Transforming the Network into a Filter: Distributed Firewall Rules for Time-Critical Traffic

Lukas Bechtel*†, Samuel Müller*, Michael Menth§, Tobias Heer*†
*University of Applied Sciences Esslingen, Germany, {lukas.bechtel,samuel.mueller,tobias.heer}@hs-esslingen.de
§Chair of Communication Networks University of Tuebingen, Germany, menth@uni-tuebingen.de
†Belden Inc., Neckartenzlingen, Germany

***Abstract*—Industrial networks require strict security policies while supporting time-sensitive communication for automation and control processes. Traditional centralized firewalls enforce security by filtering traffic between network segments but introduce unacceptable delays for real-time applications. This paper presents a novel approach that distributes firewall rules across industrial switches, leveraging their Access Control Lists (ACLs) to transform the entire network into a unified, low-latency filtering system. The proposed algorithm accounts for challenges such as rule semantics adaptation, dynamic end-device locations, network redundancy, and ACL resource constraints. It systematically analyzes network topology, calculates rule placement strategies, and ensures correct filtering behavior across distributed enforcement points. Evaluation results demonstrate that our approach significantly reduces filtering delays while maintaining security, enabling real-time communication in industrial environments.**

***Index Terms*—TSN, Network Security, Network Segmentation**

I. Introduction

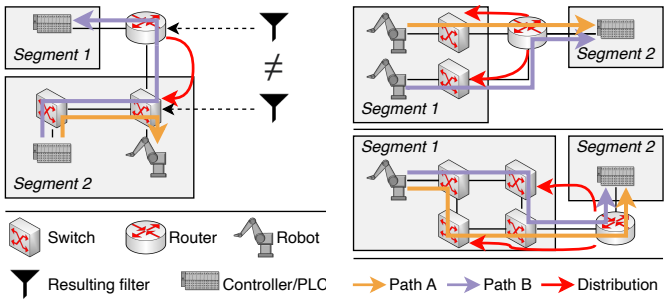
Industrial networks are the core of modern manufacturing and automation processes. These industrial networks must support time-critical communications for automation and manufacturing processes while maintaining strict security requirements. A typical industrial network organizes devices into segments based on functionality, such as machines or production cells. Firewalls filter all traffic between network segments to maintain security. Administrators can deploy central firewalls to filter traffic using only a few devices, making configuration easy by centralizing all rules. However, central firewalls fail to filter time-critical traffic quickly enough [1] because of long rulesets and the delays of software-based filtering. Therefore, time-critical traffic currently stays within one network segment of industrial networks without passing a firewall.

This paper proposes the improvement of packet filtering performance by distributing a large ruleset from a central firewall to multiple devices. Each device will filter a subset of the original ruleset. Specifically, we propose distributing rulesets from the central firewall to industrial switches that use Access Control Lists (ACLs). ACLs are available but commonly unused simple stateless hardware filters on most industrial switches. This proposal transforms a whole network segment into a distributed filter. However, administrators cannot distribute rules directly from a firewall to ACLs without modifications, as the distribution of rules opposes several challenges we solve in this paper.

One of the main challenges is the changing semantics of a single rule based on its position in a network, as shown in Figure 1a. Initially, the rule is located on the router, acting as a central firewall. At this location, the ruleset only receives the purple traffic. Initially, the rule only specifies the source of the traffic, as the location of the firewall limits the possible destinations to the receiver of the purple traffic. However, if distributed to an ACL on one of the switches in *Segment 2*, the ruleset receives the purple and orange traffic. Hence, a rule specifying the source and not the destination also matches the orange traffic, which is unintended.

Figure 1b shows two additional challenges during the distribution of a rule. The upper example displays a rule matching packets from two different senders. As a result, the distribution algorithm must distribute the rule to both switches in *Segment 1*. In the lower example, packets can take different paths to their targets due to network redundancy. Therefore, the distribution algorithm might need to distribute the rule to multiple switches to cover all paths of the packets.

The highlighted challenges are examples of the complexity of the distribution problem. Further challenges are the different numbers of supported ACL rules of different switch types, typically less than 256 rules [2]–[5], and dynamic device locations. These challenges amplify the complexity shown in Figures 1a and 1b. Our approach addresses these challenges through a novel algorithm that ensures correctness, optimizes rule placement, and significantly reduces filtering delays.



(a) Changing semantics of rules with implicit filter parameters.

(b) Topological challenges during the distribution of rules.

Fig. 1: Ruleset distribution from a central firewall to distributed ACLs.

This work has been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 528745080 - FIP 68. Additionally, this research was supported by the project "FHP: Qualifizierung und Entwicklung des professoralen Personals der Hochschule Esslingen für zukunftsweisende Themen" (FKZ: 03FHP115) as part of the Federal States Program "FH-Personal", funded by the BMBF and MWK Baden-Württemberg. The authors alone are responsible for the content of the paper.

Accepted for publication in Proceedings of *IEEE International Conference on Factory Communication Systems (WFCS)*, June 10-13, 2025

In the next section, we present the use case for the rule distribution in more detail. Section III details the algorithm and the correctness of the presented approach. In Section IV, we evaluate the runtime and efficiency of the distribution algorithm. As this is a novel approach, Section V presents a differentiation from other types of optimizations in the context of firewalls. Finally, we conclude the paper in Section VI.

II. Use-Case and Motivation

This section details the structure and configuration of industrial networks. Based on this knowledge, we motivate the distributed filtering approach.

Industrial networks are hierarchical, grouping devices into segments based on their role in the production process. Figure 2 shows a hierarchical example network with three layers. On the lowest layer, industrial networks have machine networks executing the actual physical processes using sensors and actuators. Today, this layer includes the Programming Logic Controllers (PLCs) to control the physical process. Depending on the size of the network, multiple layers above the machine networks aggregate networks to production cells, production lines, and a factory backbone.

Communication in today's industrial networks follows the hierarchical structure of the network. Time-critical communication to control the process resides within a network segment. Figure 2 presents the time-critical communication in *machine A* between the *controller* and the *robot*. Monitoring traffic crosses the boundaries of network segments to report the status of components, such as the *robot* in *machine B*, to central monitoring workstations, such as the *workstation* in *production line A*. Typical machine networks have a size of 10 to 100 devices, which need to be controlled by a PLC and monitored by a monitoring system. In future networks, virtual PLCs (vPLC) in local datacenters will replace the PLCs in machine networks to enhance backup and maintenance capabilities. Therefore, also the control traffic will cross the network segments, as shown in Figure 2 between *machine C* and the *local datacenter*.

Industrial networks must comply with different regulations and fulfill market-specific certifications. A dominant example of such security guidelines is the IEC 62433 standard [6], which introduces the concept of *zones and conduits*. In a hierarchical network, every independent network segment is a zone, and firewalls act as conduits, enforcing access control. The designed and intended communication relationships between devices in different zones define the ruleset on the firewalls to filter the traffic. The hierarchical structures in industrial networks with *future control traffic* and *monitoring traffic* typically result in 10 to 200 firewall rules above the machine networks. The firewalls in the *factory backbone* aggregate traffic from multiple machine and production line networks, having more than 1,000 rules.

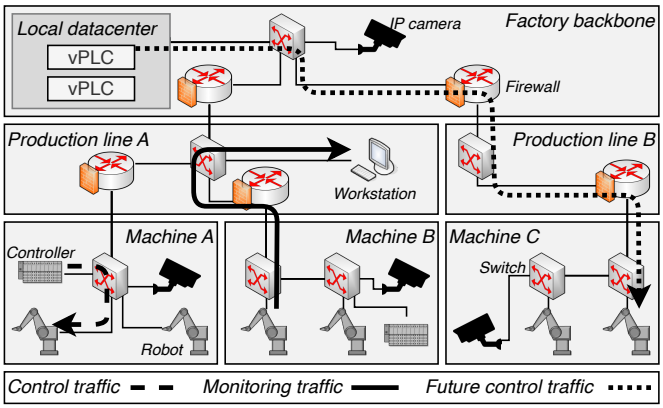


Fig. 2: Example industrial network with hierarchical network segments.

For control loops on PLCs in industrial applications, the network needs to forward packets within application-specific end-to-end latencies. Applications with drives, such as conveyor belts or robots, require a total end-to-end latency of less than 500 µs. This latency includes the forwarding along the complete communication path with 10 to 40 switches and one to five firewalls for control applications in local datacenters. In previous work, we measured the forwarding latency of state-of-the-art industrial firewalls and ACLs [1]. For firewalls, we measured a forwarding latency of 60 µs with a single firewall rule, 90 µs for 200 rules, and 200 µs for 1,000 rules. Therefore, the typical number of firewall rules can easily contradict the time-critical requirements of industrial applications. In comparison, the typical industrial switch has a forwarding latency of 1 µs to 5 µs, and the ACL configuration on switches does not affect this latency. Consequently, every rule moved to an ACL on a switch reduces the forwarding latency on that path. For example, distributing 800 rules from a firewall with 1,000 initial rules reduces the forwarding latency by around 110 µs. Distributing an additional 199 rules saves 20 µs, and distributing the least firewall rule and removing the firewall altogether saves an additional 70 µs. Reducing latency motivates the distribution of firewall rules to ACLs, making the control applications in local datacenters possible.

III. Algorithm Design

The presented algorithm is generic as it distributes rules from devices with filter capabilities to others, not just from a firewall to switches. First, we present the challenges during this distribution and formulate the requirements for a correct distribution. Second, we detail the algorithm itself. Third, we discuss two additions for broader applicability.

A. Distribution Complexity and Requirements

Many switches support ACL-based filtering, but filtering on multiple switches in combination requires the configuration to be aligned across all switches. If the configuration on one switch is not consistent, packets might be dropped or forwarded without intention. Tracing the effect of unintended missing or forwarded packets to the root cause is very complex, limiting the use of ACLs in current deployments. The automated distribution of rules from a firewall to the ACLs must maintain the consistency of the configuration across all switches. This section summarizes these challenges and states six requirements for the correct distribution of firewall rules.

TABLE I: Example ruleset used throughout the paper for distribution discussions.

ID	Source	Destination	Action
A01	192.168.0.16/31	192.168.100.40/32	accept
A02	192.168.0.18/32	192.168.100.40/32	accept
A02	192.168.0.19/32	192.168.100.40/32	accept
A04	192.168.0.20/32	*	accept
A05	*	*	drop

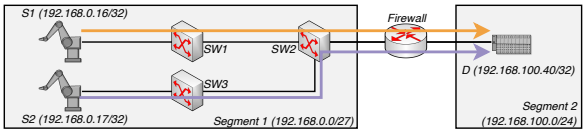
The first challenge we address is the limited number of supported ACL rules in hardware. Not all industrial switches feature ACLs, and those that do, support varying rule counts based on their hardware. Specifically, the resource constraints on switches with ACLs lead to challenges in distributing multiple rules. Therefore, the distribution algorithm must distribute rules based on the available resources.

Table I presents an example firewall ruleset with four accept rules and a default drop rule. We only present rules matching the source and destination IP addresses for simplicity and omit other parameters a firewall rule normally specifies, i.e., IP protocol, TCP/IP source port, and TCP/IP destination port. In this example, all traffic is dropped unless it matches one of the four accept rules. In Figure 3, we visualize five challenges for distributing the example rules. Each example includes a topology with two network segments, i.e., *192.168.0.0/27* and *192.168.100.0/24*, separated by a firewall, where we discuss distributing a single rule from the firewall to one or more ACLs. In each figure, we highlight sources with the letter *S*, destinations with the letter *D*, and switches with *SW*.

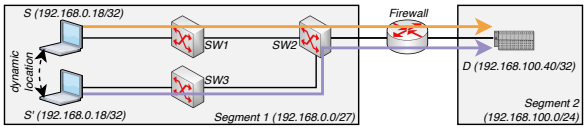
The distribution algorithm must ensure that the distribution results in filtering for every end device specified by the rule. Figure 3a shows two sources *S1* and *S2* defined by the rule *A01* with the IP range *192.168.0.16/31*. Both devices are connected at different switches in the network segment, resulting in the orange and purple paths. Initially, the rule *A01* is located on the firewall, filtering all traffic. Therefore, the algorithm must ensure the distributed rule filters the orange and purple traffic. In the given example, the algorithm can move the rule *A01* to *SW2*, which is on the orange and purple traffic path. Alternatively, the algorithm can duplicate the rule to the switches *SW1* and *SW3* to cover both paths.

Not all end devices in an industrial network have static locations. Devices such as service laptops or mobile robots can access the network at different locations, requiring the distribution algorithm to cover all possible locations of a device. Rule *A02* defines the communication between two individual hosts *S* (*192.168.0.18/32*) and *D* (*192.168.100.40/32*). Figure 3b shows the end device *S* being able to connect at a second location in the network labeled with *S'*. For example, two patch ports within the machine allow a service engineer to connect a laptop. Again, the distribution algorithm can either move the rule to the switch *SW2* or duplicate the rule to the switches *SW1* and *SW3* to cover all possible paths in the network.

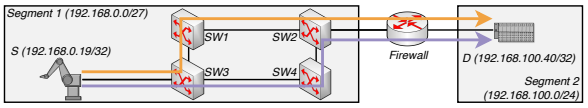
As a fourth challenge, industrial networks use network redundancy to enhance the resilience to device and link failures. The purpose of the deployed protocols for network redundancy is a fast reaction time in case of a failure. Industrial



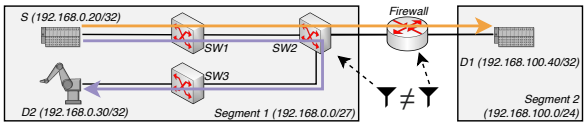
(a) Distributing rule A01, a rule matching multiple end-devices.



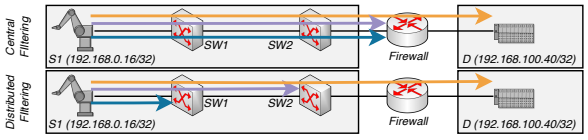
(b) Distributing rule A02, a rule matching dynamic end-devices.



(c) Distributing rule A03 with network redundancy.



(d) Distributing rule A04, a rule with implicit context.



(e) Distributing the filter logic for rule A04.

Fig. 3: Challenges while distributing the ruleset in Table I.

redundancy protocols continuously check for a failure and know the specific reconfiguration in case of a failure upfront. Hence, calculating a new rule distribution on fault detection is always slower than applying the known backup plan. Figure 3c shows a small example with a ring topology. If any of the four links in the ring, or switches *SW1* and *SW4*, have a failure, the connection between the source *S* and destination *D* of rule *A03* is still active. To cover all potential paths in the network, the distribution algorithm can either move the rule *A03* to the switches *SW2* or *SW3* without duplicating the rule or duplicate the rule to the switches *SW1* and *SW4*. The calculation of the potential path depends on the redundancy protocol used.

As a fifth challenge, rules have a specific meaning, depending on their location, not just based on their parameters. For example, Figure 3d shows two network segments and the rule *A04* on the firewall in between. The rule *A04* filters traffic for a specific source *S* (*192.168.0.20/32*) in the source network segment. The rule does not directly filter the destination, as it matches all destinations indicated with the wildcard *. However, based on the topology, only traffic to the destination *D1* in the network segment *192.168.100.0/24* can match the rule. If the algorithm moves this rule to a switch in *Segment 1*, the rule meaning changes. For example, if the rule is moved to switch *SW2*, the rule filters all traffic to *D1* (*192.168.100.40/32*),

TABLE II: Updated rule A04 to contain its explicit meaning.

ID	Source	Destination	Action
A04	192.168.0.20/32	192.168.100.0/24	accept

as before, but also to the destination *D2* (*192.168.0.30/32*). This destination was not included in the meaning of the rule before but is included in the meaning after the distribution. Therefore, the algorithm must analyze the meaning of a single rule before and after the distribution. If this meaning is altered during the distribution, the rule must be adjusted to its original meaning. In the example, the algorithm can adjust the rule *A04* by explicitly specifying the destination network segment *192.168.100.0/24*, as presented by Table II.

ACLs in switches are normally written as default accept rulesets whereas firewall rulesets are written as default drop rulesets. This mismatch of default rules defines the sixth challenge for distributing firewall rules to ACLs. In default accept rulesets, most rules are drop rules, i.e., defining which packets to drop, and the ruleset has a default rule to forward all traffic not matching a rule. In default drop rulesets, most rules are accept rules, i.e., defining which packets to forward, and the ruleset has a default rule to drop all traffic not matching a rule. Hence, the algorithm normally would need to distribute accept rules from a firewall into default accept rulesets on a switch. This kind of distribution removes the effect of the default drop rule on the firewall. Figure 3e shows the central filtering on a firewall on the top, i.e., the firewall drops all undesired traffic (purple, red, and blue). On the bottom, Figure 3e shows an example output where undesired traffic is dropped earlier, i.e., the switches *SW1* and *SW2* filter the undesired traffic. For the distribution algorithm, this challenge defines the requirement that the filter logic must be preserved, even if the default rules change.

B. Algorithm Operation

The presented algorithm distributes each rule independently and verifies the correctness of the individual rule distribution. Therefore, the algorithm operates in two nested for-loops, as presented by Figure 4. 1) The outer for-loop iterates over all firewalls in the network. 2) The inner for-loop iterates over all rules in the ruleset of a specific firewall. 3) For each rule, the algorithm calculates the paths the distribution must cover, selects a set of switches the rule is distributed to, and adjusts the rules after the distribution. We detail these three steps in the following sections.

1) Calculation of Paths: Section III-A details the challenges in distributing a rule from a central firewall between network segments to the ACLs of switches within the network segments. The algorithm analyzes the network topology and rule definition for each rule to ensure the correct filter coverage after the distribution. This section highlights the necessary calculations for this step.

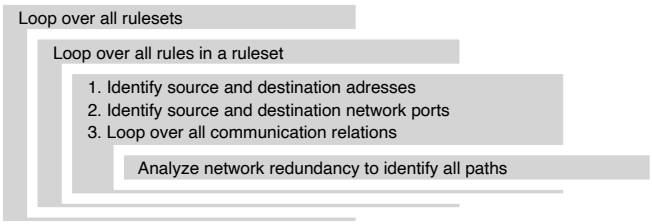


Fig. 4: Algorithm operation.

Each rule on a firewall specifies a set of source nodes talking to a set of destination nodes defined with the addresses. With this information, the algorithm evaluates the network configuration to derive the possible locations for each source and destination. The result is a set of source network ports and destination network ports. The rule filters the communication between a combination of any of the source network ports to any of the destination network ports, i.e., the cartesian product of the list of source and destination network ports. This cartesian product covers the multiple hosts and dynamic end devices per rule requirement.

As the next step, the algorithm analyzes the network topology for each communication relationship, specifically the configured redundancy mechanisms. The algorithm will find two valid paths in the network for ring protocols like *MRP* or *DLR*. For protocols to support meshed networks, such as *RSTP*, the number of paths depends on the actual topology. Therefore, we calculate all *simple paths* for a communication relation, i.e., all paths without loops.

2) Rule Distribution: The distribution of the rules depends on calculating all paths, as described in Section III-B1. When distributing the rules, the algorithm must cover each path and potentially duplicate rules to achieve that goal. Many paths in the calculated set have the same nodes, i.e., overlapping sub-paths. Hence, distributing the rules to these overlapping sub-paths saves resources in the switches, as the algorithm does not need to duplicate rules. The sub-goal of the distribution is to move a rule towards the sources of the rule. For the security of the network, this sub-goal does not make a difference, as all rules apply to the required traffic in both cases. However, filtering traffic earlier results in less undesired traffic in the network and enhances its robustness.

As explained in Section III-A, rulesets can contain accept or drop rules. In the following, we differentiate between the distribution of accept rules and drop rules.

a) Accept Rules: For the distribution of accept rules, it is important to ensure that every node on the path accepts and forwards the packet. Additionally, the drop semantics of the original ruleset must be implemented in at least one node on the path. Therefore, the algorithm must evaluate the filter on every node on the path of the communication relation. Specifically, there are two scenarios the algorithm must consider per node. First, a ruleset of the node is a default drop ruleset. In this scenario, the accept rule to distribute must always be added to the ruleset. Second, if a ruleset has a default accept rule, the traffic is forwarded without adding the rule explicitly. However, at least one node on the path must be a default drop ruleset to implement the original drop semantics. Therefore, the algorithm converts every default accept ruleset to a default drop ruleset. In this step, the algorithm adds

TABLE III: Resulting default drop ruleset on a switch in 192.168.0.0/27.

ID	Source	Destination	Action
1	192.168.0.0/27	192.168.0.0/27	accept
2	*	*	drop

an accept rule covering the network segment addresses to avoid disturbing the traffic within the initial network segment. Table III presents the result for the inversion of an empty ruleset on a switch in the network segment *192.168.0.0/27*.

b) Drop Rules: In general, only one node on the path must drop the communication, as all following nodes no longer receive the traffic. Additionally, a drop rule keeps its filter semantics independent of the default rule. Therefore, the algorithm can distribute a drop rule to any filter on the path of the defined communication relation. Specifically, the presented algorithm distributes the drop rules towards the sender to avoid unwanted traffic in the network.

The complete implementation is available for reference on GitHub[1]. This implementation is a greedy algorithm that does not produce the best solution, but it is correct and easy to follow. The iteration over all rules in the network and the independent distribution of a single rule can result in filters running out of capacity before all rules are distributed. Future implementations can leverage techniques like Integer Linear Programming for perfect optimization.

3) Rule Adjustment: Firewall rules often use wildcards, assuming implicit traffic constraints based on location. When distributed, these constraints may no longer apply and must be adjusted. To mitigate this violation, the algorithm compares the rule definition with the original behavior of the switch. For this comparison, the algorithm generates a virtual rule matching all source and destination addresses of the network segment. Next, this virtual rule is subtracted from the distributed rule. Using the result of this comparison, the algorithm adjusts the rule to avoid overlapping the network segment internal traffic.

C. Applicability

The applicability of the presented distribution algorithm has two limitations. First, the number of rules available in the ACLs on the switches is limited compared to the firewall rulesets. Second, the ACLs on the switches cannot filter all the types of rules a firewall can. For example, ACLs are stateless filters that cannot track the connection state for stateful rules.

1) Prepare Rulesets: Distributing accept rules requires the algorithm to place the rule in the ruleset of every switch on the path. This approach requires every rule to be placed in multiple switches, thus consuming more network resources. Therefore, we propose to convert all rules to *drop rules* and distribute only such to reduce the number of distributed rules. Therefore, the algorithm must invert the default drop ruleset of a firewall to a default accept ruleset, as discussed previously.

Figure 5 shows three rectangles symbolizing the packet space, similar to a Venn diagram. As in the previous ruleset

[1]Our Code: https://github.com/hs-esslingen-it-security/hses-FReD

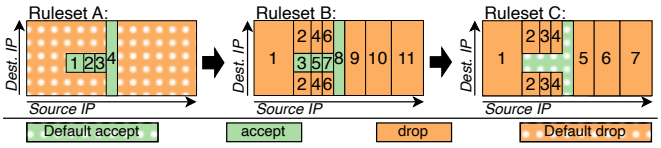


Fig. 5: Inversion of rulesets with sub-step of decorrelation (cf. Tables I, IV, and V).

example, the figure only shows the source IP and destination IP. A complete representation of the rules requires as many axes as the rules have parameters. Typically, a firewall rule consists of at least five parameters to cover the TCP/IP header, also known as the five-tuple, which consists of source and destination IP addresses, IP protocol, as well as source and destination IP ports. Additional parameters could, for example, be the network interfaces on the firewall. The ruleset *A* in Figure 5 represents the four accept rules from Table I as green squares. The dotted background represents the default drop rule, matching all packets not matching an accept rule.

To invert a firewall ruleset, our algorithm subtracts all rules from the default drop rule, resulting in many drop rules not overlapping with the accept rules. This subtraction mechanism is called decorrelation and was initially presented by Patz et al. [7]. Summarizing the decorrelation, the algorithm calculates the rules without any overlap. The image at the center of Figure 5, as well as Table IV, shows the result of the decorrelation. 11 explicit rules cover the complete space of all possible packets. Hence, no default rule exists in this configuration. The number of rules required to fill the remaining space depends on the IP ranges to be defined. Some hardware filters allow custom ranges between IP addresses, while others require following the CIDR subnet notion, as presented in Table IV. In the last step of the ruleset inversion, the algorithm removes all accept rules and introduces a default accept rule. Ruleset *C* in Figure 5 and Table V show the resulting seven drop rules.

Executing our implementation, the user can select whether accept or drop rules shall be distributed. The accept rule distribution inverts all ACLs to default drop rulesets. The drop rule distribution inverts the firewall set before the distribution.

2) Stateful Rule Distribution: Stateful filtering, offered by most software firewalls, combines the filtering based on the values in the packets with the tracking of the connection state. The connection state is often stored in a hash table

TABLE IV: Decorrelated example ruleset for distribution (cf. Figure 5).

ID	Source	Destination	Action
B01	192.168.0.0/28	192.168.100.0/24	drop
B02	192.168.0.16/31	! 192.168.100.40/32	drop
B03	192.168.0.16/31	192.168.100.40/32	accept
B04	192.168.0.18/32	! 192.168.100.40/32	drop
B05	192.168.0.18/32	192.168.100.40/32	accept
B06	192.168.0.19/32	! 192.168.100.40/32	drop
B07	192.168.0.19/32	192.168.100.40/32	accept
B08	192.168.0.20/32	*	accept
B09	192.168.0.21/32	192.168.100.0/24	drop
B10	192.168.0.22/31	192.168.100.0/24	drop
B11	192.168.0.24/29	192.168.100.0/24	drop

TABLE V: Inverted example ruleset for distribution (cf. Figure 5).

ID	Source	Destination	Action
C01	192.168.0.0/28	192.168.100.0/24	drop
C02	192.168.0.16/31	! 192.168.100.40/32	drop
C03	192.168.0.18/32	! 192.168.100.40/32	drop
C04	192.168.0.19/32	! 192.168.100.40/32	drop
C05	192.168.0.21/32	192.168.100.0/24	drop
C06	192.168.0.22/31	192.168.100.0/24	drop
C07	192.168.0.24/29	192.168.100.0/24	drop
C08	*	*	accept

and represents the direction in which a connection is opened. Stateless filters cannot track the state of the connection as they filter based on the values in the packets. As long as the hardware does not support the connection tracking, there is no equivalent distribution possible. Still, a stateless filter can limit the initial packets in the wrong direction for TCP traffic by filtering the TCP flags. For UDP traffic, the IP ports have specific ranges that can be filtered, i.e., the source port is typically in the ephemeral port range.

Stateful filtering is very beneficial in environments with dynamic and short-living connections. Industrial networks are not very dynamic, and the connections exist for long periods of time. Therefore, we propose converting stateful rules into stateless rules before distribution by filtering for the direction of *SYN* packets and port ranges. With this conversion, the algorithm can distribute the stateless rules, whereas stateful rules cannot be distributed.

IV. Evaluation

In this section, we investigate the behavior and performance of the presented algorithm. First, we describe our evaluation data, its origins, and the covered scenarios. Second, we analyze the distribution behavior and evaluate the latency optimizations. Lastly, we measure the runtime of the distribution algorithm and discuss the observed behavior.

A. Topology Generation and Latency Model

The evaluation in this paper is based on 420 different topologies and network configurations. We generate these topologies and their configurations with GeNESIS [8], a deterministic topology generator we presented in previous work. GeNESIS is available online [9] and generates realistic industrial network topologies with firewall configurations. The topologies used in this evaluation are reproducible with the GeNESIS-TAG[2] as input for GeNESIS.

[2]genesis:v1.1#496·276·718::1·2·100::1·0·0::10::1·2·2·1:S2·I·C2:L::2.30.2·2.16.2·2.32.2·1.4.1:I·O·C2:L#

TABLE VI: Example switching chips used for reference ACL capacities.

Company	Switching chip	ACL rules	Ports
Broadcom Inc.	BCM56072 [2]	256	48
Broadcom Inc.	BCM53112 [3]	128	15
MARVELL	88E6390X [4]	$\leq$ 256	11
Microchip	KSZ9896C [5]	96	6

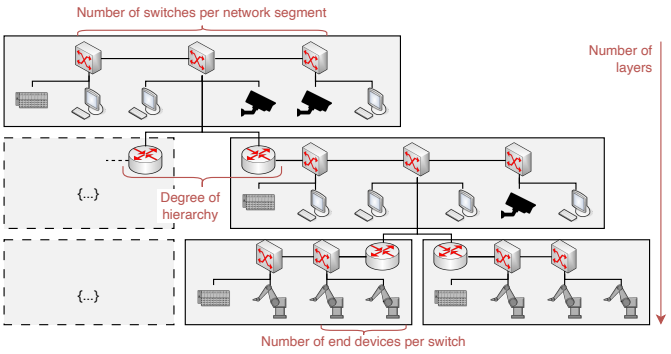


Fig. 6: Varying factors of topologies in evaluation scenarios generated with GeNESIS.

To evaluate the different effects of the distribution algorithm, we instruct GeNESIS to generate a base scenario and then create related scenarios by scaling different parameters individually. Figure 6 presents these scaled parameters: a) *number of switches per network segment*, b) *number of end devices per switch*, c) *degree of hierarchy*, and d) *number of layers* in the network. The generated topologies range from three network segments with only 12 end devices up to 31 network segments, 92 switches, and 184 end devices.

With the presented algorithm, we aim to reduce the forwarding latency by utilizing the ACLs in switches to reduce the number of firewall rules. In previous work, we introduced a model for end-to-end latency calculations in time-critical networks with routers and switches [10]. The latency for transmitting a packet depends on the link speed and packet size for the link, as well as the processing delay of the switching hardware. For simplicity and comparability between configurations, we assume 1 Gbit/s links and a frame size of 500 B, resulting in a latency of 3.8 µs for every link. Vendors of switching chips typically do not disclose measurements for the processing delay of a switch. Therefore, we use 1.5 µs as processing delay, based on measurements conducted in previous work [10]. Literature does not document the latency of software firewalls well, e.g., the load or specific hardware information is missing. For this evaluation, we use measurements of previous work [1] and assume 72 µs for the delay of a software firewall without rules and 24 ns delay for each configured rule at a load of 2 Mbit/s. With higher loads, the latency increases, making firewalls less suitable and distribution to ACLs even more necessary. The number of available ACLs per switch depends on the fabric and additional settings, as the chip shares hardware resources between ACLs and, e.g., Quality of Service policers. Table VI summarizes four switching chips documenting their ACL capabilities, leading to the selection of 128 rules per switch in this evaluation.

B. Distribution Success Rate

The presented algorithm distributes all rules sequentially until no ACL on a switch has any remaining filter capacities. Previously, in Section III-C, we presented the drawbacks of distributing accept rules and proposed the distribution of drop rules instead. For all generated topologies, we execute both distribution types, i.e., 1) the distribution of accept rules and

2) the distribution of drop rules. For each scenario, we analyze the filter utilization of all the switches in the network. Additionally, we calculate the theoretical requirements necessary to distribute all firewall rules for the scenarios with the presented algorithm. Figure 7a shows these utilization results per switch in all 420 topologies in the CDF diagram and highlights the exceeded limits with a red line at 100% utilization. The algorithm will leave the rules exceeding the limits in the original firewall. During the distribution of accept rules, the algorithm exceeds the filter resources in 38% of the executions, presented on the y-axis. During drop rule distribution, the algorithm adheres to resources in 98.8% of all scenarios, with violations occurring only in scenarios with many routers and, thus, firewalls to distribute. Only in examples with many switches and a few firewalls does the distribution of accept rules use fewer rules than the distribution of drop rules. Consequently, the applicability of the distribution of drop rules outperforms the distribution of accept rules. Therefore, we only analyze the distribution of drop rules in the following.

C. Latency Optimizations

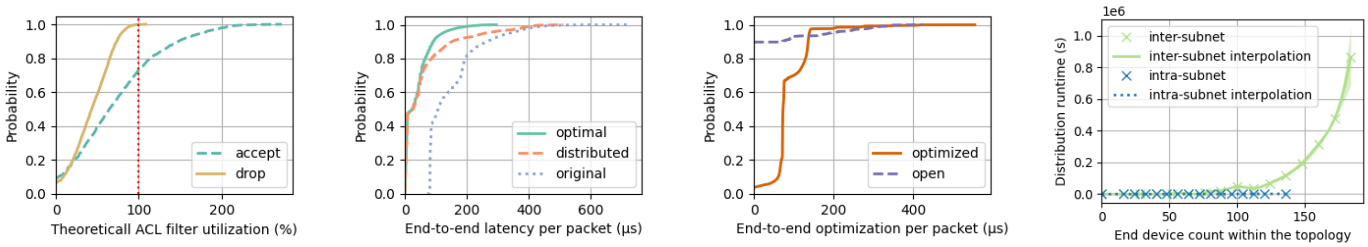
Combined with the topology and configuration for the firewalls, GeNESIS outputs a set of allowed packets. We calculate the latency introduced along the complete path for each allowed packet, i.e., cables, switches, and filtering in ACLs or firewalls, before and after the distribution. Also, we calculate the optimal latency for each packet without any filter delay on the path, just the regular end-to-end delay. Figure 7b combines the calculated latencies for all packets in all 420 topologies in a CDF per packet, whereas Figure 7a presents the ACL utilization per switch.

Every packet observes a minimum latency of 80 µs, mostly from one firewall with 100 rules in the rulesets. With an increasing number of switches, firewalls, and rules, the end-to-end latency increases to 730 µs. Figure 7c presents the benefit of the presented algorithm with the distribution for the optimized latency, i.e., the per-packet difference between the original configuration and the distributed ruleset. The algorithm improves the latency for 96% of all packets in the 420 topologies and never degrades the latency. The network resources do not allow a complete distribution of all rules, resulting in the *open* optimization potential, i.e., the difference between distributed and optimal. 89% of the packets no longer cross a firewall with rules, as the distribution algorithm distributes all rules from the firewall to ACLs, resulting in very low end-to-end latencies, equal to the optimal latency. For the remaining 11%, the algorithm cannot distribute all rules from the firewalls to the ACLs, leaving further potential for optimizations (cf. *open*). The maximum *optimal* latency equals a latency of 311 µs, caused by a line of 58 switches and 59 links between the end devices.

All together, the latency improvement between the original and distributed rulesets is significant and underlines the need for further research in this area. For example, a non-greedy algorithm could utilize the ACL resources more efficiently and distribute more rules from firewalls to ACLs.

D. Runtime Evaluation

The runtime of an optimization algorithm is crucial, as the duration of the distribution delays the configuration deployment. We grouped the generated topologies in two different scenario groups for the evaluation of runtime impacts: 1) *inter-subnet* scaling, i.e., scenarios with an increased number of network segments, and 2) *intra-subnet* scaling, i.e., scenarios with an increased number of devices per network segment. In all scenarios, the number of firewall rules per firewall stayed the same. Figure 7d shows the runtimes of the presented distribution algorithm for the differently scaled network topologies, measured on a desktop PC with an Intel i7-7700K @ 4.2 GHz and 16 GB RAM. The x-axis the network size with the number of all end devices within a single topology. We executed at least ten measurements per mark in the diagram and visualize the variance with the width of the interpolation line. Important to recognize is the linear increase of the runtime for intra-subnet scaled topologies, requiring 2 s for small topologies and 45 s for large topologies. Hence, the size of the subnets does not have a major impact on the runtime. However, scaling the number of network segments causes exponential runtime growth as the number of firewalls increases. This drastic increase from 2 s to 2.3 days is caused by two calculations in the algorithm: 1) The inversion of rulesets requires time per ruleset, i.e., per firewall, and 2) the distribution depends on the number of total rules in the network, which increases with the number of firewalls. Specifically, the path calculations as part of calculation 2) have the biggest impact on the runtime. Future work must optimize the algorithm to calculate the required paths for each rule more efficiently, e.g., by using a divide-and-conquer strategy.



(a) Theoretically required ACL filter utilization for all switches comparing the accept and drop rules distribution.

(b) Forwarding latency distribution for all packets comparing original rulesets and distributed rulesets.

(c) Per-packet difference between original and distributed (optimized), as well as distr. and optimal (open).

(d) Runtime analysis with increased number of devices per segment (intra) or number of segments (inter).

Fig. 7: Distribution, latency, optimization, and runtime evaluation of the presented algorithm in 420 topologies.

V. Related Work

We group related work into the categories: A) optimizing a firewall ruleset and B) distributing rules within the network.

A. Firewall Ruleset Optimization

Firewall rule optimization improves filtering efficiency by reducing rule complexity and processing overhead. Casado et al. [11], Porras et al. [12], and Al-Shaer and Hamed [13] propose aggregation and compression of rules to reduce lookup times and optimize storage. Xu et al. [14] demonstrate the use of machine learning for dynamic rule refinement. Multiple researchers [15], [16] explore rule reordering to ensure frequently matched rules are evaluated first. While these studies primarily optimize software-based firewall processing, our approach aims to replace software firewalls with ACLs.

B. Distributed Rule Enforcement

The distribution of firewall rules across network devices enhances scalability and reduces processing delays. Al-Shaer et al. [17] and Yuan et al. [18] discuss the challenges of distributed filters and provide algorithms for their analysis. Their approaches only encounter firewalls on router and do not include the distribution to ACLs in switches. He et al. [19] propose ACL-based rule enforcement in switches to offload filtering from centralized firewalls. However, their methods do not account for network redundancy, device mobility, and rule-sets with a mix of accept and drop rules. Comaneci et al. [20] and Fayaz et al. [21] introduce SDN-based approaches for dynamic rule management. However, they rely on centralized controllers and specific hardware, making them less suitable for industrial networks. Specifically, SDN approaches require rules in every ACL on the path, thus requiring similar amounts of resources as the distribution of accept rules. Unlike these studies, our approach operates on existing and commodity hardware, leveraging existing industrial infrastructure more efficiently to maintain real-time security enforcement.

VI. Conclusion

This paper presents a novel approach to firewall rule distribution in industrial networks, transforming network devices into a unified, distributed filtering system. By leveraging Access Control Lists (ACLs) on industrial switches, our method minimizes filtering delays while maintaining security, consistency, and adaptability to dynamic network conditions. The proposed algorithm ensures correct rule placement by analyzing communication paths, accounting for redundancy mechanisms, and modifying rules to preserve their semantics when moved from a centralized firewall to distributed enforcement points. Our evaluation demonstrates that distributing firewall rules significantly reduces packet forwarding latency, making time-sensitive industrial applications more feasible while maintaining filtering accuracy. The approach successfully handles resource limitations in ACLs by optimizing rule placement and reducing redundant filtering operations. Future work could explore advanced optimization techniques to improve rule distribution efficiency and runtime in large-scale, high-complexity networks.

References

[1] L. Wüsteney, M. Menth, R. Hummen, and T. Heer, "Impact of Packet Filtering on Time-Sensitive Networking Traffic," in *IEEE International Conference on Factory Communication Systems (WFCS)*, Linz, Austria, Jun. 2021.

[2] Broadcom Inc. (2025) BCM56072/BCM56071N Low-Power 440G Switch. Accessed: Feb. 15, 2025. [Online]. Available: https://docs.broadcom.com/doc/56072-56071N-DS1-PUB

[3] ——. (2025) BCM53112. Accessed: Feb. 15, 2025. [Online]. Available: https://docs.broadcom.com/doc/53112-PB101

[4] Marvell Technology, Inc. (2025) Marvell® Link Street® 88E6390X. Accessed: Feb. 15, 2025. [Online]. Available: https://www.marvell.com/content/dam/marvell/en/public-collateral/switching/marvell-switching-link-street-88e6390x-product-brief-2015-05.pdf

[5] Microchip Technology Inc. (2025) KSZ9896C. Accessed: Feb. 15, 2025. [Online]. Available: https://ww1.microchip.com/downloads/aemDocuments/documents/OTH/ProductDocuments/DataSheets/KSZ9896C-Data-Sheet-DS00002390C.pdf

[6] International Electrotechnical Commission (IEC), "IEC 62443-3-2," *Security for Industrial Automation and Control Systems – Part 3-2: Security Risk Assessment for System Design*, 2020.

[7] G. Patz, M. Condell, R. Krishnan, and L. Sanchez, "Multidimensional security policy management for dynamic coalitions," in *Proceedings DARPA Information Survivability Conference and Exposition II. (DISCEX)*. Anaheim, CA, USA: IEEE, Jun. 2021.

[8] L. Bechtel, S. Müller, M. Menth, and T. Heer, "GeNESIS: Generator for Network Evaluation Scenarios of Industrial Systems," in *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Padova, Italy, Sep. 2024.

[9] HS Esslingen. (2025) GeNESIS. Accessed: Feb. 15, 2025. [Online]. Available: https://github.com/hs-esslingen-it-security/hses-GeNESIS

[10] L. Wüsteney, D. Hellmanns, M. Schramm, L. Osswald, R. Hummen, M. Menth, and T. Heer, "Analyzing and Modeling the Latency and Jitter Behavior of Mixed Industrial TSN and DetNet Networks," in *Proceedings of the International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, Roma, Italy, Dec. 2022.

[11] M. Casado, T. Garfinkel, A. Akella, and J. Stribling, "Sane: A protection architecture for enterprise networks," in *USENIX Security Symposium*, Vancouver, B.C. Canada, Jul. 2006.

[12] P. A. Porras, S. Cheung, M. W. Fong, K. Skinner, and V. Yegneswaran, "Securing the software defined network control layer," in *Network and Distributed System Security (NDSS) Symposium*, Feb. 2015.

[13] E. S. Al-Shaer and H. H. Hamed, "Modeling and management of firewall policies," *IEEE Transactions on Network and Service Management*, vol. 1, no. 1, pp. 2–10, 2004.

[14] Z. Xu, J. Lu, X. Wang, J. Zhang, M. Alazab, and V. García-Díaz, "AI and Machine Learning for the Analysis of Data Flow Characteristics in Industrial Network Communication Security," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 37, no. 3, pp. 125–136, 2021.

[15] T. Y. Woo, "A modular approach to packet classification: Algorithms and results," in *IEEE International Conference on Computer Communications (INFOCOM)*. Tel Aviv, Israel: IEEE, Mar. 2000.

[16] P. Gupta and N. McKeown, "Packet classification on multiple fields," in *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. ACM, 1999.

[17] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan, "Conflict Classification and Analysis of Distributed Firewall Policies," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 10, 2005.

[18] L. Yuan, H. Chen, J. Mai, C.-N. Chuah, Z. Su, and P. Mohapatra, "FIREMAN: A Toolkit for FIREwall Modeling and ANalysis," in *IEEE Symposium on Security and Privacy (S&P)*. Oakland, CA, USA: IEEE, May 2006, pp. 15–pp.

[19] J. He, K. Ota, M. Dong, L. T. Yang, M. Fan, G. Wang, and S. S. Yau, "Customized network security for cloud service," *IEEE Transactions on Services Computing*, vol. 13, no. 5, pp. 801–814, 2017.

[20] D. Comaneci and C. Dobre, "Securing networks using sdn and machine learning," in *IEEE International Conference on Computational Science and Engineering (CSE)*. Bucharest, Romania: IEEE, Oct. 2018.

[21] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey, "Bohatei: Flexible and Elastic DDoS Defense," in *USENIX Security Symposium*, Stanford, CA, USA, Aug. 2002, pp. 817–832.