MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Fine-grained access management for eduVPN

Bachelor's Thesis

VÍT LABUDA

Brno, Spring 2026

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Fine-grained access management for eduVPN

Bachelor's Thesis

VÍT LABUDA

Advisor: RNDr. Petr Velan, Ph.D.

Department of Computer Systems and Communications

Brno, Spring 2026


FACULTAS ARTIS INFORMATICÆ
UNIVERSITAS MASARYKIANA

Declaration

Hereby I declare that this thesis is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

During the preparation of this thesis, I used ChatGPT to improve my writing style. No source code in this thesis was written by AI. I declare that I used this tool in accordance with the principles of academic integrity. I checked the content and took full responsibility for it.

Vít Labuda

Advisor: RNDr. Petr Velan, Ph.D.

Acknowledgements

Firstly, I would like to thank my advisor, RNDr. Petr Velan, Ph.D., and my consultant, Bc. Michal Drobný, for valuable discussions and advice regarding this thesis. My next thanks go to Ing. Roman Alexander Mariančík, who granted me access to the virtual machines running the current eduVPN deployment, which helped me greatly with understanding the overall eduVPN infrastructure at Masaryk University.

Last but not least, I am immensely grateful towards my family, especially my mother, for persistent support and patience throughout the entire period of writing this bachelor's thesis.

Abstract

Proper network-level access controls are vital for ensuring the security of Virtual Private Networks (VPNs), as they may be used by geographically distant clients whose physical security may easily be compromised. The eduVPN system at Masaryk University currently relies on a coarse-grained, profile-based access control architecture, which has several limitations; therefore, a new, fine-grained architecture is proposed in the thesis, and its viability is explored using a proof-of-concept implementation. A central database has been designed to store permission information, and a software solution has been created which implements the proposed architecture by automatically managing firewall configuration on eduVPN Nodes. The solution has been tested and measured, showing that the architecture is functional in a testing environment. Its deployment to a production environment is out of scope of this thesis and may be the focus of further research.

Keywords

eduVPN, VPN, nftables, firewall, access controls, network security, computer networking

Contents

List of Tables

List of Figures

Introduction

Enforcing proper network-level access controls in computer networks is a vital part of ensuring their security. This holds true for both physical networks and Virtual Private Networks (VPN), which utilize physical networks in conjunction with tunneling protocols with encryption to create a virtual network consisting of devices that may be physically present in geographically distant locations. Since this geographic separation prevents accurate identification of the individuals operating the devices and therefore increases the probability of misuse, enforcing proper network-level access controls is even more essential in VPNs as compared to physical networks, where such user identification is far more likely to be possible.

The VPN service provided to students and employees of Masaryk University is implemented using the eduVPN software system [1]. The current access control architecture in this system is based on the concept of profiles, which are created for faculties and departments and which are made available to users affiliated with them. Since all users who have access to a particular profile have necessarily access to the same set of network services, this profile-based access control architecture is considered to be overly coarse-grained in case targeting individual users is preferred. In addition, since a device can only be connected via a single profile at one point, this results in discomfort for users who need to access services only available from distinct profiles at the same time.

Therefore, this thesis proposes a new, fine-grained access control architecture which eliminates the need for managing multiple profiles and replaces them with a single, unified profile accessible to all users. In this architecture, individual users connected via this profile are permitted network-level access only to services they necessarily need by firewalls on eduVPN Nodes. The configuration of these firewalls is dynamically generated and is based on permission information stored in a central database.

This central database is designed and implemented as part of this thesis, as well as custom software components which utilize it. These software components enable the database to be synchronized with an external data source, integrate it into the eduVPN system for the

purpose of receiving notifications about user connections and disconnections, and facilitate the automatic generation and dynamic management of firewall configurations on eduVPN Nodes. It should be noted that the implemented software is considered proof-of-concept, and its integration into the production eduVPN deployment at Masaryk University is out of scope of this thesis.

In Chapter 1, key concepts and technologies necessary for understanding the rest of the thesis are explained. In Chapter 2, both the current and the proposed access control architectures are described in detail. In Chapter 3, the architecture of the central database, eduVPN Node firewalls and the custom software is designed. In Chapter 4, the proof-of-concept software implementation is presented on a per-component basis. Finally, in Chapter 5, the impact of extensive firewall configurations on network throughput is investigated.

1 Technologies

This chapter describes the key concepts and technologies on which both the architecture (Chapters 2 and 3) and the implementation (Chapter 4) of the proposed solution were built upon. Firstly, the concept of a *Virtual Private Network* (VPN) is explained with a specific focus on the *eduVPN* software system, and secondly, *firewalls* are explained as they have an essential role in enforcing network-level access controls with a focus on the *nftables* firewall implementation.

While some other concepts and technologies such as the Python programming language, relational databases and various standardized networking protocols (e.g. IPv4, IPv6, TCP or UDP), also have a key role in the proposed solution, they are not described in this chapter, as they are considered to be neither non-standard nor specialized.

1.1 VPN

A *Virtual Private Network*, often abbreviated as *VPN*, is "a way to simulate a private network over a public network, such as the Internet" [2]. Network tunneling protocols with authentication and encryption layers which form the core of VPNs enable devices with no physical access to a local network (such as a private network of a home or institution) to access that network remotely in a secure manner. This allows users of these devices to use services and resources to which access is not permitted from the public Internet, for example for security or legal (e.g. academic sources protected by copyright) reasons. Another benefit is that it protects their network traffic from various cyberattacks on public networks, such as those in airports and cafes [1].

VPNs can be described as either *split tunnel* or *full tunnel*. The term *split tunnel* is used if they are configured to tunnel only network traffic destined towards specific IP ranges, such as those on which the aforementioned restricted services or resources are available, whereas the term *full tunnel* is used if they tunnel all traffic, including traffic to the public Internet. The VPN operated by Masaryk University can be used in both of these modes [1].

1.1.1 Tunneling protocols

Many network tunneling protocols are used to establish VPNs. Some of them are part of proprietary commercial software solutions, and some of them are free and open-source [2]. In the following paragraphs, the open-source *WireGuard* and *OpenVPN* protocols are outlined, as these are used by the *eduVPN* software system.

The **WireGuard** tunneling protocol operates at L3 (Network Layer) and aims to be performant and simple [3]. Unlike *OpenVPN*, it intends to be primarily implemented in kernel space (although user-space implementations also exist), and this Linux kernel implementation can consist of less than 4,000 lines of code — making it viable for auditing and verification [3]. The tunneled network traffic is carried in an encrypted form by UDP datagrams; TCP mode is not natively supported, which is a known limitation [4].

OpenVPN, which refers to both the tunneling protocol and the user-space software implementation, can operate on both L2 (Data Link Layer) and L3 (Network Layer). The implementation depends on the *OpenSSL* library and usually uses the TLS protocol for ensuring security [5]. For carrying the tunneled network traffic, both UDP and TCP protocols are supported [6].

Natively, both *WireGuard* and *OpenVPN* do not offer any configuration distribution mechanisms for both servers and clients. If a deployment needs to have such a mechanism, an external third-party software needs to be used, for example *eduVPN*.

1.1.2 eduVPN

eduVPN is an open-source software system specifically designed to manage VPN access in an academic environment, which originally "started as a project at the Dutch research and education network SURF" [7]. Internally, it uses *WireGuard* (either natively over UDP, or over TCP using *ProxyGuard*) or *OpenVPN* for tunneling network traffic, and manages the configuration of these protocols' implementations on both the server side and the client side.

The system consists of several components, both client-side and server-side, most notably [8]:

- `vpn-user-portal` — "The user and admin portal and API for eduVPN [...] allowing for self-management by users and administrative tasks by designated administrators." [9]
- `vpn-daemon` — A "daemon [...] providing a HTTP(S) API as an abstraction on top of the management socket of (multiple) OpenVPN server process(es) and WireGuard." [10]
- ***ProxyGuard*** — A daemon for relaying UDP traffic over HTTP(S) [11]. This enables WireGuard to be used via TCP, for example on networks where UDP is blocked.
- ***eduVPN for Linux/Android/iOS/macOS/Windows*** — Client-side GUI applications for user-friendly access to eduVPN from various operating systems.

For connecting to *eduVPN*, users can either use the aforementioned GUI applications, or they can generate a static configuration[1] in the web interface of `vpn-user-portal` which can then be directly imported into one of the native *WireGuard*/*OpenVPN* implementations. This allows users to make advanced custom configuration changes that are not possible in the GUI applications, such as changing the IP ranges for *split tunneling*, or to use *eduVPN* on platforms which are not supported by the GUI applications, including those which have no GUI at all.

On the server side, *eduVPN* deployment consists of at least one *Portal* (also called *Controller*) machine and at least one *Node* machine. Both of these machines can be virtual and must run Linux. *Portal* machines run the `vpn-user-portal` software that implements the web interface and API with which the *eduVPN* GUI applications communicate. *Node* machines then run the `vpn-daemon` software, and through these machines, the actual VPN network traffic flows as they host WireGuard, ProxyGuard or OpenVPN server instances. [12]. In simple deployments, one (virtual) machine can have both the *Portal* and *Node* roles; by contrast, highly available and/or scalable deployments can consist of multiple *Portal* and *Node* machines [12]. Different *Nodes* can be configured to handle different sets of *profiles* [13].

1. These *static* configurations are time-limited, but the user can extend the expiration date any number of times using the web interface. All attributes of these configurations, such as public/private key pairs used to encrypt the tunneled data and IP addresses, remain unchanged when extended.

eduVPN Profiles

eduVPN is "built on the concept of *profiles* that allow access control at the network level" [1]. There is a many-to-many relationship between users and *profiles* — each user can have access to zero or more *profiles*, and each *profile* can be used by zero or more users simultaneously. Each *profile* has several attributes assigned to it, most notably [14]:

- `profileId`, `displayName` — unique profile identifier, and user-friendly display name;
- `defaultGateway`, `routeList` — whether the profile should be used for *full tunneling*, and if not, what IP ranges to use for *split tunneling*; and
- `wRangeFour`, `wRangeSix`, `oRangeFour`, `oRangeSix` — from which IPv4 and IPv6 ranges (subnets) to assign IP addresses to connecting users via *WireGuard* and *OpenVPN* respectively.

This *profile* system hereby ensures that connected users can only generate network traffic from IP addresses belonging to *profiles* to which they have access to; therefore, firewalls on both *eduVPN Nodes* and servers hosting services for these users can match packets based on these source IP addresses, and consequently enforce network-level access controls.

1.2 Firewall

A *firewall* is "any device, software, or arrangement or equipment that limits network access" [15]. Firewalls inspect network traffic (packets) traversing through them and decide whether they will be allowed further or dropped according to an administrator-defined policy. Such decisions can be made based on the packets' contents, such as IP addresses and port numbers, and/or metadata about them, such as their originating network interface or association to a *flow* (e.g. an established TCP connection).

Additionally, some firewalls have the integrated ability to perform certain modifications of the packets' contents. When IP addresses and/or port numbers are replaced, the term NAT (Network Address Translation) is used to describe that functionality.

The `netfilter` framework is a component of the Linux kernel that "allows kernel modules to register callback functions at different locations of the Linux network stack" [16]. Such functions are registered, for example, by the `conntrack` connection tracking subsystem or by the `iptables` and `nftables` firewalling software packages. This allows devices running the Linux operating system to act as firewalls.

1.2.1 State and connection tracking

Firewalls can be described as either *stateless* or *stateful*. They are considered *stateless* if they only inspect packets individually, without maintaining any information about previously inspected packets. On the contrary, firewalls are *stateful* if they are able to identify that a packet belongs to a certain *conversation* or *flow* and leverage that information while assessing it.[2] For transport protocols with connection establishment and termination procedures (handshakes), such as TCP, identifying these *flows* is trivial, whereas for connectionless protocols, such as UDP, some heuristic needs to be used, for example: "UDP datagrams with the same IP addresses and port numbers within a certain time interval".

Within the Linux kernel, packet *flow* (connection) identification is implemented by the `nf_conntrack` family of modules. Packets are classified into the following states [17]:

- `NEW` — This packet is the first one of a tracked connection, e.g. a TCP SYN packet.
- `RELATED` — This packet is the first one of a tracked connection, and it is related to an already tracked connection in a certain manner, e.g. an ICMP error message pertaining to a tracked connection or the TCP SYN packet of a FTP data connection corresponding to an established FTP control connection.
- `INVALID` — This packet does not "follow the expected behavior of a connection" [17], e.g. a TCP SYN-ACK packet not associated to any preceding TCP SYN packet.

2. In some literature [15], *stateless* firewalls are referred to as *packet filters* and *stateful* firewalls as *circuit-level gateways*.

- `ESTABLISHED` — This packet is *not* the first one of a tracked connection; "this state is reached when the firewall has seen two-way communication" [17].

These states are then available for querying from within firewall rules, thereby making firewalls in Linux *stateful*. By allowing all packets classified as `ESTABLISHED` and `RELATED`, and by blocking all `INVALID` packets, it is possible to configure a firewall policy that effectively only needs to assess `NEW` packets. This results in the firewall being effective despite the possible large size of the overall ruleset.

1.2.2 nftables

Within the Linux kernel, two firewalling solutions exist: the older yet still maintained `iptables` and the newer, more flexible and performant [16] `nftables`. The software presented in Section 4.6 of this thesis uses `nftables` and depends on some of its features that `iptables` does not possess, such as the *generic set infrastructure* and JSON input/output. Therefore, `nftables` will be explained further in this section.

The kernel-space part of `nftables` consists of a "virtual machine [...] that is able to execute bytecode to inspect a network packet and make decisions on how that packet should be handled" [18]. From user-space, configuration can be performed using either the command-line `nft` tool or from any other program linking the `libnftables` library.

The overall policy of a `nftables` firewall is referred to as a ruleset. Rulesets consists of various types of objects, most notably [19]:

- **tables** — Top-level containers for chains and named sets and maps. Each table is associated with an address family, for example `ip` (IPv4-only), `ip6` (IPv6-only) or `inet` (both IPv4 and IPv6).

- **chains** — Containers for a sequence of rules. The following two types of chains exist:
  - **base chains** — Chains that are hooked to `netfilter`, and therefore entered to from a specific point of the network stack. These must have a `type` (e.g. `filter` or `nat`), `hook` (e.g. `prerouting`, `input`, `forward`, `output` or `postrouting`),

`priority` and `policy` (whether to accept or drop packets that reach the end of the chain) associated with them.
  - **regular chains** — Chains that need to be invoked explicitly through the use of a `jump` (or `goto`) verdict from within another chain (base or regular) or a verdict map. These can be "used for better rule organization" [19].

- **rules** — Rules specify that certain actions shall be performed for certain packets. The packets to which a rule will apply are specified using *match expressions*, each of which compares a certain aspect of packets to a value or a set, interval, ... thereof. As an example, IP addresses or port number contained within the packets or metadata, such as the `conntrack` state (see Section 1.2.1), can be used in the expressions. Actions are then specified using *statements*, such as verdicts (e.g. `accept` — allow the packet immediately without evaluating any further rules, `drop` — block the packet immediately, `jump` — continue by evaluating rules in some other chain) or NAT statements (used to perform Network Address Translation).

- **sets** — Containers with an associated type, such as `ipv4_addr`, `ipv6_addr` or `inet_service` (port numbers), which may contain elements of that type or intervals thereof. Sets can be either anonymous, in which case they are part of a certain rule and immutable, or named — then they are standalone and elements can be added and removed from them dynamically. Maps and verdict maps are derived from sets, and together they are referred to as *generic set infrastructure* [20]:
  - **maps** — Containers for key-value pairs, which may for example be useful in more complex NAT setups [21].
  - **verdict maps** — Containers that map a key (such as an IP address) to a verdict (such as `accept`, `drop` or `jump`) that should be taken for that key. This is useful for building more complex firewall rulesets that, for example, need to perform different actions for different (source or destination) IP addresses, while maintaining high performance[3].

3. Since the *generic set infrastructure* internally uses "performance data structures such as hashtables and red-black trees"[20].

2 Access control architecture

In the context of this thesis, the term *access controls* is used to describe the set of means that when orchestrated, ensure that users connected to eduVPN, hereinafter sometimes referred to as *clients*, are only granted access to network services that are necessary for carrying out their activities. In particular, *network-level access controls* is the part of access controls enforced by firewalls, i.e. on the level of the computer network; one other example of access controls within the eduVPN system are profiles, as described on page 6. An *access control architecture* then defines a specific way in which access controls are orchestrated to properly carry out their role in ensuring security.

In the current deployment of eduVPN at Masaryk University, the smallest access control unit is a profile, to which users are granted access to based on their affiliation with a certain faculty or department, and for each profile, firewall chains are created that filter network traffic from/to users of that profile. This implies that only groups of users can be targeted by network-level access controls, but not individual users. Since this coarse-grained approach does not strictly adhere to the Principle of Least Privilege, which states that "[...] every user of the system should operate using the least set of privileges necessary to complete the job" [22], in addition to some other limitations stated in Section 2.1, a need existed for a new approach.

The goal of this thesis is therefore to devise the details and then create a proof-of-concept software implementation of a new, fine-grained access control architecture that was outlined by the employees of the Infrastructure Department of the Institute of Computer Science (ICS) at Masaryk University (MU) who are responsible for administering MU's eduVPN deployment, which would eliminate the need of multiple eduVPN profiles, and which would use an unified profile for all users and then perform per-user network-level access controls directly on eduVPN Nodes. As a result of this thesis, it should be known whether such access control architecture is implementable, and if yes, whether it is suitable for further use within Masaryk University.

In Section 2.1, the current eduVPN deployment with particular focus on the coarse-grained access control architecture and its limitations is detailed, and in Section 2.2, the requirements for the new

access control architecture and its implementation are defined. The architecture of the proof-of-concept software implementation of that access control architecture is not the subject of this chapter, but rather of Chapter 3.

2.1 Current access control architecture

The access control architecture within the eduVPN system at Masaryk University which is currently deployed, as detailed in Section 2.1.3, is substantially dependent on both eduVPN profiles which are administratively categorized into so-called *scopes*, as described in Section 2.1.1, and proper firewalling of client IP subnets on both eduVPN Nodes and servers intending to provide or not provide network services to eduVPN clients, as described in Section 2.1.2. The limitations of this architecture are then listed in 2.1.4.

The information presented in the following sections is based on the internal, employee-only eduVPN Technical Documentation portal [23], personal discussions with Institute of Computer Science's employees, and on information gathered during the self-examination of software and configuration deployed on Masaryk University's eduVPN Portal and Node virtual machines, to which the author of this thesis has administrator-level access.

2.1.1 Scopes and profiles

Each profile that is configured within Masaryk University's eduVPN deployment belongs to exactly one of the following three *security scopes*, or simply *scopes* [24]:

- **Student**
- **Employee**
- **Other**

This division of eduVPN profiles is entirely administrative — it affects what IP subnets are assigned to the profiles, on which eduVPN Nodes they are deployed, users with which legal relationship with Masaryk University can be granted access to them, and when new profiles can be created, but there is no equivalent to a *scope* in eduVPN's built-in data or security model.

The addressing and firewalling constraints of the individual scopes are described in Section 2.1.2, the deployment ones in 2.1.3 and the administrative ones in the list that follows [24]:

- **Student** — Profiles to be used only by students of Masaryk University. One profile exists for each faculty (and an additional one for students without faculty affiliation), and no custom profiles can be requested. Users are liable for their own actions while using these profiles.

- **Employee** — Profiles to be used only by "employees of the Masaryk University [*sic*] and people with legal ties with MUNI" [24]. One profile exists for each faculty and institute, and custom profiles can be requested, but only for departments and teams and not for a specific purpose (like accessing a particular server). Users are liable for their own actions while using these profiles.

- **Other** — Profiles to be used by people who cannot be categorized as either *Students* or *Employees*, e.g. external researchers or "external companies managing on-premise equipment" [24]. No profiles in this scope implicitly exist — all of them need to be explicitly requested by an employee of Masaryk University who is then liable for all actions of the profile's users, and as such these profiles are supposed to be severely limited with regard to the set of services they are able to access. Unlike in the case of the *Employee* scope, profiles can be also created "for special purposes — accessing various equipment" [24], i.e. for use by machines rather than only humans.

Relation to identity management systems

An *Identity Management System*, abbreviated as *IdM*, is "responsible for ensuring the quality of identity information such as identifiers, credentials, and attributes and using it for authentication, authorization, and accounting processes" [25]. Within Masaryk University's eduVPN deployment, the Perun IdM along with the tightly integrated *MUNI Unified Login* SSO (Single Sign-On) service have an essential role in ensuring that eduVPN users are only able to connect via profiles to which they have been granted access [26].

Profiles which are configured in the eduVPN system have their equivalents in Perun. Each profile is bound to one or more Perun groups, and these groups contain users which shall have access to that profile in eduVPN [26]. These groups can either be managed manually or be automatically synchronized from other information systems (such as *workplace* groups containing all students or employees of a certain faculty or institute).

Users are required to use *MUNI Unified Login* when logging in to eduVPN (both via the web interface and the GUI application). Upon successful login, eduVPN is signaled to which profiles the user shall have access to; only these profiles are then shown to the user in the interface and permitted for establishing VPN connections [26]. As a result, users are limited to sending network traffic only from IP addresses associated with profiles accessible to them; this fact can then be used when configuring firewalls.

2.1.2 Addressing and firewalling

Each scope is associated with a pair of IP subnets — an IPv4 one and an IPv6 one — which serve as pools for assigning IP subnets to individual eduVPN profiles within that scope [27]:

- **Student** — `100.65.0.0/16`, `2001:718:801:900:100::/72`
- **Employee** — `100.67.0.0/16`, `2001:718:801:900:200::/72`
- **Other** — `100.72.0.0/16`, `2001:718:801:900:300::/72`

All profiles and clients are always assigned both IPv4 and IPv6 subnets/addresses, as eduVPN is an exclusively dual-stack VPN system [28]. IP subnets assigned to profiles do not overlap with each other and do not change; by contrast, users may be assigned different IP addresses by the eduVPN system (from the profiles' IP subnets) each time they connect. Since within Masaryk University's eduVPN deployment, profiles are always hosted on a pair of Nodes, their IP subnets are split between them as is further detailed in Section 2.1.3.

Firewall on eduVPN Nodes

The first part of the responsibility for enforcing network-level access controls lies with eduVPN Nodes. On these Nodes, all network traffic originating from eduVPN clients is decapsulated and forwarded

further into the network, and conversely, all traffic from the network destined towards eduVPN clients is encapsulated there. It is therefore essential to have a proper firewall configuration on eduVPN Nodes, as it guarantees that no undesirable traffic from eduVPN clients can reach the network and vice versa.

Within Masaryk University's eduVPN deployment, firewalling on Nodes is performed using `nftables`. A base chain with the `forward` hook captures all packets from or to VPN clients[1]; in this chain, all packets with the `ESTABLISHED` and `RELATED` connection tracking states are immediately accepted, and `INVALID` ones dropped. `NEW` packets are then assessed in profile-specific chains, to which the base chain jumps using verdict maps containing an IP subnet entry for each profile that is handled by the Node [29].

A pair of chains named `<profile-id>-input` and `<profile-id>-output` exists for filtering traffic destined towards and originating from both IPv4 and IPv6 subnets of each profile [29]. The default policy enforced by them, which is dependent on the profile's scope, is indicated in Table 2.1; this policy always applies to implicitly created profiles for faculties and institutes, whereas for custom profiles, a custom firewall configuration can be requested [24].

Table 2.1: Default firewall policy per scope and direction [29]

	Student	**Employee**	**Other**
Input	drop all	drop all	drop all
Output	accept all	accept all	drop all

For profiles in the *Other* scope, a custom configuration is necessary, as otherwise all traffic would be dropped in both directions [24]. It should be noted here that with appropriate configuration of the `<profile-id>-input` chain, client-to-client communication within eduVPN is possible, and some profiles in the *Other* scope make use of this functionality.

1. Except packets which directly originate from or are destined to an network interface of the client's eduVPN Node itself — these are instead handled by `input`- and `output`-hooked base chains, respectively. Since there is no requirement for the proposed solution to handle this special case, any further explanations and considerations thereof are omitted in this thesis.

Firewall on servers

The other part of the responsibility for enforcing network-level access controls lies with server administrators. No eduVPN access control architecture can entirely eliminate the need of proper firewall configuration on servers, as non-eduVPN network traffic also needs to be filtered. However, proper filtering of eduVPN traffic is no less essential in the currently deployed architecture, especially since there is no filtering of outbound client traffic from profiles in the *Student* and *Employee* scopes by default directly on eduVPN Nodes.

Based on the IP subnets assigned to profiles, network traffic from (or to) certain profiles can be targeted by firewall rules and therefore filtered. However, if targeting individual users is required, this coarse-grained profile-based access control architecture is not able to facilitate that; the only option is targeting a specific device of a user using a workaround whose description follows.

Static IP addresses

In the eduVPN system, it is not possible to assign a static IP address to a user in the strict sense, i.e. for an indefinite time period [30]. However, if a static WireGuard configuration is generated in the eduVPN web interface, it is assigned a pair of IP addresses (IPv4 and IPv6) which are bound to that configuration until it expires or is deleted [30]. If firewall rules on servers target these IP addresses, it is possible to allow traffic to a network service only from a specific user's device instead of an entire eduVPN profile.

However, the administrators of Masaryk University's eduVPN deployment consider this approach an anti-pattern and a workaround due to the following issues:

- Since each WireGuard configuration is intended to be used on a single device, targeting users directly instead of individual devices of theirs within firewall rules is not possible.

- In case the WireGuard configuration expires due to the user not periodically extending its validity in the eduVPN web interface [30], or if it is deleted either manually by the them or automatically when they lose access to the eduVPN system entirely (for

example when their employment contract is terminated), the IP addresses assigned to the configuration are released and may be reassigned to a different user of the same profile. As a result, if firewall configuration on all servers which were targeting these specific IP addresses are not manually adjusted, that different user will be able to access restricted network services despite them being not authorized to do so.

- Static OpenVPN configurations are not assigned IP addresses upon creation, but rather upon each connection, equally to how connections from within the eduVPN GUI application behave (both via WireGuard and OpenVPN) [31]. Consequently, this workaround is inaccessible to non-advanced users, who do not possess the necessary knowledge to work with static configurations, and in situations where using OpenVPN is a necessity.

NAT

Since eduVPN clients at Masaryk University are assigned IPv4 addresses from the IANA-reserved `100.64.0.0/10` subnet which is not globally routable [32], Network Address Translation (NAT) is performed on clients' packets on the edge of Masaryk University's network with CESNET[2] in order for them to be able to communicate with the public Internet. By default, the public IPv4 subnets used for this purpose are scope-specific. If IPv4 network traffic from a specific profile needs to be identifiable in the public Internet, a custom NAT subnet can be requested for it [30].

As for IPv6 network traffic, NAT is neither needed nor performed, as eduVPN clients are assigned publicly routable IPv6 addresses, by which their profiles can be distinguished within both Masaryk University's network and the public Internet.

2.1.3 Deployment

The current deployment of eduVPN at Masaryk University consists of a PostgreSQL database and 10 virtual machines which are depicted in Figure 2.1. These are [33]:

2. Czech Education and Scientific Network

- two Portal machines in a High-Availability setup with fail-over;
- two Node machines for each scope;
- a machine for generating and managing TLS certificates; and
- a monitoring machine.

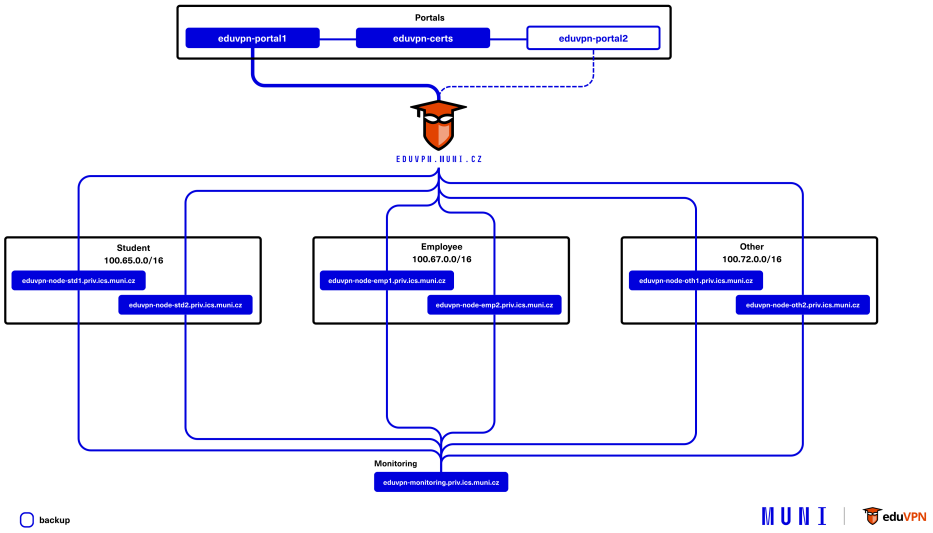

Figure 2.1: Current eduVPN deployment architecture [33]

When a user is connecting to this eduVPN deployment, their connection request reaches one of the Portal machines (`portal1` under normal operation) which then configures WireGuard or OpenVPN to allow that connection on one of the Node machines. The Portal usually selects the Node from the per-scope pair which is under the least load. Not all profiles in this deployment support OpenVPN — some are WireGuard-only, and OpenVPN support is planned to be phased out in favor of ProxyGuard.

The specific IP address that a client is assigned upon connection depends on the eduVPN profile, Node and the VPN protocol — each profile's IP subnet is split in half, each of which is assigned to each of its corresponding two Nodes, and if the profile supports OpenVPN, then the subnet is split in half once more on each node, and each half is assigned to each VPN protocol. For example, the `student-phil` profile

which is assigned the `100.65.0.0/19` IP subnet[3] as a whole [34], is split into four subnets as shown in Table 2.2.

Table 2.2: Per-node and per-protocol IP subnets of `student-phil` [34]

	WireGuard	**OpenVPN**
`node-std1`	`100.65.0.0/21`	`100.65.8.0/21`
`node-std2`	`100.65.16.0/21`	`100.65.24.0/21`

2.1.4 Limitations

As has been suggested on numerous occasions in the preceding sections, the current access control architecture has several limitations:

1. **Coarse-grainedness.** The current access control architecture is too coarse-grained in the sense that it only supports targeting eduVPN profiles in firewall configurations and not individual users. In addition, some groups of users cannot be targeted due to policy restrictions, such as arbitrary groups of students, since custom profiles may not be requested in the *Student* scope (see Section 2.1.1). Server administrators who require such fine-grained control have no alternatives but to use the unreliable and inconvenient workaround based on static WireGuard configurations described on page 15, which allows them to target individual users' devices, but still not individual users. Examples of hypothetical use cases in which more focused targeting in firewalls would be beneficial follow:

    - A lecturer would like to allow students of their course access to a certain network service from eduVPN so that they can use it at home for completing homework assignments. Since the policy at Masaryk University disallows the creation of custom profiles in the *Student* scope, they would have to allow access to the whole faculty (for which a profile implicitly exists). Such access may be excessively broad due to the restricted nature of the service, or insufficient

3. IPv6 is omitted from this explanation for simplicity.

on the contrary, for example in case the course is attended by students with no faculty affiliation.

- A team at Institute of Computer Science consists of employees who have distinct roles, e.g. software developers and server administrators. Server administrators need access to their servers using SSH, but not software developers. Since the policy at Masaryk University disallows the creation of custom profiles in the *Employee* scope for a specific purpose (like accessing a particular server), they can only request a single profile for their entire team from which everyone will be able to access the SSH service (on network level).

2. **Profile switching.** The eduVPN GUI application does not support multiple simultaneous active VPN connections on a single device, and therefore simultaneous access to multiple profiles[4]. This causes problems for users who have access to distinct network services from distinct eduVPN profiles and need to access them simultaneously, e.g. employees who work with multiple teams. Such users must then frequently switch profiles, which causes them inconveniences and inefficiencies in their workflows.

3. **Permissive firewall on eduVPN Nodes.** The default firewall configuration for eduVPN profiles in the *Student* and *Employee* scopes blocks all uninitiated incoming traffic to clients, but it allows all outbound traffic from clients (see Table 2.1). Therefore, while the firewall protects eduVPN clients from the network, it does not protect the network from the clients in any way.

4. **No central service database.** Currently, network-level access controls are primarily enforced by firewalls on individual servers and not on eduVPN Nodes, as Limitation 3 already suggested. While this decentralization of concerns may generally be considered beneficial due to the absence of a Single Point of Failure, one

4. Static configurations do not have this limitation per se, but they need to be configured in split tunnel mode with distinct tunneled IP subnets and imported into a native VPN client which supports such setups (not all of them do [24]). Consequently, they are accessible only to users with advanced understanding of computer networking and require non-trivial effort during setup and maintenance.

limitation of this approach is the absence of any central database of which network services exist, on which IP addresses and ports they are available and who is permitted to access them — the information about who is able to access which services is scattered in firewall configurations on all the individual servers which host them. This makes security auditing significantly more difficult for both server administrators (unless they keep their own records) and security teams.

5. **No profile self-management.** If the requestor of a custom eduVPN profile needs to change an attribute of their profile, including the set of IP addresses allowed or blocked by a profile's firewall on eduVPN Nodes, they cannot perform it by themselves; instead, the change must be facilitated by eduVPN administrators. This especially limits the flexibility of profiles in the *Other* scope, as their restrictive firewalls may not be rapidly adapted to new use cases by their requestors.

2.2 Proposed access control architecture

In this section, a new access control architecture which would resolve the limitations of the currently deployed architecture that were stated in Section 2.1.4 is proposed. Firstly, the requirements for the new architecture and its implementation are stated in Section 2.2.1. Secondly, in Section 2.2.2, the author proposes a set of changes to the current architecture that would allow the new architecture to be (gradually) adapted. Finally, Section 2.2.3 identifies certain regressions of the new access control architecture, i.e. undesirable traits of the new architecture that are not present in the currently deployed architecture.

2.2.1 Requirements

In general, the proposed access control architecture shall not be functionally inferior to the currently deployed one, and it shall resolve the limitations listed in Section 2.1.4. Based on these general requirements, the following specific ones have been identified:

1. **Unified profile.** The new access control architecture shall be designed around a unified (universal) eduVPN profile that can be safely made available to all eduVPN users within Masaryk University. From this single profile, they will be able to access all their permitted services without the need to switch profiles, hereby resolving Limitation 2 of the current architecture.

2. **Fine-grained, strict and dynamic firewall on eduVPN Nodes.** In order to address the security implications of the unified profile, a software shall be developed for automatic management of firewall configuration on eduVPN Nodes. This configuration shall be characterized by being:

   (a) **fine-grained**, i.e. targeting individual connected users rather than profiles, hereby resolving Limitation 1;

   (b) **strict**, i.e. blocking all undesired network traffic directly on eduVPN Nodes and therefore preventing it from reaching the network and target servers entirely, hereby resolving Limitation 3; and

   (c) **dynamic**, i.e. reacting to client connections or disconnections in a timely manner, and to changes in permissions which are assumed to be less frequent.

3. **Restricted and non-restricted users.** The new access control architecture shall have a notion of whether users are *restricted* or *non-restricted* and reflect this attribute when generating firewall configuration for them on eduVPN Nodes. *Restricted* users shall be granted network-level access only to explicitly allowed network services, whereas *non-restricted* users shall be granted access to the whole network except to disallowed services. This requirement originates from the fact (described in Section 2.1.1) that the current architecture permits users with no legal ties with Masaryk University to use only profiles in the *Other* scope, which have restrictive firewalls (see Table 2.1), whereas other users can use profiles in the *Student* and/or *Employee* scopes, which have permissive firewalls.

4. **Source-IP-based user group identification.** A significant trait of the current access control architecture is that network traffic from an eduVPN profile is identifiable by network services by source IP addresses. For any service that would still, for any reason, need to perform such source-IP-based identification, a mechanism shall exist in the new architecture that will ensure that network traffic from clients with granted access to that service will come from a defined dedicated IP subnet.

5. **Client-to-client communication.** The firewall generated on eduVPN Nodes shall support allowing network communication between eduVPN clients, taking into account that different clients may be connected to different Nodes (see Requirement 7).

6. **Central source-of-truth database.** For the purpose of eliminating Limitation 4, a database shall be designed that will serve as a central *source of truth* for firewall configuration — it shall contain all the information necessary to generate firewall configuration on eduVPN Nodes from scratch in accordance with Requirements 2, 3, 4 and 5. Furthermore, the following database-related use cases shall be taken into account:

(a) **Debugging & auditing.** In addition to containing the necessary information to generate firewall configurations, the database shall be able to store information that would aid debugging and security auditing, such as contact e-mail addresses.

(b) **Proof-of-concept synchronization.** The proof-of-concept software implementation shall contain a component which will prove that the database can be synchronized with some type of external data source containing permission information. In case of a potential future production deployment, such source could be an Identity Management System (IdM), where network service administrators would be able to manage attributes of and access to their services, hereby eliminating Limitation 5. However, integration with a specific IdM is out of scope of this thesis.

7. **Multi-machine support.** The software implementing the new access control architecture shall be able to operate within a multi-Portal and multi-Node eduVPN deployment similar to the current one at Masaryk University, as described in Section 2.1.3.

8. **Compatibility with the current access control architecture.** The proposed access control architecture and its implementation shall be designed in a way in which it does not interfere with the simultaneous deployment of the current architecture. As a result, it will be possible to deploy the new architecture gradually.

2.2.2 Modifications to the current architecture

As implied by Requirement 8, the adoption of the new access control architecture shall not require the complete decommissioning of the current architecture. For this reason, the adoption process shall comprise of a set of additions and modifications to the current architecture, as described in Section 2.1, instead of a full redesign thereof.

In accordance with the focus of this thesis on a proof-of-concept solution, the additions and modifications described in this section shall not be reflected in the production deployment of eduVPN at

Masaryk University — it is fully sufficient to only propose and describe them.

Scopes and profiles

Requirement 1 states that a new unified eduVPN profile which will be accessible to all eduVPN users shall be created. Since within the current access control architecture, all profiles are placed into exactly one scope associated with only a certain group of users — students, employees or others (see Section 2.1.1) — it is argued that the new, universally-accessible profile shall exist outside of this established system of scopes.

The profile shall be assigned an IPv4 and IPv6 subnets that do not overlap with these assigned to the scopes. This ensures that server administrators who target the IP subnets of the current scopes or profiles in firewalls will not be required to make any configuration changes upon deployment of the new access control architecture. In addition, administrators will not be forced to adopt the new architecture immediately upon its deployment — during the period of simultaneous deployment of both architectures, they may prepare their services for the new architecture and only after that allow access to them from the unified profile by allowing network traffic from the non-overlapping, newly-assigned IP subnets.

Generally, this above-described separation of the current scopes, profiles and their IP subnets from the new ones enables deployment of the new architecture without affecting the old one, supporting Requirement 8.

Firewalls

The firewalls on eduVPN Nodes have a significant role in the overall enforcement of the new access control architecture, as suggested by Requirement 2. Furthermore, these firewalls are also required to perform NAT (Network Address Translation) in order for the new architecture to be compliant with Requirement 4. Since these firewalls are generated dynamically by a software component, their configuration structure and other characteristics are described in detail as part of the software architecture, in Section 3.2.

On the contrary, firewalls on servers providing services to eduVPN clients have less significant role in the new access control architecture than in the current one. When adapting their servers to the new architecture, administrators will be required to allow access to network services from the IP subnets assigned to the unified profile. However, since all users have access to this profile, the administrators shall register the service in the central database (see Requirement 6) beforehand, which will cause access to it to be limited in firewalls on eduVPN Nodes. In case permissions in the database change, no modifications to the configuration of server firewalls are needed — the firewalls on eduVPN Nodes will be automatically reconfigured instead. As the IP subnets assigned to the unified profile are separate from those assigned to the current profiles, allowing or not allowing access to the unified profile in server firewalls will not affect access from the old profiles, indicating compliance with Requirement 8.

In general, it can be stated that the new access control architecture shifts focus from server firewalls to eduVPN Node firewalls. In the current architecture, access to a service was permitted to a group of users, i.e. a profile, by allowing the profile's IP subnets in the server firewall, while the eduVPN Node firewalls only played a significant role with profiles in the *Other* scope (see Table 2.1). In the new architecture, on the contrary, user access to a service is enforced by firewalls on eduVPN Nodes, whereas server firewalls are only able to allow or disallow access from the entire unified profile.

Deployment

The first modification of the current eduVPN deployment at Masaryk University, as described in Section 2.1.3, which will be necessary to accommodate the new access control architecture will need to occur on the pair of eduVPN Portal machines. On these machines, client connection and disconnection requests are processed — as the new access control architecture is required to dynamically modify eduVPN Node firewall configuration on user connections and disconnections due to Requirement 2c, a software component will need to hook these events and inform other components about them occurring. This software component, which is described in Section 4.5, will need to be

deployed to Portal machines and appropriately integrated into the eduVPN Portal software.

Then, eduVPN Nodes for handling network traffic from users connected to eduVPN using the universal profile will need to be deployed. Since, as has been stated previously, the universal profile is unsuitable for inclusion in any of the scopes on which the current access control architecture is based (see Section 2.1.1), it is argued that a new set of eduVPN Node virtual machines shall be created specifically for the universal profile. On these Nodes, a software component for the automatic management of firewall configuration in accordance with Requirement 2 shall be deployed, whereas the currently deployed per-scope Nodes will not be affected in any way, supporting Requirement 8. This software component is described in Section 4.6, and it is designed for simultaneous deployment on an arbitrary number of Nodes in accordance with Requirement 7.

2.2.3 Regressions

Even though it has been shown that the proposed access control architecture resolves the limitations of the current architecture described in Section 2.1.4, the author has identified the following undesirable trait of the new architecture that is not present in the current architecture.

Centralization — a single point of failure

The first regression that has been identified is the overall centralization of the new architecture's security model on firewalls on eduVPN Nodes.

The security model of the current architecture is formed by a triad of factors: the profiles, which are made accessible to only certain groups of users by the eduVPN Portal software, firewalls on servers, which only permit access from a certain set of profiles, and firewalls on eduVPN Nodes, which despite being permissive in the outbound direction (see Limitation 3) still function to protect the clients in the inbound direction (see Table 2.1). This separation of concerns forms the basis of a balanced security model, where the impact of a failure of a single component may be limited by the other components.

On the contrary, in the proposed architecture, the unified profile is available to all eduVPN users, and network traffic from all clients using the unified profile profile must be allowed in firewalls on servers in order to make the hosted services available from that profile[5]. Consequently, all the responsibility for ensuring that only desired users will be able to access the hosted services lies with the firewalls on eduVPN Nodes.

Additionally, the configuration of these firewalls is managed automatically by a software component, whose failure may cause the overall failure of the entire security model of the proposed access control architecture. However, it is argued that since a core requirement of the new architecture is a fine-grained filtering of network traffic on a per-user basis in firewalls (Requirement 2a), its direct integration into the eduVPN system is justified as this system contains information about the assignment of IP addresses to users, which is necessary for configuring such firewalls.

It is therefore concluded that if the proof-of-concept software solution presented in Chapters 3 and 4 was to be transformed to a production-ready solution, a thorough security audit of the software shall be carried out.

5. If these services utilize the functionality described in Requirement 4, access shall be allowed only from the dedicated IP subnets instead of the entire IP subnets of the universal profile. However, since this custom NAT is also performed on eduVPN Nodes — see Section 3.2 — this does not meaningfully increase the overall security.

3 Software architecture

The new access control architecture proposed in Section 2.2 requires custom software in order to be functional in its entirety. This fact has been noted on numerous occasions in the aforementioned section; to be more specific, custom software components are required for performing:

- the synchronization of the central database with external data sources (Requirement 6b);
- the processing of client connection and disconnection events (Requirement 2c); and
- the automatic reconfiguration of firewalls on eduVPN Nodes (Requirement 2).

The custom software components along with the database, which were designed and implemented by the author of this thesis and are considered part thereof (see Appendix A), have been collectively named **edufirewall**[1]. The software is implemented in the **Python** programming language with type hinting, blends both procedural and object-oriented paradigms and is available as open-source under the **3-clause BSD License** (also referred to as *BSD License 2.0*).

The software has been programmed with emphasis on correctness rather than performance in order to address the security implications described in Section 2.2.3. While the scope of this thesis is only the creation of a proof-of-concept software solution, its transformation into a production-ready solution may follow in the future provided that the software will be assessed as capable for this purpose by the people in charge at the Institute of Computer Science, Masaryk University.

In this chapter, architectural aspects of the custom software are detailed. Firstly, in Section 3.1, the schema and other characteristics of the central database, which has an essential role in the software both for data storage and as an real-time communication mechanism, are described. Then, the separation of the implemented software system into individual components is outlined in Section 3.3. Finally, the

1. A portmanteau of eduVPN and firewall.

structure of the firewall ruleset generated and automatically managed by this software is covered in Section 3.2. Implementational aspects are not subject of this chapter, but rather of Chapter 4.

3.1 Database architecture

The custom software is substantially dependent on a central database, which, as outlined in Requirement 6, serves as a source of truth for generating firewall configuration on eduVPN Nodes. For designing the database, the **relational model** has been used, and for implementing it, the **PostgreSQL** RDBMS has been chosen.

This choice has been made for the reason of consistency with the current deployment of eduVPN at Masaryk University. As has been mentioned in Section 2.1.3, the database of the eduVPN system itself is currently hosted using PostgreSQL, and therefore, it was logical for this software's database to be implemented using this RDBMS as well.

Furthermore, the choice is supported by certain features of PostgreSQL that are beneficial for the implemented software. Firstly, PostgreSQL uses the client-server model [35], which is necessary so as to support the software's ability to be distributed across multiple machines, as mandated by Requirement 7. Secondly, the notification mechanism built in to PostgreSQL proved to be invaluable for the software, as described in Section 3.1.2.

3.1.1 Schema

A series of manually-created migrations, i.e. files containing SQL commands, is distributed with the source code, which, when applied sequentially in a PostgreSQL database, forms the database schema shown in the Entity Relationship Diagram (ERD) in Figure 3.1.

The `efw_users` and `efw_services` tables contain users and services, respectively, and each row in the `efw_users_services` table represents that a certain user is allowed access to a certain service. Stable identifiers of users and services are essential for reliable synchronization with an external source (Requirement 6b), and they are therefore not auto-generated as in the other tables.

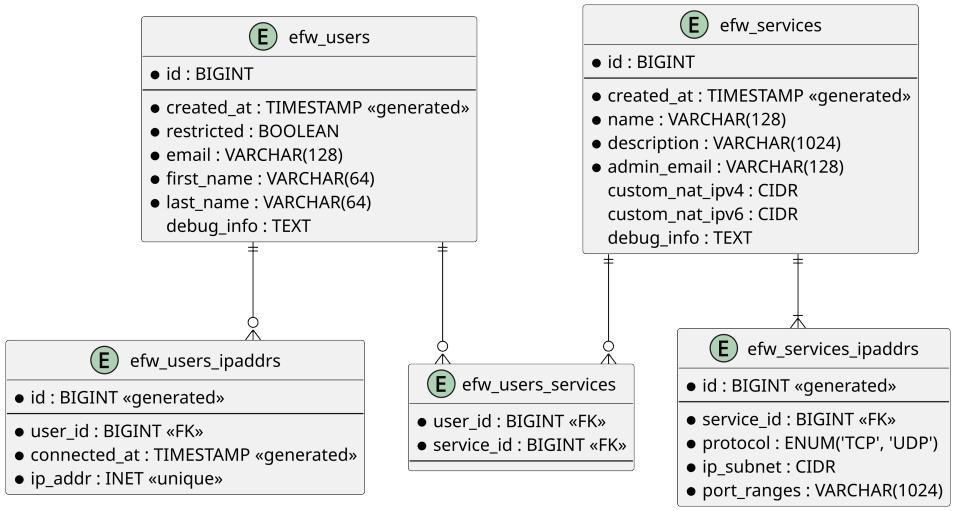

Figure 3.1: edufirewall database schema

The `restricted` attribute of users contributes to the fulfillment of Requirement 3, and the `custom_nat_*` pair of attributes of services does likewise for Requirement 4. The name, description, email, timestamp and debug info attributes are not strictly necessary for firewall generation; instead, they exist for auditing and debugging purposes, as per Requirement 6a; the `debug_info` attribute is not interpreted by the software in any way and may contain arbitrary textual information.

Each row in the `efw_users_ipaddrs` table represents that a user is currently connected to eduVPN under a specific IP address, and a so-called **access point** of a service is represented by each row in the `efw_services_ipaddrs` table. Each access point implies that a network service is accessible on all IP addresses within a certain IP subnet, using a certain transport protocol (TCP or UDP) and via a set of port numbers. This set of port numbers is represented as a string containing comma-delimited port numbers or ranges thereof, such as:

- `22`
- `80,443`
- `23,25,1024-65535`

3.1.2 Notifications

Due to Requirement 2c, a real-time communication mechanism for notifying instances of the firewall management software component running on eduVPN Nodes about database changes needed to be implemented. While the initial software design envisaged the usage of dedicated network sockets with a custom TCP/TLS-based protocol for this purpose, the PostgreSQL `NOTIFY/LISTEN` mechanism emerged as a superior solution for this use case due to its integration into the already designed database and ease of both implementation and deployment.

The notification mechanism is based on a pair of non-standard SQL commands — `NOTIFY` and `LISTEN`. By using the `LISTEN` command, a database client registers its desire to receive asynchronous notifications on a given channel. If any client uses the `NOTIFY` command (or the `pg_notify()` function) within a transaction, then when that transaction is successfully committed, all clients currently listening on the given channel are delivered the asynchronous notification. Each notification may carry a string payload, and the notification may be sent either manually or automatically from database triggers configured to react to modifications of certain tables [36].

Within the *edufirewall* software system, the following two types of notifications are used, each delivered via a separate channel:

- **`efw_connection_change_event`** — Triggered on insertions and deletions of rows in the `efw_users_ipaddrs` table, representing client connections and disconnections, respectively. The payload string of each notification encodes the affected user ID and IP addresses, enabling firewall configuration adjustments immediately upon notification delivery, without the need of any further database queries.

- **`efw_database_change_event`** — Triggered on any modifications of any other tables (see Figure 3.1). Compared to user connections and disconnections, these modifications are expected to be significantly less frequent and their reflection in firewall configuration is not as time-critical. Therefore, these notifications carry no payload strings, and their receival shall trigger a full database reload and firewall reconfiguration.

Since these notifications are sent automatically from within database triggers, no explicit cooperation of any software modifying the database is needed in this regard, which would not be the case if the aforementioned socket-based notification mechanism was employed instead. The eduVPN Node firewall management component (described in Section 4.6) listens on both of the aforementioned channels simultaneously, and takes appropriate action when notifications are received.

3.2 Firewall architecture

As has been previously mentioned on numerous occasions, firewalls on eduVPN Nodes have an essential role in enforcing the new access control architecture. Equally to eduVPN Node firewalls within the current architecture, the `nftables` firewalling software for Linux, whose capabilities have been outlined in Section 1.2.2, is utilized in the new architecture.

The firewall configuration dynamically managed by the software component described in Section 4.6 is designed to integrate into an already configured firewall ruleset rather than to entirely replace it. In other words, it is designed in such a manner that the software dynamically managing is never forced to flush the entire ruleset. This stems partly from the limited scope of the dynamically generated firewall, which is only concerned with firewalling eduVPN clients and not, for example, with filtering inbound traffic to network services hosted directly on eduVPN Nodes (such as WireGuard and/or OpenVPN servers), and partly due to the possible presence of other software relying on custom firewall configuration on the Nodes (such as containerization engines). This design decision is reflected in the names of all firewall structures (chains and verdict maps) managed by *edu-firewall*. These names are always prefixed with "`efw_`", implying that structures without this name prefix may be safely configured by the administrator — the software will never affect them in any manner.

3.2.1 Optimizations

Even though the performance of the proof-of-concept software was not the primary concern while implementing it, as mentioned in Chap-

ter 3, this does not hold true for the firewall. Its structure, as detailed in Sections 3.2.2 and 3.2.3, was designed with regard to several optimizations.

The first optimization is based on the expectation that the number of users simultaneously connected to eduVPN is likely to be significantly lower than the total number of users with access to the eduVPN system [37]. For this reason, per-user chains are only generated for users who are connected from at least one device, i.e. under at least one IP address. If the user disconnects their last device, their per-user chains are removed from the firewall and are regenerated in case of reconnection. Consequently, the number of per-user chains is linearly dependent on the number of connected users rather than total number of users present in the database, which contributes to vastly smaller sizes of firewall rulesets on eduVPN Nodes.

Another factor contributing to less extensive rulesets, but also to a less complex design of the software generating them, is that in a multi-Node eduVPN deployment, firewall on each Node is configured independently of other Nodes, and that only clients, or rather their IP addresses, connected to a particular Node are taken into consideration. As Table 2.2 illustrated, IP subnets of profiles, including the unified profile, are split between eduVPN Nodes — the dynamic firewall configuration process on each Node is aware of this split, and only considers clients whose IP addresses are from the parts of the IP subnets handled by that Node. As a result, if a user is connected only to a certain Node, all the other Nodes will not have per-user chains for that user configured within their firewall rulesets.

Then, per-user chains could be empty if the user in question had access to no services. Therefore, another ruleset size optimization which the software performs is that if it detects that an empty chain would be generated, such chain will not be generated.

Moreover, the number of rules in each chain is reduced by splitting the chain by IP version, i.e. instead of a single chain, two chains are generated — one for only IPv4 service access points and one for IPv6 ones. This separation is supported by the fact that verdict maps, which often contain jump verdicts to these chains, cannot simultaneously contain both IPv4 and IPv6 addresses [20]. It should be noted here that as a result of this optimization, the empty chain optimization

described in the previous paragraph is performed for each IP version separately.

Finally, perhaps the most impactful performance optimization results from the generated firewall only assessing the first packet of each new network connections, i.e. packets in the `NEW` conntrack state. Consequently, the speed of already established connections is not reduced regardless of the size of the generated firewall ruleset. This optimization is already used within the current access control architecture, as described in Section 2.1.2.

3.2.2 Filtering

The filtering of packets originating from or destined to eduVPN clients begins in a `nftables` base chain with the `forward` hook, which must be present in a table with the `inet` family, since eduVPN is exclusively a dual-stack VPN system [28]. This chain is not managed automatically by the software, but rather manually by the administrator. Firstly, this chain shall contain a rule to accept all packets in the `ESTABLISHED` and `RELATED` conntrack states, as well as drop those in the `INVALID` state (see the optimization described above).

Then, after possibly adding rules to for example allow ICMP Echo Request (Ping) or DNS packets, the administrator shall define rules jumping to certain *edufirewall*-managed chains based on the packet's input and output network interfaces. There are three such chains, two of which are the consequence of Requirement 5:

- **`efw_this2this_filter`** — Filters `NEW` packets from eduVPN clients connected to this Node to other clients connected to the same Node, i.e. packets both originating from and destined to VPN network interfaces.

- **`efw_othr2this_filter`** — Filters `NEW` packets from clients connected to other Nodes (in a multi-Node deployment) to clients of this Node, i.e. packets originating from a physical network interface and destined to a VPN interface.

- **`efw_this2inet_filter`** — Filters `NEW` packets from clients connected to this Node to either clients of other Nodes or to the

outside network (including the entire Internet), i.e. packets originating from a VPN network interface and destined to a physical interface.

Each of these chains is used to allow network-level access to a particular type of service access points (see Section 3.1.1). The type of each access point is determined by its IP subnet:

- **this-Node**, if the IP subnet contains exactly one IP address that belongs to the part of an IP subnet assigned to the unified profile which is declared to be handled by the eduVPN Node on which the firewall is generated;
- **other-Node**, if the IP subnet contains exactly one IP address that belongs to another part of an IP subnet assigned to the unified profile than that declared to be handled by the eduVPN Node on which the firewall is generated; and
- **Internet** or **non-unified** otherwise, i.e. if the IP subnet (which may contain more than one IP address in this case) does not overlap with any IP subnets assigned to the unified profile.

It should be noted here that the client-to-client communication within the unified eduVPN profile, which is mandated by Requirement 5 and facilitated by *this-Node* and *other-Node* service access points, is realistically possible only if the accessed client is assigned a static IP address[2]. Given that assigning *truly* static IP addresses within eduVPN is not possible, as detailed on page 15, this requirement can be considered problematic.

The `efw_this2this_filter` chain

The `efw_this2this_filter` chain is configured by the software to contain three rules — the first rule applies a verdict map to the source IPv4 address of a connected client's packet and jumps to a per-user IPv4 chain based on it, the second rule does likewise for IPv6, and the last rule drops all remaining packets.

2. Broadcast- or multicast-based IP address auto-discovery mechanisms, such as multicast DNS, are not functional within eduVPN as it operates on ISO/OSI Layer 3.

The per-user chains then contain rules accepting packets destined towards *this-Node* access points of services which the user is permitted to access. Finally, a rule to drop all remaining packets is added to the end of the chain regardless of the *restricted* status of the user (see Requirement 3) — client-to-client traffic needs to be allowed explicitly in all cases.

The `efw_othr2this_filter` chain

As opposed to the previously described firewall chain, the `efw_othr2this_filter` chain is not configured with rules consulting source IP addresses with verdict maps containing IP addresses of currently connected clients. This is because these IP addresses belong to clients connected to other Nodes, which are ignored during the firewall generation process, as stated in Section 3.2.1. Instead, the rules only check whether the packets' source IP address belongs to an IP subnet handled by another Node, and if so, jump to a per-IP-version chain. Once again, the last rule of the chain drops all remaining packets.

As the previous paragraph implies, there are no per-user chains, but rather only a pair of per-IP-version chains, which are furthermore not generated at all if the software is configured to run in a single-Node deployment. These chains accept packets destined towards all *this-Node* service access points in the database, and drop the others.

As can be inferred, no checks are in place in this chain that would validate whether a particular user (connected to another Node) is allowed to access a particular service (hosted by a client of this Node). However, security is not impacted, as these checks are performed in the `efw_this2inet_filter` chain on the other Nodes.

The `efw_this2inet_filter` chain

The basic structure of the `efw_this2inet_filter` chain is identical to the structure of the `efw_this2this_filter` chain. However, what differs are the sequences of rules in per-user chains, which depend on whether the user is restricted or not.

If the user is restricted, the rules first accept packets destined towards allowed *other-Node* service access points, then packets destined

towards allowed *Internet* access points, and finally drop all remaining packets. Otherwise, i.e. if the user is non-restricted, the rules:

- accept packets destined towards *other-Node* service access points to which the user has permitted access;
- drop all packets destined towards all the IP subnets assigned to the unified eduVPN profile, hereby ensuring that no other clients than those explicitly allowed can be reached;
- accept packets destined towards *Internet* service access points to which the user has permitted access;
- drop all packets destined towards all the *Internet* service access point IP subnets present in the database; and
- accept all remaining packets, i.e. packets destined towards *Internet* IP subnets/addresses not registered within any service access point in the database.

3.2.3 NAT

As explained on page 16 within Section 2.1.2, NAT of eduVPN clients' private IPv4 addresses in the current access control architecture is occurring on the edge of Masaryk University's network. In the new architecture, it is argued that a public IPv4 subnet separate from those used by the current architecture shall be used for performing NAT of clients of the unified profile, as per Requirement 8.

As mandated by Requirement 4, services that require to be accessed from a defined source IP subnet (different from those assigned to the unified profile) must be supported in the new architecture. This requirement is implemented by performing custom NAT on eduVPN Nodes — the software component responsible for dynamically configuring the firewall supports generating rules for this purpose. As indicated in Figure 3.1, each service may define a pair of IP subnets for performing this custom NAT — one IPv4 and one IPv6.

The custom NAT is carried out through the automatically managed `efw_this2inet_nat` chain, which shall be jumped to from a administrator-managed `postrouting`-hooked base chain of type `nat`. The jump shall occur for packets which have been routed to a physical network interface — services with at least one non-*Internet* access point shall not have a custom NAT subnet defined in the database. Similar to the `efw_this2inet_filter` chain, this chain contains verdict-map-

based jumps to per-user and per-IP-version chains; these chains then contain a source NAT (`snat to`) rule for each *Internet* access point of each service with a custom NAT subnet defined in the database allowed for that user. Unlike the filter chains described in Section 3.2.2 which were designed to always reach a terminal verdict (`accept` or `drop`), this NAT chain is designed to never reach a verdict, i.e. it always returns after it is jumped to.

This mechanism ultimately ensures that all packets from eduVPN clients to services which have a custom NAT subnet defined in the database will originate from IP addresses from that subnet. It should be noted here that in order for two-way traffic to be functional, the custom NAT subnet must be routed to eduVPN Nodes — the administrator is responsible for ensuring that. While this is sufficiently straightforward in single-Node eduVPN deployments, the situation becomes more complex in multi-Node deployments, as a distinct IP subnet needs to be routed to each Node.

In order to facilitate that, the administrator is required to split the database-defined custom NAT subnets into smaller ones and route each such subnet to each Node. Then, the configuration of the firewall management software component on each Node provides a dictionary for mapping the database-defined subnets to their smaller parts, which will then be used within the generated firewall rules. This dictionary also doubles as an allowlist for the custom NAT subnets.

3.3 Component architecture

As has been frequently and repeatedly mentioned in the preceding sections of this thesis, the *edufirewall* software system is collectively formed by several components. Each component is constituted by a Python package which provides a well-defined piece of functionality and which is to be deployed on a well-defined part of the eduVPN infrastructure. All the components are detailed separately in Chapter 4 — this section solely provides a list of them.

Firstly, two **library** components, i.e. Python packages which provide shared functionality which is imported and used by all the program components listed below, are implemented:

- `efw_baselib` — Base Library (Section 4.2)

- `efw_datalib` — Data Access Library (Section 4.3)

And secondly, three **program** components, i.e. Python packages with defined execution entrypoints, are available for use:

- `efw_idmhook` — Identity Manager Hook (Section 4.4)
- `efw_vpnhook` — eduVPN Hook (Section 4.5)
- `efw_nftablesd` — nftables Configuration Daemon (Section 4.6)

4 Implementation

In this chapter, the proof-of-concept software implementation of the new access control architecture proposed in Section 2.2 is detailed on a per-component basis. Firstly, elements common to more than one component are covered in Section 4.1, and then, each component as listed in Section 3.3 is elaborated upon in an independent section.

It should be noted here that this chapter describes the implementation in a somewhat abstract manner, since specific low-level implementation details are instead described in the source code itself, in the form of documentation strings or code comments. Usage and/or deployment guides for administrators are then provided in the form of per-component `README.md` files — see Appendix A.

4.1 Common modules

In more than one component of *edufirewall*, certain Python modules or concepts which serve an identical purpose are present. This section outlines these modules and concepts.

4.1.1 Exception hierarchy

Throughout the entire *edufirewall* software system, error handling is based on a custom-defined hierarchy of exception classes. `EdufirewallError` is the base class of this hierarchy. Each component then defines a subclass of this class (e.g. `VPNHookError` for `efw_vpnhook`), which in turn serves as the base class for all concrete exception classes raised[1] from that component. All these exception classes, both base and concrete, are defined the `exc.py` module which is separately present in every component.

Within the source code of the components, the convention of handling all error conditions before they reach the public interface of a class or module is followed — the errors are then reported by raising exceptions from the above-described hierarchy, which all hold a human-readable diagnostic message. If an exception outside of this

1. The Python-specific term of raising exceptions rather than throwing them is used in this thesis.

hierarchy crosses the public interface boundary, it is considered a bug. This includes instances of `AssertionError`, which may be caused by the `assert` statements often used within the code. These conventions proved to be of great assistance when striving for the correctness of the software system.

4.1.2 Configuration

Deployment-specific configuration parameters that each of the program components requires in order to properly function are provided to it in the `config.py` module. This module is not shipped as-is with the source code — instead, the `config.example.py` module is provided, which contains detailed configuration instructions for administrators in the form of code comments. This example module is intended to be copied to `config.py`, edited to fit the deployment's needs and deployed as part of the program's Python package.

Each configuration module is separated into the following triad of logical sections:

- **Logging** — Throughout the *edufirewall* software system, logging is based on the built-in Python `logging` module — this module shall be configured in this section.
- **Database** — All program components are extensively reliant on the database described in Section 3.1. The necessary credentials and options for accessing it are provided using this section.
- **Program-specific options** — Certain configuration options are only relevant for a single program component. As an example, the IP subnets assigned to the unified eduVPN profile are provided to `efw_nftablesd` in this section.

4.1.3 Program initialization

The `main.py` module contains the first code that is executed when an *edufirewall* program component is launched. This code implements certain steps which are necessary to initialize the program before its operation; while the exact steps vary from program to program, in general, the module notably:

- imports the `config.py` module and validates the configuration options defined in it;
- parses command-line arguments;
- starts an `asyncio` event loop;
- tests connection to the database;
- registers handlers for POSIX signals;
- instantiates *tasks* and hands over control to the *task manager* (see Section 4.2.1); while
- providing top-level error handling for the program, logging any raised exceptions before terminating it.

4.2 Base Library

The *edufirewall Base Library* component, packaged as `efw_baselib`, provides various small and isolated pieces of shared functionality which are imported and used by the other components. This includes definitions of shared type aliases for IP addresses and subnets, lightweight wrappers for logging and POSIX signal handling, and also the *task* abstraction, which is further described in the following subsection due to its role in the overall architecture of the implementation.

4.2.1 Tasks and their management

The `asyncio` library enables the development of concurrent programs in Python. It allows the declaration of coroutines using the `async def` syntax, which are essentially functions whose execution may be suspended when the `await` keyword is used within their body to wait for the completion of an asynchronous operation. Coroutines may then be wrapped in `Task` objects and scheduled for execution on an event loop. When a task is suspended, a different one may be resumed by the event loop, thereby enabling cooperative multitasking within the program. In addition, suspended tasks may be canceled and may communicate using synchronization primitives such as queues [38, 39, 40]. This library is of great assistance when implementing event-driven programs such as `efw_nftablesd` (described in Section 4.6), which needs to wait for database notifications a pair of channels

simultaneously. As a result, the *edufirewall* software system utilizes the `asyncio` library extensively.

The `efw_baselib` library provides two abstractions for the primitives outlined in the previous paragraph: the `Task` base class[2] and the `TaskManager` class. Each subclass of `Task` contains the implementation of an asynchronous algorithm which is to be scheduled on the event loop and then managed by `TaskManager`.

More specifically, `TaskManager` schedules all given tasks on the event loop, monitors them for potential error conditions (indicated by raised exceptions) and then ensures their proper termination by `await`-ing all of them, regardless of whether any other tasks have previously raised exceptions or not. `TaskManager` guarantees that all tasks are given the opportunity to perform necessary cleanup actions, such as disconnecting from the database, in all cases. Furthermore, `TaskManager` simultaneously waits for a POSIX signal indicating a request to the program to terminate, canceling (and properly `await`-ing) all still running tasks upon its receival.

4.3 Data Access Library

The *edufirewall Data Access Library*, packaged as `efw_datalib`, constitutes the data access layer of the *edufirewall* software system. The primary purpose of the library is to provide a programming interface for retrieving and storing data in the software system's PostgreSQL database, which was described in Section 3.1, as well as provide an object-oriented representation of that data to be used within the program components. Furthermore, `efw_datalib` provides a pair of utility modules which operate on that data.

4.3.1 Data access and representation

The `DatabaseAccessor` class provides methods for establishing database sessions for specific purposes — reading data, reading and writing data, and receiving notifications. These methods manage connections to the database, set up locks and transactions appropriately for a particular purpose, and return instances of so-called *handle* classes,

2. Not to be confused with the `asyncio.Task` class provided by the `asyncio` library.

for example `DBReadHandle` or `DBWriteHandle`. The methods of these classes may then be used to manipulate the data in the database.

The retrieved or to-be-stored data are represented by instances of so-called *container* classes. Such class exists for every entity present in the database schema (see Figure 3.1), as well as for both types of database notifications (see Section 3.1.2). In certain cases, two container classes are implemented for a single database entity — one for synchronization with external sources, and one for generating firewall rulesets. This relates to Requirement 6a, which forces the database to be able to store certain auditing and diagnostic data that are, however, unnecessary while generating firewall rulesets.

4.3.2 Utility modules

The first utility module, `port_ranges.py`, provides a function for validating and parsing *port ranges strings*, as described in the end of Section 3.1.1, into an object-oriented canonical internal representation. This representation can be reliably compared for equality, which is crucial for performing synchronization of the database with external sources. The raw strings may not be compared in this manner, as different strings may represent the same set of ports, for example `22-25` and `24-25,22-23`. Conversely, a function for converting this internal representation back to a string is implemented in this module.

The second utility module, `event_queue.py`, provides a custom priority queue for received database notifications. This queue prioritizes *database change events* over *connection change events*, as well as silently ignores any event insertions while the former event is present in it. This behavior stems from the fact that a *database change event* is supposed to cause a full database reload, and since such reload will also necessarily involve all the changes represented by the ignored events, processing such events would be redundant.

4.4 Identity Manager Hook

The *edufirewall Identity Manager Hook*, packaged as `efw_idmhook`, is a program which facilitates the synchronization of the *edufirewall* database with an external data source. More specifically, it synchro-

nizes information about users, services, service access points and so-called *bindings*, which represent that a particular user shall have access to a particular service, into the database. The only type of information which is stored by the database but not synchronized by this program are IP addresses of users (see Figure 3.1) — these are instead managed by the *eduVPN Hook* program described in Section 4.5.

Each time the program is launched, it sequentially performs the following actions:

1. It loads and validates the complete intended state of data from an external data source using the *data loader class* which was chosen by the user when launching the program.

2. It compares the intended state of data with the currently stored data in the *edufirewall* database and calculates what database modifications must be performed in order to reach the intended state in the database.

3. It performs these database modifications, thereby reaching the intended state[3]. In other words, this action ensures that the data about users, services and bindings stored in the *edufirewall* database are identical to the data stored in the external data source.

By implementing *data loader classes*, the program can be adapted to synchronize with any source which provides the necessary data in their complete form. Requirement 6b suggests that such source may be an Identity Management System (IdM), which is therefore mentioned in the name of this program. However, as that requirement also states, "integration with a specific IdM is out of scope of this thesis"; consequently, the only data loader class which is currently implemented is `TestFileDataLoader`, which is primarily intended to be used for testing and which loads data from a local JSON file.

3. In addition, the program can be launched in a *dry run mode*, in which the modifications are not performed, but rather only logged for diagnostic purposes.

4.5 eduVPN Hook

The *edufirewall eduVPN Hook*, packaged as `efw_vpnhook`, is a program for managing IP addresses of users in the *edufirewall* database. The program is able to operate in two distinct modes: the *eduVPN mode* and the *propagation mode*.

In the **eduVPN mode**, the program is intended to be deployed on eduVPN Portal machines and run as a *Script Connection Hook* [31]. This functionality of the `vpn-user-portal` software makes it possible to "launch a (custom) script when a VPN client connects, and/or disconnects" [31]. This script is provided various information about the connection or disconnection event through environment variables, including, but not limited to, the user ID and IP addresses [31]. In this mode, `efw_vpnhook` uses this information to reflect the event in database, which triggers database notifications to be sent to listening instances of `efw_nftablesd` (see Section 4.6), resulting in firewalls on eduVPN Nodes being adjusted accordingly. Since this above-described hooking into the eduVPN system is the primary use case of this program, it is reflected in its name.

In the **propagation mode**, the program synchronizes IP addresses of users in a similar manner to how `efw_idmhook` synchronizes the other information in the database, as described in Section 4.4. In this mode, the program loads users' IP addresses from a CSV file, calculates the difference between the loaded data and data in the database, and then equalizes the difference. This is an auxiliary mode which may be of assistance for example during database recovery or testing.

4.6 nftables Configuration Daemon

The *edufirewall nftables Configuration Daemon* program component, packaged as `efw_nftablesd`, is responsible for generating and managing firewall rulesets on eduVPN Nodes. It shall be deployed on all eduVPN Nodes handling the unified profile, where it will configure fine-grained `nftables` firewalls in the form described in Section 3.2 based on the data stored in the *edufirewall* database. Furthermore, the program will wait for notifications informing about changes to that data and react to them accordingly. Therefore, the program is designed

to run as a daemon, i.e. indefinitely and without direct user interaction, and it will terminate only upon receiving either the `SIGTERM` or `SIGINT` POSIX signal.

Architecturally, the implementation of this component is more complex than in the case of the previously described components. Its functionality is separated into three types of tasks in four instances, which are scheduled concurrently on an event loop as outlined in Section 4.2.1. These tasks communicate using the custom priority event queue described in Section 4.3.2 by passing event objects through it. The tasks are described in more detail in the following subsections.

4.6.1 The reload-on-signal task

This task enables administrators to trigger a full database reload on demand. It waits for the `SIGHUP` POSIX signal to be delivered, and for each such delivery, it generates a *database change event* and inserts it into the event queue. This capability of this program it may be of assistance for diagnostic purposes.

4.6.2 The database notification receiver tasks

This task manages a connection to the PostgreSQL database which is set to be listening on a particular notification channel, receives notifications from it, and inserts them into the event queue. Unlike the other tasks, this task exists in two instances in the program — one for database change notifications and one for connection change notifications (see Section 3.1.2).

4.6.3 The event consumer task

In an infinite loop, this task consumes events from the event queue, which are produced by the other tasks, and acts according to them. If it obtains a **database change event**, which is also automatically inserted into the queue upon the program's startup and as a result of various error conditions, it performs the following actions:

- Firstly, it discards any previous internal state related to firewall configuration, and reloads data about all users, services and their bindings from the *edufirewall* database.

- Then, it performs certain checks of the loaded data, thereby ensuring that all IP addresses and subnets contained within them are valid; in case any invalid or inconsistent data are detected, the program does not exit with an exception — instead, the problematic data are discarded and warning messages are logged.

- Furthermore, while performing the checks, various additional information necessary for generating the resulting firewall is computed from the loaded data, such as the classification of service access point IP subnets as either *this-Node*, *other-Node* or *Internet* (see Section 3.2). Since the results of this classification are naturally different on each eduVPN Node, they cannot be reliably computed in advance, while storing the data into the database.

- Finally, it clears any previously generated `nftables` firewall configuration, generates a new configuration in the form detailed in Section 3.2 based on the loaded and validated data, and applies it in the operating system. It does so in an atomic manner and using the JSON API of the `libnftables` library.

If the task obtains a **connection change event**, which represents that a particular user has either connected to or disconnected from eduVPN under a particular IP address, it takes the following actions:

- It reflects the information represented by the event in the internal state of the program, while performing certain consistency checks. If such check fails, for example when the affected user does not exist or the newly connected IP address is already assigned to a different user in the internal state, a full database reload is triggered.

- It incorporates the state modifications into the currently configured `nftables` ruleset. This involves adding or removing the affected IP address in the per-user and per-IP-version verdict maps, but also adding or removing per-user chains in certain cases (see Section 3.2.1).

5 Performance testing

As part of this thesis, the testing of the proof-of-concept software presented in Chapters 3 and 4 was performed, and the impact of the firewall rulesets generated by it on the network throughput of eduVPN Nodes was measured. The measurements were carried out using the `iperf3` tool, whose raw outputs are part of Appendix B, and they are analyzed in this chapter.

Due to the proof-of-concept nature of the software implementation, its performance was not subject to measurement considering that the software's speed was not the primary focus while implementing it, as stated in Chapter 3. Further optimization thereof, or even its complete rewrite in a more performant programming language, is likely to improve it.

5.1 Environment

For testing purposes, three virtual machines have been made available to the author of this thesis by the people in charge at Institute of Computer Science, Masaryk University. The virtual machines are located in a Proxmox cluster with an AMD EPYC 7713 CPU, are connected to the network using a paravirtualized network interface (`virtio_net`), and are detailed individually in Table 5.1.

Table 5.1: Details of testing virtual machines

Hostname	CPU	RAM	Usage
`labuda-eduvpn`	4 cores	8 GB	eduVPN Portal & Node
`labuda-eduvpn-perf1`	4 cores	4 GB	iperf server
`labuda-eduvpn-perf2`	8 cores	8 GB	eduVPN & iperf client

In terms of software, the Debian GNU/Linux 13 (trixie) operating system is installed on all the virtual machines. On `labuda-eduvpn`, the eduVPN system has been deployed as per the official deployment instructions [41]; in addition, all the *edufirewall* components listed in Section 3.3 have been successfully deployed on that machine according to the instructions within their `README.md` files (see Appendix A),

thereby showing that the proof-of-concept software implementation is functional.

While performing the measurements, network traffic was generated using the `iperf3` tool on `labuda-eduvpn-perf2`, routed via VPN through `labuda-eduvpn` and destined towards `labuda-eduvpn-perf1`.

5.2 Baseline

Firstly, in order to be able to conclude whether the generated firewall rulesets reduce network throughput, it is necessary to measure the throughput without them. Therefore, a measurement was performed while the `efw_nftablesd` daemon was stopped and while the `forward`-hooked `nftables` chain was empty (and was configured to accept all packets); the results of this measurement are presented in Table 5.2.

Table 5.2: Baseline network throughput

	Min	**Avg**	**Max**
WireGuard	0.99 Gb/s	1.06 Gb/s	1.13 Gb/s
ProxyGuard	0.35 Gb/s	0.37 Gb/s	0.38 Gb/s

As can be seen, the network throughput while eduVPN is using the ProxyGuard software is significantly lower than when using pure WireGuard. This is to be expected, as ProxyGuard proxies WireGuard datagrams through HTTPS [11], or, to be more specific, through the web server which is hosting the eduVPN Portal. Therefore, this decrease of throughput can be attributed to the additional TCP, TLS and HTTP layers.

5.3 Small dataset

Then, network throughput was measured with a firewall ruleset generated from a small dataset comprising of 2 users, with one of them connected (the one who performed the measurements), and 5 services[1]. The results of this measurement are displayed in Table 5.3.

1. The small dataset, which was inserted into the database using `efw_idmhook`, is included in the source code archive (see Appendix A) — `src/efw_idmhook/test_data/small_real.json`.

Table 5.3: Small dataset network throughput

	Min	Avg	Max
WireGuard	0.95 Gb/s	1.05 Gb/s	1.12 Gb/s
ProxyGuard	0.35 Gb/s	0.36 Gb/s	0.37 Gb/s

Since the generated firewall ruleset was naturally not extensive, there was no measurable impact on the network throughput compared to when no firewall was active, as expected.

5.4 Large dataset

Finally, a large dataset was randomly generated[2] with 65,000 users (approximately the number of active users at Masaryk University) and 400 services. Then, 384 simultaneously connected users were simulated (using the *propagation mode* of `efw_vpnhook`[3] — see Section 4.5), and one additional user was actually connected to the VPN for the purpose of carrying out the measurement. The results of this measurement, with the extensive generated firewall ruleset active, are presented in Table 5.4.

Table 5.4: Large dataset network throughput

	Min	Avg	Max
WireGuard	0.91 Gb/s	1.02 Gb/s	1.08 Gb/s
ProxyGuard	0.34 Gb/s	0.37 Gb/s	0.39 Gb/s

Despite the extensive firewall ruleset, no significant impact on the network throughput could be measured compared to when no firewall or a small firewall was active. It is argued that this is, at least partially, the result of the numerous optimizations of the dynamically generated firewall ruleset (see Section 3.2.1).

2. The script for randomly generating such large datasets is bundled with the source code (see Appendix A) — `src/efw_idmhook/test_data/generate_big_json.py`.
3. The input data for this propagation have been randomly generated by a script which is distributed in the source code archive (see Appendix A) — `src/efw_vpnhook/test_data/generate_big_csv.py`.

5.5 Evaluation

Based on the undertaken measurements, it was found that the size of the dataset, i.e. the size of the contents of the *edufirewall* database, which is reflected in the size of the generated firewall ruleset, does not have any significant impact on the network throughput of eduVPN Nodes, as Figure 5.1 illustrates. Therefore, the author argues that this firewalling approach is suitable for use even in large institutions with extensive eduVPN user bases, such as Masaryk University, from the performance standpoint.

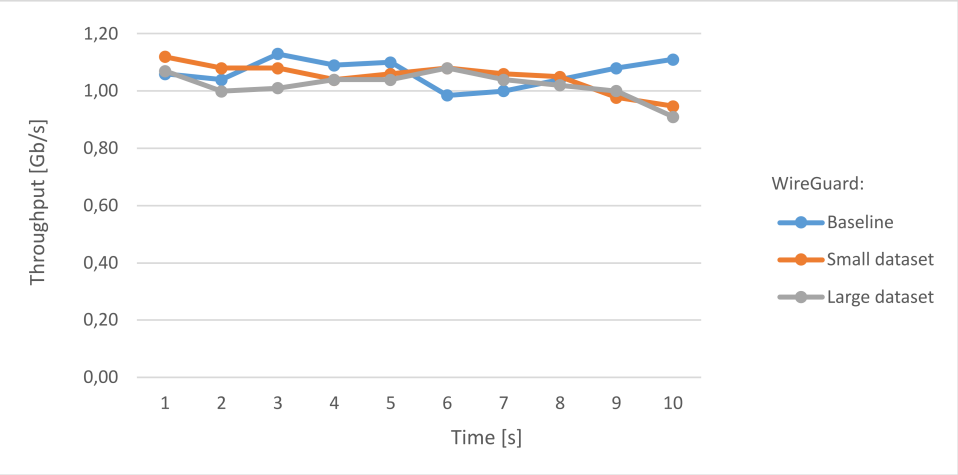

Figure 5.1: Measurement results comparison

Conclusion

The purpose of the thesis was to propose and then create a proof-of-concept implementation of a fine-grained access control architecture, which would eliminate the security- and usability-related disadvantages of the currently deployed profile-based architecture within Masaryk University.

In the first part of this thesis, key concepts and technologies were presented. Then, the access control architecture currently deployed within the eduVPN system at Masaryk University was described in detail. This architecture has several limitations, such as its coarse-grained nature and the need for profile switching.

In the second part of this thesis, the proposal of the new access control architecture followed, beginning with the list of requirements for it. Since one of the requirements mandated the compatibility of both architectures, the description of the new architecture took the form of a set of modifications to the current one. The new architecture resolves the limitations of the current one; however, it was identified that it centralizes the overall security model.

The new access control architecture is substantially dependent on custom software, and therefore, this software was designed and implemented next. The software utilizes a central database storing information about users and services, and it consists of several components: `efw_idmhook`, which enables the synchronization of permission information into the database, `efw_vpnhook`, which notifies the system about eduVPN client connections and disconnections, and `efw_nftablesd`, which automatically generates firewall configurations on eduVPN Nodes based on the information stored in the database and dynamically reacts to its changes. Furthermore, common functionality, such as the database access layer, was factored out into a pair of library components. The solution was collectively named *edufirewall*.

Finally, the implemented solution was tested and the impact of extensive firewall configurations generated by it was investigated with regard to the network throughput of eduVPN Nodes. Since it was found that the size of the firewall does not result in a noticeable impact on the throughput, it was argued that this firewalling approach is suitable for further use within Masaryk University. However, the

thesis was only concerned with creating a proof-of-concept solution, which has not yet been deployed in a production environment, as well as subjected to security testing — this may be the focus of further research.

Bibliography

1. *eduVPN - Remote Connection to the University Network* [online]. Brno: Masaryk University, 2026 [visited on 2026-04-26]. Available from: https://it.muni.cz/en/services/eduvpn.
2. SCOTT, Charlie; WOLFE, Paul; ERWIN, Michael. *Virtual Private Networks*. 2nd ed. Sebastopol, CA: O'Reilly Media, 1999.
3. DONENFELD, Jason A. Wireguard: next generation kernel network tunnel. In: *NDSS*. 2017, pp. 1–12.
4. *Known Limitations - WireGuard* [online]. Jason A. Donenfeld, 2015/2022 [visited on 2026-04-26]. Available from: https://www.wireguard.com/known-limitations/.
5. *OpenVPN Cryptographic Layer* [online]. OpenVPN Inc., 2026 [visited on 2026-04-26]. Available from: https://openvpn.net/community-docs/openvpn-cryptographic-layer.html.
6. *OpenVPN Protocol* [online]. OpenVPN Inc., 2026 [visited on 2026-04-26]. Available from: https://openvpn.net/community-docs/openvpn-protocol.html.
7. *About eduVPN* [online]. eduVPN, 2026 [visited on 2026-04-26]. Available from: https://www.eduvpn.org/about-eduvpn/.
8. *Source Code* [online]. eduVPN, 2026 [visited on 2026-04-26]. Available from: https://docs.eduvpn.org/server/v3/source-code.html.
9. *vpn-user-portal README* [online]. eduVPN, 2026 [visited on 2026-04-26]. Available from: https://codeberg.org/eduVPN/vpn-user-portal/src/branch/v3/README.md.
10. *vpn-daemon README* [online]. eduVPN, 2026 [visited on 2026-04-26]. Available from: https://codeberg.org/eduVPN/vpn-daemon/src/branch/main/README.md.
11. *proxyguard README* [online]. eduVPN, 2026 [visited on 2026-04-26]. Available from: https://codeberg.org/eduVPN/proxyguard/src/branch/main/README.md.

12. *eduVPN - High Availability (HA)* [online]. eduVPN, 2026 [visited on 2026-04-26]. Available from: https://docs.eduvpn.org/server/v3/ha.html.

13. *eduVPN - Multi Node Deployments* [online]. eduVPN, 2026 [visited on 2026-04-26]. Available from: https://docs.eduvpn.org/server/v3/multi-node.html.

14. *eduVPN - Profile Configuration* [online]. eduVPN, 2026 [visited on 2026-04-26]. Available from: https://docs.eduvpn.org/server/v3/profile-config.html.

15. CHESWICK, William R; BELLOVIN, Steven M; RUBIN, Aviel D. *Firewalls and internet security*. 2nd ed. Boston, MA: Addison Wesley, 2003. Addison-Wesley professional computing series.

16. *netfilter/iptables project homepage* [online]. The netfilter.org project, 2025 [visited on 2026-04-29]. Available from: https://www.netfilter.org/.

17. AYUSO, P. Netfilter's connection tracking system. *Login: The Usenix Magazine*. 2006, vol. 31, pp. 34–39.

18. *The return of nftables* [online]. LWN.net, 2013 [visited on 2026-04-30]. Available from: https://lwn.net/Articles/564095/.

19. *Man page of NFT* [online]. The netfilter.org project, 2023 [visited on 2026-04-30]. Available from: https://www.netfilter.org/projects/nftables/manpage.html.

20. *Sets - nftables wiki* [online]. The netfilter.org project, 2023 [visited on 2026-04-30]. Available from: https://wiki.nftables.org/wiki-nftables/index.php/Sets.

21. *Maps - nftables wiki* [online]. The netfilter.org project, 2021 [visited on 2026-04-30]. Available from: https://wiki.nftables.org/wiki-nftables/index.php/Maps.

22. SALTZER, Jerome H; SCHROEDER, Michael D. The protection of information in computer systems. *Proceedings of the IEEE*. 1975, vol. 63, no. 9, pp. 1278–1308.

23. *eduVPN Documentation* [online]. Brno: Masaryk University, 2024 [visited on 2026-05-03]. Available from: https://docs.eduvpn.muni.cz/. Internal technical documentation portal; access restricted to Masaryk University employees.

24. *Profiles - eduVPN Documentation* [online]. Brno: Masaryk University, 2024 [visited on 2026-05-03]. Available from: https://docs.eduvpn.muni.cz/export/documentation/profiles/. Internal technical documentation portal; access restricted to Masaryk University employees.

25. SILVA, Edelberto Franco; MUCHALUAT-SAADE, Débora Christina; FERNANDES, Natalia Castro. ACROSS: A generic framework for attribute-based access control with distributed policies for virtual organizations. *Future Generation Computer Systems*. 2018, vol. 78, pp. 1–17.

26. *Perun - eduVPN Documentation* [online]. Brno: Masaryk University, 2024 [visited on 2026-05-04]. Available from: https://docs.eduvpn.muni.cz/export/documentation/architecture/perun/. Internal technical documentation portal; access restricted to Masaryk University employees.

27. *Network Segmentation - eduVPN Documentation* [online]. Brno: Masaryk University, 2024 [visited on 2026-05-04]. Available from: https://docs.eduvpn.muni.cz/export/documentation/architecture/network-segmentation/. Internal technical documentation portal; access restricted to Masaryk University employees.

28. *eduVPN - Firewall (iptables) - Reject IPv6 Client Traffic* [online]. eduVPN, 2026 [visited on 2026-05-04]. Available from: https://docs.eduvpn.org/server/v3/firewall-iptables.html#reject-ipv6-client-traffic.

29. *Firewall - eduVPN Documentation* [online]. Brno: Masaryk University, 2024 [visited on 2026-05-04]. Available from: https://docs.eduvpn.muni.cz/export/documentation/profiles/firewall/. Internal technical documentation portal; access restricted to Masaryk University employees.

30. *Addressing - eduVPN Documentation* [online]. Brno: Masaryk University, 2024 [visited on 2026-05-04]. Available from: https://docs.eduvpn.muni.cz/export/documentation/architecture/addressing/. Internal technical documentation portal; access restricted to Masaryk University employees.

31. *eduVPN - Script Connection Hook* [online]. eduVPN, 2026 [visited on 2026-05-05]. Available from: https://docs.eduvpn.org/server/v3/script-connection-hook.html.

32. WEIL, Jason; KUARSINGH, Victor; DONLEY, Chris; LILJENSTOLPE, Christopher; AZINGER, Marla. *IANA-Reserved IPv4 Prefix for Shared Address Space* [RFC 6598]. RFC Editor, 2012. Request for Comments, no. 6598. Available from DOI: 10.17487/RFC6598.

33. *Architecture - eduVPN Documentation* [online]. Brno: Masaryk University, 2024 [visited on 2026-05-05]. Available from: https://docs.eduvpn.muni.cz/export/documentation/architecture/. Internal technical documentation portal; access restricted to Masaryk University employees.

34. *100.65.0.0/19 - NetBox* [online]. Brno: Masaryk University, 2026 [visited on 2026-05-05]. Available from: https://netbox.ics.muni.cz/ipam/prefixes/27096/prefixes/. Internal information system; access restricted to selected Masaryk University employees.

35. *PostgreSQL: Documentation: 18: 51.2. How Connections Are Established* [online]. The PostgreSQL Global Development Group, 2026 [visited on 2026-05-11]. Available from: https://www.postgresql.org/docs/18/connect-estab.html.

36. *PostgreSQL: Documentation: 18: NOTIFY* [online]. The PostgreSQL Global Development Group, 2026 [visited on 2026-05-12]. Available from: https://www.postgresql.org/docs/18/sql-notify.html.

37. *eduVPN - Novinky, statistiky a řešené problémy* [online]. Brno: Masaryk University, 2025 [visited on 2026-05-12]. Available from: https://docs.eduvpn.muni.cz/blog/2025/06/26/eduvpn-novinky-statistiky-a-resene-problemy/. Internal technical documentation portal; access restricted to Masaryk University employees.

38. *asyncio - Asynchronous I/O* [online]. Python Software Foundation, 2026 [visited on 2026-05-16]. Available from: https://docs.python.org/3/library/asyncio.html.

39. *A Conceptual Overview of asyncio* [online]. Python Software Foundation, 2026 [visited on 2026-05-16]. Available from: https://docs.python.org/3/howto/a-conceptual-overview-of-asyncio.html.

40. *Coroutines and tasks* [online]. Python Software Foundation, 2026 [visited on 2026-05-16]. Available from: https://docs.python.org/3/library/asyncio-task.html.

41. *eduVPN - Deploying on Debian and Ubuntu* [online]. eduVPN, 2026 [visited on 2026-05-16]. Available from: https://docs.eduvpn.org/server/v3/deploy-debian.html.

A Source code

The source code of the proof-of-concept software named *edufirewall*, whose architecture was described in Chapter 3 and implementation in Chapter 4, is an essential part of this thesis. It is packaged in the `Bachelor-thesis_source-code_Vit-Labuda.zip` file, which is available in the archive of this thesis and whose top-level structure is the following:

- `LICENSE` — The license notice.
- `README.md` — A read-me document with a short description of the software project, as well as general deployment instructions.
- `db/` — The definition and documentation files pertaining to the *edufirewall* database. It contains its own `README.md` document with database-related instructions.
- `src/` — The Python packages of which this software system is comprised in individual subdirectories. Each of the packages comes with its own `README.md` document describing its usage and/or deployment.

B Performance testing outputs

The analysis of the performance testing presented in Chapter 5 is based on measurements of network throughput, which have been carried out as part of this thesis. The raw measurement results, as well as the commands used to generate them, are packaged in the `Bachelor-thesis_measurement-results_Vit-Labuda.zip` file, which is available in the archive of this thesis and which contains the following files:

- `01_baseline.txt` — The outputs analyzed in Section 5.2.
- `02_small-dataset.txt` — The outputs analyzed in Section 5.3.
- `03_large-dataset.txt` — The outputs analyzed in Section 5.4.