target.dst` to the `NATBox` and the `NATdst` to the hostname of `target.NATdstIp`. After this process, the solution proceeds with the Consolidation step, described in Section 3.1.

For outgoing traffic, first, the source address is processed. Initially, the NAT source address (*target.NATsrc*) is renamed to a familiar name. Thereafter the target source (*target.src*) is set to the NAT's address. Finally, the destination IP address is verified to set a friendly name, as for the firewall specialization.

3.3. Nile extensions

Nile [7] was chosen as the default language to represent intents, since it meets this work's requirements of readability and abstraction. Nile's constructs are capable of representing various network components and actions. However, Nile natively does not support a NAT behavior. Thus, Nile's grammar was extended with NAT constructs to allow for the representation of a NAT behavior. The entire grammar of the extended Nile is shown in full in Appendix.

The extension includes the `forward` and `rename` keywords, which adds support for a fully functional NAT forward behavior. The NAT forward behavior requires the following information: source address or hostname, source port, and destination port. Since the NAT forward is defined by redirecting incoming traffic in a specific port to a specified address in a specific destination port, the protocol can be inferred from source and destination ports. For instance, incoming traffic in Hypertext Transport Protocol (HTTP) can be abstracted to "traffic on port 80". The representation of a NAT port's preservation is facilitated in the syntax by allowing omitting the destination port. In this case, the destination port is the same as the source port.

The `forward` keyword was added to represent a forward NAT action. For this keyword to be declared, a ⟨target⟩ symbol is required. While the actual Nile ⟨target⟩ symbol allows for the representation of groups, services, endpoints, and traffic only, it cannot represent a pair of IP addresses and ports as a target for operation. To overcome this limitation of ⟨target⟩, the new symbol ⟨address⟩ was created to express a pair of IP addresses for hosts and a port (*cf.* Listing 1 for the syntax).

Listing 1: NAT Forwarding Behavior in Extended Nile

```
<forward> ::= 'forward' <address>
<address>
  ::= 'address (' <addrterm> ')'
  | 'address (' <addrterm> ', '
    <port> ')'
<addrterm>
  ::= <ip>
  | <host>
<middlebox>
  ::= 'middlebox('<term>')'
```

With the `forward` and `rename` keyword, plus address symbol, NAT actions can be represented in Nile. The ⟨address⟩ symbol is composed of the ⟨addrterm⟩ and ⟨port⟩ terms. The ⟨addrterm⟩ symbol represents a target source or destination, which is a host or IP address. The symbol ⟨host⟩ can be composed only of valid characters for a hostname and the symbol ⟨ip⟩ represents grammatically valid IP addresses.

The syntax of a `forward` action requires two endpoints representing the source and destination hostnames or IP addresses, a service or a protocol (*e.g.*, TCP), and a port. For a traditional NAT forward operation for incoming traffic, the source is an external address, and the destination represents a NAT, which forwards the traffic for a given address and port, expressed by the symbol ⟨address⟩. For instance, a scenario containing a NAT at the network's edge manages all input traffic. Internally, there is also a "Web Server" with IP address 10.0.0.5, listening for HTTP traffic on port 8080. Thus, the NAT receives input connections on port 80 for HTTP traffic and redirects all this traffic to the Web Server. By applying the extended version of Nile, this scenario is now represented in Listing 2 for incoming traffic. Here, a NAT action was represented for incoming traffic.

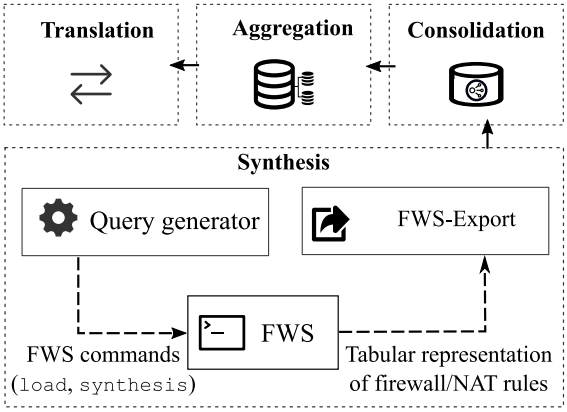


Fig. 5. Implementation architecture.

Listing 2: NAT Forward in Nile

```
define intent forwardIntent:
 from endpoint ('All')
 to endpoint ('NAT Device')
 for protocol ('HTTP')
 forward address ('Web Server', 8080)
```

4. Implementation

The prototype was implemented to validate the approach and it runs on top of SCRIBE [23] implementing specializations of the process as well as focusing on firewalls and NAT configurations. The system leverages diverse input sources to process low-level firewall rules and queries all configurations available on firewall filtering and NAT tables.

4.1. System architecture and use

The architecture is composed out of modules that follow the process steps described in Section 3.1. In addition, this implementation includes the creation of additional modules that interact with third-party synthesizers (*e.g.*, FWS [9]), which were used as a base solution to parse network configurations. The general implementation architecture of the tool developed (*cf.* Fig. 5), includes the Query Generator module to deliver commands to the synthesizer. These commands include (*i*) load-low-level configuration rules, (*ii*) synthesization of rules, and (*iii*) querying synthesizer results. The FWS-Export module is responsible for exporting FWS tables into a Comma Separated Values (CSV) format.

4.2. System modeling

The prototype developed is extensible and capable of representing several network functions, thus, utilizing modules, each representing a different network function. Initially, the implementation of the system follows the model using Target, Traffic, and Action entities as explained in Section 3.1. These entities are composed out of attributes relating to parts from the configuration. The model was specialized to represent firewall and NAT through mapping from input files to abstract entities.

The initial approach is to map the fields from the input files to system classes according to the structure of input files. For this, the network was modeled using an Objected Oriented Programming (OOP) paradigm. An object represents an instance of a network component for the model, such as an IP address. In the first moment, utility functions read the input configuration and CSV files before generating entities. Most firewall input entries can be directly mapped from the input to the proposed entities, as shown in Table 2.

Table 2
Mapping tabular model to system entities.

Information	Input column	Entity
Source IP	srcIP	Target.SrcIP
Destination IP	dstIP	Target.DstIP
Source Port	srcPort	Traffic.SrcPort
Destination Port	dstPort	Traffic.DstPort
Protocol	protocol	Traffic.Protocol

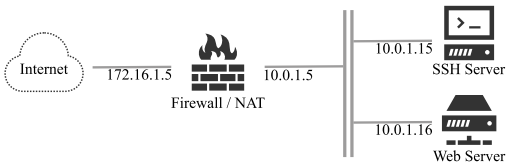


Fig. 6. Corporate network with a NAT.

5. Evaluation and discussion

The implementation is evaluated in the context of three real-world case studies to demonstrate the applicability in different network contexts and to prove the approach via the accuracy of results by exploiting real datasets containing firewall rules from different servers and institutions [10].

5.1. Case studies

Assuming that a new operator initiates at a company and wants to understand the network configuration, the operator intends to assert whether the network behaves as expected. For this, there is a need to understand the existing network configuration. The operator could use the output of the proposed solution to understand the network state before adding a new rule as needed. In this subsection, three case studies illustrate the solution's applicability in real-world scenarios to extract intents from already deployed network configurations.

The deployment of new intents in a traditional network requires that operators discover all network devices and their respective configurations. To do this task, operators must have platform-specific expertise, besides network administrative tools knowledge. Such a situation demands a network operator to *(i)* discover all devices in the network and their respective IP addresses, *(ii)* read the entire configuration from various network devices, and, if needed, *(iii)* provide some documentation to help understand the network behavior for the next alteration.

Three scenarios were analyzed to prove the concept and the technical feasibility of the proposed solution. The first scenario represents a small company containing a single firewall device at the network's input that manages firewall rules and translates NAT addresses. This scenario also includes servers and a workstation, both isolated by NAT. The second scenario explores a Science DMZ campus network containing ACLs directly in a DMZ router due to the high-speed characteristic in this network, and a firewall to secure the internal network and provide NAT isolation. Finally, a case study illustrates a scenario with IP cameras secured by a firewall.

Case study #1: Small company with a NAT

This scenario represents a typical small company with a unique firewall device managing all incoming and outgoing traffic as well as NAT rules. The studied network contains a Web Server, an SSH Server, and a Firewall/NAT middlebox, which manages the network input. Servers cannot be accessed directly from external devices; instead, external connections need to pass through the "Firewall/NAT" middlebox. All input traffic in Firewall/NAT middlebox is redirected to the corresponding server for access to them, according to the traffic type. The firewall device has two interfaces and also translates NAT addresses. For this scenario, the firewall was configured using `iptables`, which also manages NAT rules. The representation of this scenario is shown in Fig. 6.

In this context, the proposed solution can help by parsing the `iptables` code, processing all deployed network configurations, and representing NAT and firewall rules in the Nile language, assisting operators in understanding the network behavior into a language closer to the natural language. At the beginning of the process, the solution queries synthesizer for all configurations, resulting in a table indicating all allowed services in this network. After, the solution processes the resulting table, extracts entities, and generates the model as described in Section 3.1. Friendly hostnames can be aggregated together with the service running in each port indicated by the firewall rule, enriching the model, providing more high-level information. Finally, the system generates meta-intents with higher-level rules and translates them to the Nile language. The generated Nile code for this scenario is shown in Listing 3.

Listing 3: Nile Intents for Company with a NAT

```
define intent firewallIntent1:
   from endpoint ('all')
   to endpoint ('firewall')
   allow protocol ('SSH'),
         protocol ('HTTP')

define intent natIntent1:
   from endpoint ('all')
   to endpoint ('firewall')
   for protocol ('SSH')
   forward address ('SSH Server')

define intent natIntent2:
   from endpoint ('all')
   to endpoint ('firewall')
   for protocol ('HTTP')
   forward address ('Web Server')
```

After extracting intents, the generated Nile code can help the operator to comprehend network behavior. It is simple to infer the primary security behavior of this scenario by observing the extracted intents in Listing 3. From the first intent, it can be noted that (*i*) only protocols HTTP and SSH are allowed to enter on this network. The remaining of the generated intents describe the NAT behavior of the network. The second produced intent in Listing 3 indicates that (*ii*) all SSH traffic will be redirected to the SSH Server, which is internally addressed by the IP address 10.0.1.15. The last intent express that (*iii*) all HTTP traffic will be redirected to the Web Server, which is internally addressed by the IP address 10.0.1.16. The default behavior for this scenario is the default block; thus (*iv*), all other connections will be denied.

Case study #2: Science DMZ

This case study explores a university scenario making use of the design pattern Science DMZ [24]. This scenario comprises high-speed Data Transfer Nodes (DTN) in a DMZ, a router with ACLs, and an internal firewall in the network under study. Fig. 7, shows the scenario. In this network, DTNs are connected directly to a router for high-speed transfers [24], as passing this traffic through a stateful firewall can reduce the bandwidth significantly.

A DMZ is used to isolate special servers from the machines of the internal network. It is essential to maintain public servers separated from the internal network so an attacker cannot get to all the systems. On the other hand, a stateful firewall may degrade the network performance and is not suitable to realize the transfer of large amounts of data (*e.g.*, for data-intensive scientific experiments) in high-speed connections. To keep these DMZ secure, ACLs are inserted directly in the router in a stateless manner, aiming to maximize transfer speeds [24].

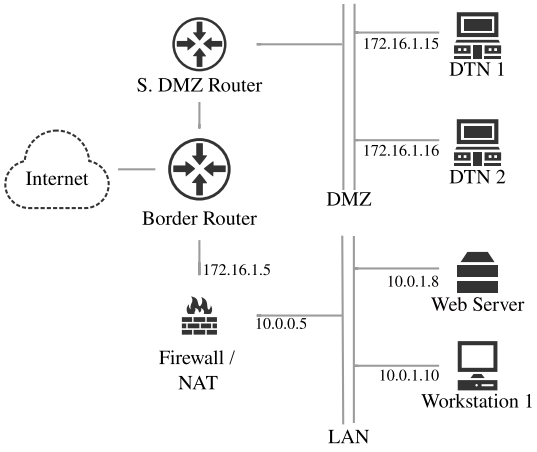


Fig. 7. Science DMZ network.

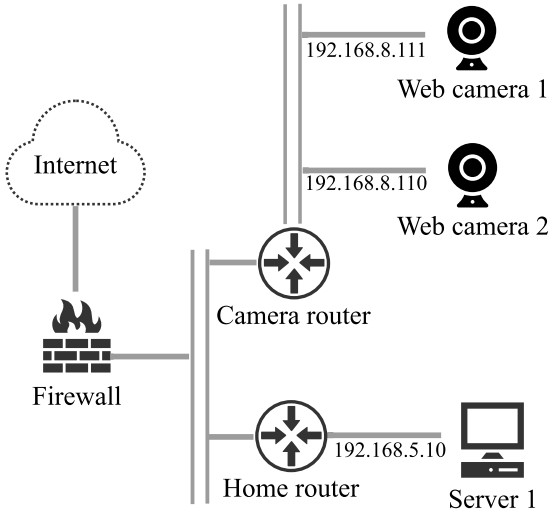


Fig. 8. Camera network.

Several intents can be inferred after applying the bottom-up process to the configuration from this network. The ACL router begins with a default-blocked setting. It is possible to add network IPs and prefixes in a "allowed list", indicating which server is allowed to perform a high-speed connection with a specific DTN. NAT isolates the internal network, and the NAT box, which translates external addresses to internal addresses, is coupled with the firewall device. Besides blocking suspected protocols, firewall rules deal with internal traffic, such as UDP or TCP for some specific internal hosts.

Listing 4: Nile Intents for DMZ

```
define intent firewallIntent1:
   from endpoint ('University A')
   to endpoint ('DTN 1')
   allow traffic ('all')

define intent firewallIntent2:
   from endpoint ('University B')
   to endpoint ('DTN 2')
   allow traffic ('all')

define intent firewallIntent3:
   from endpoint ('Intranet')
   to endpoint ('Intranet')
   allow traffic ('all')

define intent natIntent1:
   from endpoint ('all')
   to endpoint ('Firewall / NAT')
   for protocol ('HTTP')
   forward address ('Web Server')

define intent natIntent2:
   from endpoint ('all')
   to endpoint ('Firewall / NAT')
   for protocol ('HTTPS')
   forward address ('Web Server')
```

From the intents listed in Listing 4, it can be inferred the base intents that derive from this scenario. The first two intents (*i*) represents an allow policy from Universities A and B for their corresponding authorized DTNs (Data Transfer Nodes). The third intent of Listing 4 (*ii*) describes the behavior of allowing all traffic in the internal network. By using an intuitive alias, as an example, the alias "Intranet" refers to the prefix 10.0.0.0/8 – prefix of the internal network, which all devices belong to. Last, intents `natIntent1` and `natIntent2` suggest (*iii*) a NAT translation from incoming HTTP and HTTPS traffic, indicating that the traffic using these protocols will be forwarded to the Web Server, which is only visible in the internal network. In addition, it can be noted that the proposed tool help reduces the complexity to understand global configurations, as in configuring two separate devices to manage ACLs in a high-speed DMZ and a traditional stateful firewall managing internal traffic.

Case study #3: IP camera network

This case study explores a typical scenario of a camera network secured by a firewall. Security mechanisms are needed to ensure the security of IP cameras, as malicious devices can use them as a vector for DDoS attacks [25,26]. This scenario is adapted from [27], a scenario containing an IP camera with `iptables`, and it includes two available IP cameras separated in their own network. Besides the cameras, there are two routers, one that connects the cameras, and another connects to the Workstation. The home router and its network has the prefix 192.168.8.0/24, and the camera network has the prefix 192.168.5.0/24. Both connect to a firewall, which ensures that only the necessary traffic can reach these networks. In addition, the NAT box is coupled with the Firewall middlebox. The representation of this scenario is shown in Fig. 8.

The cameras need to connect to an external FTP server directly, and they are managed by a server available at the Workstation. The firewall protects both routers and all their devices within the network, limiting the inbound and outbound camera connections to only required ports and protocols. The following firewall rules are applied: (i) FTP outgoing traffic is allowed from the camera network to the Internet (all), (ii) ICMP, HTTP, and SSH traffic are allowed from the private network to the camera network (for ping and camera management), (iii) all internal traffic is allowed for the private network, (iv) allow HTTP outgoing traffic from the Workstation to the Internet. For NAT traffic, the following forward rules apply: (i) outgoing FTP traffic from cameras needs to be mapped to a public address, (ii) outgoing HTTP traffic from the Workstation needs to be mapped to a public address (for remote management).

Listing 5: Nile Intents for IP Cameras

```
define intent firewallIntent1:
   from endpoint ('Cameras')
   to endpoint ('all')
   allow protocol ('FTP')

define intent firewallIntent2:
   from endpoint ('Server')
   to endpoint ('all')
```

```
    allow protocol ('HTTP')

define intent firewallIntent3:
    from endpoint ('Cameras')
    to endpoint ('Private network')
    allow protocol ('ICMP'),
          protocol ('SSH'),
          protocol ('HTTP')

define intent firewallIntent4:
    from endpoint ('Private network')
    to endpoint ('Private network')
    allow traffic ('all')

define intent natIntent1:
    from endpoint ('Cameras')
    to endpoint ('Firewall')
    for protocol ('FTP')
    rename source to ('143.54.11.1')

define intent natIntent2:
    from endpoint ('Firewall')
    to endpoint ('all')
    for protocol ('HTTP')
    rename source to ('143.54.11.2')
```

The derived firewall and NAT intents for this scenario are shown in Listing 5, in a default-blocked setting. The first two intents (`firewallIntent1` and `firewallIntent2`) represent an allow policy from an element within the network to the Internet. Also, these intents hide the complexity involved to configure two separated networks: internal and cameras. The third firewall intent (`firewallIntent3`) represents an allow policy between the two networks for different protocols, thus allowing, for instance, traffic such as ping and HTTP control between the cameras and the server. The last firewall intent (`firewallIntent4`) can be easily inferred as a policy to allow all traffic in the private network. From intents `natIntent1` and `natIntent2`, it can be observed (*iii*) a NAT policy from all outgoing traffic to have their source address changed to a public IP, in this example anonymized to a random address belonging to the UFRGS infrastructure.

5.2. Accuracy evaluation

The evaluation of the bottom-up process contains the analysis of output intents, *i.e.*, to identify, whether intents extracted represent the identical configuration as the low-level configuration. Thus, metrics are defined and collected from the implementation through several real-world network configurations.

The datasets used for this evaluation are composed of raw firewall rules collected from repositories publicly available [28] and processed using the FWS synthesizer to generate the tabular representation. The rules processed use a flattened representation [29], which is divided into four different files: Input, Output, Forward, and Loopback, corresponding to their firewall tables, one for each firewall table. Each of these files contains two types of rules: filtering, representing firewall allow policies, and NAT, representing the policies for forwarding.

Two main aspects were evaluated to assert the feasibility of the bottom-up process: (*i*) the trustworthiness of generated intents and (*ii*) support for the most present functionality in real-world firewall dumps. To evaluate the trustworthiness of intents generated, it is essential to check all characteristics of original configuration files being present in extracted intents. Several measurements were performed using [28], while selected public IP and Medium Access Control (MAC) addresses have been anonymized.

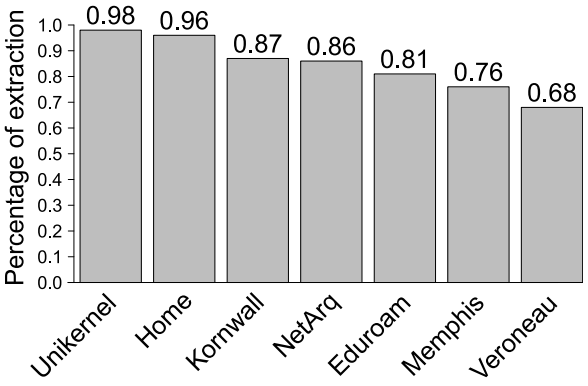


Fig. 9. Correctly extracted intents for each firewall dump.

Translation accuracy

Static analysis was performed first to validate whether each intent generated accurately represents the configuration. Each of these characteristics analyzed represents a configuration element of a rule. For instance, source address, port number, and action (*e.g.*, allow) are characteristics of a traditional allow rule. A tool developed was used to extract intents from the aforementioned datasets as well as to evaluate the accuracy. Those parameters present in raw configurations were dumped. The analysis consists in verifying whether these parameters were also present in generated intents by counting how many parameters of the configuration files are also present in generated intents. Fig. 9 presents the accuracy of the intents. The X-axis displays the name (compact) of the firewall dump. Y-axis shows the rate of configuration parameters extracted correctly, *i.e.*, the percentual of configuration parameters that were present in low-level configurations, and also present in Nile intents.

Fig. 9 shows that the accuracy of the intents generated is high, capable of representing the majority of elements present in the firewall dumps, missing very few configuration details. The missed configurations were related to characteristics in firewall dumps that could not be represented in the actual intents, for instance, rate-limiting rules. The analysis of the worst-case in terms of Veroneau (a dump of *veroneau.net*) indicates several directives of Internet Control Message Protocol (ICMP) messages and directives of rate-limiting, both still not supported by the synthesizer used as a base here. However, even for this case, the implementation can express most of the firewall configuration's functionality. Furthermore, it can be observed that network configurations can be represented using high-level intents in Nile, closely resembling the natural language and still with fidelity to the original configuration.

Functionality support

This evaluation of the functionality support verifies most used configuration in firewall and NAT configurations provided. This evaluation consists of obtaining firewall configurations from [28] and checking which of them are currently supported. Most of these `iptables` options can refer to command, parameter, match, target, and listing options [30]. Command options instruct `iptables` to perform specific actions, such as append or delete a new rule. Parameter options are used to define a packet filtering rule. Match options are used for specialized matching options such as protocols and ports and can be extended through modules. Target options represent actions to take if a packet is matched, such as ACCEPT or DROP actions. Listing options represent options to list rules on the screen.

Functionality is represented by `iptables` parameters, as well as matches and target options. This evaluation is performed by a script dumping all parameters existing as well as matches and target options existing that represent functionality in configuration files. Thus, the number of functionality existing in the dataset was counted and

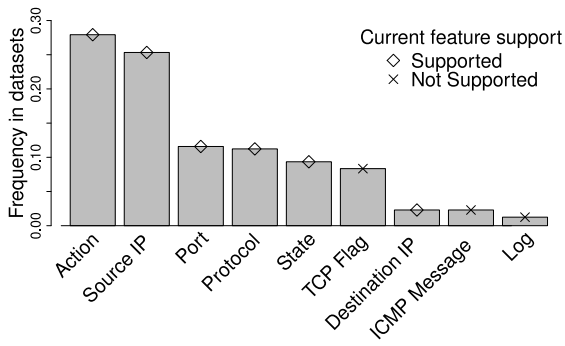


Fig. 10. Support for the most used functionality in datasets.

compared with functionality supported by the tool developed, demonstrating the most used functionality in the datasets provided and their support (*cf.* Fig. 10 with displaying functionality on the X-axis and the frequency of their appearance on the Y-axis).

Results show that the solution implemented supports the most used functionality parameters from these datasets, which represent real-world firewall configurations [28]. Due to this wide support, it can represent existing firewall functionality in the dataset in Nile as an intent-level representation. Most of the unsupported functionality of firewalls refer to logging, TCP flags, and ICMP messages. At the moment, the developed prototype does not support the rate-limiting functionality either. However, the rate-limiting functionality is rarely used directly in `iptables` firewall configurations, according to the public repositories of firewall configurations [28].

6. Summary and future work

Current network management relies on humans in the control loop, where human insight is fundamental for network maintenance [17]. Furthermore, traditional networks are generally hard to maintain due to the gap between high-level specifications and low-level configuration directives. In a traditional network, it is hard to infer high-level, network-wide specifications from device-level configuration directives, because operators often need to learn different low-level configuration languages to express configurations for each network device from different vendors. Thus, these management activities are error-prone and cumbersome for operators, which can lead to incorrect device configurations and, in turn, poor network performance.

Therefore, this new approach here work designed and implemented prototypically a solution to fill this gap between high-level specifications and device-level configurations by introducing a novel bottom-up approach for the extraction of security intents, which synthesizes and represents existing network configurations in an intent-level language. The proposed approach facilitates operators' understanding of actual network configurations by extracting intents from such low-level configuration directives, assisting both novice and experienced operators in understanding existing configurations. Moreover, this solution helps operators on migrating from legacy networks to Intent-based Networks (IBN).

The prototype was validated with respect to its feasibility and correctness under these three case studies describing real-world scenarios. They provided evidence that intents can be extracted from databases of traces with firewall and NAT rules, such that they represent the network behavior in an intent-level language, which closely resembles natural language (in this case, English). Experiments had proven the solution's concept by demonstrating (*i*) the trustworthiness of intents extracted and (*ii*) the support of the developed prototype for most of functionality present in real-world firewall configurations. Furthermore, these results demonstrate that the solution accurately expresses configurations in an intent-level language.

While the implementation was performed through a prototype tool running on top of SCRIBE, the tool developed implements the bottom-up process. Due to its prototypical nature the support of device-level directives represent configurations of certain firewalls, such as `iptables`, IP Firewall (IPFW), and Cisco IOS. Moreover, since the output is directly Nile code reproducing these configurations in the expected intent-level, operators can directly use the intent-level output as input for modern tools in the IBN domain.

Future work of such an approach includes ((i) a full exploration of conflict finders to identify conflicts between a synthesized configuration and those intents being specified, (*ii*) the support of firewall features with respect to rate-limiting, logging, and ICMP messages, so far already included in datasets available, and (*iii*) an extension of the approach to support wider network functions such, as Open Shortest Path First (OSPF) and Border Gateway Protocol (BGP) routing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

While the work had been performed in a Master Thesis at the Computer Networks Group, Institute of Informatics, Federal University of Rio Grande do Sul, Brazil, this paper had been written at the time, when the first author had already moved to University of Zürich UZH, Switzerland as a Ph.D. student.

Appendix. Extended Nile Grammar

This section shows the complete grammar of the extended version of the Nile language in the EBNF syntax [31].

⟨*intent*⟩ ::= '`define intent`' ⟨*term*⟩':' ⟨*operations*⟩

⟨*operations*⟩ ::= ⟨*path*⟩ ⟨*operation*⟩ ' ' ⟨*operation*⟩

⟨*path*⟩ ::= [⟨*from_to*⟩ | ⟨*targets*⟩]

⟨*from_to*⟩ ::= '`from`' ⟨*endpoint*⟩ '`to`' ⟨*endpoint*⟩

⟨*operation*⟩ ::= (⟨*middleboxes*⟩
| ⟨*qos*⟩
| ⟨*rules*⟩)+ [⟨*interval*⟩]

⟨*middleboxes*⟩ ::= [add | remove] ⟨*middlebox*⟩ ','
| ', \n' ⟨*middlebox*⟩

⟨*middlebox*⟩ ::= '`middlebox`('⟨*term*⟩')'

⟨*qos*⟩ ::= [set | unset] ⟨*metrics*⟩

⟨*metrics*⟩ ::= ⟨*metric*⟩ (','
| ', \n') ⟨*metric*⟩

⟨*metric*⟩ ::= [bandwidth | quota]('[`max`' | '`min`']',' ⟨*term*⟩')'

⟨*rules*⟩ ::= [allow | block] ⟨*matches*⟩ [⟨*matches*⟩]

⟨*targets*⟩ ::= '`for`' ⟨*target*⟩ (',' | ', \n') ⟨*target*⟩

⟨*target*⟩ ::= [⟨*group*⟩ | ⟨*service*⟩ | ⟨*endpoint*⟩ | ⟨*traffic*⟩]

⟨*matches*⟩ ::= [⟨*service*⟩ | ⟨*traffic*⟩ | ⟨*protocol*⟩]

⟨*endpoint*⟩ ::= 'endpoint('⟨*term*⟩')'

⟨*group*⟩ ::= 'group('⟨*term*⟩')'

⟨*service*⟩ ::= 'service('⟨*term*⟩')'

⟨*traffic*⟩ ::= 'traffic('⟨*term*⟩')'

⟨*protocol*⟩ ::= 'protocol('⟨*term*⟩')'

⟨*date_time*⟩ ::= 'datetime('⟨*term*⟩')'
| 'date('⟨*term*⟩')'
| 'hour('⟨*term*⟩')'

⟨*term*⟩ ::= [a-z0-9]+

⟨*forward*⟩ ::= 'forward' ⟨*address*⟩

⟨*address*⟩ ::= 'address (' ⟨*addrterm*⟩ ')'
| 'address (' ⟨*addrterm*⟩ ', ' ⟨*port*⟩ ')'

⟨*addrterm*⟩ ::= ⟨*ip*⟩
| ⟨*host*⟩

⟨*middlebox*⟩ ::= 'middlebox('⟨*term*⟩')'

⟨*ip*⟩ ::= ([0-9]{1,3}.){3}[0-9]{1,3}

⟨*host*⟩ ::= [a-zA-Z]{1}[a-zA-Z0-9_]*

⟨*port*⟩ ::= [0-9]{1,5}

⟨*rename*⟩ ::= ⟨*nat_target*⟩ 'to (' ⟨*addrterm*⟩ ')'

⟨*nat_target*⟩ ::= 'source'
| 'destination'

References

[1] K. Benzekki, A. El Fergougui, A. Elbelrhiti Elalaoui, Software-defined networking (SDN): A survey, Secur. Commun. Netw. 9 (18) (2016) 5803–5833.

[2] Z. Kerravala, Business-Critical Services, Technical Report October, ZK Research, 2018.

[3] S. Arezoumand, K. Dzeparoska, H. Bannazadeh, A. Leon-Garcia, MD-IDN: Multi-domain intent-driven networking in software-defined infrastructures, in: 2017 13th International Conference on Network and Service Management (CNSM), IEEE, New York, NY, USA, 2017, pp. 1–7.

[4] M. Behringer, M. Pritikin, S. Bjarnason, A. Clemm, B. Carpenter, S. Jiang, L. Ciavaglia, Autonomic Networking: Definitions and Design Goals, RFC 7575, RFC Editor, 2015.

[5] A. Clemm, L. Ciavaglia, L.Z. Granville, J. Tantsura, Intent-Based Networking - Concepts and Definitions, draft-irtf-nmrg-ibn-concepts-definitions-03, Internet Engineering Task Force, 2021, Work in Progress.

[6] Cisco, Intent-Based Networking and Extending the Enterprise, Cisco public, 2020.

[7] A.S. Jacobs, R.J. Pfitscher, R.A. Ferreira, L.Z. Granville, Refining network intents for self-driving networks, in: Proceedings of the Afternoon Workshop on Self-Driving Networks, in: SelfDN 2018, Association for Computing Machinery (ACM), New York, NY, USA, 2018, pp. 15–21.

[8] B. Tian, X. Zhang, E. Zhai, H.H. Liu, Q. Ye, C. Wang, X. Wu, Safely and automatically updating in-network ACL configurations with intent language, in: Proceedings of the 2019 Conference of the ACM Special Interest Group on Data Communication, Association for Computing Machinery (ACM), New York, NY, USA, 2019, pp. 214–226.

[9] C. Bodei, P. Degano, L. Galletta, R. Focardi, M. Tempesta, L. Veronese, Language-independent synthesis of firewall policies, in: 2018 IEEE European Symposium on Security and Privacy (Euro S&P), IEEE, New York, NY, USA, 2018, pp. 92–106.

[10] C. Diekmann, J. Michaelis, M. Haslbeck, G. Carle, Verified iptables firewall analysis, in: 2016 IFIP Networking Conference (IFIP Networking) and Workshops, Association for Computing Machinery (ACM), New York, NY, USA, 2016, pp. 252–260.

[11] S. Martínez, J. Cabot, J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia, A model-driven approach for the extraction of network access-control policies, in: Proceedings of the Workshop on Model-Driven Security, in: MDsec '12, Association for Computing Machinery (ACM), New York, NY, USA, 2012, pp. 5:1–5:6.

[12] A. Kheradmand, Automatic inference of high-level network intents by mining forwarding patterns, in: Proceedings of the Symposium on SDN Research, in: SOSR '20, Association for Computing Machinery (ACM), New York, NY, USA, 2020, pp. 27–33.

[13] A.S. Jacobs, R.J. Pfitscher, Ferreira, R.H. Ribeiro, R.A. Ferreira, L.Z. Granville, W. Willinger, S. Rao, Hey, lumi! using natural language for intent-based network management, in: 2021 USENIX Annual Technical Conference (USENIX ATC 21), USENIX Association, 2021.

[14] A. Fogel, S. Fung, L. Pedrosa, M. Walraed-Sullivan, R. Govindan, R. Mahajan, T. Millstein, A general approach to network configuration analysis, in: Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation, in: NSDI'15, USENIX Association, Berkeley, CA, USA, 2015, pp. 469–483.

[15] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, D. Walker, Network configuration synthesis with abstract topologies, in: Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, in: PLDI 2017, Association for Computing Machinery (ACM), New York, NY, USA, 2017, pp. 437–451.

[16] A. Tongaonkar, N. Inamdar, R. Sekar, Inferring higher level policies from firewall rules, in: Proceedings of the 21st Conference on Large Installation System Administration Conference, in: LISA'07, USENIX Association, Berkeley, CA, USA, 2007, pp. 2:1–2:10.

[17] R. Birkner, D. Drachsler-Cohen, L. Vanbever, M. Vechev, Net2Text: Query-guided summarization of network forwarding behaviors, in: 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), USENIX Association, Renton, WA, 2018, pp. 609–623.

[18] R.H. Ribeiro, A.S. Jacobs, R. Parizotto, L. Zembruzki, A.E. Schaeffer-Filho, L.Z. Granville, A bottom-up approach for extracting network intents, in: L. Barolli, F. Amato, F. Moscato, T. Enokido, M. Takizawa (Eds.), Advanced Information Networking and Applications, Springer International Publishing, Cham, 2020, pp. 858–870.

[19] R.H. Ribeiro, A Bottom-Up Approach for Extracting Network Intents, Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, 2020, pp. 1–59.

[20] A. Mercian, F. Ahmed, P. Sharma, S. Wackerly, C. Clark, Mind the semantic gap: Policy intent inference from network metadata, in: 2021 IEEE 7th International Conference on Network Softwarization (NetSoft), 2021, pp. 312–320.

[21] E.J. Scheid, C.C. Machado, M.F. Franco, R.L. dos Santos, R.P. Pfitscher, A.E. Schaeffer-Filho, L.Z. Granville, INSpIRE: Integrated NFV-based intent refinement environment, in: 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), IEEE, New York, NY, USA, 2017, pp. 186–194.

[22] IANA, Service Name and Transport Protocol Port Number Registry, IANA, 2021, https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml.

[23] R.H. Ribeiro, SeCuRity intent-based extractor (SCRIBE), GitHub Repos. (2020) https://github.com/ComputerNetworks-UFRGS/SCRIBE.

[24] E. Dart, L. Rotman, B. Tierney, M. Hester, J. Zurawski, The science DMZ: A network design pattern for data-intensive science, Sci. Program. 22 (2) (2014) 173–185.

[25] N. Vlajic, D. Zhou, IoT as a land of opportunity for ddos hackers, Computer 51 (7) (2018) 26–34.

[26] Y.-J. Lee, N.-K. Baik, C. Kim, C.-N. Yang, Study of detection method for spoofed IP against ddos attacks, Pers. Ubiquitous Comput. 22 (1) (2018) 35–44.

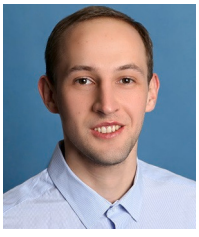
[27] Mind Chasers Inc, Configure Netfilter (iptables) on Linux to Protect Your Networked Devices, Mind Chasers Inc, 2019, https://mindchasers.com/dev/netfilter.

[28] C. Diekmann, Net-network, GitHub Repos. (2015) https://github.com/diekmann/net-network.

[29] K. Jayaraman, N. Bjørner, G. Outhred, C. Kaufman, Automated Analysis and Debugging of Network Connectivity Policies, Technical Report, Microsoft Research, Redmond, WA, NY, USA, 2014.

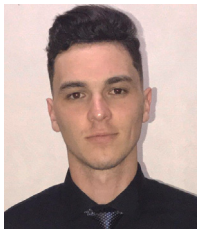
[30] RedHat, 18.3. Options Used within iptables commands, Options Used Within Iptables Commands (2005).

[31] 14977:1996, ISO/IEC, Information technology — Syntactic metalanguage — Extended BNF, Technical Report, ISO/IEC, 2018.



Rafael Hengen Ribeiro joined the Communication Systems Group (CSG) of University of Zürich (UZH) in November 2020 in order to obtain his doctoral degree under the supervision of Prof. Dr. Burkhard Stiller. He obtained his master's degree in 2020 under the supervision of Prof. Dr. Lisandro Zambenedetti Granville at the Federal University of Rio Grande do Sul (UFRGS). His research interests include Intent-Based Networking (IBN), Segment Routing (SR), and Software-Defined Networking (SDN).

Arthur Selle Jacobs is a second year Ph.D. student in Computer Science, under the supervision of Prof. Dr. Lisandro Granville, at the Federal University of Rio Grande do Sul, in Brazil. His research interests include network management, Network Functions Virtualization, Self-Driving Networks, programmable networks and artificial intelligence. Arthur is currently researching the use of machine learning for network management, as a mean to achieve Self-Driving Networks.

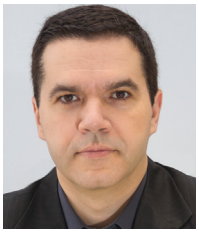


Luciano Zembruzki is a Ph.D. student in Computer Science at Federal University of Rio Grande do Sul (UFRGS) advised by Prof. Dr. Lisandro Zambenedetti Granville. He achieved his B.Sc. Degree in Computer Science from the Integrated Regional University (URI) - Campus Frederico Westphalen, in 2017. He hold his M.Sc. Degree in Computer Science from UFRGS, in 2020. His research interests include Internet Measurements, Network Functions Virtualization (NFV), Software-Defined Networking (SDN), and Distributed Systems.

Ricardo Parizotto is a Ph.D. student under the supervision of Prof. Dr. Alberto Schaeffer-Filho at the Federal University of Rio Grande do Sul (UFRGS) in Brazil. He obtained his master's degree in 2019 under the supervision of Prof. Dr. Alberto Schaeffer-Filho at the Federal University of Rio Grande do Sul (UFRGS). His current research focuses on programmable networks and in-network computing systems.

Eder John Scheid is a Junior Researcher and Ph.D. candidate pursuing his Ph.D. since December 2017 under the supervision of Prof. Dr. Burkhard Stiller at the University of Zürich UZH, Switzerland, within the Communication Systems Group CSG of the Department of Informatics IfI. Eder holds an M.Sc. degree in Computer Science from the Federal University of the Rio Grande do Sul (UFRGS), Brazil, which he obtained in 2017 under the supervision of Prof. Dr. Lisandro Zambenedetti Granville. Eder focuses his research on blockchain, Smart Contracts, and network management.

Alberto Schaeffer-Filho holds a Ph.D. in Computer Science (Imperial College London, 2009) and is Associate Professor at Federal University of Rio Grande do Sul (UFRGS), Brazil. From 2009 to 2012 he worked as a research associate at Lancaster University, UK. Dr. Schaeffer-Filho is a CNPq-Brazil Research Fellow and his areas of expertise are network/service management, network virtualization and software-defined networks, policy-based management, and security and resilience of networks. He has authored over 50 papers in leading peer-reviewed journals and conferences related to these topics, and also serves as TPC member for important conferences in these areas, including: CNSM (2018), IEEE/IFIP NOMS (2018), CNSM (2017), IFIP/IEEE IM (2017), IEEE CCNC (2017), and IEEE INFOCOM CNTCV Workshop (2017). He is the general chair for SBRC 2019, co-chair for IEEE ICC 2018 CQRM Symposium, demo co-chair for IFIP/IEEE IM 2017, and workshop co-chair for ManSDN/NFV 2015. He is also a member of the IEEE.



Lisandro Zambenedetti Granville is full Professor at the Institute of Informatics of the Federal University of Rio Grande do Sul. He served as the TPC Co-Chair of IFIP/IEEE DSOM 2007, IFIP/IEEE NOMS 2010, and IEEE NetSoft 2018, General Co-Chair of IFIP/IEEE CNSM 2014, IEEE/IFIP NOMS 2022, and IEEE GLOBECOM 2022, and the TPC Vice-Chair do IEEE ICC 2018. He is the current board member of the Brazilian Computer Society. His interests include network management, software-defined networking, and network functions virtualization.

Burkhard Stiller received the Informatik-Diplom (M.Sc.) in Computer Science and the Dr. rer.-nat. (Ph.D.) degree from the University of Karlsruhe, Germany, in 1990 and 1994, respectively. In his research career he was with the Computer Lab, University of Cambridge, U.K. (1994–1995), ETH Zürich, Switzerland (1995–2004), and the University of Federal Armed Forces Munich, Germany (2002–2004). Since 2004 he chairs the Communication Systems Group CSG, Department of Informatics IfI, at the University of Zürich UZH, Switzerland. Besides being a member of the editorial board of the IEEE Transactions on Network and Service Management, Springer's Journal of Network and Systems Management, and the KICS' Journal of Communications and Networks, Burkhard is the past Editor-in-Chief of Elsevier's Computer Networks journal. His main research interests are published in well over 300 research papers and include systems with a fully decentralized control (Blockchains, clouds, peer-to-peer), network and service management (economic management), Internet-of-Things (security of constrained devices, LoRa), and telecommunication economics (charging and accounting).