Extending SDN-ACL Automation with User Groups, Authentication Events, and Intrusion Detection System Integration

Florian Grießer*, Hirokazu Hasegawa‡, Hajime Shimada§
*School of Computation, Information and Technology, Technical University Munich, Chair of Security in Information Technology, Germany
‡Center for Strategic Cyber Resilience R&D, National Institute of Informatics, Japan
§Information Technology Center, Nagoya University, Japan
florian.griesser@tum.de, hasegawa@nii.ac.jp, shimada@itc.nagoya-u.ac.jp

***Abstract*—As cyberattacks become more common and advanced, traditional networks fall behind because they depend on static settings and manual adjustments. Software-Defined Networking (SDN) provides more flexibility and can be used to solve these problems. In this work, we present a system that automatically creates Access Control Lists (ACLs) in SDN environments. The system links access control to the User Database and generates rules automatically, which reduces the effort for administrators. With Port Access Control, only authenticated devices are allowed to use network resources. In addition, the system integrates an Intrusion Detection System (IDS): suspicious clients are first monitored through mirrored traffic, and if violations go beyond a threshold, their traffic is redirected for inline inspection. We tested the system in three use cases: connecting new clients, adapting dynamically to authentication events, and redirecting malicious hosts through the IDS. The results show that our approach not only reduces manual work but also enforces role-based security and prevents IDS overload by escalating only persistent or severe attacks.**

Keywords*-*Software-Defined Networking; Authentication; Access Control Lists; Intrusion Detection Systems

I. Introduction

This article is an extended version of our previous work presented at the International Conference on Networks (ICN) [1], where we introduced a system for automatically generating Access Control Lists (ACLs) within Software-Defined Networking (SDN) environments. While the conference paper focused on the system architecture and initial evaluation, this extended version expands the analysis in several directions. In particular, we introduce a more comprehensive *Formal Security Model and Threat Analysis* (Section V) and extend the system with an *Intrusion Detection System (IDS)* for dynamic policy adaptation. We further examine how identity and authentication events can drive the automated creation of fine-grained access control rules in SDN environments and how such mechanisms adapt to user behavior while maintaining scalability, security, and low administrative overhead. The core challenge we address is how to integrate authentication-driven, identity-aware access control into SDN in a way that adapts dynamically to user behavior while preserving scalability and security. The primary aim of this work is to demonstrate the feasibility and practical applicability of the proposed system in realistic scenarios, with a focus on validating the architectural concept and its security properties rather than conducting an exhaustive performance study.

Digital transformation has exponentially increased the complexity of network architectures, presenting significant challenges in maintaining robust security frameworks [2]. In this ever-evolving digital landscape, cybersecurity threats have become more sophisticated, leveraging the linkage of modern infrastructures to exploit vulnerabilities at an alarming rate. Traditional network security mechanisms, which mainly rely on static configurations and manual oversight, are increasingly proving inadequate against this backdrop of dynamic and evolving threats [2]. The inherent limitations of these conventional approaches, characterized by their inflexibility and slow response times, underline the urgent need for more adaptable, responsive security measures.

Software-Defined Networking (SDN) is a paradigm that promises to redefine network management and security [3]. At its core, SDN separates the network's control logic from the underlying hardware, facilitating a centralized and programmable framework that transcends traditional hardware limitations [4]. This separation enhances network flexibility and management and introduces agility and adaptability that were unachievable with conventional network architectures until now. According to a report by Global Market Insights, the SDN market, valued at USD 28.2 billion in 2023, is expected to experience significant growth, with a projected expansion rate exceeding 17% annually from 2024 to 2032 [5]. Through SDN's capabilities, networks gain the flexibility to adapt swiftly to evolving security demands. This flexibility enables the immediate implementation of tailored security measures and configurations to counter new threats effectively, as illustrated in the study by Ali et al. [6].

Furthermore, our contribution is complemented by the work of Yakasai et al. in FlowIdentity, which advances virtualized network access control within SDN through a role-based firewall [7]. We also build on the architectural insights of Casado et al. in Ethane, demonstrating the power of centralized policy enforcement [8], and the approach of Mattos et al. in AuthFlow, focusing on authentication and access control mechanisms in SDN environments [9].

Additionally, this approach was refined by incorporating a structured analysis of authentication logs, drawing upon the work of Xing et al. in SnortFlow, which explores an OpenFlow-based intrusion prevention system in cloud environments [10], and the study by An Le et al. on a flexible network-based intrusion detection and prevention system on Software-Defined Networks [11].

The paper progresses from reviewing related SDN security work in Section II to foundational concepts in Section III. Section IV describes our system for automating ACLs, followed by Section V, which presents the formal security model and analyzes potential threats. Section VI details the implementation, Section VII evaluates the system's performance and demonstrates the benefits of the IDS integration, and Section VIII encapsulates concluding thoughts and future directions.

II. Related Work

Network security and access control advancements are crucial in the evolving landscape of SDN. The following studies demonstrate that emerging technologies and frameworks are pivotal in addressing these challenges.

A. Intrusion Detection with Authentication Events

In the study by Chu et al. [12], "ALERT-ID," an intrusion detection system for large-scale network infrastructures, is presented. The system distinguishes between normal operations and potential security threats through real-time analysis of authentication, authorization, and accounting (AAA) system logs. It employs behavioral models built on historical access patterns and user profiles, efficiently identifying potential intrusions and misuse. Notably, ALERT-ID balances the need for thorough security monitoring with a manageable false alarm rate, demonstrating the importance of dynamic security measures in complex network environments.

Building on this, Janabi et al. provide a comprehensive survey of intrusion detection systems in Software-Defined Networking [13]. Their work categorizes existing approaches into anomaly-based, signature-based, and hybrid detection mechanisms, and highlights key challenges such as scalability, controller bottlenecks, and false alarm reduction. This survey underscores the urgency of developing IDS solutions that are tailored to the dynamic characteristics of SDN environments.

A more specialized perspective is given by Susilo et al., who present an SDN-based intrusion detection system that leverages deep learning techniques [14]. By training models on traffic data, they achieve high detection rates for a variety of attack types, demonstrating the potential of machine learning to significantly enhance IDS performance in SDN settings.

In a related direction, Wang et al. propose an AI-powered network threat detection system [15]. Their approach integrates artificial intelligence methods such as Random Forests and neural networks into SDN Controllers to enable real-time threat detection. The study highlights the effectiveness of AI in reducing false positives and improving scalability, though it also raises concerns regarding explainability and computational overhead.

B. Dynamic Access Control in SDN

Transitioning to dynamic access control, the work by Nayak et al. introduces "Resonance: Dynamic Access Control for Enterprise Networks" [16]. Resonance implements dynamic security policies with a registration phase, complemented by real-time monitoring and inference mechanisms specified by administrator rules.

A more recent study by Shah and Yadav integrates IEEE 802.1X authentication into SDN to control port-level access [17]. Their approach provides admission control based on authentication status, yet it does not support dynamic ACL updates or identity-based rule generation. This gap is central to our contribution, which leverages authentication events and user group information to automatically derive and update fine-grained, identity-aware ACLs.

Further extending the concept of network security, the study by Martins et al. [18] introduces an access control architecture for SDN leveraging the ITU X.812 standard. This framework incorporates Role-Based Access Control (RBAC) with traffic prioritization rules, advancing towards more granular access control based on predefined role mappings. While powerful, this approach depends on extensive manual configuration efforts to establish complete rule sets, limiting its ability to adapt dynamically to user behavior or evolving authentication contexts.

C. Formal Security Models & Threat Modeling in SDN

Beyond practical implementations, several works emphasize the necessity of formal reasoning in SDN security. Sharma and Tyagi present a structured threat model for SDN environments, covering controller protection, inter-controller communication, and malicious switch scenarios [19]. Their taxonomy illustrates how different layers of the SDN stack are exposed to unique attack vectors, motivating the need for systematic threat analysis.

Pradeep et al. propose EnsureS, an SDN security model that validates service paths based on efficient hashing and tag verification mechanisms [20]. Their design provides both efficiency and security guarantees, reducing the risk of path manipulation and enhancing packet integrity.

Finally, Meng et al. introduce a policy model transformation and verification framework [21]. By automatically converting high-level security policies into flow-level configurations and applying formal verification techniques, they ensure consistency between intended policies and deployed rules. This approach demonstrates the potential of combining formal methods with SDN programmability to achieve provable security properties.

D. Policy Conflict Detection and Resolution in SDN

In parallel with dynamic access control, substantial work has addressed the challenge of policy conflict detection and resolution in SDN environments. Systems like FortNOX [22] and FlowGuard [23] provide real-time enforcement mechanisms that prevent new flow rules from violating established security policies by checking for conflicts during rule insertion. Verification tools such as VeriFlow [24] and NetPlumber [25] take a verification-oriented approach, intercepting flow updates to ensure they do not violate global invariants such as isolation or reachability. More recent frameworks, including PGA [26] and Brew [27], focus on composing and reconciling policies from different modules or tenants, producing a consistent, conflict-free rule set. These efforts highlight the importance of handling interactions between dynamic ACLs, legacy firewall configurations, and other network policies, which is particularly relevant in hybrid and large-scale deployments.

These studies illustrate a significant progression in SDN security research, ranging from intrusion detection systems to dynamic access control and formal threat modeling. Building on these developments, our work reduces administrative burden while enabling adaptive, security-driven policy updates.

III. Preliminary Concepts

This section introduces foundational concepts relevant to network management and security, including OpenFlow switches in SDN, 802.1X Port-Based Network Access Control, access management with Active Directory and LDAP-based directory services, and the Extensible Authentication Protocol over LAN (EAPOL).

A. The Role of OpenFlow Switches in SDN

Software-Defined Networking (SDN) represents a paradigm shift in network management by decoupling the control plane from the data plane. OpenFlow, one of the earliest and most widely adopted SDN protocols, provides a standardized interface for communication between the centralized controller and the network devices. Within this architecture, OpenFlow switches serve as the essential data plane components, responsible for forwarding packets based on flow rules received from the controller [28].

These programmable switches enable dynamic network control, granular traffic engineering, and the implementation of advanced security policies. Because the control logic resides in a centralized controller, network behavior can be reconfigured on the fly without the need for manual reprogramming of individual devices. This centralized programmability simplifies policy enforcement and enables rapid adaptation to changing security requirements.

While OpenFlow switches provide strong flexibility, they also introduce security challenges. Because switches rely on the controller for all flow decisions, a compromised controller could install malicious rules or bypass policies. In addition, limited flow-table capacity makes them vulnerable to exhaustion attacks such as flow flooding [29].

The communication channel between switches and the controller, typically protected by TLS, can also be a point of failure if improperly configured [3]. Authentication, certificate management, and key distribution therefore become crucial aspects of securing the SDN infrastructure. Finally, the lack of default policies or fallbacks in many OpenFlow implementations can result in a denial-of-service condition if the controller becomes unreachable [29].

To address these issues, several countermeasures have been proposed. These include distributed controller architectures to eliminate a single point of failure, flow aggregation to reduce table pressure, and anomaly detection systems to identify malicious flow behaviors [3, 4]. Proper integration with access control mechanisms, such as 802.1X and centralized identity management, can further enhance the trustworthiness of OpenFlow-based networks [30, 31, 32].

In summary, OpenFlow switches provide the flexibility and programmability needed for next-generation networks but must be deployed with careful attention to security design, controller hardening, and flow policy management.

B. Port-Based Authentication with IEEE 802.1X

IEEE 802.1X Port-Based Network Access Control significantly strengthens network security by implementing stringent access control at the physical port level. As a foundational component of network admission control, 802.1X ensures that only authenticated devices and users gain access to the network, thereby maintaining integrity and reducing the risk of unauthorized intrusion [30].

The 802.1X framework operates with three core entities: the supplicant (client device), the authenticator (typically a switch or wireless access point), and the authentication server (commonly a Remote Authentication Dial-In User Service (RADIUS) server). When a device connects to a network port, it is initially placed into an unauthorized state. The authenticator acts as an intermediary, forwarding authentication messages between the supplicant and the server using the Extensible Authentication Protocol over LAN (EAPOL) [33].

The authentication sequence involves an initial access request, followed by a secure exchange of identity and credentials, and concludes with the authentication server evaluating the credentials and instructing the authenticator whether to grant or deny access. This structured process prevents unauthorized devices from entering the network and defends against threats such as MAC spoofing, rogue clients, and replay attacks. Mutual authentication using Extensible Authentication Protocol – Transport Layer Security (EAP-TLS) further enhances security by preventing credential interception and significantly reducing the risk of man-in-the-middle attacks.

Advanced deployments of 802.1X often integrate dynamic network configurations, such as VLAN assignment and ACL enforcement, based on the identity of the authenticated entity. This enables granular policy control and flexible segmentation of network resources. For example, guests, employees, and devices can be placed in separate VLANs with tailored permissions, immediately upon authentication.

While 802.1X offers robust security, its implementation can be complex. Challenges include managing certificates for mutual authentication, ensuring client compatibility, and avoiding service disruptions due to misconfiguration [30, 33]. Nonetheless, when properly deployed and integrated with directory services like Active Directory, 802.1X forms a critical layer of defense in modern enterprise networks.

In conclusion, IEEE 802.1X is an essential mechanism for enforcing authenticated, role-based access at the network edge. It combines strong identity verification with flexible policy application, forming a resilient barrier against unauthorized access and internal threats.

C. Centralized Access Management via Active Directory and LDAP Authentication

Access control is a foundational building block of network security, ensuring that only authorized users can access critical systems and resources. Two widely used technologies for implementing centralized access management are Microsoft's Active Directory (AD) [31] and the Lightweight Directory Access Protocol (LDAP) [32]. Both systems facilitate authentication, authorization, and user management across a range of services and devices, albeit with different implementations and scopes.

Active Directory (AD) is a directory service developed by Microsoft for Windows domain networks. It acts as a centralized repository of user credentials, group memberships, and security policies. AD enables administrators to enforce Role-Based Access Control (RBAC) through Group Policy Objects (GPOs), automate login scripts, and manage user rights at scale. It also supports Kerberos-based authentication, which enhances security through ticket-based access mechanisms.

LDAP, on the other hand, is an open and vendor-neutral protocol used to access and manage distributed directory information services. While AD itself supports LDAP as one of its interfaces, LDAP can also be implemented in a platform-agnostic manner using solutions such as OpenLDAP or Apache Directory. LDAP directories typically organize information in a hierarchical structure and are used for centralizing authentication across services such as email servers, intranet portals, VPNs, and UNIX systems.

In enterprise environments, a centralized identity and access management (IAM) system is often built around AD, which integrates seamlessly with LDAP-aware services. This centralization facilitates consistent enforcement of security policies, simplifies user onboarding and offboarding, and reduces the administrative burden associated with managing multiple local accounts. It also enables Single Sign-On (SSO), allowing users to authenticate once and gain access to a variety of authorized services without repeated logins.

Moreover, integration with federated identity systems and multi-factor authentication (MFA) further strengthens the security posture. AD and LDAP are often combined with other authentication frameworks, such as Security Assertion Markup Language (SAML) and OAuth (Open Authorization), especially in hybrid environments that span both on-premises and cloud infrastructures.

Overall, the use of Active Directory and LDAP for access management supports scalability, interoperability, and a high level of control, making them indispensable components of modern enterprise security architectures.

IV. Proposed System

This section presents a comprehensive overview of our proposed system, designed to significantly enhance network security and efficiency through advanced ACL management and authentication mechanisms.

A. Previous Works: Problem and Approach

Building on our established framework for automating ACL generation through statistical analysis of communication patterns, this work seeks to further leverage and enhance the existing infrastructure. Our initial efforts in [34] laid the foundational groundwork for this approach, which saw significant development and refinement in subsequent studies, such as [35]. A primary challenge identified in our exploration was addressing the need for authentication proof for IP addresses.

We have refined our approach to take advantage of a typical user database, like Active Directory [31], a standard part of a company network. This centralized database offers a significant advantage because user and group management and their corresponding resource access permissions are already handled there. We aim to leverage this existing infrastructure to streamline the process and eliminate redundant tasks for administrators.

B. Architecture of the Proposed System

In the proposed system, we enhance network security through Port Access Control, which limits network port access exclusively to authenticated users. This approach is grounded in a security model where each port is individually secured and requires authentication before granting access. As a result, each user undergoes an authentication process, enhancing the network's overall security posture. The process for user connection is designed with precision to ensure a secure and efficient authentication mechanism and can be found in Figure 1.

Initially, the system is configured to allow only EAPOL messages, which are then directed to an authenticator component. This step ensures that there is no communication before the client authenticates.

The SDN Controller checks its internal state for pre-configured users based on MAC and IP addresses. This information is crucial for comparing against new data received during authentication. Authentication messages for EAPOL are forwarded to a RADIUS server, which validates the credentials against a common user database, typically an Active Directory.

Upon successful authentication, the system assigns an IP address to the specific MAC address by inserting a record in a DHCP server. This procedure ensures that the assigned IP address corresponds to a specific MAC address and is associated with a specific port. The system then generates User-Specific Access Control Lists tied to a particular port by requesting user groups for the specific username from the Active Directory via LDAP. The fundamental idea is that users in the same group as a specific server should also have access to that server. For instance, if the user *Ben* is a member of the *Mail* group, to which our Mailserver also belongs, the system creates ACLs permitting this specific traffic. Consequently, a whitelist is established to allow this connection while blocking all other traffic.

The system can identify users labeled as servers, which differ from standard clients, through a unique identifier group

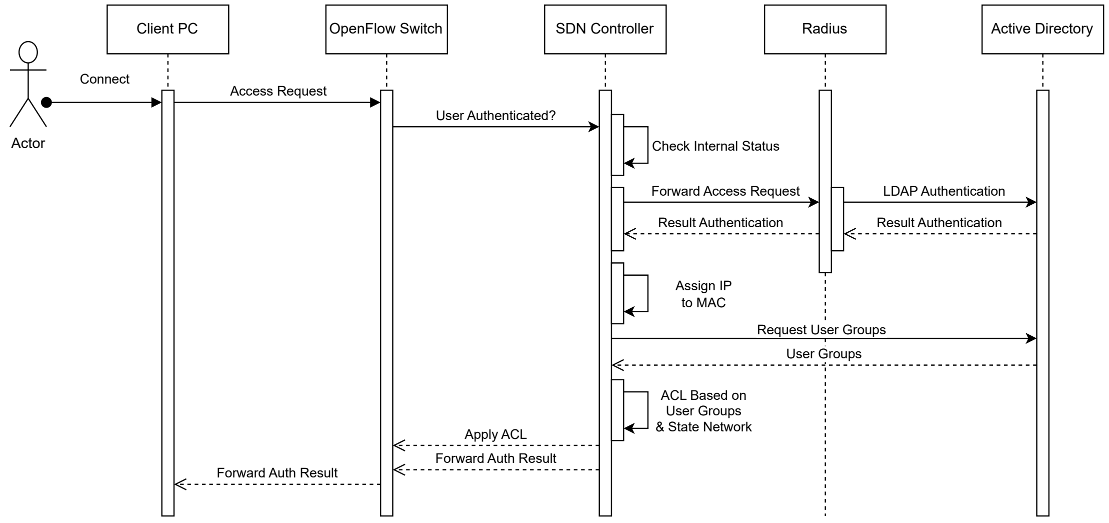

Figure 1. Dynamic ACL Adaptation based on Authentication Events

assigned explicitly to servers. The same concept could also be implemented using specific roles depending on existing usage in a company network, but in our case, we focused on groups. Thus, any user belonging to this shared group is able to establish a connection with the designated server. A port scan is conducted for servers to identify open ports and protocols. This information is linked to the user group of the server. For clients, the system constructs ACLs based on user groups and existing database information about servers, ensuring only communication between the user's MAC and IP address and the server's IP address and port. Since only the port is required for enforcing ACLs, the mechanism naturally extends to multiple SDN switches, as the ACL is bound to a port on a switch rather than a specific device.

Administrators can create templates for specific scenarios, such as restricting SSH access to administrators only. For example, port 22 can be explicitly bound to the *Administrator* group. These templates are scanned before actual ACL generation, with higher priority than user roles and dynamically discovered ports. Templates are also essential for managing Internet traffic, with administrators defining routing rules that cannot be derived automatically from database information.

Since updates of active users and checks for new ports on servers occur periodically (once a day by default, configurable by administrators), the system maintains a minimal ACL set and removes unused entries if a host is no longer connected, thereby improving efficiency [36].

C. Dynamic Authentication Events and IDS Integration

In addition to static ACL generation, the system accounts for dynamic authentication events. An LDAP proxy continuously monitors authentication activities to detect suspicious behavior, such as repeated failed login attempts for a particular service. Inspired by ALERT-ID [12], our approach extends the concept by directly adapting the network configuration when malicious behavior is observed.

Suspicious traffic is identified based on configurable thresholds. For example, an alarm is triggered when more than fifty packets are sent to invalid IP addresses, when communication attempts are made with five or more distinct blocked IPs (a common indicator of scanning activity), or when repeated connection attempts target sensitive ports such as SSH or non-standard high ports. When such conditions are met, the SDN Controller modifies OpenFlow rules to redirect all traffic from the suspicious user through the integrated Intrusion Detection System (IDS). This redirection applies to both internal and external flows, enabling centralized inspection of potentially malicious activity.

The IDS, implemented with Snort [37] in our prototype, evaluates the forwarded traffic and raises alarms that are logged in a structured JSON format. Depending on the severity of the alarm, different mitigation strategies are applied. For high- and medium-priority alarms, the system immediately updates ACLs to block the offending traffic and sends an alert to the administrator. For low-priority alarms, ACLs remain unchanged, but the administrator is notified so that the event can be investigated further.

This integration of threshold-based detection, real-time rule updates, and IDS alarms provides a layered defense mechanism. It ensures that suspicious activity is promptly identified and that the system can react adaptively, either by isolating malicious traffic or by escalating alerts to administrators.

D. Quantitative Analysis of Access Control Lists

We rely on a quantitative understanding of ACLs to evaluate the system's complexity. Using ACL counts as a metric is consistent with prior SDN research, where the number of ACL policies is directly linked to controller processing delay and scalability [36]. The ACL count is determined based on user, group, and port configurations, providing insights into the scale and complexity of the access control mechanisms.

- **Number of ACLs per User** (N_u): This metric quantifies the number of ACL entries associated with each user. It is calculated by summing the ports across all groups a user belongs to, given by

$$N_{u_i} = \sum_{g=1}^{G_i} P_{g_i} \tag{1}$$

where P_{g_i} is the number of ports for group g for user i and G_i is the total number of groups for user i.

- **Global Number of ACLs** (N_{global}): The total number of ACLs across the network reflects the overall complexity of access control. It is computed as

$$N_{global} = \sum_{i=1}^{U} N_{u_i} \tag{2}$$

where U represents the total number of users, and N_{u_i} is the number of ACLs for user i.

- **Number of ACLs per Switch** (N_s): Understanding the ACL count for each switch helps in optimizing access control at the local level. This metric is determined by

$$N_s = \sum_{i \in S} N_{u_i} \tag{3}$$

where S is the set of users connected to the switch, and N_{u_i} represents the number of ACLs for user i in the set S.

These metrics not only provide a clear measure of system complexity but also form the basis for comparing different security configurations in the subsequent evaluation.

The subsequent section introduces the formal security model and threat analysis, establishing the assumptions and adversary capabilities before moving to implementation details.

V. Formal Security Model and Threat Analysis

This section formalizes the security objectives, assumptions, and threats addressed by the proposed system. We begin by outlining the intended security goals and the structural system model from a security standpoint. This is followed by trust assumptions, a detailed adversary model, and a threat analysis. We then extend the analysis to account for Intrusion Detection System (IDS) integration, before reflecting on limitations and directions for future work.

A. Security Objectives

The proposed system aims to strengthen network security through several concrete objectives. First, authenticated network access ensures that only verified users and devices may participate in network communication. Second, fine-grained access control is enforced via dynamically generated Access Control Lists (ACLs) based on user group membership, which are centrally managed through Active Directory. The system introduces accountability by logging authentication attempts and access control decisions, enabling forensic analysis. A key focus lies in dynamic response to emerging threats: repeated authentication failures trigger ACL updates, while only client traffic that violates an ACL is redirected to the IDS. Adhering to the principle of least privilege, clients are granted access only to services required by their role. Finally, the system ensures that security measures balance confidentiality and integrity with availability, so that malicious clients can be isolated without impairing legitimate traffic.

B. System Model

From a security perspective, the system model comprises multiple entities and communication interfaces. Clients represent end-user devices that must authenticate before being granted access to the network. Servers provide services such as mail or GitLab and are associated with defined user groups. The authenticator component is realized through an OpenFlow switch that enforces port-based access control using IEEE 802.1X. The authentication backend consists of a RADIUS server and an Active Directory instance that validates credentials and provides group membership information. A logically centralized SDN Controller orchestrates ACL generation and enforcement, while an LDAP proxy monitors authentication behavior. Suspicious traffic can be mirrored to a monitoring application or redirected through an IDS for deeper inspection.

All communication between components is assumed to be secured using encrypted and authenticated channels. EAPOL messages are transmitted between clients and the authenticator, TLS is used between the SDN Controller and the OpenFlow switch, and LDAP traffic is protected when transiting to the directory server. This design confines the trust boundary to a minimal set of components.

An architectural overview of these entities and their interactions is shown in Figure 2. While this section describes the model from a security standpoint, the detailed technical implementation is presented in Section VI.

C. Trust Assumptions

Several trust assumptions underpin the security guarantees of the system. The SDN Controller is assumed to be secure and uncompromised, as it orchestrates all access policies. The authentication server is assumed to correctly validate credentials and return accurate group memberships. The underlying user database, such as Active Directory, is assumed to provide accurate and up-to-date user and group information. Communication channels are presumed to be encrypted and authenticated to prevent interception or tampering. Switch firmware and enforcement logic are assumed to function correctly and reliably execute ACL rules. IDS components such as Snort are assumed to operate as specified, correctly classifying traffic according to known signatures. Finally, administrator-defined templates are considered trustworthy and free from malicious intent.

D. Threat Model

The system accounts for both external and internal adversaries. External adversaries may attempt to gain unauthorized

TABLE I. Threats, Vulnerabilities, and Mitigations

Threat	Vulnerability	Impact	Mitigation
Unauthorized access by unauthenticated device	Absence of port-level control	High	Port-based authentication with IEEE 802.1X
Brute-force or credential stuffing attacks	No rate limiting on login attempts	Medium	LDAP proxy monitors failures; ACLs updated dynamically
Privilege escalation by authenticated users	No role-based access control	High	ACLs based on user groups and administrator templates
MAC/IP spoofing	Identities not bound to MAC/IP	High	IP bound to authenticated MAC and port
Controller compromise	Centralized SDN control	Critical	Controller hardening, TLS restriction, redundancy
ACL evasion or bypass	Misconfigured or permissive rules	Medium	Automatically generated fine-grained ACLs
IDS overload	Indiscriminate traffic mirroring	Medium	Two-stage detection: monitoring threshold before IDS redirection

access without valid credentials or employ brute-force methods to compromise legitimate accounts. Internal adversaries include compromised users seeking to escalate privileges or move across the network.

Attackers are assumed capable of passively observing or actively injecting traffic. They may attempt to impersonate other users via spoofed MAC or IP addresses, perform credential stuffing, or exploit misconfigurations. Their primary goals include bypassing authentication, accessing restricted services, evading detection, and maintaining persistence. In addition, adversaries may deliberately generate high volumes of suspicious traffic to overwhelm the IDS or exploit delays between detection and mitigation.

E. Threat Analysis and Mitigation

Table I summarizes key threats addressed by the system. The classification follows established SDN threat taxonomies that group attacks into spoofing, unauthorized access, controller compromise, and policy manipulation, as discussed for example by Sharma and Tyagi [19] and Farooq et al. [29]. Each entry outlines the exploited vulnerability, its impact, and the mitigation strategy implemented. The system integrates multiple layers of defense, ranging from preventive mechanisms such as 802.1X-based port authentication to reactive measures like dynamic ACL updates and IDS redirection. Together, these mechanisms minimize attack surfaces, detect anomalous behavior, and respond swiftly to emerging threats.

F. Unauthorized Network Access

Multiple attack scenarios fall under unauthorized access. Privilege escalation occurs when users attempt to access services beyond their roles, such as SSH to administrative servers. ACL templates restrict such access to authorized groups. Brute-force or credential stuffing attacks target services like GitLab, attempting logins with common credentials. These are detected by counting failed authentication events via the LDAP Proxy and mitigated by blocking the offending client or redirecting its traffic through the IDS. Spoofing is thwarted by binding IP addresses to authenticated MAC addresses and ports, preventing impersonation.

An additional concern is compromise of the authentication backend, such as the LDAP server. While part of the trusted computing base, a compromise could undermine group-based ACL generation. Evasion attempts, such as behaving benignly during authentication but launching attacks later, are addressed by behavior-based detection. In such cases, traffic can be mirrored or redirected to the IDS for deeper inspection, ensuring continuous protection.

G. Security Limitations

While the proposed system improves security and reduces administrative overhead, certain limitations remain. First, the architecture relies on a trusted and centralized controller, which represents a single point of failure. The trust assumptions are static and do not verify runtime integrity, leaving the system exposed if a core component is compromised. Second, ACLs are restricted to header-based inspection. This minimizes performance overhead but prevents detection of application-layer threats. The IDS redirection strategy also operates in a reactive, threshold-based manner, which may delay detection of more adaptive attacks. A further limitation concerns reliance on IEEE 802.1X. Many legacy or IoT devices do not support this protocol and must be handled through weaker fallback mechanisms such as VLAN isolation or MAC-based controls, reducing the uniformity of the security model. Moreover, the system inherently depends on the correctness of the central user database. Incorrect group assignments or outdated entries immediately affect ACL generation, as no secondary validation layer is present. The current evaluation also remains qualitative. Metrics such as detection latency, mitigation time, or ACL update overhead have not been quantified and should be explored in future work.

Finally, while the IDS integration demonstrates feasibility, the accuracy of detection has not been evaluated. False positives or false negatives were not measured, and the overall impact of IDS misclassification on network behavior remains unexamined.

In summary, the formal model and threat analysis establish the security guarantees and limitations of the proposed system. Having defined these foundations, we now turn to the implementation, where the architecture is realized and integrated into a working prototype.

VI. Implementation

Following the conceptual framework outlined in Section IV, the practical implementation of the Port Access Control system integrated various components. The goal was to create a working prototype that demonstrates the feasibility of fine-grained access control and prepares the ground for the evaluation. The design in Figure 2 presents how different parts work together.

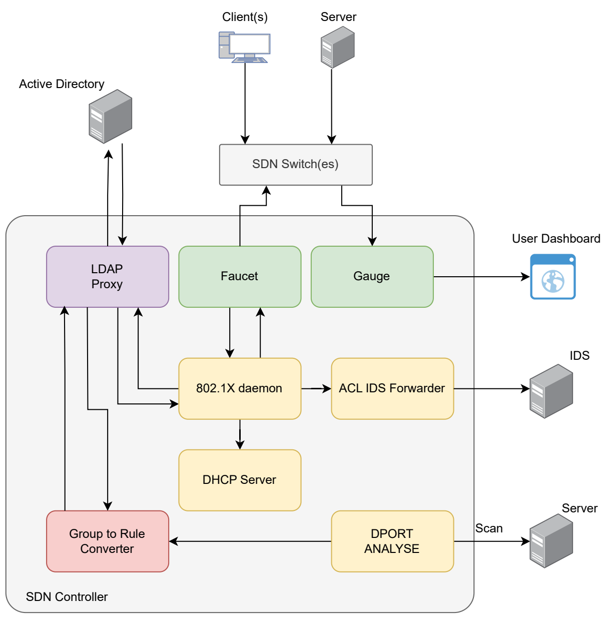

Figure 2. Architecture of the SDN Controller

We chose the Faucet SDN Controller [38] for our prototype, which uses Ryu [39] in the backend and Gauge to view events on the switch. It has the considerable advantage that the rules are defined in YAML (YAML Ain't Markup Language). One significant advantage of this architecture is that these files are human-readable and easy to understand. The initial setup of the OpenFlow switch contains only port information and requires authentication before connecting to the network. Furthermore, specific default rules, such as special treatment for the SDN Controller and the Active Directory, were specified beforehand, as these settings are essential when configuring a new network. We used the 802.1X daemon Chewie [38] as a starting point, and then it was heavily adapted to obtain user groups via a simple LDAP proxy. A second service called *Group to Rule* applies the ACLs as discussed in Section IV. An example rule can be found in Figure 3. It shows the resulting rule with a defined protocol, port, source MAC and IP address, and destination IP address. Since this is directly applied to the port, no other traffic can pass the OpenFlow switch port.

A Python script that searches for UDP and TCP ports on the server provides the open ports needed to craft the ACLs. It then saves this information into a database with the corresponding MAC address and IP address. One problem is that the server does not directly have an IP address when we try to scan it. We must wait until the IP address is handed over via the DHCP server to start scanning. Therefore, for a server, the ACLs can only be applied later on and not directly, which is

```
acls:
  mac_whitelist_user_ben:
    - rule:
        dl_type: 0x800       # ipv4
        nw_proto: 6          # tcp
        tcp_dst: 80          # port
        eth_src:  32:90:43:57:f2:01
        ipv4_src: 192.168.0.1
        ipv4_dst: 192.168.0.9
        actions:
            allow: True
    - rule:
            actions:
                allow: False
                mirror: 3
```

Figure 3. FAUCET ACL Configuration

not a problem since the default rules still block all access to the network and only DHCP is then allowed to obtain an IP address. A simple folder structure was defined for the templates where an administrator can place templates for groups and specific ports as well as initial network operations such as DHCP and DNS.

The dynamic adaptation of ACLs depending on authentication data is realized via the LDAP Proxy. All servers in the network attempt to authenticate their users via LDAP bind requests to the proxy, which then forwards them to the Active Directory. This setup allows us to track whether authentication was successful. We implemented a counter for each user with a configurable threshold (six failed attempts by default), which resets after a timeout period, similar to the mechanism described in ALERT-ID [12]. A lightweight monitoring script parses the authentication logs and forwards suspicious events to the SDN Controller.

For demonstration purposes, any traffic that would normally be blocked is mirrored to a dedicated port where a simple Python counter application is connected. This service maintains per-host violation counts (based on MAC and IP addresses) and alerts the SDN Controller once a threshold is exceeded. When thresholds are crossed, such as repeated attempts to access non-standard ports, scanning activities against multiple blocked IPs, or persistent connections to invalid destinations, the controller updates the OpenFlow configuration and enforces stricter handling of the suspicious host.

At this point, the IDS integration becomes active. In our prototype we used Snort [37], which inspects all traffic from hosts that exceeded the violation threshold. Critical alarms, such as confirmed scanning or exploitation attempts, immediately trigger ACL updates that block the corresponding host and alert the administrator. Medium-severity alarms also result in blocking, but the administrator receives a detailed event log for further analysis. Low-severity alarms are logged and reported without enforcing automatic blocking, leaving the final decision to the administrator.

This two-stage design ensures that the IDS is not overloaded by benign or low-level violations. Faucet's mirroring capability

TABLE II. ACL CONFIGURATION FOR FIVE CONNECTED CLIENTS

OpenFlow Port	Source MAC	Source IP	Group	Destination IP	Destination Port	Description
3	1C:69:7A:6D:C6:27	192.168.11.11	mail	192.168.11.101	25, 993, 995	Mailserver
3	1C:69:7A:6D:C6:27	192.168.11.11	gitlab	192.168.11.102	22, 80, 443	GitLab
4	1C:69:7A:43:7C:12	192.168.11.12	mail	192.168.11.101	25, 993, 995	Mailserver
5	1C:69:7A:6D:C8:B0	192.168.11.13	mail	192.168.11.101	25, 993, 995	Mailserver
5	1C:69:7A:6D:C8:B0	192.168.11.13	gitlab	192.168.11.102	22, 80, 443	GitLab
6	1C:69:7A:6D:C7:EE	192.168.11.14	mail	192.168.11.101	25, 993, 995	Mailserver
7	1C:69:7A:6D:C8:16	192.168.11.15	mail	192.168.11.101	25, 993, 995	Mailserver

enables efficient pre-filtering via the counter application, and only persistent or severe violations cause full redirection through the IDS. As a result, IDS integration is not a standalone add-on but an embedded part of the access control pipeline managed by the SDN Controller.

The implementation phase reaffirmed the proposed system's potential to enhance network security through fine-grained access controls and adaptive IDS support. At the same time, it highlighted the complexities of managing an extensive rule set, especially in larger networks where automated rule aggregation and optimization become critical.

VII. Evaluation

In this evaluation chapter, we begin with a detailed examination of the technical aspects of our experimental setup, laying the groundwork for a thorough assessment. We then delve into the feasibility of the proposed system, followed by a comparative analysis of its efficiency and complexity against existing systems. This analysis sets the stage for a nuanced discussion synthesizing our findings and their implications.

A. Experimental Conditions

Our experimental setup was designed to mirror a realistic environment consisting of multiple physical PCs and servers to simulate a conventional corporate network infrastructure. The network configuration included five Windows clients. We assigned the clients to different user groups in the Active Directory. The configuration of each client and its connected port can be found in Table III. In setting up our experiment, we went with a mix that one would typically find in an office: a mail server for emails and a GitLab instance for the devs to collaborate on code. This way, we could see how different roles, like developers needing GitLab and managers relying on emails, would interact with the system. It is a practical approach that helps us understand how our setup performs in a real-world scenario.

TABLE III. CLIENTS IN THE NETWORK

Client Name	Port	Source MAC	Groups
Client1	3	1C:69:7A:6D:C6:27	mail, gitlab
Client2	4	1C:69:7A:43:7C:12	mail
Client3	5	1C:69:7A:6D:C8:B0	mail, gitlab
Client4	6	1C:69:7A:6D:C7:EE	mail
Client5	7	1C:69:7A:6D:C8:16	mail

At the core of our network was an Active Directory on a Windows Server 2019, connected to a dedicated port at the OpenFlow switch. This switch was a Linux PC running Ubuntu 22.10, with an Intel(R) Core(TM) i7-8700 CPU supporting OpenFlow protocol version 1.3.

This detailed setup provides a solid foundation for evaluating the system's feasibility, performance, and complexity.

B. Feasibility

The project aimed to demonstrate the feasibility of such a system and highlight its advantages. Therefore, we conducted multiple experiments to verify the system's operability to achieve this. In our earlier conference paper [1], we presented two experiments demonstrating feasibility: initial connection and ACL enforcement, and failed login handling. In this extended version, we add a third experiment that evaluates the integration of an IDS for redirecting suspicious traffic. Together, these three experiments provide a comprehensive assessment of the system.

1) Experiment 1: Connection to the network: In our initial experiment, we aimed to verify the functionality of the system's initial configuration and the practical application of Access Control Lists. We began by attempting to connect a server to the network. Initially, all packets except EAP packets were blocked, preventing any network connection without proper authentication. To facilitate authentication, we configured the server's wpa_supplicant with EAP after setting up a dedicated user account in the Active Directory for the server, marked by the "server" group identifier, to distinguish it as such. Additionally, the server was assigned to the "mail" group to define its access rights. The authentication process utilized standard Username and Password credentials defined within the Active Directory.

Upon initiating these configurations, we observed successful authentication, followed by the server obtaining an IP address via DHCP. The IP address assignment was managed by the SDN Controller, ensuring the server's connectivity post-authentication. We then proceeded with a port scan, which was feasible only after the OpenFlow switch recognized the server's IP, confirming that the server was operational. The procedure was repeated for the second GitLab server.

Subsequently, we connected a client machine to the network. Like the server setup, this client was denied network access until authentication credentials were provided. After authentication, the SDN Controller dynamically generated ACLs based on the client's group memberships.

For example, the first client, identified as a developer, was granted access to both the mail server and GitLab, as reflected in the applied ACLs (refer to Table II, lines 1 and 2). This access control was strictly enforced, with all unauthorized traffic being blocked at the port level based on the authenticated source MAC address and specified port. In contrast, a client identified as a manager, and thus only requiring access to the mail server, demonstrated restricted network access in line with their role (refer to Table II, line 3). Attempts to access GitLab by this client were blocked, illustrating the ACLs' role-based access control. After connecting all clients, each client and port results can be found in Table II.

Upon issuing a logoff command to the RADIUS server, all associated ACLs were cleared, reverting the system to its default state of blocking all traffic from the disconnected client. Logging in with a different username on the same PC triggered a reallocation of ACLs, aligning with the new user's access rights. This experiment demonstrated the feasibility of initially creating and effectively applying ACLs within our network environment.

2) Experiment 2: Failed Logins: In the second experiment, we tested failed login attempts to evaluate the system's response mechanisms. This test simulated incorrect authentication attempts on the GitLab server to observe the system's reaction.

The experiment began with a series of failed login attempts, with each unsuccessful attempt logged by the SDN Controller. After the sixth failed attempt, the SDN Controller adjusted the ACLs, cutting off the client's access to the server and other network components. An alert was automatically sent to the network administrator, who could either restore the client's access after a successful re-authentication or suspend the client for further investigation.

Additionally, we tested the network's traffic mirroring feature. In this part of the experiment, despite multiple failed login attempts, the client was not disconnected from the network. Instead, the client's traffic was mirrored to a specific port on an OpenFlow switch. This procedure was verified using tcpdump to confirm that the traffic mirroring was functioning as intended, without the integration of an IDS, since this was not in the scope of the experiment.

3) Experiment 3: Redirecting Suspicious Traffic to IDS: The third experiment evaluated the system's ability to identify suspicious clients and to enforce dynamic redirection through the IDS. The scenario consisted of the following key steps:

1) Introduce a malicious host that violated predefined policies.
2) Detect repeated violations until defined thresholds are exceeded.
3) Redirect all traffic from the host through the IDS.
4) Generate malicious traffic samples to trigger Snort rules.
5) Apply mitigation actions depending on alert severity.

To simulate malicious activity in detail, we introduced a host that attempted to connect to non-standard ports, repeatedly accessed invalid IP addresses, and performed network scanning across multiple targets. Each of these actions incremented the violation counter maintained by the monitoring application, which received mirrored traffic from the OpenFlow switch.

Once the violation thresholds were exceeded, the SDN Controller updated the Faucet rules for the offending host. Instead of simply dropping further packets, the rules were rewritten so that all subsequent traffic from the host was forwarded through the IDS. In our prototype, this redirection was achieved by replacing the `mirror` action with an `output` action that directed the traffic to the IDS ingress port.

To further assess the system's responsiveness, we generated malicious traffic using publicly available attack samples and network scanning tools in order to trigger common Snort rules. Snort [37], deployed as the IDS in our setup, successfully identified the injected traffic and produced alarms. Depending on the severity of the alert:

- Critical alerts caused the controller to immediately block the host and notify the administrator.
- Medium alerts also resulted in blocking but generated detailed logs for review.
- Low alerts were logged and reported without automatic blocking.

Figure 4 shows the CPU usage of the Snort instance during a 600-second evaluation run. The annotated timeline of events is as follows:

- **0s:** Start of experiment, baseline traffic only.
- **152s:** First client exceeded thresholds and was redirected to IDS.
- **307s:** A second malicious client was added, increasing IDS load.
- **531s:** Snort raised a critical alert for the first client; the SDN Controller blocked it, reducing IDS load.

This progression highlights how each additional client measurably increased CPU usage on the IDS, while removing a client reduced load again. CPU utilization is strongly correlated with traffic volume, consistent with prior measurements by Lukaseder et al. [40]. Although our test traffic was deliberately crafted to repeatedly trigger Snort rules (leading to a higher per-client load than in production), the experiment demonstrates that selective redirection scales with the number of suspicious hosts while preventing the IDS from being overwhelmed with benign traffic.

C. Complexity and Efficiency

To evaluate our system's complexity and OpenFlow rule management capability, we compared it against other SDN security methods by examining the number of OpenFlow rules in different scenarios. Our analysis included a baseline scenario without ACLs, a basic ACL setup, and scenarios involving VLANs. The scenario with Basic ACLs has only rules for direct IP access. That means we only specify that user X can access server Y without further defining which ports or protocols. The VLAN example does not have any specific ACLs. It splits the users into two groups, usually some kind of department in a corporate network. This option has the disadvantage of allowing clients from the same department to

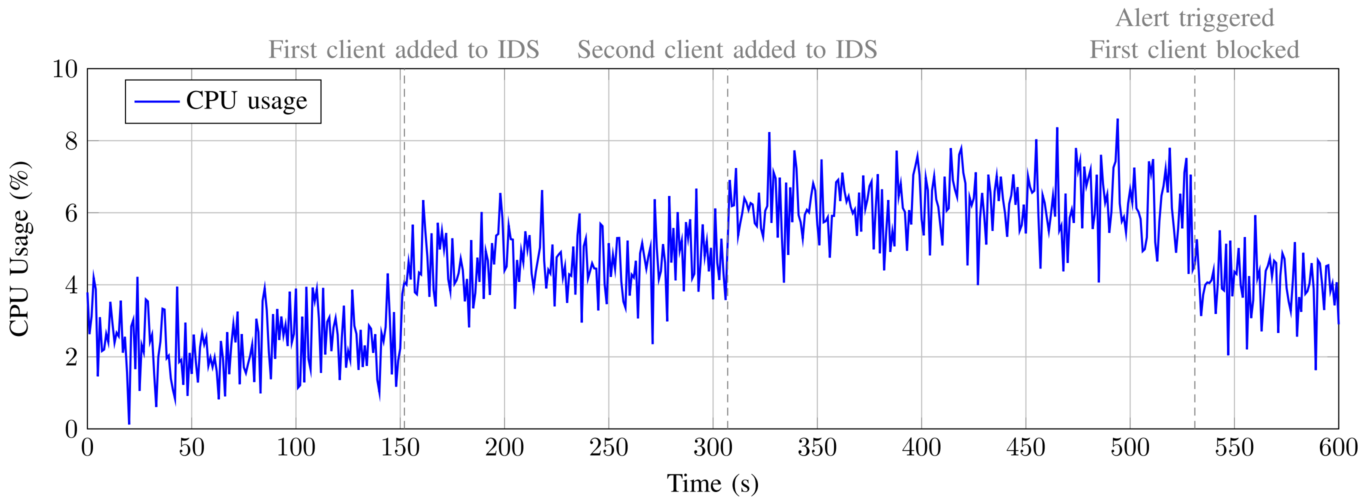

Figure 4. CPU usage over time with key IDS events highlighted.

communicate, which does not prevent malware from spreading.

Table IV summarizes the OpenFlow rule count for each scenario. As observed, the number of OpenFlow rules directly reflects the count of Faucet ACLs. As discussed in Equation (2), the number of ACLs will increase linearly with the number of users. The dynamic ACL configuration, while more complex, demonstrates the system's flexibility and responsiveness to network changes without significantly impacting performance.

TABLE IV. RULE COUNT COMPARISON

Scenario	Faucet ACLs	OpenFlow Rules
No ACLs	0	27
Basic ACLs	8	67
VLANs	N/A	67
Dynamic ACLs	32	91

D. Discussion

In discussing the outcomes and implications of our experiments, it is essential to consider both the implemented system's strengths and potential challenges. To offer a comprehensive understanding of our system's enhanced performance and its innovative approach to network security and management, we performed an extensive comparison across several key metrics, including security level, scalability, and manageability. This comparison, detailed in Table V, is based on empirical data from ACL number analytics, a comparative analysis of system architectures, and their maintenance needs, highlighting our proposed system's superiority in terms of security, scalability, and ease of management.

The introduction of automated, fine-grained whitelist ACLs represents a significant step forward in network security management. The configuration process substantially decreases administrative overhead and mitigates the risk of human error, which is prevalent in manual configurations. A crucial advantage of this approach is the centralization of security decisions, such as access rights, in a singular user database. This consolidation ensures that modifications to access rights are uniformly applied across the network and all services utilizing this common user database, thereby enhancing consistency and security within the system. As demonstrated in Experiment 1, the automation of ACL configuration significantly reduced administrative overhead since all needed restrictions are applied individually for each client without the need for additional adaptation by the administrator. In contrast, this centralized, automated approach ensures that only authenticated users and their associated MAC addresses are actively maintained in the system, limiting access to authorized entities and inherently reducing the risk of unauthorized access. Another significant benefit of our approach is its scalability and ease of integration across multiple switches without additional overhead. Since ACLs and clients are bound to specific ports and not to the physical switches themselves, our system can seamlessly scale to accommodate an extensive network infrastructure with multiple SDN switches. ACLs are applied uniquely to each switch, as delineated in Equation (3), ensuring efficient and tailored security measures are in place, irrespective of the size or complexity of the network.

However, this automation and simplification come at the cost of increased complexity due to the more significant number of ACLs required to maintain fine-grained control over network access. The number of ACLs does not directly impact the system's performance since it only inspects the TCP header to minimize performance impact, compared to, for example, complex rules that inspect the TCP payload. According to Cabrera et al. [41], the time required to check the payload is, on average, 4.5 times longer than that required for header checks. Therefore, even with many ACL rules, the focus on header information ensures minimal impact on network throughput, as even a single ACL with a TCP header rule necessitates the inspection of every packet. One drawback is that we need to prepare the clients and the server to perform

TABLE V. COMPARISON OF NETWORK SECURITY AND MANAGEMENT APPROACHES

Metric	**No ACLs**	**Basic ACLs**	**VLANs**	**Resonance**[16]	**ACL Based on X812**[18]	**Proposed System**
Security Level	Low	Medium	Medium	High	Very High	Very High
Port Security	None	None	None	None	Full	Full
Performance Impact	Low	Medium	Medium	Moderate	Moderate	Moderate
Scalability	High	Moderate	Good	Moderate	Moderate	Excellent
Manageability	Easy	Moderate	Moderate	Moderate	Moderate	Easy
Centralization	None	Low	Low	Medium	Medium	High
Flexibility	Low	Moderate	Low	Very High	Very High	High
Cost Efficiency	High	Moderate	Moderate	Low	Moderate	High
Integration Capability	Seamless	Moderate	Challenging	High	High	Low
Resilience	Low	Medium	Medium	High	High	High
Automation & Dynamic Response	None	None	None	Semi-Automated	Semi-Automated	Fully Automated
ACLs based on Authentication	None	None	None	None	None	Supported

an 802.1X authentication.

The experiments involving IDS redirection further confirmed the system's scalability and efficiency. As shown in Figure 4, the CPU usage of Snort increased proportionally with the number of malicious clients redirected, but decreased again once clients were blocked by the SDN Controller. This demonstrates that the selective forwarding mechanism ensures the IDS is only engaged for traffic that truly warrants inspection, preventing overload and allowing for efficient use of IDS resources. Such integration provides a stronger security posture without sacrificing performance for benign traffic, which continues to be handled exclusively by the ACL framework.

One of the more critical considerations is the system's approach to handling failed login attempts, as demonstrated in Experiment 2. Completely blocking access after a series of incorrect credentials can safeguard against brute-force attacks but also pose a risk to business continuity. For instance, automated tools using outdated credentials could inadvertently trigger these security measures, leading to unnecessary disruptions. This aspect of the system necessitates a careful balance between maintaining robust security and ensuring uninterrupted business operations.

Integrating traffic mirroring for suspicious hosts presents a nuanced approach to enhancing security monitoring without overloading the network or the IDS. By selectively mirroring traffic from potentially compromised hosts, the system can focus on analyzing and responding to genuine threats, improving overall security efficiency. This concept aligns with the approach discussed in [42], which proposes a clustering-based flow grouping scheme that assigns network flows to various IDSs based on routing information and flow data rates, aiming to optimize the load distribution among IDSs and enhance attack detection capabilities.

VIII. Conclusion and Future Work

In conclusion, the proposed system presents a straightforward yet powerful framework that significantly enhances network security by enforcing fine-grained access control rules. By leveraging a common user database, such as Active Directory, and binding access controls to specific MAC addresses, the system ensures that only authenticated users can access network ports, thereby establishing a robust security posture.

A key contribution of this work is the integration of an Intrusion Detection System into the access control framework. Rather than deploying the IDS inline for all traffic, our approach mirrors suspicious flows to a monitoring application and selectively redirects offending hosts once violation thresholds are reached. This two-stage mechanism ensures that benign traffic is not subjected to costly IDS inspection, while malicious clients are efficiently isolated and analyzed. As demonstrated in our evaluation, this selective redirection prevents IDS overload, allows scalable operation, and improves the responsiveness of the SDN Controller to detected threats.

Although the results demonstrate the feasibility of the proposed architecture, several limitations remain. The evaluation was conducted in a controlled laboratory environment, which restricts the generality of the findings. The approach relies on a trusted and centralized SDN Controller as well as correct group assignments in the user database, and these assumptions may not hold in all deployments. IDS behavior was not evaluated with respect to false positives or false negatives, and performance under larger or more dynamic network conditions remains open for further study.

Future work should extend the evaluation and broaden the applicability of the system. A first step is a more detailed performance study, including metrics such as ACL update latency, controller processing times, IDS alert latency, and the behavior of the system under higher client loads. The IDS integration should also be assessed in a more systematic way, for example by measuring false-positive and false-negative rates and by comparing different inspection strategies. Additional experiments with larger deployments would help validate scalability, while integration with advanced monitoring systems such as Zeek [43] could widen the visibility of network flows and enable deeper analyses. Another line of research concerns the interaction between dynamically generated ACLs, existing firewall rules, and higher-level network policies. Incorporating policy conflict detection and resolution mechanisms from related SDN research could further strengthen robustness in hybrid environments. Finally, evaluating unknown attack classes is orthogonal to the access-control-focused design of this system but may complement future studies that investigate more general SDN-based threat detection frameworks.

Acknowledgment

The authors would like to thank Prof. Hiroki Takakura for useful advice. This work was partially supported by JSPS KAKENHI Grant Number JP23K28086 and JP24K14959.

References

[1] F. Grießer, A. Shinoda, H. Hasegawa, and H. Shimada, "Automating SDN-ACLs with user groups and authentication events," in *Proceedings of the Thirteenth International Conference on Networks (ICN 2024).* Barcelona, Spain: IARIA, May 2024, pp. 5–12.

[2] H. Zhou, C. Wu, M. Jiang, B. Zhou, W. Gao, T. Pan, and M. Huang, "Evolving defense mechanism for future network security," *IEEE Communications Magazine*, vol. 53, no. 4, pp. 45–51, 2015.

[3] S. T. Ali, V. Sivaraman, A. Radford, and S. Jha, "A survey of securing networks using software defined networking," *IEEE Transactions on Reliability*, vol. 64, no. 3, pp. 1086–1097, 2015.

[4] F. Bannour, S. Souihi, and A. Mellouk, "Distributed SDN control: Survey, taxonomy, and challenges," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 333–354, 2017.

[5] P. Wadhwani, "Software defined networking (sdn) market report, 2024–2032," Market Research Report, Global Market Insights Inc., Report ID: GMI2395, 2024, accessed: 2025-12-09. [Online]. Available: https://www.gminsights.com/industry-analysis/software-defined-networking-sdn-market

[6] F. S. Ali, R. Amin, M. Majeed, and M. M. Iqbal, "Dynamic ACL Policy Implementation in Software Defined Networks," in *2022 International Conference on IT and Industrial Technologies (ICIT)*, Oct 2022, pp. 01–07.

[7] S. T. Yakasai and C. G. Guy, "FlowIdentity: Software-defined network access control," in *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN).* IEEE, 2015, pp. 115–120.

[8] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: taking control of the enterprise," in *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 1–12.

[9] D. M. Ferrazani Mattos and O. C. M. B. Duarte, "AuthFlow: authentication and access control mechanism for software defined networking," *Annals of Telecommunications*, vol. 71, pp. 607–615, 2016.

[10] T. Xing, D. Huang, L. Xu, C.-J. Chung, and P. Khatkar, "SnortFlow: A OpenFlow-Based Intrusion Prevention System in Cloud Environment," in *2013 Second GENI Research and Educational Experiment Workshop*, March 2013, pp. 89–92.

[11] A. Le, P. Dinh, H. Le, and N. C. Tran, "Flexible Network-Based Intrusion Detection and Prevention System on Software-Defined Networks," in *2015 International Conference on Advanced Computing and Applications (ACOMP)*, Nov 2015, pp. 106–111.

[12] J. Chu, Z. Ge, R. Huber, P. Ji, J. Yates, and Y.-C. Yu, "ALERT-ID: analyze logs of the network element in real time for intrusion detection," in *Research in Attacks, Intrusions, and Defenses: 15th International Symposium, RAID 2012, Amsterdam, The Netherlands, September 12-14, 2012. Proceedings 15.* Springer, 2012, pp. 294–313.

[13] A. H. Janabi, T. Kanakis, and M. Johnson, "Survey: Intrusion detection system in software-defined networking," *IEEE Access*, vol. 12, pp. 164 097–164 120, 2024.

[14] B. Susilo and R. F. Sari, "Intrusion detection in software defined network using deep learning approach," in *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC).* IEEE, 2021, pp. 0807–0812.

[15] B.-X. Wang, J.-L. Chen, and C.-L. Yu, "An ai-powered network threat detection system," *IEEE Access*, vol. 10, pp. 54 029–54 037, 2022.

[16] A. K. Nayak, A. Reimers, N. Feamster, and R. Clark, "Resonance: dynamic access control for enterprise networks," in *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*, ser. WREN '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 11–18.

[17] V. Shah and P. Yadav, "An implementation of dot 1x for secure network access in sdn," in *2025 6th International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV).* IEEE, 2025, pp. 1264–1269.

[18] B. J. C. de A. Martins, D. M. Mattos, N. C. Fernandes, D. Muchaluat-Saade, A. B. Vieira, and E. F. Silva, "An Extensible Access Control Architecture for Software Defined Networks based on X.812," in *2019 IEEE Latin-American Conference on Communications (LATINCOM)*, 2019, pp. 1–6.

[19] P. K. Sharma and S. Tyagi, "Security enhancement in software defined networking (sdn): A threat model," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 9, 2021.

[20] S. Pradeep, Y. K. Sharma, U. K. Lilhore, S. Simaiya, A. Kumar, S. Ahuja, M. Margala, P. Chakrabarti, and T. Chakrabarti, "Developing an sdn security model (ensures) based on lightweight service path validation with batch hashing and tag verification," *Scientific Reports*, vol. 13, no. 1, p. 17381, 2023.

[21] Y. Meng, Z. Huang, G. Shen, and C. Ke, "A security policy model transformation and verification approach for software defined networking," *Computers & Security*, vol. 100, p. 102089, 2021.

[22] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for openflow networks," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, 2012, pp. 121–126.

[23] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao, "Flowguard:

Building robust firewalls for software-defined networks," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, 2014, pp. 97–102.

[24] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey, "Veriflow: Verifying network-wide invariants in real time," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, 2012, pp. 49–54.

[25] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte, "Real time network policy checking using header space analysis," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, 2013, pp. 99–111.

[26] C. Prakash, J. Lee, Y. Turner, J.-M. Kang, A. Akella, S. Banerjee, C. Clark, Y. Ma, P. Sharma, and Y. Zhang, "Pga: Using graphs to express and automatically reconcile network policies," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 29–42, 2015.

[27] S. Pisharody, J. Natarajan, A. Chowdhary, A. Alshalan, and D. Huang, "Brew: A security policy analysis framework for distributed sdn-based cloud environments," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 6, pp. 1011–1025, 2017.

[28] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[29] M. S. Farooq, S. Riaz, and A. Alvi, "Security and Privacy Issues in Software-Defined Networking (SDN): A Systematic Literature Review," *Electronics*, vol. 12, no. 14, 2023.

[30] "IEEE Standard for Local and Metropolitan Area Networks–Port-Based Network Access Control," *IEEE Std 802.1X-2020 (Revision of IEEE Std 802.1X-2010 Incorporating IEEE Std 802.1Xbx-2014 and IEEE Std 802.1Xck-2018)*, pp. 1–289, 2020.

[31] B. Desmond, J. Richards, R. Allen, and A. G. Lowe-Norris, *Active Directory: Designing, Deploying, and Running Active Directory*. " O'Reilly Media, Inc.", 2008.

[32] J. Sermersheim, "Lightweight Directory Access Protocol (LDAP): The Protocol," RFC 4511, Jun. 2006.

[33] J. Vollbrecht, J. D. Carlson, L. Blunk, D. B. D. Aboba, and H. Levkowetz, "Extensible Authentication Protocol (EAP)," RFC 3748, Jun. 2004.

[34] H. Hasegawa, Y. Sato, and H. Takakura, "Construction of Secure Internal Network with Communication Classifying System Using Multiple Judgment Methods," *International Journal on Advances in Telecommunications*, vol. 13, no. 3 & 4, 2020.

[35] Y. Sato, H. Hasegawa, and H. Takakura, "Construction of Secure Internal Networks with Communication Classifying System," in *ICISSP*, 2019, pp. 552–557.

[36] M. Ali, N. Shah, and M. A. Khan Khattak, "DAI: Dynamic ACL Policy Implementation for Software-Defined Networking," in *2020 IEEE 17th International Conference on Smart Communities: Improving Quality of Life Using ICT, IoT and AI (HONET)*, Dec 2020, pp. 138–142.

[37] M. Roesch, "Snort: Lightweight intrusion detection for networks," in *Proceedings of the 13th USENIX Conference on System Administration (LISA '99)*. USENIX Association, 1999, pp. 229–238.

[38] FaucetSDN, "Faucet," 2024, accessed: 2025-11-20. [Online]. Available: https://github.com/faucetsdn/faucet

[39] Ryu SDN Framework Community, "Ryu sdn framework," https://ryu-sdn.org/, 2024, accessed: 2025-11-20.

[40] T. Lukaseder, J. Fiedler, and F. Kargl, "Performance evaluation in high-speed networks by the example of intrusion detection," *arXiv preprint arXiv:1805.11407*, 2018.

[41] J. B. Cabrera, J. Gosar, W. Lee, and R. K. Mehra, "On the statistical distribution of processing times in network intrusion detection," in *2004 43rd IEEE Conference on Decision and Control (CDC)(IEEE Cat. No. 04CH37601)*, vol. 1. IEEE, 2004, pp. 75–80.

[42] T. Ha, S. Yoon, A. C. Risdianto, J. Kim, and H. Lim, "Suspicious flow forwarding for multiple intrusion detection systems on software-defined networks," *IEEE Network*, vol. 30, no. 6, pp. 22–27, 2016.

[43] T. Z. Project, "Zeek: Network security monitor," https://github.com/zeek/zeek, 2024, accessed: 2025-11-27.