POLITECNICO DI TORINO
Repository ISTITUZIONALE

Parallelizing Automated Firewall Configuration via Traffic-Aware Decomposition

Original
Parallelizing Automated Firewall Configuration via Traffic-Aware Decomposition / Pizzato, F., Bringhenti, D., Valenza, F.. - In: IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT. - ISSN 1932-4537. - ELETTRONICO. - (In corso di stampa). [10.1109/TNSM.2026.3747990]

Availability:
This version is available at: 11583/3016987 since: 2026-10-26T07:10:03Z

Publisher:
IEEE

Published
DOI:10.1109/TNSM.2026.3747990

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©9999 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

02 November 2026

Parallelizing Automated Firewall Configuration via Traffic-Aware Decomposition

Francesco Pizzato, Daniele Bringhenti, Fulvio Valenza

***Abstract*—Automating the configuration of distributed firewalls by refining high-level security requirements is a well-established approach to improve correctness and manageability in modern computer networks. However, existing solutions formulate this task as a single monolithic constraint-solving problem, whose scalability suffers as network size and policy complexity increase. To overcome this limitation, this paper proposes a traffic-aware decomposition approach for the automated, parallel configuration of distributed firewalls. The method decomposes the global refinement problem into cluster-scoped subproblems using a flow-weighted network representation and solves them concurrently before merging the results into a single deployable configuration. The approach is implemented on top of a constraint-based framework and evaluated on representative topologies. The results show substantial runtime reductions, with gains increasing with problem size and reaching up to ~98% on larger instances, while maintaining satisfactory optimization quality. Moreover, the approach enables the resolution of firewall configuration problems beyond the limits of monolithic formulations.**

***Index Terms*—firewall, parallelized configuration, virtual network orchestration, network security automation**

I. Introduction

Protecting computer networks against cyberattacks requires configuring distributed firewalls, mostly packet filters, to enforce a set of Network Security Requirements (NSRs), expressing which communications should be denied or permitted. This task has become increasingly complex as networks have grown more dynamic, heterogeneous, and dependent on software-defined components. In this context, NSRs continuously change due to topology evolution, incident response, or application needs [1]. As a result, firewall configurations must be synthesized within tight time constraints and correctly, while traditional manual approaches no longer scale and remain a major source of outages and security violations.

Consequently, automated firewall configuration techniques were proposed [2], so that the allocation scheme and filtering rules of distributed firewalls could be refined automatically from high-level user-friendly NSRs. Significant progress has been made to the point that the most feature-complete approaches, e.g., [3], [4], combine automation with formal correctness assurance and optimization, typically by formulating constraint-based problems. Nevertheless, while effective, these approaches remain constrained by scalability: as networks grow and NSRs accumulate, the constraints they impose make their resolution increasingly expensive, preventing their use in large scenarios.

F. Pizzato, D. Bringhenti, and F. Valenza are with the Politecnico di Torino, Dip. Automatica e Informatica; e-mail: {first.last}@polito.it.
This project has received funding from the European Union's Horizon Europe Research and Innovation Programme under grant agreement No. 101168144.

Only a limited subset of studies has attempted to mitigate this limitation by optimizing the problem formulation [5]–[7]. However, although these methods reduce the computational overhead of each configuration refinement instance, they do not fundamentally address the dominant bottleneck, i.e., the fact that synthesis is still performed over a globally coupled problem. In particular, NSRs are expressed over end-to-end communication semantics and interact through shared routing structure and shared enforcement opportunities, which causes even optimized solutions to scale poorly.

A possible solution may be to parallelize the configuration problem, but achieving this is not trivial. First, NSRs are defined over end-to-end communications, and partitioning can split flows that must be treated consistently, e.g., when an NSR requests that a traffic flow must be permitted uninterrupted from source to destination. Second, independently synthesized sub-solutions may allocate or configure devices at shared locations within the borders of the sub-clusters, creating the need for reconciliation. Third, decomposition can degrade the global optimization objective, potentially increasing the number of allocated firewalls or the rule-set size. As a result, achieving parallelism requires a segmentation strategy that accounts for traffic locality and a merge procedure that deterministically produces a coherent, deployable configuration.

This paper addresses this gap by proposing a methodology for traffic-aware decomposition and parallelization of the automated firewall configuration problem. The proposed approach constructs a flow-weighted graph representation of the network to guide clustering via a community-detection algorithm, then generates independent firewall configuration subproblems that can be solved in parallel and finally merged into a global solution. Moreover, the approach aims to minimize resource consumption in terms of allocated firewalls and configured rules. The main contributions of this work are:

- a problem segmentation method, based on a traffic-aware clustering of a flow-weighted network representation, aimed at reducing cross-cluster traffic and enabling the construction of independent synthesis subproblems;
- a parallelized resolution strategy that executes one automated configuration synthesis instance per cluster, preserving the baseline synthesis semantics within each subproblem;
- a merge and post-processing procedure that reconciles independently synthesized firewall configurations and mitigates potential sub-optimalities.

In this work, the adopted problem segmentation method is the application of the Leiden [8] community-detection algorithm

to a flow-weighted graph representation of the network. The employed configuration synthesis engine is an extension of VEREFOO [4], as it is among the most feature-complete approaches in the literature thanks to a Maximum Satisfiability Modulo Theories (MaxSMT) formulation of the configuration problem. Instead, the reconciliation procedure is implemented via a dedicated merge procedure and a post-processing algorithm that removes redundant firewalls introduced by cross-cluster decomposition. All these choices contribute to providing a concrete realization of the methodology, while alternative partitioning strategies, synthesis engines, or reconciliation procedures could be adopted when compatible abstractions are available. The experimental evaluation across representative topologies shows that the proposed approach significantly reduces computation time compared to VEREFOO, with increasingly pronounced gains as problem size increases, reaching up to ~98% on the largest instances, while maintaining satisfactory optimization quality and scaling until system memory becomes the limiting factor.

The remainder of the paper is structured as follows. Section II introduces background on graph partitioning and VEREFOO. Section III overviews the methodology. Section IV formalizes the problem segmentation, Section V the parallel resolution, and Section VI the merge and post-processing procedures. Section VII describes the implementation and the experimental validation. Section VIII discusses related work. Section IX elaborates on the conclusions and future work.

II. Background

A. Graph Partitioning and Community Detection

Within graph theory, a central analytical task is the decomposition of a graph into subgraphs that simplify downstream reasoning, optimization, or control. Community detection addresses the graph partitioning problem by decomposing a large graph into groups of nodes, called communities, such that nodes within each community are more strongly related to each other than to nodes outside the community.

Let $G = (V, E)$ be a weighted graph with node set V and edge set $E \subseteq V \times V$. Each edge $e_{ij} \in E$ has an associated weight $w_{ij} > 0$ that encodes an application-dependent notion of relationship intensity (e.g., traffic volume, number of flows). The output of community detection is a partitioning schema $P = \{c_1, \ldots, c_k\}$ of V into nonempty communities c_i.

A desirable partitioning schema should (i) reduce cross-community interactions by minimizing cut edges, i.e., edges linking nodes in different communities, and (ii) maximize intra-community edge density. In addition, partitions that are strongly imbalanced or internally disconnected are typically undesirable, as they either concentrate complexity into a small number of communities or create communities that do not correspond to coherent subgraphs. Community detection is commonly formulated as the optimization of a quality function over the partitions that encode the aforementioned objectives.

Most commonly, there are two prominent formulations for the quality function, which are *Modularity* [9] and *Constant Potts Model* (CPM) [10]. The former compares the number of edges in a community against the expected number under a randomized null model, i.e., a graph whose nodes have the same degree distribution but whose edges are uniformly random. The latter replaces the null model with a constant density reference, proving to be more effective in detecting small but meaningful communities in large graphs. Moreover, when considering weighted graphs, the outcome of community detection algorithms depends not only on the chosen quality function, but also on how edge weights are interpreted within the optimization process. Both these aspects influence the resulting partitioning and therefore the proposed approach.

Optimizing such metrics requires computing all possible combinations, which is an NP-hard problem. Therefore, practical approaches rely on heuristic or greedy algorithms. For many years, the most popular one has been the Louvain algorithm [11], as it was found to be among the fastest and best-performing ones in different analyses [12], [13]. However, a more recent optimization of the Louvain algorithm is represented by the Leiden algorithm [8], which has been proven to be faster and able to find better partitions. Specifically, unlike Louvain, it guarantees the production of well-connected communities. Leiden iteratively moves nodes to improve the quality function, refines communities to ensure well-connected components, and aggregates them into super-nodes until no further improvement is possible.

B. The VEREFOO Models and Approach

The VEREFOO [4] approach is among the most feature-rich methodologies for automatic, optimal, and formal allocation and configuration of distributed packet-filtering firewalls in computer networks. It works on two inputs: a logical representation of the network where firewalls must be allocated, and a set of security requirements to be enforced there.

Network Topology. The first input is the logical topology of the computer network where firewalls must be configured. In VEREFOO, it is formally modeled as a directed graph, named *Service Graph* (SG), $G_S = (N_S, E_S)$. N_S is the set of vertices, representing network nodes (e.g., endpoints, routers, network address translators), whereas E_S is the set of edges, representing directed links among nodes modeling connectivity.

The SG provided as input is internally processed by VEREFOO to derive a second representation, the *Allocation Graph* (AG). The AG is similarly modeled as $G_A = (N_A, E_A)$, and it is computed from an SG by introducing *Allocation Places* (APs), i.e., special placeholder nodes where a firewall can be optionally allocated. It results in $N_A = N_S \cup N_P$, where N_P is the set of all APs. In general, an AP is introduced on every link connecting pairs of vertices belonging to the set N_S, but the creation can be constrained through specific predicates.

In the derived AG, the allocation decisions are represented by the predicate $allocated\colon N_A \rightarrow \mathbb{B}$, which is true by construction for endpoints and pre-existing service functions and is left open for APs, as the firewall allocation scheme represents an output of VEREFOO and is not initially known.

Network Functions and Traffic. Each network function is modeled based on its behavior toward the received packet classes. Each packet class $t \in T$, named traffic for the

sake of conciseness, is modeled as a predicate defined over the IP 5-tuple fields, and it is formalized as $t = (IPSrc, IPDst, pSrc, pDst, tProto)$. Consequently, the behavior of each network function is modeled with (i) a forwarding predicate *deny*: $N_A \times T \rightarrow \mathbb{B}$ determining whether an input traffic class is dropped or not by the input node, and (ii) a transformer function mapping an input traffic class to the corresponding output class after processing. With them, it is possible to characterize different function types, such as devices that only forward, devices that filter, and devices that transform headers (e.g., network address translators).

Filtering decisions, driving the interpretation of *deny*, depend on the specific network function configuration. Specifically, the configuration of a packet-filtering firewall possibly allocated in $n \in N_P$ is modeled through the *allowlist*: $N_P \rightarrow \mathbb{B}$ predicate and the rule set P_n. The former maps a firewall n to true if it is configured in allowlist, i.e., with *deny* as default action, to false if it is configured in denylist, i.e., with *allow* as default action. The latter is a set of so-called *placeholder rules*, built with free variables whose values will be set by VEREFOO as output only if they are actually used to enforce some security properties. To this purpose, the predicate *configured*: $N_P \times P_n \rightarrow \mathbb{B}$ maps a placeholder rule to true if VEREFOO wants to configure it with concrete values. In that case, its action is the opposite of the default one.

Given this representation, *traffic flows* are modeled to capture how a packet class is originated by a source endpoint and transformed as it traverses the AG to reach the destination endpoint. Specifically, a traffic flow f is represented as an alternating sequence of crossed nodes and packet classes:

$$f = [n_s, t_{sa}, n_a, t_{ab}, n_b, \ldots, n_k, t_{kd}, n_d], \tag{1}$$

where each t_{ij} denotes the packet class observed between adjacent nodes along the path.

Network Security Requirements. The second input is a set of Network Security Requirements (NSRs). Each NSR is modeled as $r = (C, a)$, where C is a condition predicate over packet fields and $a \in \{allow, deny\}$ is the required action. As VEREFOO focuses on packet-filtering firewalls, it supports connectivity requirements: isolation NSRs have $a = deny$, whereas reachability NSRs have $a = allow$.

Let F_r denote the set of flows satisfying the condition C of NSR r, $\pi(f)$ the ordered set of crossed nodes in f, and $\tau(f, n)$ the traffic class (or packet class) observed in input to node n along f. This information is computed before the formulation and resolution of the MaxSMT problem, thanks to an algorithm presented in [14], which computes all flows satisfying the NSRs while maximizing their aggregation. Specifically, that algorithm groups packet classes that traverse the same path and undergo identical transformations and forwarding decisions, thereby reducing the number of flow entities.

The satisfaction of r is then captured through different flow-based constraints, based on the type of r:

$$\exists f \in F_r.\ \forall n \in \pi(f).\ allocated(n) \Rightarrow \neg deny(n, \tau(f, n)) \tag{2}$$

$$\forall f \in F_r.\ \forall n \in \pi(f).\ allocated(n) \Rightarrow \neg deny(n, \tau(f, n)) \tag{3}$$

$$\forall f \in F_r.\ \exists n \in \pi(f).\ allocated(n) \wedge deny(n, \tau(f, n)) \tag{4}$$

These formulations are for a partial reachability NSR (2), a complete reachability NSR (3), and an isolation NSR (4). They express, respectively, the existence of at least one end-to-end allowed traffic flow matching the partial reachability NSR, the necessity of allowing every end-to-end flow matching the complete reachability NSR, and the necessity of blocking every flow that matches the isolation NSR. Finally, the user-defined NSR set is assumed to be anomaly-free, as well-known anomaly analysis techniques [15]–[17] can be used to reconduct any NSR set into an anomaly-free one.

MaxSMT Formulation. Once both inputs are defined, VEREFOO encodes the firewall allocation and configuration as a partial weighted MaxSMT problem. Hard constraints encode the semantics of network functions, allocation feasibility, and NSR satisfaction. Soft constraints encode optimization objectives, favoring solutions that minimize the number of allocated firewalls and configured rules, with weights that reflect their relative goals (e.g., configuring a new rule within an existing firewall is preferred to allocating a new firewall).

VEREFOO uses an SMT solver as the underlying engine to solve the constraint-based problem and determine the values of the open variables, thereby allocating and configuring firewalls. In particular, the output allocation scheme is derived from the interpretation established for the predicate *allocated*, whereas the output filtering configuration is derived from the interpretation of the predicate *configured*.

III. Approach

To improve the scalability of automated refinement of high-level user-defined policies into distributed firewall allocation and configuration, the proposed approach (i) partitions the network into disjoint clusters, (ii) solves the corresponding per-cluster policy-refinement subproblems in parallel, and (iii) algorithmically merges the resulting partial configurations. The objective is to reduce end-to-end computation time in large networks while maintaining near-optimal solutions in terms of firewall instances and configured rules.

As shown in Fig. 1, the workflow starts from two inputs: a logical representation of the network topology and a set of security requirements. These inputs are represented by the AG and NSR models, already presented in Section II. The present approach operates on a static snapshot of these inputs. If the network topology or the NSR set changes, the workflow can be executed again on the updated inputs to synthesize a new correct and optimized configuration. After being fed with them, the approach unfolds in three main phases:

1) *Problem Segmentation*. The inputs are processed to create an intermediate representation, i.e., a weighted graph that encodes the notion of "relation intensity" via traffic-based reasoning. Then, a graph decomposition strategy is applied to obtain a varying number of subnetworks and a partitioning of the NSRs.
2) *Parallel Configuration.* The multiple subnetworks and NSR subsets produced by the clustering phase become the inputs of the parallel configuration. For each pair of inputs, this phase produces a locally optimal, formally

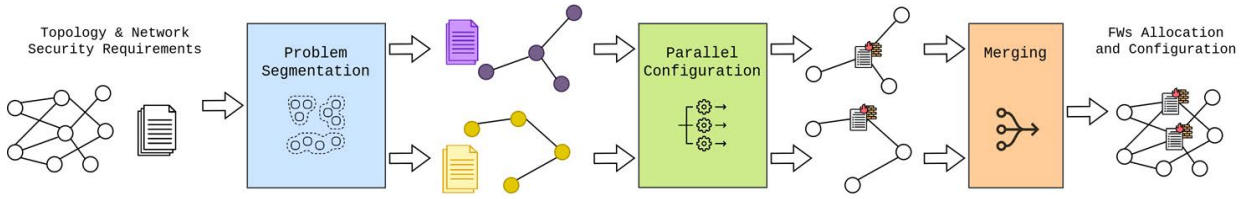


Fig. 1: The traffic-aware decomposition approach for parallel configuration.

correct allocation and configuration of firewalls that satisfy the given requirements.

3) *Merging.* The resulting partial configurations are merged into a single global configuration for the original inputs. The merge phase consolidates the allocation scheme while reconciling the rules of firewalls allocated at the borders of subnetworks and therefore shared by multiple problems. Additionally, a post-processing phase algorithmically processes the locally optimal configurations of each subnetwork and addresses potential suboptimalities.

The remainder of this section illustrates these phases through a concrete example. Then, Sections IV, V and VI will formalize them with the support of TABLE I, which summarizes the notation used throughout the paper.

Illustrative Example. Fig. 2 illustrates the three phases of the approach. Step ⓐ shows the two inputs. The AG connects two endpoint networks 10.0.0.* and 10.0.1.* through two routers, while the white circles on all network links identify the APs, i.e., placeholders indicating where the methodology may later decide to allocate a firewall. The NSRs are represented in a table; each row specifies the required action for traffic matching the corresponding condition, but it does not specify where it must be enforced. For example, the first row requires traffic from 10.0.0.1 to 10.0.1.1 to be allowed, while the fourth requires traffic from 10.0.0.2 to 10.0.1.2 to be denied.

In the first phase, these inputs are processed to obtain the weighted graph, named Flow Graph (FG), shown in ⓑ. First, a traffic flow is derived for each of the five NSRs and is depicted as a colored path. The FG retains the nodes and the links relevant to these flows, while its edge weights quantify their traffic-induced coupling. For example, a weight of 2 or 3 means that the corresponding link is traversed by, respectively, two or three NSR-induced flows.

The clustering procedure applied to the FG split the networks at the inter-router link, and the AP on this link is duplicated on both cluster-scoped topologies. The NSRs are partitioned consistently with the topology. The traffic of NSRs 2 and 3 remains within one cluster, so they are trivially assigned. NSR 4 and 5 are isolation ones whose flows cross the boundary. Blocking them in either cluster is sufficient, so the algorithm assigns NSR 4 to the right cluster and NSR 5 to the left for balance. Instead, NSR 1 is a cross-cluster reachability that must be preserved end-to-end and is therefore assigned to both clusters. At the end of this phase, the original problem has thus become the two topology-NSR pairs depicted in ⓒ.

In the second phase, the two topology-NSR pairs are given to two independent solver instances and processed concurrently. Each solver decides which of its APs must host a

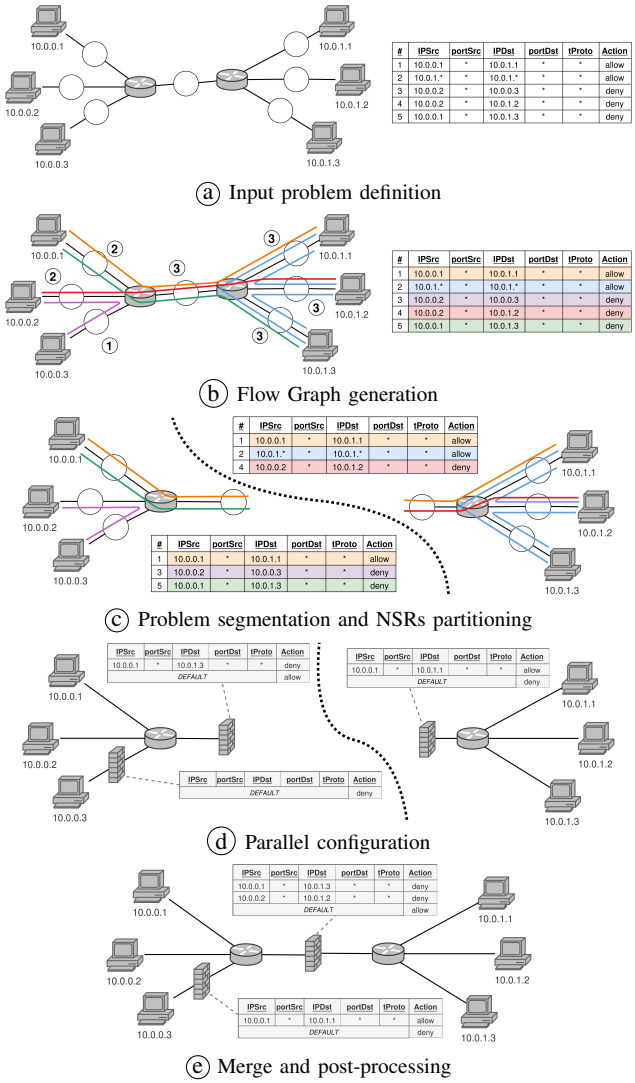


Fig. 2: Illustrative example of the proposed approach.

firewall and synthesizes the corresponding default action and filtering rules. In this case, as shown in ⓓ, the local solution for the left cluster allocates two firewalls: an allowlist one at the AP adjacent to 10.0.0.3, and a denylist one at the cluster's border with a deny rule for NSR 5. Independently, the solution for the right cluster allocates only an allowlist firewall at the cluster's border with a rule for NSR 1.

The third phase maps the two local solutions back onto the original AG. As the AP at the partitioning boundary is independently synthesized by both local problems, the merge procedures consolidate their allocation decisions and reconcile

TABLE I: Notation used throughout the paper.

Notation	Meaning	Notation	Meaning
$\mathbb{B}$	Boolean domain $\{true, false\}$	T	Set of traffic classes
$G_S = (N_S, E_S)$	Service Graph, with nodes N_S and links E_S	$G_A = (N_A, E_A)$	Allocation Graph derived from G_S
$N_A = N_S \cup N_P$	AG nodes, including the AP set N_P	$allocated(n)$	True iff a firewall is allocated in AP n
$deny(n, t)$	True iff node n drops traffic class t	P_n	Placeholder-rule set of a firewall at AP n
$allowlist(n)$	True for deny-by-default; false for allow-by-default	$configured(n, p)$	True iff placeholder p is instantiated at n
$\Pi(n)$	Set of configured rules, i.e., the filtering policy at n	R, $r = (C, a) \in R$	NSR set; each NSR has condition C and action a
F_r, $F_R = \bigcup_{r \in R} F_r$	Flows induced by NSR r; flows induced by all NSRs	$\pi(f)$, $\tau(f, n)$	A flow ordered node path and its traffic class at node n
$\Lambda(f)$	Set of directed hops traversed by flow f	$\|X\|$, $\mathcal{P}(X)$	Cardinality and power set of X
$G_F = (N_F, E_F, w)$	Flow Graph used for clustering, with $N_F = N_A$	$w(\{u, v\})$	Number of NSR-induced flows traversing link $\{u, v\}$
$P = \{N_{F,c_1}, \ldots, N_{F,c_k}\}$	Partition of N_F into clusters; c_i is a cluster label	γ	Resolution parameter controlling cluster granularity
$\mathcal{L}(n)$	Cluster label(s) associated with node/AP n	$\mathcal{C}(f)$, $\mathcal{A}(r)$	Clusters crossed by flow f, or touched by NSR r
R_{c_i}, F_r^{cut}	NSRs assigned to cluster c_i; cut flows induced by r	$G_{A,c_i} = (N_{A,c_i}, E_{A,c_i})$	Cluster-scoped AG for cluster c_i
E_{A,c_i}^{int}, E_{A,c_i}^{cut}	Internal and preserved boundary edges of G_{A,c_i}	$\Pi_{c_i}(n)$, $\Pi(n)$	Local policy for n in cluster c_i; merged global policy
$allocated_{c_i}$, $allowlist_{c_i}$, $configured_{c_i}$	Decisions produced by the solver instance for cluster c_i	*ConvertAllowList*, *ConvertDenyList*	Functions converting local policies to a common default action
$\Pi_A(n)$, $\Pi_D(n)$	Merged allowlist and denylist candidate policies at n	$\mathcal{L}_a(n)$	Cluster instances allocating a firewall at shared AP n
$H = \sum_{f \in F_R} \|\pi(f)\|$	Total length of all NSR-induced flow paths	R_I	Isolation NSRs, i.e., NSRs whose action is *deny*
S	Candidate firewalls considered for removal	F_m^I, C_g	Isolation flows crossing m; alternative APs for flow g
$allocated_{new}(n)$	Post-processed allocation decision at AP n	$allowlist_{new}(n)$, $\Pi_{new}(n)$	Post-processed default action and filtering policy at AP n

their filtering policies into a single firewall configuration. The result is the deployable configuration in ⓔ for the input AG, despite the configuration problem having been solved through parallel, cluster-scoped instances.

IV. Problem Segmentation

The first phase decomposes the global problem into cluster-scoped subproblems at two levels: (i) network segmentation, by constructing a traffic-centric weighted graph and applying community detection; and (ii) NSR partitioning across the resulting subnetworks.

A. Flow Graph Construction

Given the input AG $G_A = (N_A, E_A)$ and the set of flows F_R derived from the NSR set R, i.e., $F_R = \bigcup_{r \in R} F_r$, an intermediate flow-centric representation, named *Flow Graph* (FG), is built. This is an undirected weighted graph used exclusively to drive network clustering, modeled as $G_F = (N_F, E_F, w)$.

The node set N_F is preserved as it is in the AG, i.e., $N_F \equiv N_A$. Instead, the edge set E_F may differ from that of the AG from which it derives, i.e., from E_A. Specifically, edges in E_F are *traffic-induced*: given two nodes u and v, if E_A contains at least a (directed) communication link among $e_1 = (u, v)$ and $e_2 = (v, u)$, and at least one flow in F_R traverses e_1 or e_2, a corresponding undirected edge $e' = \{u, v\}$ is added to E_F. Formally, let $\pi(f) = \langle n_0, n_1, \ldots, n_{|\pi(f)|-1} \rangle$ denote the ordered node sequence crossed by flow $f \in F_R$, and let the function $\Lambda : F_R \to \mathcal{P}(N_A \times N_A)$ denote the set of directed hops induced by $\pi(f)$ for flow f:

$$\Lambda(f) = \{(n_{i-1}, n_i) : i = 1, \ldots, |\pi(f)| - 1\} \tag{5}$$

Given this notation, the edge set E_F of the FG is defined as:

$$E_F = \left\{\{u, v\} \subseteq N_F : \exists f \in F_R.\ (u, v) \in \Lambda(f) \ \vee \ (v, u) \in \Lambda(f)\right\} \tag{6}$$

Hence, if no traffic flow in the set F_R traverses a communication link connecting two nodes of G_A, then no edge will connect those nodes in G_F.

Edge weights encode the "relation intensity" induced by the NSR set and its associated traffic. In particular, the function $w : E_F \to \mathbb{N}$ maps an input FG edge $\{u, v\} \in E_F$ to the number of flows in F_R that crosses any link of E_A insisting on the same nodes u and v. The behavior of this function is expressed as:

$$w(\{u, v\}) = |\{f \in F_R : (u, v) \in \Lambda(f) \ \vee \ (v, u) \in \Lambda(f)\}| \tag{7}$$

Operationally, $w(\{u, v\})$ is computed by scanning all flows $f \in F_R$ and incrementing a counter associated to $\{u, v\}$ whenever $\Lambda(f)$ contains hop (u, v) or (v, u). Directionality is intentionally ignored at this stage, since the objective is to capture the strength of traffic-induced coupling for clustering rather than forwarding configuration.

Because edges are instantiated only when supported by the presence of a flow in F_R crossing the corresponding link in the AG of derivation, the FG may not be fully connected. Each connected component of G_F identifies a maximal subset of nodes that are mutually coupled by at least one NSR-induced flow. This is the ideal scenario in which R already induces disjoint traffic domains: components can be treated as independent sub-networks, and community detection operates independently within each component to further partition it.

Complexity. Let $H = \sum_{f \in F_R} |\pi(f)|$ denote the total length of the NSR-induced flow paths. The FG is constructed by visiting each hop once and updating the corresponding edge counter. Assuming constant-time dictionary operations, its time complexity is $O(H)$, while its additional space complexity is $O(|N_F| + |E_F|)$, excluding the AG and flow-path representation, which are not uniquely related to this phase.

B. Traffic-Aware Network Clustering

Given the FG $G_F = (N_F, E_F, w)$, a partition of N_F into traffic-coupled sub-networks is computed via community detection. The algorithm operates on $w(\cdot)$ and therefore favors grouping nodes connected by links traversed by a larger number of flows, with the goal of increasing internal policy coupling while reducing cross-cluster policies.

For simplicity, let G_F be assumed a connected graph. This assumption is without loss of generality, since even if the creation of G_F results in a disconnected graph, the same clustering procedure could be trivially applied independently to each connected subgraph.

Among the possible graph decomposition strategies, this work employs the Leiden algorithm [8] for weighted com-

munity detection, primarily for its proven superiority in performance and guaranteed creation of connected communities, i.e., subgraphs. Other partitioning strategies could be adopted if they better optimize different objectives or topology-specific constraints. An experimental analysis comparing Leiden with other partitioning strategies is presented in Section VII-D. The output is a partition of the N_F into disjoint clusters, i.e., $P = \{N_{F,c_1}, \ldots, N_{F,c_k}\}$, with $\bigcup_{i=1}^{k} N_{F,c_i} = N_F$ and $N_{F,c_i} \cap N_{F,c_j} = \emptyset$ for $i \neq j$, each with a cluster's label $c_i \in \mathbb{N}$.

The partition produced by Leiden depends on the selected quality function and its parameterization. Their impact on the overall approach is evaluated in Section VII-A.

C. Network Security Requirement Partitioning

Given the clustering output $P = \{N_{F,c_1}, \ldots, N_{F,c_k}\}$, the NSR set R is accordingly partitioned into cluster-scoped subsets so as to obtain independent inputs for the refinement problem. The partitioning is flow-driven also for the NSRs, since each $r \in R$ is expanded into its induced flow set F_r, and assignments are derived from the cluster labels of the nodes crossed by such flows.

Let $\mathcal{L} : N_F \to \mathbb{N}$ map each node to its cluster label. For each flow $f \in F_R$, the function $C : F_R \to \mathcal{P}(\mathbb{N})$ produces the set of crossed cluster labels:

$$C(f) = \{\mathcal{L}(n) : n \in \pi(f)\} \tag{8}$$

Accordingly, each $r \in R$ is associated with the set of clusters it touches through its induced flows through the function $\mathcal{A} : R \to \mathcal{P}(\mathbb{N})$, defined as:

$$\mathcal{A}(r) = \bigcup_{f \in F_r} C(f) \tag{9}$$

If $\mathcal{A}(r) = \{c_i\}$ is a singleton, then the NSR r is fully contained within cluster N_{F,c_i} and is assigned to the local requirement set R_{c_i}. Otherwise, the NSR r induces cross-cluster coupling and must be handled through boundary-aware partitioning.

For isolation NSRs, satisfaction is per-flow. Consequently, every $f \in F_r$ must be denied at least once along its path. Let a flow f be *internal* if all the nodes on its path belong to the same cluster, i.e., $|C(f)| = 1$; otherwise, f is *cut* as it crosses multiple clusters, i.e., $|C(f)| > 1$. For internal flows induced by r, assignment is again trivial as there is a single cluster. Note that, in contrast to the case $\mathcal{A}(r) = \{c_i\}$, here the NSR r is potentially duplicated across multiple clusters, since each internal $f \in F_r$ implies an independent assignment. In contrast, to block isolation cut flows, the assignment is not immediate, as multiple clusters (i.e., any of the crossed ones) may be configured to satisfy it. A straightforward approach is to assign the same NSR r to all crossed clusters, i.e., to all R_{c_i} with $c_i \in C(f)$. However, this would significantly degrade optimality, as it may lead to redundant enforcement and duplicated allocations, since each subcluster blocks the same f. The proposed solution, while still potentially suboptimal, is to delegate r to exactly one cluster in $C(f)$ without loss of correctness. Specifically, each isolation cut flow $f \in F_r$ triggers the assignment of r to a single cluster in $C(f)$, selected by a load-balancing heuristic (i.e., the cluster with the smallest number of delegated flows).

For partial reachability NSRs, the model semantics is existential, i.e., r is satisfied if at least one end-to-end flow in F_r is allowed. Under decomposition, existential semantics is not compositional: independent subproblems may yield different candidate flows to satisfy the reachability NSR, and the merged configuration may still block all end-to-end candidates due to inconsistent local choices. To prevent this problem, if a reachability NSR r induces a cut flow, we conservatively strengthen it to complete reachability, requiring that all flows in F_r are allowed. In particular, internal complete reachability flows are trivially assigned to a single cluster, whereas cut flows are assigned to all clusters' labels in $C(f)$. As reachability must be guaranteed end-to-end, this scenario does not allow any heuristic assignment.

Complexity. NSR partitioning is derived directly from the FG labeling $\mathcal{L}$. For each flow $f \in F_R$, the crossed-cluster set $C(f)$ is computed in $O(|\pi(f)|)$ time through constant-time lookups of $\mathcal{L}(n)$ along its path. This results in about $O(H)$ time for the R set. Computing $\mathcal{A}(r)$ and applying the load-balancing heuristic do not increase this bound, since both operate on crossed clusters and are bounded by $|\pi(f)|$. Overall time complexity is $O(H)$. The additional space complexity is $O(|P||R|)$ in the worst case, since each NSR may be assigned to at most all $|P|$ cluster-scoped NSR sets.

V. Parallel Configuration

The problem segmentation phase produces a variable set of inputs for the next phase, where each input consists of (i) a subnetwork topology and (ii) a corresponding subset of NSR. The goal of the parallel configuration phase is to compute, for each cluster, an optimal and formally correct firewall allocation and configuration independently of the other clusters. Possible refinements and optimizations of the local configurations are handled with a post-processing phase described in Section VI.

Let $P = \{N_{F,c_1}, \ldots, N_{F,c_k}\}$ be the partitioning of N_F returned by the traffic-aware clustering presented in Section IV-B, and let R_{c_i} denote the subset of NSR assigned to the cluster labeled with c_i as described in Section IV-C. From this information, for each cluster c_i, an AG $G_{A,c_i} = (N_{A,c_i}, E_{A,c_i})$ is obtained by restricting the original AG $G_A = (N_A, E_A)$ to the nodes and edges belonging to c_i.

Due to the treatment of boundary conditions, the cluster-scoped AGs may overlap at the borders. The edge set E_{A,c_i} and the node set N_{A,c_i} for a given local instance are constructed by evaluating both internal and cut edges of the original graph.

Concerning edge assignment, internal edges, i.e., edges where both adjacent nodes belong to the same cluster, are directly assigned to the corresponding AG, G_{A,c_i}. Formally, the set of internal edges for G_{A,c_i} is defined as:

$$E^{int}_{A,c_i} = \{(u,v) \in E_A \mid \mathcal{L}(u) = c_i \wedge \mathcal{L}(v) = c_i\} \tag{10}$$

Instead, cut edges, which connect adjacent nodes belonging to different clusters, require a specific assignment logic to handle cross-cluster cut flows. When a cut edge occurs between two non-AP nodes, the corresponding link is removed from

both cluster-scoped AGs, so that the resulting subgraphs are completely disjoint. The set of such edges is defined as:

$$\{(u,v) \in E_A : \mathcal{L}(u) = c_i \wedge \mathcal{L}(v) = c_j \wedge c_i \neq c_j \wedge v \notin N_P \wedge u \notin N_P\} \tag{11}$$

Conversely, when one of the adjacent nodes of a cut edge is an AP, the cluster that initially lacks that AP is expanded to include it. This means the AP is replicated across the adjacent clusters to preserve local enforcement opportunities for cut traffic. Formally, the set of boundary cut edges preserved for a cluster c_i is defined as:

$$E^{cut}_{A,c_i} = \{(u,v) \in E_A : (\mathcal{L}(u) = c_i \oplus \mathcal{L}(v) = c_i) \wedge (u \in N_P \oplus v \in N_P)\} \tag{12}$$

Consequently, the complete edge set for the cluster-scoped AG is $E_{A,c_i} = E^{int}_{A,c_i} \cup E^{cut}_{A,c_i}$.

Concerning node assignment, the local node set N_{A,c_i} is then naturally derived as the set of all nodes native to the cluster c_i, i.e., N_{F,c_i}, combined with any shared boundary APs connected via the preserved cut edges.

$$N_{A,c_i} = N_{F,c_i} \cup \{n \in N_P : \exists u \in N_A : (u,n) \in E^{cut}_{A,c_i} \vee (n,u) \in E^{cut}_{A,c_i}\} \tag{13}$$

In the end, these operations produce multiple sets of inputs (G_{A,c_i}, R_{c_i}) that are encoded as a partial weighted MaxSMT problem by reusing the baseline VEREFOO formulation presented in Section II. All instances are solved in parallel by launching independent solver executions, each producing a locally optimal interpretation of the problem that specifies the truth assignment of the allocation decision variables for APs in N_{A,c_i} and the corresponding configuration decisions. Since border APs may appear in multiple instances, the resulting local configurations are not immediately deployable as a single global configuration, as a reconciliation is needed and deferred to the merge phase described in Section VI.

Finally, note that global optimality with respect to the original monolithic instance is not guaranteed in general due to the decomposition and boundary handling rules for cut flows. A discussion and quantification of the optimality degradation is presented in Section VII.

VI. Merge and post-processing

The parallel configuration phase returns one local solution for each cluster-scoped instance. Each local solution contains (i) the set of allocation decisions for the APs considered by that instance and (ii) the filtering configuration synthesized for each allocated firewall. These solutions are local and correct with respect to the cluster-restricted topology and assigned NSRs, but they are not directly deployable on the original network. The reason is that, due to the boundary strategy introduced in Section V, some APs may be replicated across adjacent cluster-scoped AGs, potentially leading to multiple local solutions that independently allocate and configure a firewall on the same AP of the original AG. During the merge phase, this issue is solved by consolidating replicated allocation decisions for the same original AP and reconciling the corresponding local filtering policies. Note that internal APs, i.e., APs belonging to exactly one cluster-scoped instance, do not require any merge operation and are copied unchanged to the final solution. Next, a post-processing phase is executed to optimize the merged solution, aiming to remove some sub-optimalities introduced by the parallelization of the global configuration problem.

Let $P = \{N_{F,c_1}, \ldots, N_{F,c_k}\}$ be the network partitioning obtained according to Section IV, and let $\mathcal{L}(n)$ denote the set of labels of the cluster-scoped instances containing node $n \in N_P$. By construction, $|\mathcal{L}(n)| = 1$ for internal APs and $|\mathcal{L}(n)| > 1$ for shared APs. Then, for each cluster $c_i \in \mathcal{L}(n)$, let $allocated_{c_i}(n) \in \mathbb{B}$ denote the local allocation decision for node n under the solution's instance for AG G_{A,c_i}. When $allocated_{c_i}(n) = true$, then $allowlist_{c_i}(n) \in \mathbb{B}$ denote the local default action decision and $\Pi_{c_i}(n)$ denote the local filtering policy synthesized for n under the solution's instance for AG G_{A,c_i}, i.e., $\Pi_{c_i}(n) = \{p \in P_n : configured_{c_i}(n,p) = true\}$.

A. Merge

The merge operation is executed only on APs and is applied to each original AP independently. For a given AP $n \in N_P$, there are four possible scenarios.

1) *No allocated firewall.* If $allocated_{c_i}(n) = false$ for all $c_i \in \mathcal{L}(n)$, then no firewall is present in the global solution.
2) *Exactly one allocated firewall.* If there exists exactly one $c_j \in \mathcal{L}(n)$ such that $allocated_{c_j}(n) = true$, then the global solution inherits that allocation decision and copies the configuration unchanged, i.e., $allowlist(n) \equiv allowlist_{c_j}(n)$ and $\Pi(n) \equiv \Pi_{c_j}(n)$.
3) *Multiple allocated firewalls with the same default action.* In this case, there are multiple local configurations in which n is allocated in the cluster-scoped AGs. However, all of them have the same default action. Given the auxiliary set of allocated firewalls defined as $\mathcal{L}_a(n) = \{c_i \in \mathcal{L}(n) \mid allocated_{c_i}(n) = true\}$, then $allowlist_{c_i}(n) = allowlist_{c_j}(n)$ for any $c_i, c_j \in \mathcal{L}_a(n)$. As different local solvers may synthesize different specific rules at the same shared AP n, but with the same action, the correct global configuration is obtained by preserving the common default action and taking the union set of all cluster-scoped firewall rules:
$$\Pi(n) = \bigcup_{c_i \in \mathcal{L}_a(n)} \Pi_{c_i}(n) \tag{14}$$
4) *Multiple allocated firewalls with different default action.* The non-trivial case is when $|\mathcal{L}_a(n)| > 1$ and the local configurations allocated at cluster-scoped APs corresponding to the original AP n do not share the same default action, i.e., $\exists c_i, c_j \in \mathcal{L}_a(n) : allowlist_{c_i}(n) \neq allowlist_{c_j}(n)$. A direct union of the local rule sets is not acceptable, because rule semantics depend on the default action. In particular, an allowlist firewall blocks by default and explicitly permits selected traffic classes, whereas a denylist firewall allows by default and explicitly blocks selected traffic classes. Therefore, the local solutions must be projected onto a common representation before being merged. To this end, two candidate global policies are constructed. The first one is an allowlist candidate, obtained by converting every local configuration at n into

an equivalent allowlist representation and then taking the union of the resulting rule sets:

$$\Pi_A(n) = \bigcup_{c_i \in \mathcal{L}_a(n)} \mathit{ConvertAllowList}(\Pi_{c_i}(n)) \tag{15}$$

The second one is a denylist candidate, obtained analogously by converting every local configuration into an equivalent denylist representation:

$$\Pi_D(n) = \bigcup_{c_i \in \mathcal{L}_a(n)} \mathit{ConvertDenyList}(\Pi_{c_i}(n)) \tag{16}$$

Finally, the merged firewall is allocated at n using the most optimal of the two equivalent representations, i.e., the one configuring fewer firewall rules:

$$(\mathit{allowlist}(n), \Pi(n)) = \begin{cases} (\mathit{true}, \Pi_A(n)), & |\Pi_A(n)| \le |\Pi_D(n)|, \\ (\mathit{false}, \Pi_D(n)), & \text{otherwise} \end{cases} \tag{17}$$

The equivalence between the two representations is guaranteed only over the NSR-induced traffic classes. Consequently, changing the default action in the merged configuration may affect traffic not covered by any NSR, potentially impacting the overall security posture. However, this is consistent with the policy model adopted in this work and inherited from VEREFOO, where formal security guarantees apply only to traffic explicitly covered by user-defined NSRs, while the behavior of all other traffic is left unspecified and may be determined according to the optimization objective. Therefore, users should specify all traffic requiring a particular security behavior.

The complete merge processing, including all four scenarios, is formalized in Algorithm 1. There, the two auxiliary functions, i.e., *ConvertDenyList* and *ConvertAllowList*, convert an allowlist to a denylist and vice versa. The former computes the set of flows that a firewall n blocks thanks to its default action, by identifying each $f \in F_{R_{c_i}}$ crossing it and for which the *deny* predicate maps the n and $\tau(f, n)$ traffic pair to true. For each $\tau(f, n)$ thus identified, a corresponding denying rule p is created with that traffic as a condition, and set as configured, i.e., $\mathit{configured}_{c_i}(n, p) = \mathit{true}$. The default action is finally set to allow, i.e., $\mathit{allowlist}_{c_i}(n) = \mathit{false}$. The latter follows an analogous procedure considering the flows crossing n that are permitted by it.

The union of the filtering policies of independently configured firewalls may lead to suboptimal results. Therefore, the union of rules must be further processed to resolve possible discordances. This problem is orthogonal to this contribution and has been extensively discussed in other works [15]–[17], which the current one simply uses to address this issue.

Given the complexity and importance of the merge phase, we provide a formal argument for its correctness below. This phase must preserve the satisfaction of the original NSRs in R. We first show that cluster-scoped solutions cannot impose conflicting NSR decisions at shared APs, and then that Algorithm 1 preserves such decisions in the merged configuration.

Consistency of NSR decisions. Internal APs require no merging and are therefore trivially consistent. Instead, considering a shared AP n, suppose by contradiction that two NSRs require opposite decisions for the same traffic class t at n. In particular, let an isolation flow f_I be denied at n by cluster c_i, i.e., $\mathit{allocated}_{c_i}(n) \wedge \mathit{deny}_{c_i}(n, \tau(f_I, n))$, while a reachability flow f_R satisfies $\tau(f_R, n) = \tau(f_I, n)$. Since n is a boundary AP, f_R is a cut flow and is assigned to every cluster it crosses. Hence, the correctness of the solution for c_i requires $\neg \mathit{allocated}_{c_i}(n) \vee \neg \mathit{deny}_{c_i}(n, \tau(f_R, n))$, contradicting the isolation decision. Therefore, NSR-induced decisions at shared APs must be globally consistent. Notice that $\tau(f, n)$, the traffic class observed at n, already accounts for header transformations, e.g., NAT, occurring earlier along the path. Traffic flows and their per-node traffic classes are computed before clustering and then reused by each cluster-scoped problem. Therefore, splitting a flow does not alter its traffic representation. For example, consider the simplified flow $f = [n_1, t_x, n_2, t_x, n_{\mathrm{NAT}}, t_y, n_3, t_y, n_4]$, where the NAT transforms t_x into t_y. If clustering introduces a boundary between n_{NAT} and n_3, the downstream cluster still reasons on t_y. Hence, traffic classes are preserved across cluster boundaries and already reflect transformations performed upstream, even when they occur outside the considered cluster.

Preservation of decisions during merge. According to Algorithm 1, if $|\mathcal{L}_a(n)| = 0$, no firewall is allocated at n, whereas if a single cluster c_i allocates n, its configuration is copied unchanged. If multiple local configurations have

Algorithm 1 Merge of Local Firewall Configurations

Require: Local cluster solutions $\big(\mathit{allowlist}_{c_i}(\cdot), \Pi_{c_i}(\cdot), \mathit{allocated}_{c_i}(\cdot)\big)$, $\forall c_i \in P$
Ensure: Global configuration $\big(\mathit{allowlist}(\cdot), \Pi(\cdot), \mathit{allocated}(\cdot)\big)$

for $n \in N_P$ **do**
  $l_c \leftarrow \mathcal{L}_a(n)$
  if $l_c = \emptyset$ **then**
    $\mathit{allocated}(n) \leftarrow \mathit{false}$
    $\Pi(n) \leftarrow \emptyset$
  else if $|l_c| = 1$ **then**
    $\mathit{allocated}(n) \leftarrow \mathit{true}$
    $\Pi(n) \leftarrow \Pi_{l_c[0]}(n)$
    $\mathit{allowlist}(n) \leftarrow \mathit{allowlist}_{l_c[0]}(n)$
  else if $\forall i \in l_c : \mathit{allowlist}_i(n) = \mathit{true}$ **then**
    $\mathit{allocated}(n) \leftarrow \mathit{true}$
    $\mathit{allowlist}(n) \leftarrow \mathit{true}$
    $\Pi(n) \leftarrow \bigcup_{c_i \in l_c} \Pi_{c_i}(n)$
  else if $\forall c_i \in l_c : \mathit{allowlist}_{c_i}(n) = \mathit{false}$ **then**
    $\mathit{allocated}(n) \leftarrow \mathit{true}$
    $\mathit{allowlist}(n) \leftarrow \mathit{false}$
    $\Pi(n) \leftarrow \bigcup_{c_i \in l_c} \Pi_{c_i}(n)$
  else
    $\Pi_A \leftarrow \emptyset,\ \Pi_D \leftarrow \emptyset$
    for $c_i \in l_c$ **do**
      $\Pi_A \leftarrow \Pi_A \cup \textsc{ConvertAllowList}(\Pi_{c_i}(n))$
      $\Pi_D \leftarrow \Pi_D \cup \textsc{ConvertDenyList}(\Pi_{c_i}(n))$
    end for
    $\mathit{allocated}(n) \leftarrow \mathit{true}$
    if $|\Pi_A| \le |\Pi_D|$ **then**
      $\mathit{allowlist}(n) \leftarrow \mathit{true}$
      $\Pi(n) \leftarrow \Pi_A$
    else
      $\mathit{allowlist}(n) \leftarrow \mathit{false}$
      $\Pi(n) \leftarrow \Pi_D$
    end if
  end if
end for

the same default action, their explicit rules are combined. With default *allow*, these are deny rules; with default *deny*, they are allow rules. Their union preserves all NSR-induced decisions because, as shown above, no conflicting obligation exists for the same traffic class. Overlapping or wildcard rules do not alter this argument: they are aggregations of compatible placeholder rules defined over specific packet classes and therefore preserve their forwarding semantics. If local configurations instead use different default actions, *ConvertDenyList* and *ConvertAllowList* first map them to a homogeneous representation. The former makes the required deny decisions explicit before setting the default to *allow*, while the latter makes the required allow decisions explicit before setting the default to *deny*. Hence, the representation changes, but the decisions over the NSR-induced traffic classes are preserved and grounded in underlying placeholder rules.

End-to-end NSR preservation. For every isolation NSR r, decomposition assigns each induced flow to one of the clusters it crosses, such that

$$\forall f \in F_r.\exists c_i \in C(f).\exists n \in \pi(f).allocated_{c_i}(n) \wedge deny_{c_i}(n, \tau(f, n)). \tag{18}$$

Since the merge preserves this decision, the resulting configuration satisfies

$$\forall f \in F_r.\exists n \in \pi(f).allocated(n) \wedge deny(n, \tau(f, n)). \tag{19}$$

For complete reachability, every crossed cluster satisfies

$$\forall f \in F_r.\forall c_i \in C(f).\forall n \in \pi(f).allocated_{c_i}(n) \rightarrow \neg deny_{c_i}(n, \tau(f, n)). \tag{20}$$

Since these decisions are preserved by the merge, the resulting configuration satisfies

$$\forall f \in F_r.\forall n \in \pi(f).allocated(n) \rightarrow \neg deny(n, \tau(f, n)). \tag{21}$$

Finally, a cut partial-reachability NSR is strengthened to complete reachability before decomposition. Hence, the merged configuration satisfies every $r \in R$.

Complexity. Let $Q = \sum_{n \in N_P} \sum_{c_i \in \mathcal{L}_a(n)} |\Pi_{c_i}(n)|$ denote the total number of rules in the local firewall configurations. The merge processes each AP and its local policies once, requiring $O(|N_P| + Q)$ time when their default actions coincide. When different default actions must be reconciled, the traffic classes observed at each shared AP n are also inspected for each local configuration that allocates it. Since $|\mathcal{L}_a(n)| \leq |P|$ and the aggregate number of traffic observations is bounded by H, the overall time complexity is $O(|N_P| + Q + |P| \cdot H)$. Additional space complexity is $O(|N_P| + Q + H)$: $O(|N_P|)$ stores the merged configuration of all APs; $O(Q)$ accounts for the local rule sets of each cluster-scoped solution before merging; and $O(H)$ covers the temporary $\Pi_A(n)$ and $\Pi_D(n)$ representations.

B. Post-processing

The merge phase produces a semantically correct and deployable global configuration, but not necessarily one that is optimal with respect to the original monolithic MaxSMT instance. Because cluster-scoped problems are solved independently, different local solutions may allocate multiple firewalls to block the same traffic at different APs along a common path. To mitigate this effect, Algorithm 2 is applied to remove redundant firewalls from the merged solution.

Algorithm 2 Optimization of Duplicate Firewalls

Require: Global configuration $(allowlist(\cdot), \Pi(\cdot), allocated(\cdot))$
Ensure: Optimized configuration $(allowlist_{new}(\cdot), \Pi_{new}(\cdot), allocated_{new}(\cdot))$

1: $S \leftarrow \emptyset$
2: **for all** $m \in N_A$ **do**
3: $\quad allocated_{new}(m) \leftarrow allocated(m)$
4: $\quad allowlist_{new}(m) \leftarrow allowlist(m)$
5: $\quad \Pi_{new}(m) \leftarrow \Pi(m)$
6: **end for**
7: **for all** $r \in R_I = \{r \in R : r.a = deny\}$ **do**
8: $\quad$ **for all** $f \in F_r^{cut}$ **do**
9: $\quad\quad S \leftarrow S \cup \{\, m \in \pi(f) \cap N_P \mid allocated_{new}(m) \,\}$
10: $\quad$ **end for**
11: **end for**
12: $F_{R_I} \leftarrow \bigcup_{r \in R_I} F_r$
13: **for all** $m \in S$ **do**
14: $\quad F_m^I \leftarrow \{\, g \in F_{R_I} \mid m \in \pi(g) \,\}$
15: $\quad del_{flag} \leftarrow true$
16: $\quad$ **for all** $g \in F_m^I$ **do**
17: $\quad\quad C_g \leftarrow (\pi(g) \cap N_P) \setminus \{m\}$
18: $\quad\quad$ **if** $\nexists m' \in C_g : deny(m', \tau(g, m'))$ **then**
19: $\quad\quad\quad del_{flag} \leftarrow false$
20: $\quad\quad\quad$ **break**
21: $\quad\quad$ **end if**
22: $\quad$ **end for**
23: $\quad$ **if** del_{flag} **then**
24: $\quad\quad allocated_{new}(m) \leftarrow false$
25: $\quad\quad \Pi_{new}(m) \leftarrow \emptyset$
26: $\quad$ **end if**
27: **end for**

The optimization targets isolation requirements only, as they induce firewall allocation by requiring at least one denying enforcement point on every matching flow. Let $R_I = \{r \in R \mid r.a = deny\}$ be the set of isolation NSRs. During initialization (lines 1–6), it creates a copy of the merged global configuration, from which it will later only remove firewalls and rules, without introducing new enforcement points or changing the semantics of existing ones.

Specifically, the algorithm builds the set of candidate firewalls (lines 7–11) by considering only cut flows induced by isolation NSRs. For each $r \in R_I$ and each $f \in F_r^{cut}$, it collects the allocated APs crossed by f and stores them in the set S. Next, the algorithm evaluates each $m \in S$ independently (lines 13–22). For a candidate firewall m, it computes the set of all isolation flows whose path includes m:

$$F_m^I = \{g \in \bigcup_{r \in R_I} F_r \mid m \in \pi(g)\} \tag{22}$$

Firewall m can be removed only if every flow in F_m^I remains denied elsewhere. Accordingly, for each $g \in F_m^I$, the algorithm considers the alternative enforcement points on the same path, $C_g = (\pi(g) \cap N_P) \setminus \{m\}$, and checks whether there exists some $m' \in C_g$ such that $deny(m', \tau(g, m'))$ holds. If this condition fails for even one flow, then m is necessary and must be retained. Otherwise, the filtering effect of m is fully subsumed by other firewalls on the same paths, and m is redundant.

When this redundancy condition holds, the algorithm deallocates m and removes its rule set from the optimized configuration (lines 23–26). Since the procedure removes only firewalls

whose denying effect is already provided by other allocated firewalls, it preserves the satisfaction of all reachability NSRs as well as of every isolation NSR.

Finally, as Algorithm 2 is greedy, its output may depend on the order in which the candidate firewalls in S are processed. Consider three cut isolation flows f_1, f_2, f_3 and three allocated firewalls m_1, m_2, m_3, such that m_1 denies all three flows, m_2 denies f_1 and f_2, and m_3 denies f_3. If m_1 is evaluated first, it is removed because m_2 and m_3 jointly preserve all denials, leaving two firewalls. Conversely, if m_2 and m_3 are processed first, both are removed because m_1 still denies the corresponding flows. Therefore, different processing orders can produce different, yet correct, configurations.

Complexity. Copying the input configuration requires $O(|N_A|)$ time, while identifying candidate firewalls by traversing the cut isolation flows costs $O(H)$. For each cut isolation flow f, up to $a_f = |\pi(f) \cap N_P|$ candidate firewalls (i.e., APs) may be evaluated, each against at most $a_f - 1$ alternative APs, resulting in at most $a_f(a_f - 1)$ checks. Across all isolation flows, this contributes $O\left(\sum_{f \in F_{R_I}} a_f^2\right)$ time. Since $\sum_{f \in F_{R_I}} a_f \leq \sum_{f \in F_{R_I}} |\pi(f)| \leq H$, this term is upper-bounded by $O(H^2)$ in the worst case. The resulting time complexity is $O(|N_A| + H^2)$. The additional space complexity is $O(|N_A| + Q + H)$, accounting for the copied configuration, its firewall policies, and the candidate-firewall and isolation-flow associations used during redundancy checks.

VII. Implementation and validation

The proposed approach was implemented in Java as an extension of the VEREFOO framework, adding support for the different phases presented in Section III. The clustering step relies on a publicly available Java implementation of the Leiden algorithm [18], while MaxSMT solving is performed using the Z3 theorem prover [19]. All experiments were run on a machine equipped with an Intel Core Ultra 7 265 processor, 64 GB of RAM, and Z3 version 4.14.1.

The validation campaign considers three topology families. The first family is based on the Texas 2000 communication network of the Texas electric transmission grid [20], modeled as a hierarchical three-level architecture with one Balancing Authority, multiple Utility Control Centers, and multiple Substations. The tested instances are generated by increasing the number of Utility Control Centers and Substations, as well as the number of NSRs. Therefore, the topology size and the number of APs vary across the different instances. The second family is based on the Stanford backbone network [21], derived from real configuration data for a campus-style topology. In this case, the topology is fixed in size, and only the number of NSRs varies. Texas 2000 and Stanford were chosen as representative realistic network topologies. The third family is a Hub&Spoke topology, introduced to complement the reference benchmarks with a more controlled setting in which both topology size and NSR cardinality can be varied directly. For all scenarios, the relative NSR distribution was fixed to 50% isolation and 50% reachability NSRs.

The evaluation process consisted of two phases. First, we performed a preliminary analysis to determine a suitable parameterization of the clustering phase. Since the Leiden algorithm's output depends on multiple parameters, this step was necessary to ensure a balanced final decomposition. Second, the selected parameterization was used to compare the proposed parallelized approach with the monolithic VEREFOO implementation regarding both performance and optimality, and to carry out an ablation study.

A. Clustering Analysis

Because partition quality directly affects the proposed approach, we conducted a preliminary analysis to select suitable parameter configurations for the Leiden algorithm. The analysis considered all topology families introduced above, along with increasing NSR cardinalities, to capture both the effect of topology size and the traffic flows induced by the NSRs. For each topology and instance size, we tested all admissible discrete values and representative values for continuous parameters. In particular, we sampled the resolution parameter logarithmically between 10^{-3} and 10, as preliminary runs showed negligible changes beyond this range.

Each parameter combination was evaluated through a decomposition-quality score designed to capture both performance and optimality. The score therefore combines three factors: (i) the number of cut flows, (ii) the balance of the number of nodes across clusters, and (iii) the balance of the number of assigned NSRs across clusters. The first captures optimality, since cut flows can introduce duplicated enforcement after merging. The other two capture performance, since the slowest cluster-scoped instance determines the parallel critical path.

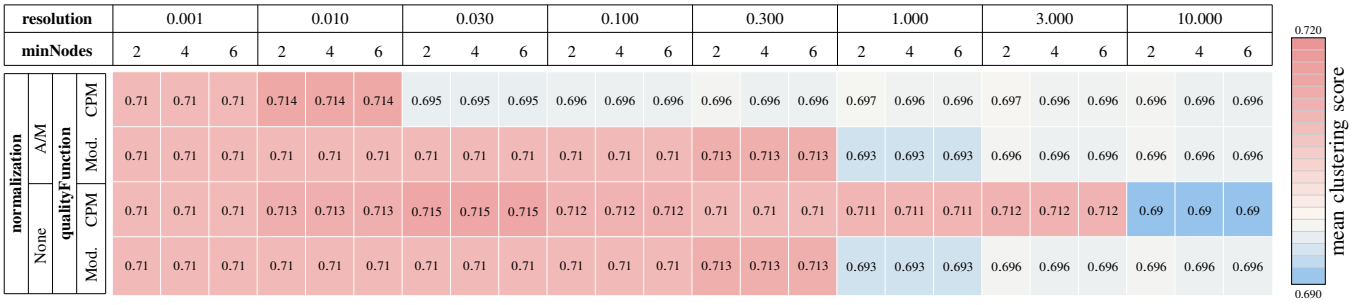
Let $G_A = (N_A, E_A)$ be the input AG, let R be the NSR set, and let $P = \{N_{F,C_1}, \ldots, N_{F,C_k}\}$ be the partition of the flow-graph nodes into k clusters. Each cluster C_i induces one subproblem, described by a cluster-scoped AG $G_{A,C_i} = (N_{A,C_i}, E_{A,C_i})$ and an assigned NSR subset R_{C_i}. We define the decomposition-quality score as:

$$S = \begin{cases} 0, & \text{if } |P| = 1 \\ & \vee\, \exists\, G_{A,C_i} : |N_{A,C_i}| > N_{\max} \\ & \vee\, \exists\, R_{C_i} : |R_{C_i}| > R_{\max} \\ \lambda \cdot S_O + (1-\lambda) \cdot S_P, & \text{otherwise,} \end{cases} \tag{23}$$

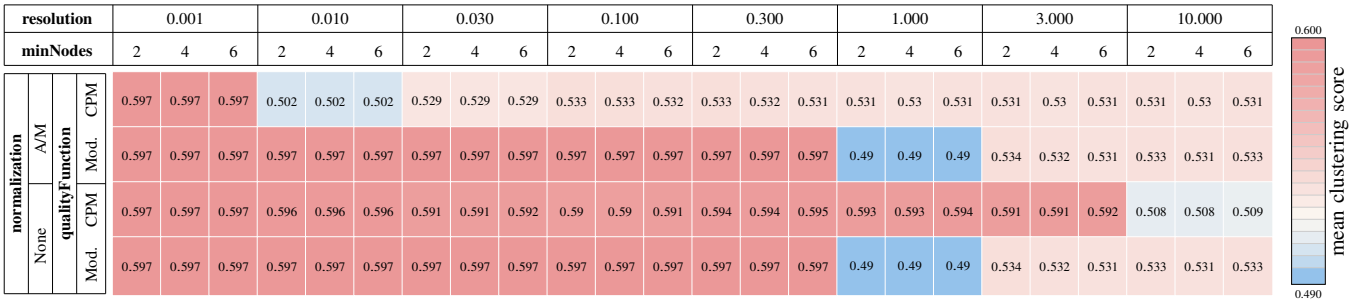
$$S_P = \frac{1}{2}\left(\frac{1}{1 + \frac{\sigma(\{|N_{A,C_i}|\}_{i=1}^{k})}{\mu(\{|N_{A,C_i}|\}_{i=1}^{k})}} + \frac{1}{1 + \frac{\sigma(\{|R_{C_i}|\}_{i=1}^{k})}{\mu(\{|R_{C_i}|\}_{i=1}^{k})}} \right), \qquad S_O = 1 - \frac{|F_R^{cut}|}{|F_R|} \tag{24}$$

The parameter λ controls the trade-off between optimality and performance. Since part of the optimality loss is later mitigated by the post-processing phase, we set $\lambda = 0.3$, thereby assigning slightly higher weight to the performance component. The score is forced to zero whenever the partition is trivial, i.e., when a single partition corresponds to G_A, or any of the generated subproblems exceeds hard-complexity limits. These limits are derived from benchmarking the monolithic VEREFOO implementation and are intended to exclude decompositions that are balanced in relative terms but whose derived subproblems remain too large to solve efficiently.

Scores were averaged over 10 independent runs per parameter combination. A preliminary single-factor analysis reduced



(a) 50 NSRs



(b) 100 NSRs

Fig. 3: Heatmaps of average clustering score for Stanford topology at different NSR values

the search space: Leiden iterations were fixed to 5, the random seed showed negligible influence, *minNodes* = 1 was discarded due to consistently poorer results, and the A and M normalization strategies were collapsed into a single representative level, since they showed indistinguishable behavior. The remaining parameters were retained for further analysis.

The role of the other parameters is summarized as follows. The resolution parameter controls partition granularity: smaller values tend to produce fewer and larger clusters, while larger values favor finer decompositions. The parameter *minNodes* imposes a lower bound on cluster size and limits the creation of very small subnetworks. The quality function, i.e., CPM or modularity, determines the objective Leiden optimizes. After filtering, the remaining parameter combinations were analyzed separately for each NSR cardinality using heatmaps.

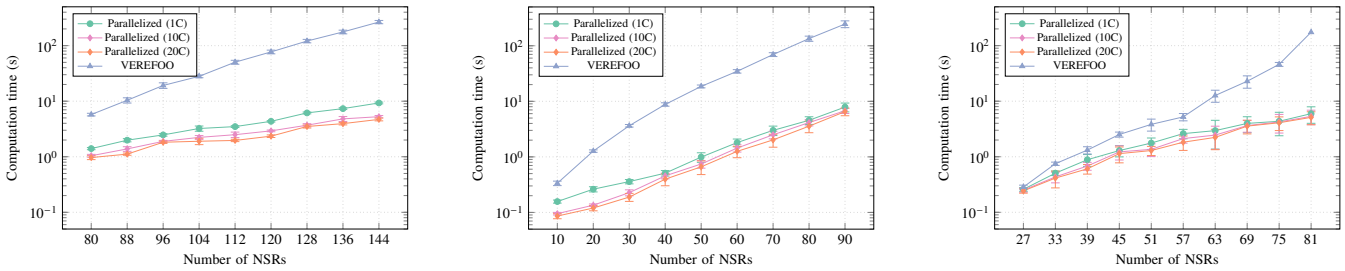
Fig. 3 shows a representative example for the Stanford topology. At 50 NSRs, the score surface in Fig. 3a is relatively flat, indicating limited sensitivity to parameter choice and suggesting that many configurations are adequate. At 100 NSRs, the effect of parameter selection becomes more visible in Fig. 3b, and configurations with *resolution* = 1.0 consistently yield lower scores. The same trend remains visible at larger NSR cardinalities: lower-resolution values tend to provide better decompositions, whereas the effect of *minNodes* is more limited on this topology, indicating that Leiden rarely produces excessively small clusters in this setting. Overall, the preferred parameterization varies with problem size, but broad near-optimal regions emerge rather than isolated maxima.

For this reason, we did not select the single best-scoring configuration for each NSR cardinality. Instead, we retained the near-optimal region, defined as the set of parameter combinations whose scores lie within a fixed tolerance of the best observed score. For each NSR cardinality, we selected one representative configuration by choosing the values that appeared most consistently within that region, and, in case of ties, preferring simpler settings. The same procedure was repeated for all topology families. For Texas 2000, Hub&Spoke, and Stanford, respectively, the selected settings are *resolution* = $\{0.10, 0.10, 0.01\}$ and *minNodes* = $\{2, 2, 4\}$; CPM is used for the first two and modularity for Stanford, while A/M normalization is enabled only for Texas 2000.

B. Performance Evaluation

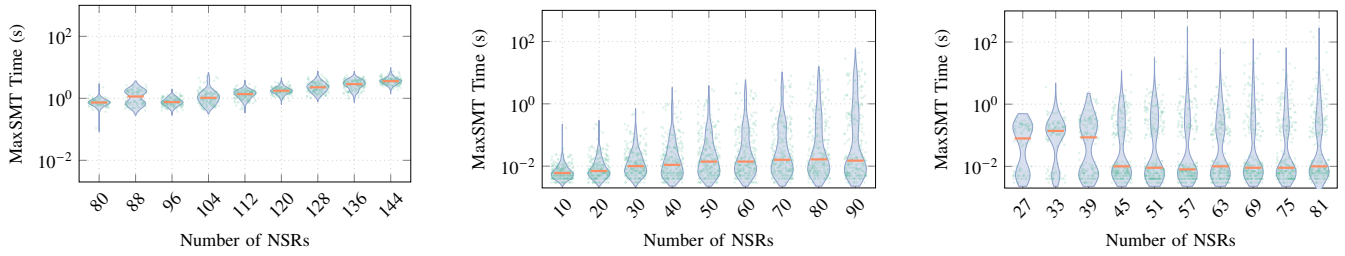
Fig. 4 compares the computation time of the proposed approach against the original monolithic VEREFOO. Each point reports the average over 50 runs and its 95% confidence interval. We measure computation time end-to-end, from the problem inputs to the final configuration. Consequently, both measurements include shared phases, such as computing the NSR-induced traffic flows F_R, which affect their execution times equally. The parallelized approach additionally includes the costs of problem segmentation and merging, accounting for ∼10–50 ms and ∼0–10 ms, respectively. These costs are negligible compared with the overall runtime, which is dominated by MaxSMT solving. We primarily evaluate the proposed approach using 1 and 20 logical cores, representing the minimum and maximum degrees of concurrency available on the experimental platform. An intermediate configuration using 10 logical cores is also included in Fig. 4 to characterize intermediate scaling behavior.

The 1 core configuration captures the benefits of problem decomposition without inter-cluster parallelism, whereas its comparison with the 20 core configuration isolates the additional contribution of parallel execution. We quantify these effects through $G_D = 1 - T_{1C}/T_V$, $G_P = 1 - T_{20C}/T_{1C}$, and $G_O = 1 - T_{20C}/T_V$, which measure the runtime reduction due to



(a) Texas 2000 (b) Stanford (c) Hub&Spoke

Fig. 4: Time performance comparison between the parallelized approach and the original VEREFOO



(a) Texas 2000 (b) Stanford (c) Hub&Spoke

Fig. 5: Distribution of cluster-scoped MaxSMT resolution times

	Texas 2000			Stanford			Hub&Spoke		
NSRs	**80**	**112**	**144**	**10**	**50**	**90**	**27**	**51**	**81**
G_D	75.6%	93.1%	96.5%	53.1%	94.7%	96.8%	9.1%	53.9%	96.6%
G_P	31.0%	43.6%	49.4%	45.2%	33.2%	17.1%	6.2%	26.7%	14.3%
G_O	83.2%	96.1%	98.2%	74.3%	96.4%	97.3%	14.7%	66.2%	97.1%

TABLE II: Runtime reduction breakdown

decomposition, parallel execution, and their combined effect, respectively. Here, T_V, T_{1C}, and T_{20C} denote the execution times of VEREFOO and of the parallelized approach using one and 20 logical cores. TABLE II reports these values for three representative sizes of each topology.

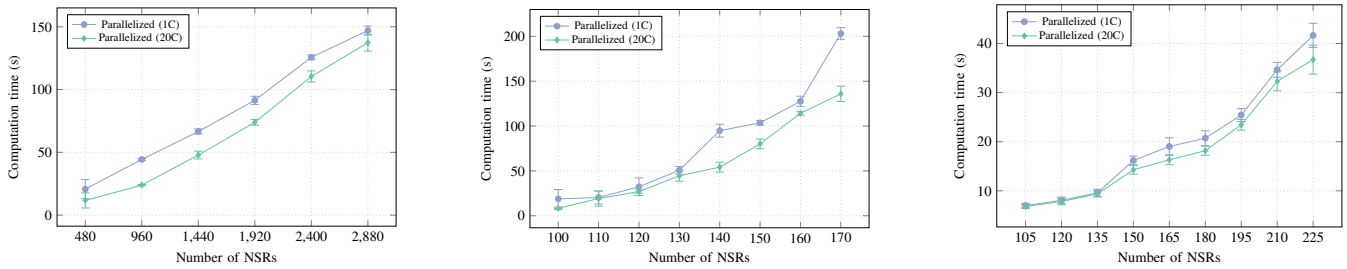
Across all topology families, the proposed approach exhibits a significantly slower runtime growth as the number of NSRs increases. This behavior reflects the effect of decomposition: instead of solving a single MaxSMT instance, the problem is divided into smaller, loosely coupled subproblems, whose solutions are later reconciled.

On Texas 2000 (Fig. 4a), the performance gap widens steadily with the number of NSRs. Using 20 logical cores, the proposed method increases from 0.97 s at 80 NSRs to 4.71 s at 144 NSRs, while monolithic VEREFOO grows from 5.75 s to 265.87 s over the same range, corresponding to an overall runtime reduction from 83.2% to 98.2%. This trend is expected, as the topology expands together with the number of requirements: newly introduced NSRs tend to remain localized within new network regions, promoting balanced partitions and limiting cross-cluster dependencies. TABLE II shows that both effects contribute to this gain. G_D increases from 75.6% to 96.5%, while G_P increases from 31.0% to 49.4%. Thus, as the topology grows, decomposition produces increasingly smaller problems relative to the monolithic one, while the expanding cluster set also exposes more useful parallelism.

The Stanford topology (Fig. 4b) exhibits the same qualitative advantage. Using 20 logical cores, the proposed approach grows from 0.086 s at 10 NSRs to 6.53 s at 90 NSRs, whereas monolithic VEREFOO increases from 0.34 s to 246.37 s, yielding an overall runtime reduction from 74.3% to 97.3%. Unlike Texas 2000, Stanford has a fixed topology, so an increasing number of requirements is absorbed by the same cluster set. As a result, decomposition reduces solver coupling but cannot fully offset the growth of local problem size. Nevertheless, the parallelized approach remains clearly preferable. Indeed, TABLE II shows a different balance: G_D increases from 53.1% to 96.8%, whereas G_P decreases from 45.2% to 17.1%. Thus, decomposition accounts for most of the gain on larger instances, while the fixed topology limits the additional parallelism available as the NSR set grows.

A similar trend is observed in the Hub&Spoke topology (Fig. 4c). In this case, using 20 logical cores, the proposed approach increases from 0.24 s at 27 NSRs to 5.11 s at 81 NSRs, whereas VEREFOO grows from 0.29 s to 174.38 s. The overall reduction is limited on the smallest instance (14.7%) but reaches 97.1% on the largest. Beyond 60 NSRs, the monolithic formulation degrades rapidly, while the parallelized approach remains within a few seconds. This behavior is consistent with the regular structure of the topology, which is well-suited to traffic-aware partitioning and limits inter-cluster dependencies. TABLE II shows that G_D grows from 9.1% to 96.6%, whereas G_P remains smaller and non-monotonic, reaching 26.7% at 51 NSRs and 14.3% at 81 NSRs. The large-instance gain primarily stems from simplifying the MaxSMT problem via decomposition, rather than from concurrent execution.

These results show that decomposition and parallelization substantially reduce MaxSMT solving costs across all tested scenarios, although their relative contribution depends on how



(a) Texas 2000 (b) Stanford (c) Hub&Spoke

Fig. 6: Time performance scalability of the parallelized approach

	NSRs	Texas 2000			Stanford			Hub&Spoke		
		80	**112**	**144**	**10**	**50**	**90**	**27**	**51**	**81**
# Cl.	**Median**	5	8	12	7	11	9	6	13	20
	Q1,Q3	4,5	8,8	12,12	6,7	8,13	8,10	5,7	11,14	19,22
	P95	6	9	13	8	14	14	7	15	23
Nodes/cl.	**Median**	14	15	15	3	7	9	3	3	3
	Q1,Q3	8,15	8,23	12.3,21	3,5	3,13	5,27	3,3	3,3	3,3
	P95	23	26	27	7	26	77	12	13	12
NSRs/cl.	**Median**	18	16	14	1	3	4	4	4	4
	Q1,Q3	17,20	14,17	12,16	1,2	1,6	2,15	4,6	3,6	3,6
	P95	23	18	18	3	15	36	12	13	16

TABLE III: Cluster decomposition statistics

the workload is distributed among clusters. TABLE III characterizes the resulting decompositions by reporting the distribution of the number of clusters and their sizes in terms of nodes and assigned NSRs. Texas 2000 yields comparatively balanced clusters, whereas Stanford and Hub&Spoke exhibit more pronounced upper tails, with compact interquartile ranges but substantially larger P95 values, indicating that a limited number of larger subproblems coexist with many smaller ones. This behavior is reflected in Fig. 5, which shows the corresponding distribution of cluster-scoped MaxSMT resolution times, with medians highlighted by orange lines. Texas 2000 (Fig. 5a) shows comparatively balanced cluster resolution times, so multiple subproblems retain enough work to benefit from concurrent execution. Stanford (Fig. 5b) becomes increasingly skewed: most clusters remain inexpensive, while a few costly instances determine the parallel critical path. Hub&Spoke (Fig. 5c) shows a similar imbalance, with most cluster instances remaining inexpensive and only a limited upper tail requiring substantial solving time. This explains the different G_P trends in TABLE II: parallel execution is most effective when costly MaxSMT work is distributed across several clusters, rather than concentrated in a few. This imbalance limits the additional parallelization gain, but does not necessarily limit the scalability provided by decomposition.

Finally, the scalability of the proposed approach was evaluated on significantly larger problem instances beyond the practical limits of the monolithic VEREFOO, which could be executed only up to the sizes reported in Fig. 4. The parallelized approach was evaluated using both 1 and 20 logical cores until system memory saturation. The different maximum scales reflect how problem complexity is distributed across cluster-scoped instances. Stanford has a fixed topology, so increasing the NSR set progressively enlarges the same local problems. Texas 2000 and Hub&Spoke both grow with the NSR set, but distribute the resulting complexity differently across clusters, as shown in Fig. 5, leading to markedly different scalability. The results, shown in Fig. 6, confirm that the decomposed approach remains scalable at larger problem sizes and reflect the trends observed above. On Texas 2000 (Fig. 6a), runtime using 20 logical cores grows from 11.70 s at 480 NSRs to 137.32 s at 2880 NSRs, compared with 20.69 s and 147.03 s using one core. The additional parallelization gain therefore decreases from 43.5% to 6.6% as the problem grows. On Stanford (Fig. 6b), the 20-core runtime grows from 8.27 s at 100 NSRs to 135.87 s at 170 NSRs, compared with 18.80 s and 203.17 s using one core. Finally, Hub&Spoke (Fig. 6c) shows the most gradual growth: the 20-core runtime increases from 6.78 s at 105 NSRs to 36.71 s at 225 NSRs, while the corresponding one-core values are 7.01 s and 41.65 s. Thus, parallel execution provides only a limited additional gain on this topology, while decomposition still keeps the resulting subproblems inexpensive as the problem grows.

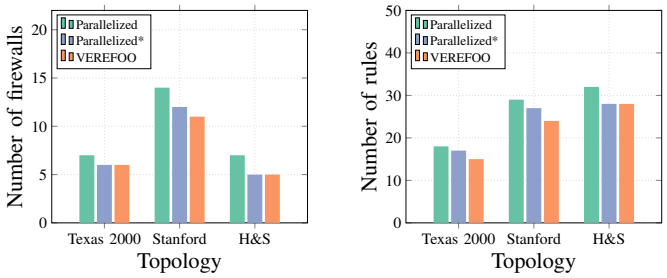
Overall, these results show that decomposition enables the resolution of substantially larger instances, while parallel execution provides an additional gain depending on cluster-workload distribution.

C. Optimality Evaluation

Fig. 7 evaluates the optimality loss induced by decomposition and the effect of post-processing. The results refer to the largest instances considered in Subsection VII-B, i.e., 144 NSRs for Texas 2000, 81 NSRs for the Hub&Spoke topology, and 90 NSRs for Stanford. The figure compares three configurations: the raw merged parallel result (*Parallelized*), the result after post-processing (*Parallelized**), and the original monolithic implementation (*VEREFOO*).

Post-processing removes redundant firewalls introduced by cross-cluster isolation flows. On Texas 2000, the *Parallelized* solution allocates, on average, 7 firewalls, whereas post-processing *Parallelized** reduces this to 6, matching the monolithic VEREFOO result. The same occurs on the Hub&Spoke topology, where the number of allocated firewalls decreases from 7 to 5. On Stanford, the raw parallel solution allocates 14 firewalls, the post-processing reduces this number to 12, the monolithic solution allocates 11. This residual gap is expected; decomposition does not guarantee recovery of global optimum.

A similar trend emerges for the number of configured rules. On Texas 2000, the raw merged solution configures 18 rules,



(a) Allocated firewalls (b) Configured rules

Fig. 7: Optimization evaluation

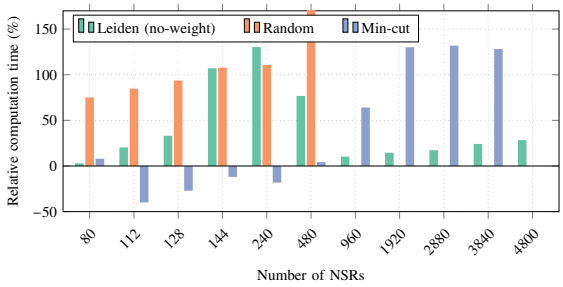


Fig. 8: Comparison with other partitioning strategies

which post-processing reduces to 17, compared with 15 rules in the monolithic baseline. On the Hub&Spoke topology, the number of rules decreases from 32 to 28, exactly matching the monolithic solution. On Stanford, the raw solution requires 29 rules, which post-processing reduces to 27, while the monolithic solution uses 24. Rule reduction is only a side effect of post-processing, as removing a redundant firewall also removes its rule set.

To isolate the contribution of merging from the results discussed above, we then analyzed the rule-set variations introduced by Algorithm 1 separately, focusing on Texas 2000, whose scalability supports larger NSR cardinalities and thus enables evaluating the merge behavior across larger problem sizes. Overall, merging happens for 37.9% of total processed APs; in these events, increases occurred in only 4.6% of runs, with a maximum observed increase of 26.7% (corresponding to +4 rules), and decreases occurred in 67.7% of the cases, with a maximum decrease of -37.4% (corresponding to −10 rules). Grouping tested instances of the same topology into small (80–96 NSRs), medium (104–120), and large (128–144) problems, case 2 of merging remains dominant and stable (51.2%, 51.6%, and 53.1%, respectively), while case 1 becomes progressively more frequent (6.1%, 24.2%, 29.8%) and case 3 decreases accordingly (35.1%, 16.2%, 10.7%). Case 4 remains instead rare (7.6%, 8.0%, 6.4%). Similar behavior was observed on the other topologies.

D. Ablation Study

Finally, we evaluate the proposed traffic-aware partitioning strategy against three alternatives: (*i*) Leiden without flow-derived edge weights, (*ii*) random partitioning, and (*iii*) recursive min-cut partitioning. We use Texas 2000 because it allows both network size and NSR cardinality to increase. For (*ii*) and (*iii*), we set the target number of partitions to that produced by traffic-aware Leiden for the same instance. All subsequent phases described in Section III remain unchanged. Results are averaged over 50 runs.

Fig. 8 reports relative computation time with respect to the proposed approach, where positive values indicate slower execution and negative values faster execution. Leiden (no-weight) is 107.14% and 130.25% slower at 144 and 240 NSRs, respectively. The gap decreases at larger scales, but remains between 10.20% and 28.35% from 960 to 4800 NSRs. Random partitioning is already 75.08% slower at 80 NSRs and reaches 543.41% at 480 NSRs, after which it exceeds the 10-minute timeout. Recursive min-cut is faster on several smaller instances, by up to 39.93% at 112 NSRs, but loses this advantage as the problem grows. From 960 NSRs onward, it is at least 64.06% slower, with the gap exceeding 128% between 1920 and 3840 NSRs, before timing out at 4800 NSRs. Overall, Leiden provides better large-scale behavior than random and recursive min-cut partitioning, while flow-derived weights further improve its end-to-end performance.

We also compare the resulting configuration optimality through the average number of allocated firewalls and configured rules. With respect to the proposed traffic-aware partitioning, Leiden (no-weight), random, and recursive min-cut allocate 0.48%, 2.20%, and 5.07% more firewalls on average, respectively. For firewall rules, Leiden (no-weight) and random require 2.34% and 2.19% more rules, respectively, whereas recursive min-cut requires 7.18% fewer rules. Overall, traffic-aware Leiden achieves the lowest average firewall allocation while remaining competitive in rule count. This result is consistent with the score in Section VII-A, which places greater weight on performance than on optimality.

VIII. Related Work

The approach proposed in this paper decomposes the automatic distributed firewall configuration problem into smaller sub-problems, so that their parallel resolution can lead to better performance. In the literature on network configuration, decomposition has been pursued mainly for other problems, e.g., configuration verification, whereas proposals for firewall configuration automation focus on different objectives.

A. Decomposition of network configuration problems

Problem decomposition has long been a well-established strategy for improving the scalability of complex computational problems. However, in network configuration, it has been primarily used for purposes other than automatic configuration synthesis, e.g., verification and monitoring.

Concerning the decomposition of the configuration verification problem, existing studies can be grouped into three directions [22]–[27]: partition-based verification of the forwarding state, incremental or differential verification of configuration changes, and modular verification of specifications and proof obligations. The most representative approach in the first direction is Libra [22], which partitions the forwarding state into prefix-specific forwarding graphs and independently

verifies them using a MapReduce workflow. The second direction includes incremental approaches such as Delta-net [23], which works on an incremental edge-labeled graph, the algorithm introduced in [24], which performs incremental network configuration verification by regenerating only the affected data-plane state, and DNA [25], which further extends this idea toward differential analysis. Finally, Cocoon [26] and the recent proposal in [27] represent the third direction, where scalability is pursued through modular abstractions and localized verification conditions, respectively.

Concerning the decomposition of the network-wide monitoring problem, HeteroSketch [28] partitions a mixed-integer optimization problem for placing monitoring tasks and allocating resources across heterogeneous devices into disjoint cluster-level sub-problems that are solved concurrently.

Although all these studies show the value of decomposition in improving scalability, they address a problem different from the one considered in this paper, i.e., the automatic synthesis of distributed firewall configurations. Instead, partially related synthesis-oriented approaches are AED [5] and Jinjing [6]. AED [5] incrementally computes configuration updates from existing router configurations by modeling changes as syntax-tree additions and removals and encoding routing semantics as SMT constraints. Jinjing [6] automatically repairs in-network ACL update plans, so as to preserve the desired reachability during ACL evolution. However, also these approaches address problems that are substantially different from the one considered in this paper. In particular, they start from an already deployed routing and access-control infrastructure and focus on updating existing configurations, whereas the approach of this paper jointly computes the firewall allocation scheme and filtering rules from network-wide security requirements. Moreover, they do not follow a parallelization strategy, but rather an incremental configuration-update approach, and they do not address the specific challenges of distributed firewall configuration, as they concern other network functions.

B. Automatic approaches for firewall configuration

In the literature on distributed firewall configuration automation, most studies focus on only one of the two subtasks of the problem, i.e., firewall allocation and rule generation. Examples are the approaches in [3], [29]–[31] for the former, and the ones in [32]–[34] for the latter. However, these works provide only partial solutions with respect to the approach of this paper, as they do not jointly address both tasks.

The only methodologies that address both firewall configuration tasks are the ones proposed in ([4], [35]–[38]). However, most of them, i.e., [35]–[38], rely on simplifying assumptions on the input network topology that limit their applicability to modern network scenarios. Specifically, the algorithms in [35]–[37] can synthesize only service function chains, even though most modern network topologies are now graphs with ramifications. Instead, the approach pursued in [38] automatically builds a service graph composed solely of firewalls interconnecting endpoints, without considering other middleboxes, such as load balancers or network address translators, which are commonly present in current networks. These studies are also limited in terms of scalability and performance results. [35] and [37] do not provide explicit information about managed network size and time performance. The approach in [36] is validated on scenarios with up to 20 functions, including firewalls, with an execution time of about 4 s, but without providing formal correctness guarantees. The technique in [38] considers scenarios with up to 60 firewalls, but focuses its evaluation on optimization quality rather than on execution time, which is not explicitly reported.

The most feature-complete and best-performing approach to date is VEREFOO [4]. In addition to being the only approach providing both formal correctness and optimization, it can also manage graphs with up to 100 APs for firewalls. Nevertheless, its execution time can still exceed several minutes on some problem instances, due to the complexity of solving a single monolithic MaxSMT formulation across the entire graph. Such execution times may sometimes be unacceptable in scenarios where rapid configuration computation is required.

In summary, existing approaches either address only part of the firewall configuration problem or suffer from limitations in topology modeling, scalability, or execution time. The proposed approach goes beyond them by pursuing an efficient parallelization strategy tailored to realistic service graphs.

IX. Conclusions

This paper presented a traffic-aware graph decomposition solution for the automated configuration of distributed firewalls, which solves cluster-scoped subproblems by parallelizing the formal VEREFOO approach. The evaluation showed increasing runtime reductions across different topologies, while maintaining acceptable levels of optimality. Despite these promising results, a limitation is the dependence on the Leiden parameterization, which directly affects decomposition quality and subproblem size. While the proposed parameter selection strategy provides robust results across the evaluated topologies, it relies on an offline analysis and does not fully adapt to arbitrary network structures and traffic patterns.

Future work will focus on improving the adaptability and generality of the clusterization phase. In particular, automated parameter selection techniques, including data-driven and machine learning-based methods that infer configurations from network features, represent a promising direction for making the solution self-tuning. Alternative delegation strategies for cut flows may also be explored, e.g., constraining enforcement toward cluster boundaries to favor subsequent consolidation. Finally, integrating the proposed decomposition with complementary paradigms, such as hierarchical solving or incremental reconfiguration as already explored for VEREFOO [7], may further extend its applicability to dynamic environments.

References

[1] Y. E. Sung, X. Tie, S. H. Y. Wong, and H. Zeng, "Robotron: Top-down network management at facebook scale," in *Proceedings of the ACM SIGCOMM 2016 Conference, Florianopolis, Brazil, August 22-26, 2016*. ACM, 2016, pp. 426–439.

[2] D. Bringhenti, G. Marchetto, R. Sisto, and F. Valenza, "Automation for network security configuration: State of the art and research trends," *ACM Comput. Surv.*, vol. 56, no. 3, pp. 57:1–57:37, 2024.

[3] M. A. Rahman and E. Al-Shaer, "Automated synthesis of distributed network access controls: A formal framework with refinement," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 2, 2017.

[4] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "Automated firewall configuration in virtual networks," *IEEE Trans. Dependable Secur. Comput.*, vol. 20, no. 2, pp. 1559–1576, 2023.

[5] A. Abhashkumar, A. Gember-Jacobson, and A. Akella, "AED: incrementally synthesizing policy-compliant and manageable configurations," in *16th Int. Conf. on emerging Networking EXperiments and Technologies, Barcelona, Spain, December, 2020*. ACM, 2020, pp. 482–495.

[6] B. Tian *et al.*, "Safely and automatically updating in-network ACL configurations with intent language," in *Proceedings of the ACM Special Interest Group on Data Communication, SIGCOMM 2019, Beijing, China, August 19-23, 2019*. ACM, 2019, pp. 214–226.

[7] F. Pizzato, D. Bringhenti, R. Sisto, and F. Valenza, "Automatic and optimized firewall reconfiguration," in *NOMS 2024 IEEE Network Operations and Management Symposium, Seoul, Republic of Korea, May 6-10, 2024*. IEEE, 2024, pp. 1–9.

[8] V. A. Traag, L. Waltman, and N. J. van Eck, "From Louvain to Leiden: guaranteeing well-connected communities," *Scientific Reports*, vol. 9, no. 1, p. 5233, Mar. 2019.

[9] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical review E*, vol. 69, no. 2, p. 026113, 2004.

[10] V. A. Traag, P. Van Dooren, and Y. Nesterov, "Narrow scope for resolution-limit-free community detection," *Phys. Rev. E*, vol. 84, p. 016114, Jul 2011.

[11] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, 2008.

[12] Z. Yang, R. Algesheimer, and C. J. Tessone, "A comparative analysis of community detection algorithms on artificial networks," *Scientific Reports*, vol. 6, no. 1, p. 30750, 2016.

[13] A. Lancichinetti and S. Fortunato, "Community detection algorithms: A comparative analysis," *Phys. Rev. E*, vol. 80, p. 056117, Nov 2009.

[14] D. Bringhenti, S. Bussa, R. Sisto, and F. Valenza, "A two-fold traffic flow model for network security management," *IEEE Transactions on Network and Service Management*, vol. 21, no. 4, pp. 3740–3758, 2024.

[15] E. Al-Shaer, H. H. Hamed, R. Boutaba, and M. Hasan, "Conflict classification and analysis of distributed firewall policies," *IEEE J. Sel. Areas Commun.*, vol. 23, no. 10, pp. 2069–2084, 2005.

[16] H. Hu, G. Ahn, and K. Kulkarni, "Detecting and resolving firewall policy anomalies," *IEEE Trans. Dependable Secur. Comput.*, vol. 9, no. 3, pp. 318–331, 2012.

[17] D. Bringhenti, S. Bussa, R. Sisto, and F. Valenza, "Atomizing firewall policies for anomaly analysis and resolution," *IEEE Trans. Dependable Secur. Comput.*, vol. 22, no. 3, pp. 2308–2325, 2025.

[18] N. J. Van Eck, V. Traag, and L. Waltman, "networkanalysis," Aug. 2023. [Online]. Available: https://github.com/CWTSLeiden/networkanalysis

[19] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *Proc. of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 337–340.

[20] P. Wlazlo *et al.*, "A cyber topology model for the texas 2000 synthetic electric power grid," in *2019 Principles, Systems and Applications of IP Telecommunications, IPTComm 2019, Chicago, IL, USA, October 14-16, 2019*. IEEE, 2019, pp. 1–8.

[21] P. Kazemian, "Stanford_backbone: Benchmark for header space analysis (hsa-python)," Bitbucket repository., 2012.

[22] H. Zeng *et al.*, "Libra: Divide and conquer to verify forwarding tables in huge networks," in *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, Seattle, WA, USA, April 2-4, 2014*, 2014, pp. 87–99.

[23] A. Horn, A. Kheradmand, and M. R. Prasad, "Delta-net: Real-time network verification using atoms," in *14th USENIX Symposium on Networked Systems Design and Implementation, Boston, MA, USA, March 27-29, 2017*, 2017, pp. 735–749.

[24] P. Zhang, Y. Huang, A. Gember-Jacobson, W. Shi, X. Liu, H. Yang, and Z. Zuo, "Incremental network configuration verification," in *HotNets '20: The 19th ACM Workshop on Hot Topics in Networks, Virtual Event, USA, November 4-6, 2020*. ACM, 2020, pp. 81–87.

[25] P. Zhang, A. Gember-Jacobson, Y. Zuo, Y. Huang, X. Liu, and H. Li, "Differential network analysis," in *19th USENIX Symposium on Networked Systems Design and Implementation, Renton, WA, USA, April 4-6, 2022*, 2022, pp. 601–615.

[26] L. Ryzhyk *et al.*, "Correct by construction networks using stepwise refinement," in *Symposium on Networked Systems Design and Implementation, NSDI Boston, MA, USA, March 27-29, 2017*, pp. 683–698.

[27] H. Chen, "Incremental network configuration verification via localized subspecification," in *Proceedings of the 2nd Workshop on Formal Methods Aided Network Operation, FMANO 2025, Coimbra, Portugal, September 8-11, 2025*. ACM, 2025, pp. 55–57.

[28] A. Agarwal, Z. Liu, and S. Seshan, "Heterosketch: Coordinating network-wide monitoring in heterogeneous and dynamic networks," in *19th USENIX Symposium on Networked Systems Design and Implementation, Renton, WA, USA, April 4-6, 2022*, 2022, pp. 719–741.

[29] E. J. Scheid *et al.*, "Inspire: Integrated nfv-based intent refinement environment," in *Proc. of the IFIP/IEEE Symp. on Integrated Network and Service Management (IM17)*, 2017.

[30] Z. Hao, Z. Lin, and R. Li, "A sdn/nfv security protection architecture with a function composition algorithm based on trie," in *Proc. of the 2nd Intern. Conf. on Computer Science and Application Engineering (CSAE18)*, 2018.

[31] A. S. Sendi, Y. Jarraya, M. Pourzandi, and M. Cheriet, "Efficient provisioning of security service function chaining using network security defense patterns," *IEEE Trans. Services Comput.*, vol. 12, no. 4, 2019.

[32] D. Ranathunga, M. Roughan, P. Kernick, and N. Falkner, "The mathematical foundations for mapping policies to network devices," in *Proc. of the 13th Int. Joint Conf. on e-Business and Telecommunications*, 2016.

[33] A. Sahu, P. Wlazlo, N. Gaudet, A. Goulart, E. Rogers, and K. Davis, "Generation of firewall configurations for a large scale synthetic power system," in *Proc. of the IEEE Texas Power and Energy Conference*. IEEE, 2022, pp. 1–6.

[34] N. Wintering, E. Lanfer, and N. Aschenbruck, "Automating network perimeter threat prevention for decentralized network administration," in *20th International Conference on Network and Service Management, Prague, Czech Republic, October 28-31, 2024*. IEEE, 2024, pp. 1–7.

[35] N. Schnepf, R. Badonnel, A. Lahmadi, and S. Merz, "Rule-based synthesis of chains of security functions for software-defined networks," *ECEASST*, vol. 76, 2018.

[36] C. Basile, F. Valenza, A. Lioy, D. R. Lopez, and A. P. Perales, "Adding support for automatic enforcement of security policies in NFV networks," *IEEE/ACM Trans. Netw.*, vol. 27, no. 2, 2019.

[37] C. Yang, X. Mi, Y. Ouyang, R. Dong, J. Guo, and M. Guizani, "SMART intent-driven network management," *IEEE Commun. Mag.*, vol. 61, no. 1, pp. 106–112, 2023.

[38] M. Yoon, S. Chen, and Z. Zhang, "Minimizing the maximum firewall rule set in a network with multiple firewalls," *IEEE Trans. Computers*, vol. 59, no. 2, 2010.



Francesco Pizzato received his M.Sc. degree in computer engineering from Politecnico di Torino, Italy, in 2023, where he is currently pursuing his Ph.D. degree in control and computer engineering. His research interests include novel network and cloud technologies, automatic orchestration and configuration of security functions, and cloud security.



Daniele Bringhenti received the M.Sc. degree (summa cum laude) and the Ph.D. degree (summa cum laude) in computer engineering from the Politecnico di Torino, Torino, Italy, in 2019 and 2022, respectively, where he is currently a fixed-term Assistant Professor. His research interests include novel networking technologies, automatic orchestration and configuration of security functions in virtualized networks, formal verification of network security policies.



Fulvio Valenza received the M.Sc. degree (summa cum laude) and the Ph.D. degree (summa cum laude) in computer engineering from the Politecnico di Torino, Torino, Italy, in 2013 and 2017, respectively, where he is currently an Associate Professor. His research activity focuses on network security policies, orchestration and management of network security functions in SDN/NFV-based networks, and threat modeling.