Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Mondrian: A Comprehensive Inter-Domain Network Zoning Architecture

Master Thesis

Claude Hähni

August 17, 2020

Advisors: Prof. Dr. Adrian Perrig, Dr. Jonghoon Kwon, Patrick Bamert

Department of Computer Science, ETH Zürich

Abstract

A central element of designing IT security infrastructures is the logical segmentation of information assets into groups sharing the same security requirements and policies, called network zones. As more business ecosystems are migrating to the cloud, additional demands for cybersecurity emerge and make the network-zone operation and management for large corporate networks challenging. In this thesis, we introduce the new concept of an inter-domain transit zone that securely bridges physically and logically non-adjacent zones in large-scale information systems, simplifying complex network-zone structures. With inter-zone translation points, we also ensure communication integrity and confidentiality while providing lightweight security policy enforcement. A logically centralized network coordinator enables scalable and flexible network management. Our implementation demonstrates that the new architecture merely introduces a few microseconds of packet processing delay.

Acknowledgments

mention the people I'd like to thank

Acknowledgments

The paper "paper name" has been written based on the work done in this thesis. [put reference to paper]

Contents

Chapter 1

Introduction

Network zoning has long been an essential part of the Internet security infrastructure, which logically partitions network and information assets into disjoint segments that share the same security requirements and policies, and functional similarities. Zones define the network boundaries and their defense requirements by stating the entities populating the zones, the entry points into the zones, and how traffic is monitored and filtered at these entry points. Informally, these zones are realized by a virtualized separation at layer 2 (e.g., IEEE 802.1q [1]) with firewalls at higher levels governing data transfers between zones [2].

Each zone is identified by a distinct level of trust, and forms a trusted/untrusted relationship with other zones [3]. To realize the unidirectional trust model, firewalls are considered to be the most viable technology and are widely used in the current practice. However, operating firewalls in large enterprises is often challenging for network operators and security architects. The access control for network zones might be dynamic, and thus it requires complex management schemes to accommodate a myriad of policies. While there are advanced technologies such as virtual firewalls [4, 5], distributed security enforcement [6, 7], and Unified Threat Management (UTM) [8], newly designed to enforce access control polices in extremely dynamic networks, network zone management and modeling still remains cumbersome [9, 10].

Bridging geographically distant network zones is very challenging today. In general, network zones are created not only for security purposes but also because of geographical, operational, or organizational factors. Large enterprises with geographically distributed branch networks, and possibly collaborative partners' networks need to be interconnected. Given that distant network zones exchange information over an untrusted network (e.g., the Internet), there is a risk that the communication exposes security-sensitive information during transit. To mitigate such threats, administrators leverage

additional security mechanisms (e.g., IPsec [11] and SSL-VPN [12]) which ensure confidentiality and integrity of the transmission over the untrusted network by encrypting and authenticating the data with securely shared cryptographic keys. Nonetheless, these technologies bring forth new challenges such as management scalability [13] and compatibility issues with other security solutions [14]—universal agreement with business partners on building collaborative security infrastructure is often problematic.

Mondrian is a new network zoning architecture that secures inter-zone communication—which operates on layer 3, supporting heterogeneous layer 2 architectures—while ensuring scalable cryptographic-key management and flexible security policy enforcement. Mondrian flattens the current hierarchically-complex network zone topology into a collection of we never show what hierarchy was used before horizontal zones connected to a unified security gateway, called Zone Translation Point (TP), thus simplifying large enterprise networks. By interconnecting zones through TPs, complex zone restructuring operations become easier with respect to new zone initializations or zone migrations. The TP ensures source authentication, zone transfer authorization, and illegitimate access filtering by acting as a secure ingress/egress point for network zones. A logically centralized control unit provides management scalability on zone classification and policy enforcement, and mediates cryptographic key establishment.

A secure zone transfer is performed in three steps: i) the security gateway acquires access policies for each network zone from its controller, ii) the gateway issues a cryptographically protected authorization token if a given zone transfer request is permitted, and iii) the network forwards only packets with a valid token. By leveraging the notion of secure tunneling between two endpoints (i.e., a pair of local and remote TPs), confidentiality and integrity of the zone transfer packets are ensured, while keeping the overhead of the authentication process small. For scalable key management, we employ a key establishment system that enables dynamic key derivation and ensures perfect forward secrecy.

We provide an implementation of Mondrian that ensures secure zone transfer for both intra and inter-domain communication at line rate, while requiring no network-stack changes from end hosts. We extensively evaluate this implementation to demonstrate the practical viability of Mondrian. The results show that the TP introduces negligible processing delay; less than 500 ns of additional delay for intra-domain zone transfer and approximately $2.5 \sim 3.5\mu$s for inter-domain zone transfer traffic. We further provide in-depth security and practicality analyses.

The main contributions of this paper are the following:

- We introduce Mondrian, a new security architecture that enables secure, flexible and viable network zoning and inter-zone communica-

tion for large enterprise networks.

- We introduce the new notion of an inter-domain transit zone that dramatically simplifies the current hierarchical zone structure, enabling flexible and cost-efficient network zone management.
- MONDRIAN enables network filtering at the edge of the network, such that it suppresses network overhead in terms of wasted network bandwidth and packet processing time.
- We implement MONDRIAN as an opensource project.

Chapter 2

Background

Using a case study we explore how network zoning is realized in modern enterprise networks, and later we derive the main challenges we confront.

2.1 Case Study

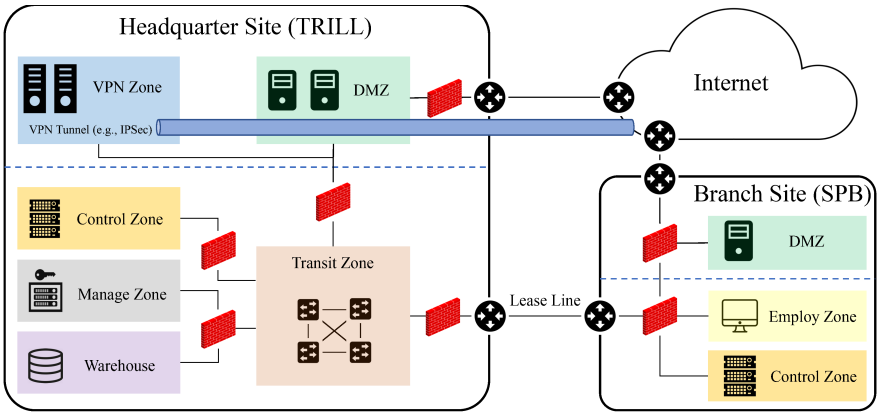

Figure 2.1: *Network zoning use case for large enterprises. Network zones are realized with heavy use of security middleboxes (e.g., Firewalls).*

Most enterprise networks have embraced the notion of layered security classification, that can be broadly split into intranet, extranet, and opennet [9]. The opennet is the least trusted network (e.g., the Internet) which is an inhospitable region where live threats exist, whereas the intranet is the most trusted network hosting business-critical systems and sensitive information. Since the intranet has rigorous access control mechanisms to protect information assets from exposure to the opennet, enterprises are forced to operate

another security layer (extranet, also known as demilitarized zone or DMZ) in between, which exposes the publicly accessible services to the opennet, while reducing the attack surface of the intranet.

Over time, these layered network structures have become more sophisticated [3] due to extreme changes in network environments—diverse demands from customers, partners and employees accessing enterprise networks with a variety of devices. As a result, many enterprise networks comprise a large number of zones defined by operational, organizational, and most importantly security factors. Figure 2.1 depicts a real-world use case for network zones running on inter-domain level with multiple involved autonomous systems (ASes). They can be categorized into three main types.

Intra-domain Zone Transfer Within a local network, multiple devices such as servers, databases, and hosts are connected through network switches. These devices are assigned with a unique IP address that belongs to a logically isolated network zone. These zones commonly consist of multiple subnets, often realized with a layer 2 virtualization technology (e.g., VLAN). Each zone is protected by a set of security middleboxes, e.g., firewalls, intrusion-prevention systems (IPS), and intrusion-detection systems (IDS), which enforce predefined security policies for all traffic passing through.

To maintain the zone-based trust model, access permission to one zone is not considered to be valid for other zones. That is, an entity must obtain access permissions from all zones on the path when accessing a non-adjacent zone. This trust model however often complicates policy management and enforcement, especially for large enterprise networks. To resolve this complication, the current practice introduces the notion of a dedicated zone in which zone transitions are handled, called Transit Zone.

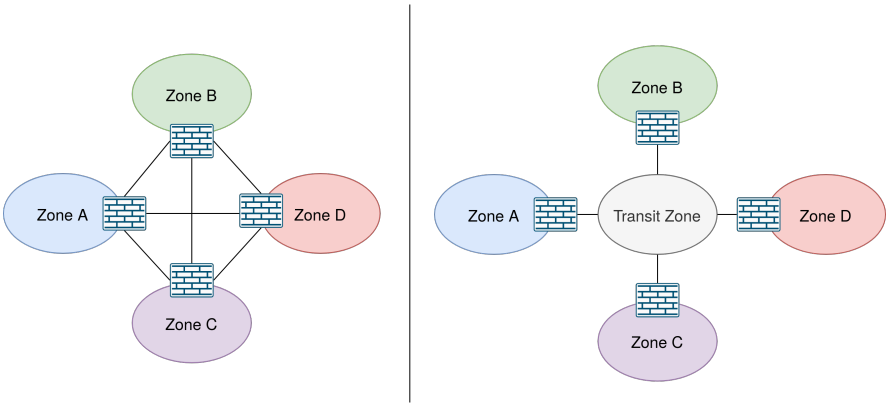

Figure 2.2: *Two ways of structuring network zones. The use of a transit zone on the right side reduces the number of required links.*

A transit zone acts like a patch panel allowing zones to be interconnected without the need of a dedicated link between each pair of zones (Figure 2.2). The Transit zone sits in the middle of all the other zones and mediates access between zones wishing to communicate with each other. It is commonly comprised of only forwarding devices (e.g., switches), interconnecting the attached zones via various ingress/egress points on which security middleboxes enforce the security policies. In a nutshell, the Transit Zone reduces the depth of zone hierarchies and thus simplifies the network zone design and management.

Inter-domain Zone Transfer To ensure that geographically distributed zones can securely communicate with each other, enterprises employ various networking technologies. The most common choice is connecting two remote sites with a physical private line, (e.g., layer 2 circuit). Enterprises can lease these lines from Internet service providers and make use of them to bridge local networks. However, purchasing private lines is costly and might come along with trust issues towards the service provider.

An alternative is a virtual private network (VPN). A VPN uses cryptographic primitives to create a virtual tunnel between two local networks, preventing information leakage during transmission over the public Internet. While the VPN technology ensures data confidentiality, typically yet another layer of overlay protocols is required to achieve virtual separation of zones. The use of such overlay protocols, however, has the disadvantage that all interconnected sites need to deploy the same protocol since such protocols generally do not offer interoperability.

Traffic from the Internet Traffic not originating in cooperative (trusted) networks can be classified into the following three types: i) public traffic, ii) authorized traffic, and iii) malicious traffic. The first case covers normal customers who access the enterprise's public services, e.g., Web servers. This traffic in general ends up at the demilitarized zone (DMZ) hosting only public services that require exposure to Internet. The second case refers to the traffic coming from temporarily authorized devices. For example, a legitimate employee outside the enterprise's premises—working from home with a personal device—may get a temporal permit to access restricted zones via VPN. The last category comprises attack traffic which is to be filtered by the security middleboxes in the frontline of defense.

describe all 6 usecases here?

2.2 Challenges

Secure Zone Transfer Transmitting security-sensitive data between zones in different physical locations (e.g., data center to branch site) over the public Internet poses a challenge. Security level information is lost in transit, requiring that the data is re-authenticated and filtered again on the receiving site even though source and destination could be part of the same logical zone. Today's overlay protocols are often used to overcome the restriction of losing security level information in transit. This, however, introduces new challenges: difficulties in deployment per zone, computational overhead, and poor management scalability.

Interoperability Even if security-level information persists in transit, different zones might not be built on the same internal protocols (e.g., SPB [15] vs Trill [16]) which makes it difficult for end systems in different zones to be able to seamlessly communicate with each other. uncomment text, maybe rephrase

Management Scalability In current local network zoning architectures, administration is being considered a tedious, time-consuming, and labor-intensive task. For example, simply adding a new zone might require existing policies to be thoroughly reviewed, updated, and re-distributed to the local network entities. The management complexity dramatically increases in a wide-area network (WAN) environment. uncomment text, maybe rephrase

Chapter 3

Related Work

move to bottom?

The majority of literature in network zoning has focused on security enforcement architecture using middleboxes such as firewalls, IPS, and IDS. Conventional security middleboxes define restricted zones and filter unwanted traffic at the entry points of protected zones [17]. As information systems and the corresponding network functions got more complicated, the notion of distributed security systems has been introduced in the late 1990's [18]. Early approaches to protect only internal information systems from external threats have further evolved to mitigate sophisticated threats, for example insider attacks, rule tampering, application-level proxies, and denial-of-service attacks [6]. Later, with emerging network virtualization technologies and cloud computing environments, virtual firewalls and collaborative security enforcement kept getting attention from both academia and the industry [7, 14].

Despite numerous research efforts, network zoning using security middleboxes has issues with respect to performance—the iMIX throughput degrades by 40 - 75% on commodity products [19]—and misconfigurations [20–22]. With MONDRIAN, we address these challenges by leveraging a cryptographic-based policy enforcement and centralized policy orchestration, achieving scalable, effective, and cost-efficient network zoning.

Network isolation through network segmentation is another essential element of network zoning. To logically segment the physical network, network virtualization technologies are heavily used in today's Internet, in particular large enterprise networks and cloud computing environments. VLAN [1] is the most frequently used network segmentation technique. It logically segments a physical LAN into up to 4094 virtual LANs by tagging the layer 2 header with a unique VLAN identifier (VID). Later, Virtual eXtensible LAN (VXLAN) [23] has been introduced for better scalability;

it expands the number of virtual LANs to up to 16 million by leveraging a 24-bit identifier. SPB [15] and Trill [16, 24] are layer 2 routing protocols enabling multi-path communication among virtual LANs within the same physical LAN. Although security is an important property when it comes to network segmentation, it unfortunately has often not been treated as a major concern. Systems lack protection for segment membership and access control. To isolate each segment, use of a large number of security middleboxes is necessary, thus increasing operational costs and management complexity.

SVLAN [25] is an architecture enhancing security in network isolation by enforcing the receiver's consent towards incoming traffic. SVLAN is the closest work related to Mondrian, but there are three major differences. First, Mondrian decouples policy establishment and enforcement. In SVLAN, the authorization delegate (i.e., controller) is responsible for establishing basic reachability policies and issuing authorization tokens to enforce the policies. The single trust model requires additional latency to fetch the authorization token for a new flow. In contrast, Mondrian leaves the policy enforcement to the data-plane, reducing overhead. Second, Mondrian provides data confidentiality. SVLAN focuses on bridging layer 2 networks and thus leaves higher-layer protocols responsible for communication security. Lastly, Mondrian is compatible with various Internet architectures, whereas SVLAN is relying on segment routing [26, 27]. Although segment routing is an emerging technology, yet it is not completely supported especially for collaborative wide-area networks, and thus constrains deployability.

Chapter 4

Overview

This chapter provides an overview of MONDRIAN. We first elaborate the fundamental goals of this research, along with requirements, and design choices (§4.1). We then outline MONDRIAN including a brief introduction of each component and their workflow (§4.2). Finally, we describe our threat model (§4.3) and state our assumptions (§4.4).

4.1 Design Principles

Goal The fundamental goal of this work is to build an architecture that combines local network zoning and inter-domain routing with regard to secure zone transfers, lightweight protocol interoperability, and incremental deployability—thereby enabling secure, scalable, and flexible network zoning on a global scale. That is, an administration domain expresses zone definitions and corresponding zone transfer rules, and deploys the policies to distributed network entities. These policies force the network to only forward authorized packets protected by cryptographically secured authenticators, ensuring secure and sustainable zone-to-zone communication.

Desired Properties We consider the following properties to achieve this goal.

- Data confidentiality: through a constructive approach, the zone transfer protocol ensures that no information is exposed while being transmitted via the public Internet.
- Management scalability: logically centralized orchestration empowers network administrators to easily migrate network topologies, update policies and mirror abstract network zones into the real network.
- Efficiency: the cryptographic primitives introduce only minor performance overhead in terms of latency, bandwidth, and operational costs.

- Deployability: the Mondrian architecture requires minimal changes to the existing network infrastructure in order to achieve compatibility. Furthermore, firewalls and VPN devices at each entry point of every zone can be replaced with one Mondrian gateway, saving operational costs for the same level of network security.

Design Choices We design Mondrian working with mature technologies that are embraced in modern enterprise environments. In particular, the following design choices are made:

- Network programmability is realized with the concept of a logically centralized controller (e.g., SDN), preserving management flexibility and scalability.
- Asymmetric cryptography is used for core operations where source authenticity is critical (e.g., symmetric key exchange).
- Symmetric cryptography is used for the rest of the data transmission, ensuring efficient packet processing.

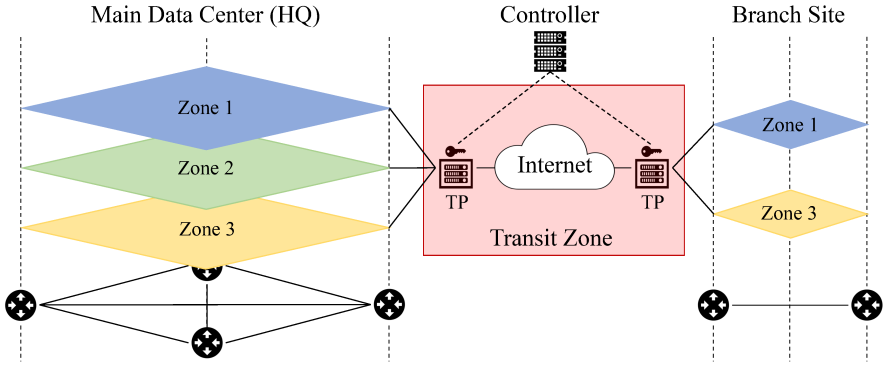

Figure 4.1: *An overview of* Mondrian *architecture. The inter-domain transit zone interconnects physically and logically distributed network zones with unified security policy enforcement.*

4.2 Mondrian Overview

Figure 4.1 illustrates an overview of Mondrian. Different branch sites of an enterprise are interconnected over a WAN (e.g., the Internet). Each site contains multiple, logically separated zones connected to the single Zone Translation Point (TP) at the corresponding site. The TP is a designated gateway for zone transfers, operating on layer 3 and interconnecting all zones at a given site of the enterprise network. All the traffic towards either internal

network or WAN, therefore, passes through the TP. Note that TPs are the endpoints of our architecture, meaning that everything beyond that point should not be modified for ease of deployment and to ensure compatibility with the modern enterprise environments.

The main task of TPs is twofold: first, they ensure that traffic adheres to a set of allowed zone transitions. For a packet originating in a zone and destined for another zone, the transition must be explicitly allowed by a policy. Second, TPs enable communication across the WAN without losing previously established security information. To this end, TPs embed a tag with cryptographically secured zone information into packets before they leave the internal network. Furthermore, TPs act as endpoints hiding sensitive information, such as internal addresses and the respective zone binding, from external entities. We also note that, for a case where zones with the same security requirements and functionality are distributed over multiple branch sites (e.g., Zone 1 and 3 in Figure 4.1), we consider them as the same logical zone (e.g., the same zone identifier). We call this concept zone extension which is not subject to zone authorization.

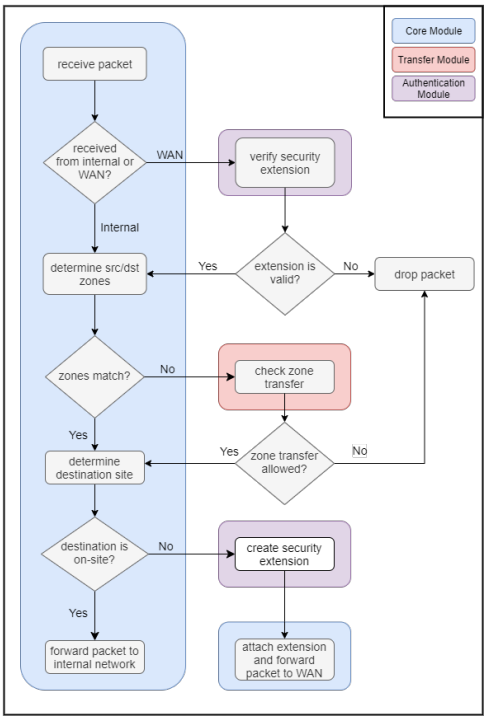

Figure 4.2: *Control flow of the Zone Translation Point*

A logically centralized controller orchestrates the TPs. The controller is owned by a single administration domain, and thus the owner of the controller owns all the zones behind the TPs. The controller provides its owner a management interface in which the owner designs an outline of zone structure and transfer policies. The explicit network configuration is then distributed to the TPs via a secure control channel to enforce the configuration at the individual premises.

Communication Flow MONDRIAN enables secure zone transfers over WAN as follows:

1. Network administrators first establish an IP-zone map and their transfer policies, which represents a virtual network configuration that explicitly specifies reachability, and upload them to the logically centralized controller.

2. Each pair of TPs exchanges symmetric keys to establish a secure tunnel. The key is being updated frequently through a state-of-art key management and distribution system that protects the secrecy of the zone transfer from being exposed to the public while ensuring high-speed data transmission.

3. The on-site TP inspects all the packets to be transmitted to another zone. The TP acquires the corresponding zone transfer policy from the controller and verifies if the transmission is authorized.

4. If a packet is authorized, the TP looks up the respective end-point TP, encrypts the packet along with the corresponding zone transfer information, and forwards it across the WAN. Otherwise, the packet is dropped.

5. The remote-site TP then decrypts the packet and forwards it according to the enclosed zone transfer information. The receiving TP could also verify the validity of the packet transmission if desired.

Figure 4.2 shows the control flow of a TP.

4.3 Threat Model

We consider a threat model in which attackers reside either on-premise (i.e., compromised end hosts) or are located outside of the cooperative networks. The goal of attackers is to access unauthorized zones to exfiltrate information assets, or disrupt networks and services. To achieve this goal, attackers can use the following strategies:

Unauthorized Access Attackers may disguise as authorized entities to blind the security middleboxes and access restricted zones. A more sophisticated attack is to override security systems by directly injecting tampered policies.

Denial-of-Service Here, the goal is to disrupt the target networks or services. Attackers can sabotage the core network systems, for example, by flooding security middleboxes that perform deep packet inspection. This might then lead to network performance degradation, causing denial-of-service for legitimate clients.

4.4 Assumptions

Public Key Infrastructure A given enterprise network has a public key infrastructure (PKI). That is, the enterprise creates a trust model for its network infrastructure, acts as a trusted certificate authority (CA), and issues certificates for the core systems. Entities can retrieve and verify the public keys of the core systems. There are open source projects, such as EJBCA[1] and OpenXPKI[2], available for setting up enterprise-grade PKIs.

Secure Cryptography Cryptographic primitives we use in MONDRIAN are secure; authenticity, integrity, and confidentiality remain intact unless the cryptographic keys are exposed.

Time Synchronization Core entities within the cooperative network have loosely synchronized system clocks with a precision of seconds (e.g., network time protocol achieves a precision of tens of milliseconds). Time synchronization is mainly used to constrain the validity of cryptographic keys.

Sole Administration A network has a sole administration domain (e.g., enterprise) which has full control over the network architecture and security policy. To access the network, collaborators must obey this policy.

[1] https://svn.cesecore.eu/svn/ejbca/trunk/ejbca/
[2] https://github.com/openxpki/openxpki/

Chapter 5

Architecture

In this chapter, we present the MONDRIAN architecture and the underlying protocols in detail. Later, we describe our key-establishment system that enables rapid key derivation and distribution in large networks.

5.1 Mondrian Bootstrapping

The bootstrapping procedure is performed when a new zone or a new remote site (a group of zones along with a TP) joins the network.

Zoning Policy A network zone is a logical concept, a group of network segments. A widely adopted segmentation technology is Virtual Lan (VLAN) which is used to separate networks on layer 2. For example, zones are made up of multiple disjoint layer 2 networks all identified by their corresponding VLAN ID (VID). A VID can be reused as long as networks using the same VID are kept separate.

In MONDRIAN, IP subnets correspond to exactly one network zone, such that the zone a host belongs to can be identified by the host IP address. The same private address spaces can be a part of different zones, and they are distinguishable by a combination of their ASN (autonomous system number) and IP subnet. Consequently, each zone is defined as:

$$ASN \times IPsubnet \mapsto VID, \tag{5.1a}$$

$$TP \times VID \mapsto zoneID. \tag{5.1b}$$

The way zones are described here corresponds to how enterprises typically segment their networks today, upholding backward compatibility and consequently deployability. Furthermore, MONDRIAN does not depend on the specific layer 2 protocols used at each site.

Every zone is under one ownership, e.g., a company. In case multiple companies want to collaborate by sharing certain network zones, it is one company that creates and owns the zones and the other companies simply join the network.

Zone Transfer Policy To allow network operators to explicitly express their zone transfer policies, we consider both denylist and allowlist-based policy establishment. This is a mature approach commonly used in modern network management systems, enabling flexible and agile orchestration of complex networking policies. The following depicts the zone transfer policy format:

$$< zone_{dst}, zone_{src} > \Rightarrow < action, priority, time > \tag{5.2}$$

action determines the corresponding action for the given source and destination zone pair, e.g., `forwarding`, `drop`, and `established`. Similar to *iptables* rules, `forwarding` would allow any incoming packets from the source zone whereas `drop` discards all traffic. `established` allows incoming traffic for all established connections.

In an event of conflict where $zone_{src}$ has different access authorizations for the $zone_{dst}$, the conflict can be resolved by the *priority* field. That is, the policy with a highest priority would be enforced. If conflicting policies have the same priority, the most recently established policy according to *time* would be enforced.

Translation Point Initialization A TP is initialized when a new remote site opens, prior to any communication between zones. The initialization process has mainly two goals: i) bootstrapping a secure channel with its controller to exchange control-plane messages, and ii) establishing secure tunnels with other TPs for data transmission.

Upon bootstrap, a TP performs a *client-authenticated TLS handshake* with its controller in order to exchange their certificate, and agree on a cipher suite and compression method. The TP finds the corresponding address in its configuration file. This means that, prior to first use, there needs to be an out-of-band setup where the TP is configured. This can either be done by the owner of the controller shipping a pre-configured machine to the new remote location or by the remote location setting up a machine and granting the controller owner management access to the given machine. Bootstraping TPs with multiple controllers will be described in §9.3 with practical considerations such as controller discovery, TP migration, and policy consistency.

The controller synchronizes with TPs to keep their list of other TPs in the network up-to-date. To this end, the controller frequently pushes a list of potential peers to TPs. This can be done either regularly (e.g., on a daily basis) or occasionally (e.g., when a new TP joins the network). TPs then

establish a secure channel with new TPs. To prevent TPs from using a stale TP list, the shared TP list is associated with a time to live (TTL) value.

For "*trust-even-before-first-use*", the TPs and controller must be able to verify each other, preventing: i) a legitimate controller pushing information to an unauthorized TP, and ii) a legitimate TP connecting to a bogus controller or TP under an attacker's control. We ensure source authenticity of the MONDRIAN entities by leveraging PKI-based identities. The main idea behind the source authenticity is that all the TPs and controllers obtain digital certificates for their IP address from a public key infrastructure (PKI). The owner of the entities (e.g., an enterprise) is a certificate authority, and issues the certificates before an entity bootstraps. To this end, we consider well-established practices, e.g., the Resource Public Key Infrastructure (RPKI) [28, 29].

5.2 Protocol Description

Now, we describe how two end hosts within different remote sites are able to communicate. Figure 5.1 illustrates a protocol level design including authorization, forwarding, and verification. A detailed header design follows.

Zone Transfer Authorization End hosts behind a TP operate as they normally would when they reside in a local network connected to a commodity gateway. That is, without any acknowledgment on network changes, a sender H_S sends a packet to the receiver H_R. If H_S and H_R are residents of the same subnet (namely the same zone), the packet will be directly steered to the destination by the local forwarding devices. If H_R is in a different subnet (or a remote site) however, the packet is first delivered to TP_S since TP_S is the gateway of H_S.

To determine if the packet is allowed to be forwarded to the given destination zone B, TP_S needs an explicit zone-transfer policy for the given source and destination pair. Ideally, the policy is cached in the TP_S's zone transfer table. In case the cache misses, TP_S acquires the policy from the controller C as follows:

1. TP_S requests a zone-transfer policy from C:

$$TP_S \rightarrow C : H_R \mid H_S \tag{5.3}$$

2. C replies with the zone-transfer policy:

$$C \rightarrow TP_S : Z_R \mid Z_S \mid rule \mid TP_R \mid ExpTime \tag{5.4}$$

The controller consults the zone-transfer policy to see if the packet is allowed to be forwarded to H_R, more specifically to destination zone Z_B. To this

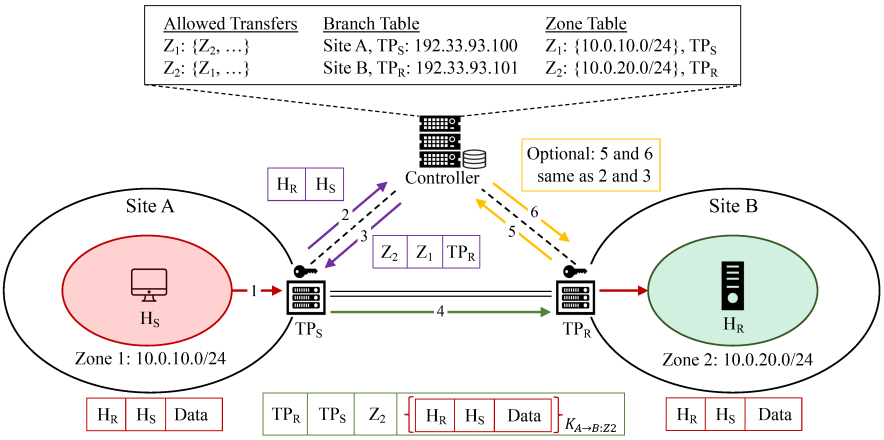

Figure 5.1: *Protocol details for data forwarding. The controller frequently updates TPs with the latest zone transfer policies.*

end, the controller first checks the corresponding zone information (Equation 5.1), and matches it with the zone transfer rules (Equation 5.2). The authorization result is then delivered to the requesting TP along with the corresponding source and destination zone identifiers (Z_R and Z_S respectively), the destination TP address (TP_R), and the expiration time ($ExpTime$) for the policy. $ExpTime$ can be an arbitrary number, but we consider it to be a small number used for policy freshness.

Data Forwarding The TP discards the packet (Host Unreachable) if $rule =$ `drop`. Otherwise, the TP looks up its routing table and transmits the packet. There exist two types of zone transfer cases: one for local (same-site) zone transfers and another for remote zone transfers as addressed in §2.1. For the local zone transfer, the TP simply rewrites the Ethernet header by following the local layer 2 protocol, and forwards it through a corresponding interface. Since the local network is assumed to be trustworthy, no additional packet processing is necessary apart from the authorization.

For the remote zone transfer, the TP is responsible for the secure transmission of the packet towards the destination TP. Recall that it is important for the inter-domain zone transfer packet to keep confidentiality and integrity in transmission. We therefore leverage the notion of secure tunneling, i.e., the IPSec tunnel mode [11, 30], meaning that the original packet is wrapped, encrypted, authenticated, and attached to a new IP header. The new packet

layout is formed as follows:

$$EIP = \{H_R \mid H_S \mid payload\}_K, \tag{5.5a}$$

$$AT = MAC_K(Z_R \mid EIP), \tag{5.5b}$$

$$TP_S \rightarrow TP_R : TP_R \mid TP_S \mid Z_R \mid AT \mid EIP. \tag{5.5c}$$

The Encrypted original IP payload (EIP) is the original packet including IP header and payload. It is encrypted with a secret key K pre-shared between TP_S and TP_R. By encrypting the original packet and encapsulating it into the new IP datagram, we ensure confidentiality on the original payload as well as the host identities. We also introduce an Authentication Token (AT) which is placed in front of the EIP and contains a message authentication code (MAC) covering EIP and the destination zone identifiers. AT provides integrity over the entire packet except the outer IP header field which could be modified in transit.

The main difference to the Encapsulating Security Payload (ESP) on IPSec tunnel mode is that, rather than having site-to-site symmetric keys, we use site-zone pairwise keys. That is, the keys used for every triplet of $\{TP_{src} \mid TP_{dst} \mid zoneID_{dst}\}$ differs, providing a variety of unique symmetric keys even for the same pair of TPs. In addition, by conveying only $zoneID_{dst}$ in the header, zone pair information, which could lead to the potential disclosure of the zone structure and their transfer rules, is not exposed.

Verification The destination TP performs two steps of verification upon packet arrival: authentication and authorization. By extracting the quartet information from the header, TP_R first derives the corresponding symmetric key and recalculates AT to see whether the MAC matches the original AT value. This step is used to verify packet integrity as well as authenticity since only the two parties can derive the same symmetric key. If the match fails, it means that either the packet integrity is compromised or source authentication failed. Therefore, the packet is discarded.

To further verify authorization, TP_R obtains H_S and H_R by decrypting EIP, and verifies if H_S is authorized for the zone transfer towards H_R. Similar to TP_S, TP_R might send a request to its controller to acquire the authorization policy when the policy is missing in its database (Equations 5.3 and 5.4).

In principle, MONDRIAN is constructed under a single administrative domain such that all the core entities, i.e., TPs and controllers, are trustworthy. One of the main advantages of this trust model is that the authorization check performed by the sender side TP is also trusted. The receiver-side TP therefore does not necessarily verify the zone transfer authorization. Upon receiving a packet, TP_{dst} checks the authenticity of the packet, decrypts EIP, and forwards the original IP packet to the destination host. This trust model

could simplify the entire verification process significantly by omitting the authorization step which requires an additional challenge-response protocol to the controller, improving practicality for TPs running at small branches limited in operational resources.

5.3 Key Management

In order for TPs to create and verify authenticators based on symmetric cryptography we need a scheme to distribute keys amongst them. Ideally, the keys used for every triplet of $\{TP_{src} \mid TP_{dst} \mid zoneID_{dst}\}$ should be different. Additionally, ease of key management is a major concern as key distribution mechanisms in today's Internet, such as IPsec [31–33], are complicated and introduce management overhead. To alleviate these problems we propose a key management system based on a state-of-art key management system called PISKES [34].

Major modifications to PISKES introduced for Mondrian key management stem from the following requirements: first, in the context of network zoning, we require a high degree of confidentiality on top of authenticity to protect sensitive information. PISKES mainly targets authenticity for network entities, not confidentiality. Second, Mondrian does not trust ASes hosting zones for enterprises. PISKES's key establishment relies on the asymmetric key pair issued by each AS, that might cause a *man-in-the-middle* (MITM) vulnerability. Furthermore, it is unlikely that an AS wants to have a dedicated key-establishment service for each enterprise, which would require additional management overheads and deployments amongst ASes. Third, Mondrian requires key management at zone granularity, while PISKES is intended to support key exchanges at a higher granularity (e.g., per host or application). Thus, there is a design headroom which could simplify the architecture, reducing functional complexity and enhancing management scalability.

Driven by this, we redesign the PISKES's key-derivation architecture removing the AS dependency (i.e., the AS keys and the dedicated key servers at each AS), meaning that an enterprise has full control over distributed network zones and does not need to trust other ASes for inter-zone networking. Additionally, we simplify the key design to work at zone granularity, supporting faster key-derivation while providing the same level of security.

Key Hierarchy PISKES introduces a key hierarchy that allows services to dynamically derive symmetric keys in a fast and easy manner. We adapt the concept to support key derivation in the context of network zoning. The key hierarchy is as follows:

- 0th-level key: S_{TP} is the secret value generated by each TP individually.

- 1st-level key: a TP derives different symmetric keys for other TPs from the local secret value S_{TP}. The derived symmetric keys are called first-level keys and are calculated as

$$K_{A \to B} = PRF_{S_{TP_B}}(A), \tag{5.6}$$

  where A and B stand for the sending and receiving TP addresses and PRF is a secure pseudo-random function. Since only the receiver B can derive this key, it is necessary for the other party, in this case A, to fetch the shared symmetric key by contacting B. We note that, in contrast to PISKES, the arrow direction in the notation indicates the communication direction for which the key is used.

- 2nd-level key: from the first-level keys, second-level keys are derived to provide diverse symmetric keys for each zone within the same source and destination TP pairs. The second-level keys are calculated as

$$K_{A \to B:Z} = PRF_{K_{A \to B}}(Z). \tag{5.7}$$

  Z is the zone ID of the target zone where the destination host resides.

This hierarchical key structure benefits us in multiple ways: first, it delivers key diversity. Since a second-level key is bound to a specific destination zone, it enables zones to have different keys even for the same pair of source and destination TPs. Second, it is easy for TPs to efficiently derive the symmetric keys as all the required inputs (i.e., local and remote TP address, and destination zone) are contained in the packet header. In particular, a remote TP thus can derive the key directly from the packet header without a memory lookup. Finally, since all second-level keys are derived directly from first-level keys, the system scales linearly with the number of TPs, not the number of zones, achieving scalability.

Bootstrapping Keys Each TP randomly generates a local secret value S_{TP}, the root of the TP-specific key hierarchy. Since the first and second-level keys are derived from the secret value recursively, they inherit the randomness and secrecy of S_{TP}. Driven by this, we consider a non-deterministic random number generator. The randomly generated secret value never leaves TP premises and is frequently renewed, e.g., on a daily basis, to achieve perfect forward secrecy [35].

Key Establishment Key establishment precedes first data transmission. To establish a first-level key, the source TP initializes the key exchange protocol by sending a key exchange request:

$$req = A \mid B \mid ValTime, \tag{5.8a}$$

$$TP_A \to TP_B : \{req\}_{K_A^-}, \tag{5.8b}$$

where *ValTime* represents the validity period of the request. The request is signed with the requesting TP's private key K_A^-, meaning that the receiving TP can verify the authenticity of the request packet. Recall that, for authenticity of Mondrian entities, each TP certifies its own public/private key pair with a certificate issued by the trusted CA, e.g., the owner of the network zones.

Upon receiving the key exchange request, TP_B verifies the source authenticity and checks the validity of *ValTime*. If the request is valid TP_B derives a first-level key from the local secret value S_{TP_B} and replies back to the requester. The reply packet is formed as follows:

$$K_{A \to B} = PRF_{S_{TP_B}}(A), \tag{5.9a}$$

$$rep = \{B \mid K_{A \to B} \mid ExpTime\}_{K_A^+}, \tag{5.9b}$$

$$TP_B \to TP_A : \{rep\}_{K_B^-}, \tag{5.9c}$$

where *ExpTime* denotes the expiration time of the first-level key, K_A^+ is TP_A's public key used for encryption, and K_B^- is TP_B's private key to sign the reply packet. Finally, the requesting TP verifies the validity of the reply packet and caches $K_{A \to B}$ until it expires. Note that the key exchange protocol could be replaced with well-established key exchange protocols, such as IKE [36].

Ideally, a TP prefetches all the first-level keys for other TPs it wishes to communicate with. The TP acquires the list of active TPs from its controller and initiates the key exchange protocol for these TPs in advance. This is feasible because the number of TPs for an enterprise is surely limited; for example, the total number of branches that the Bank of America has in 2019 is approximately 4.6k [37]. Each branch would need just one TP, which means that a TP needs to prefetch 4.6k first-level keys. Nonetheless, on-demand key fetching is also possible. In particular, when the current first-level key expires in the middle of on-going data transmission or a new TP joins, a key exchange can be initiated.

TPs are also responsible for second-level key establishment. However, this does not require any key exchange protocol. Upon data transmission, source and destination TPs are able to dynamically derive the same second-level key for the destined zone from the shared first-level key as shown in Equation 5.7.

Chapter 6

Implementation

We now describe the implementation details of each component in MONDRIAN. We implemented a prototype that comprises a software-based gateway and controller. The main development language is golang 1.14.1[1], and we used SQLite3[2] for the database. To secure control-plane channels, we also leveraged TLS 1.3[3]. The prototype is publicly available[4].

Our implementation builds on top of the SCION architecture [38]. The implementation decision has been driven by the following reasons: i) SCION provides network programmability along with the separation of control and data plane, ii) SCION comes with an embedded PKI system that can be utilized for our key management system, and iii) the opensource version of the PISKES[5] system as well as a software-based gateway working with SCION are available, thus enabling rapid prototyping.

6.1 Translation Point

To implement a prototype of TP, we extend the SCION-IP Gateway (SIG)[6]. The main functionality of SIG is to encapsulate legacy IP packets into SCION packets and vice versa. In this context, a SIG acts as a gateway between an internal (legacy) network and an external (SCION) network. Since TP is designed as a gateway that bridges LAN traffic over WAN—the underlying inter-domain routing protocol is not relevant here—the functional aspects of TP meets with what SIG provides. To be integrated with SIG, TP medi-

[1] https://golang.org/doc/go1.14
[2] https://www.sqlite.org/releaselog/3_32_0.html
[3] https://tools.ietf.org/html/rfc8446
[4] https://github.com/chaehni/scion/tree/zoning
[5] https://github.com/netsec-ethz/scion/tree/scionlab_previousversion/go/lib/drkey
[6] https://github.com/scionproto/scion

ates between the UNIX socket and SIG socket, and performs zone transfer authorization and verification for all incoming/outgoing packets.

6.1.1 Modular Design

Packet Format In order to make TPs modular, all modules must agree on a common packet format on which they operate. The packet contains the raw IP packet read from the interface, as well as metadata that is required by the modules to perform their task. Modules can read and write all fields of the packet. In particular they can modify the raw IP packet. Listing 6.1 shows the packet format.

Listing 6.1: *The abstract packet format passed between the modules of a TP.*

```
1  // Packet contains a raw IP packet with additional meta data
2  type Packet struct {
3      Ingress     bool
4      SrcHost     net.IP
5      DstHost     net.IP
6      RemoteTP    string
7      DstZone     uint32
8      RawPacket   common.RawBytes
9  }
```

The `Ingress` field identifies a packet as either an ingress packet, coming from the WAN, or an egress packet that originated in the local network. `SrcHost` and `DstHost` reflect the source and destination IP addresses of the packet. `RemoteTP` designates the remote TP. For an ingress packet that is the source TP from which the packet was received, for an egress packet it is the TP to which the packet needs to be forwarded to. `DstZone` is the Zone ID of the zone to which `DstHost` belongs.

Module Interface A module is then simply defined as a type that handles this packet format. More precisely, a module implements a very simple `Module` interface.

Listing 6.2: *The interface all modules must implement.*

```
1  // Module is a single element in the pipeline
2  // that handles IP packets.
3  // Modules must be thread safe.
4  type Module interface {
5      Handle(Packet) (Packet, error)
6  }
```

As can be seen in Listing 6.2, modules call `Handle` to process packets of the aforementioned format and then return a new, potentially modified, packet and an error. The error is `nil` if the handling of the packet was successful.

Handler Chains Modules can be linked in a specific order to create a chain of handlers. These handler chains can then be registered for a specific use case. In particular, it is possible to create different handler chains for intra-domain and inter-domain zone transfers.

6.1.2 Implemented Modules

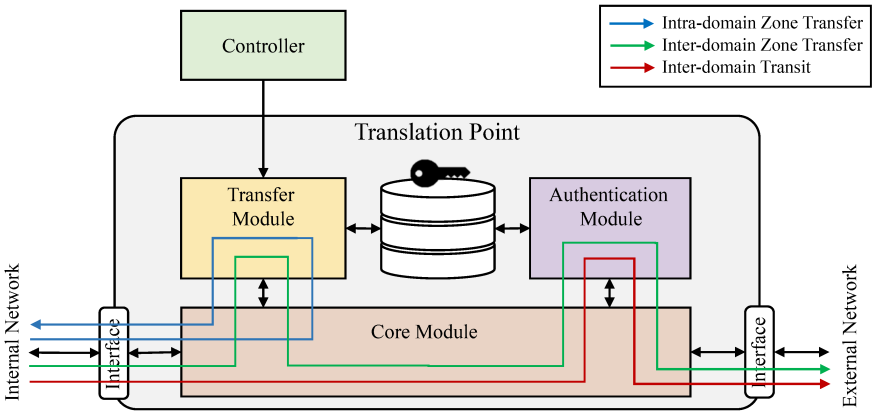

Figure 6.1: *An overview of the modularized TP implementation. Major use cases are indicated with colored arrows.*

In this section we describe the main modules that have been implemented to satisfy the protocol described in §5.2. Figure 6.1 illustrates the implementation details of the modularized TP design that consists of the three main modules: i) core module, ii) transfer module, and iii) authentication module.

Core Module The core module is the main loop of TP. It reads packets from the UNIX socket and redistributes them to the corresponding interfaces. More precisely, when receiving packets from the internal network, it retrieves metadata such as source and destination IP addresses (as illustrated in §6.1.1) from the raw packet and hands over to the transfer module. If the zone transfer is authorized ($return = 1$), the packet is then either forwarded back to the internal network or, in case the given destination is in a remote zone, once again handed over to the authentication module to be prepared for secure transmission. For packets coming from the external network, a TP first calls the authentication module for verification of the conveyed authentication token. Packets with invalid tokens are simply discarded.

Transfer Module The main objective of this module is to check the zone transfer rules. The transfer module communicates with its controller to

maintain a list of up-to-date zone transfer policies. To this end, it establishes a TLS channel with the controller, downloads policies, and populates the database. We implemented the transfer module to support different drop-in options using APIs.

- No-Op: This is for a setup in which no inter-domain zone transfers are required, but only inter-domain zone extensions.
- Standard: This mode would perform an authorization check for the requested zone transfer based on the source and destination IP addresses.
- Firewall: If needed, the module could be instantiated as a full-fledged firewall. This mode would be useful for cases where the firewall can not be replaced. should we discuss this further in the discussion section?

While the No-Op and Firewall options are both trivial to implement (the first simply accepts all packets while the latter forwards packets to the firewall instance), it is crucial for the Standard option to efficiently match IP addresses against a list of trusted IP subnets. In our implementation we leverage a compressed trie data structure (also radix trie or compact prefix tree) to check if a given pair of source and destination IP addresses are allowed to exchange data. Specifically, we use the open source cidranger[7] library. Since the lookup time of the trie is only dependant on the depth of the trie, which is fixed (32 bits for IPv4 and 128 bits for IPv6), the lookup time is constant with respect to the number of stored subnets. This ensures fast packet processing even if a TP has a large number of rules configured. Figure 6.2 shows a simplified example of how a trie stores IP subnets.

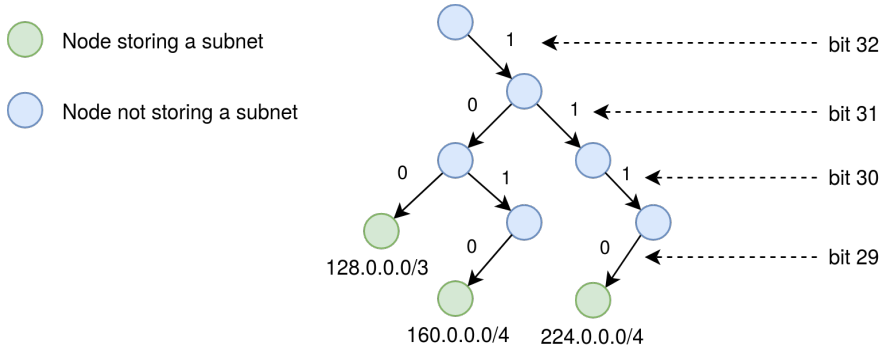

Figure 6.2: *A partial trie storing the subnets 128.0.0.0/3, 160.0.0.0/4, and 224.0.0.0/4.*

[7] https://github.com/yl2chen/cidranger

While the trie stores all configured IP subnets known to the TP, another data structure is required to store the acutal zone transfer rules, stating which zones, and by extension which subnets, are allowed to communicate with each other—recall that a subnet corresponds to exactly one zone. For this, we use a fast in-memory key-value store which maps to a given zone all other zones from which it accepts traffic.

Authentication Module For inter-domain packet transmissions, the authentication module issues authentication tokens (further discussed in §6.3) for outgoing packets. It (ideally) caches the first-level keys prefetched from other TPs and derives a second-level key to generate the authentication token. Inversely, for packets incoming from other TPs, it derives the corresponding 2nd-level key and verifies the attached authentication token. The authentication module is in itself modular by relying on an interface to fetch and derive keys. The key management logic (which is responsible for exchanging, caching, and evicting expired keys) is completely decoupled from the rest of the module. This has the advantage that the key exchange protocol can be changed independently, without a need to change other parts of the module. The interface that all implementations of such a key manager must satisfy is outlined in Listing 6.3.

Listing 6.3: *The KeyManager interface used to by the authentication module to fetch and derive keys.*

```
1  // KeyManager is a thread-safe key store managing L0 and L1 keys
2  type KeyManager interface {
3      // FetchL1Key fetches the level 1 key to be used to send
4      // data to remote. It returns the key and a bool
5      // indicating if the cached key was expired and a fresh
6      // key has been fetched from remote.
7      FetchL1Key(remote string) ([]byte, bool, error)
8      // FetchL2Key fetches the Level-2 key used to encrypt
9      // outgoing traffic
10     FetchL2Key(remote string, zone uint32) ([]byte, bool, error)
11     // Derive L1Key derives the level 1 key used to derive
12     // the L2 key.
13     DeriveL1Key(remote string) ([]byte, error)
14     // Derive L2Key derives the level 2 key used to verify
15     // incoming traffic.
16     DeriveL2Key(remote string, zone uint32) ([]byte, error)
17 }
```

6.2 Controller

We implemented the controller as a Web server written in golang with an SQLite database storing the zone information and transfer policies. The

controller offers an API which allows TPs to fetch zoning information via HTTPS GET requests.

APIs The endpoints of interest are:

1. `/api/get-subnets`
2. `/api/get-transfers`

Using these endpoints TPs fetch IP subnet and zone transfer rules. Important to note is that the controller only hands out the subset of the full set of rules which is required for the requesting TP to be operational. This minimizes the size of data transmissions and also improves security by not disclosing the full network view to every TP. For every call to the API the controller first verifies the authenticity of the caller before the request is forwarded to the corresponding handler. The handlers then load the requested data from the database and send it to the caller as JSON-formatted bytes.

Listing 6.4: *Abstraction layer function that inserts a zone transfer rules into the database.*

```
1  // InsertTransfers inserts premitted zone transfers into the backend
2  func (b *Backend) InsertTransfers(transfers map[int][]int) error {
3      stmt := 'INSERT INTO Transfers (src, dest) VALUES (?, ?)'
4
5      // do insertion in a transaction to ensure atomicity
6      tx, err := b.db.BeginTx(context.Background(), nil)
7      if err != nil {
8          return err
9      }
10
11     for src, dests := range transfers {
12         for _, dest := range dests {
13             _, err = tx.Exec(stmt, src, dest)
14             if err != nil {
15                 tx.Rollback()
16                 return err
17             }
18         }
19     }
20     tx.Commit()
21     return nil
22 }
```

Database The database consists of four tables (Zones, Sites, Subnets, Transfers), each describing one of the core elements of the architecture. The database schema is listed in Appendix A. An abstraction layer written in golang allows the controller to interface with the database using high-level calls. The abstraction layer makes use of transactional queries to ensure

consistency even in the event of errors. Furthermore, the abstraction uses prepared statements for insertions, deletions and retrievals of data. This protects against SQL-injections and improves the speed of queries. An example function of the abstraction layer is depicted in Listing 6.4.

6.3 Authentication Token

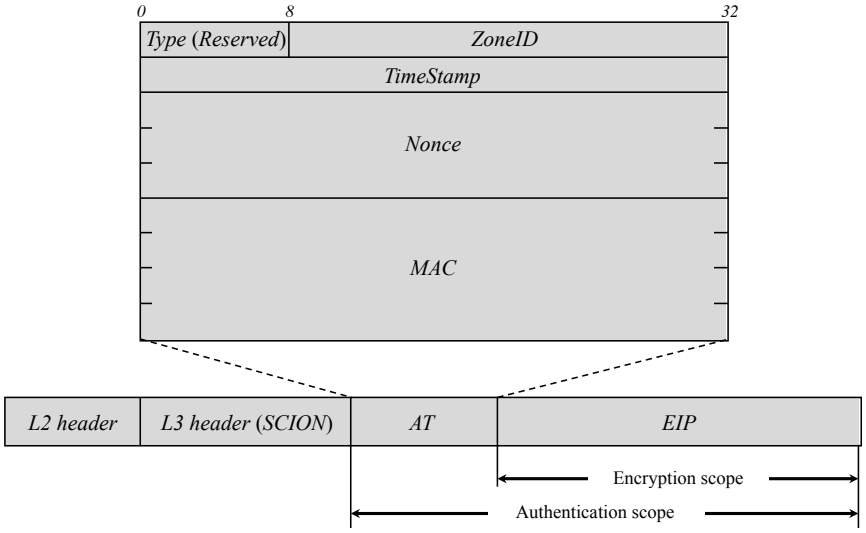

Figure 6.3: MONDRIAN *packet format for secure tunneling.*

The MONDRIAN packet format follows the IP tunneling conventions of encapsulating the original packet with a new outer IP header that indicates the two tunnel endpoints as the new source and destination. The original packet is encrypted and then authenticated along with the new packet header fields. Figure 6.3 shows the detailed packet structure and coverages of the confidentiality and integrity guarantee.

The authentication token starts with one byte of reserved space for a *Type* field. While currently unused this will be useful in the future for distinguishing different variations of the authentication token. *ZoneID* depicts the 3 byte long zone identifier of the destination zone. It is used by the receiving TP to derive the correct key for MAC verification and decryption. The next 4 bytes are occupied by a *TimeStamp* which is added by the sending TP. It is the Unix time at the point of sending the packet. The receiving TP uses this timestamp to reject replayed packets. The timestamp is followed by a *Nonce* of 12 bytes.

The nonce as well as the previous three token fields and the data to be

encrypted (*EIP*) serve as input to a *Galois/Counter Mode* (GCM) algorithm with an underlying *AES-128* block cipher as cryptographic primitive. This mode of operation is widely adopted for its performance as well as the capability to do *authenticated encryption with associated data* (AEAD). Here it provides authenticity over the header fields (*Type*, *ZoneID*, *TimeStamp*) and the data in *EIP* while *EIP* additionally also gets encrypted.

The 16 byte *MAC* generated by GCM is the last field in the authentication token. Both, the nonce and the MAC sizes follow the guidelines recommended by NIST SP 800-38D [39].

Listing 6.5 shows the interface that is used by the authentication module to create and attach an authentication token to an IP packet. Of particular interest are the functions `ToIR` and `FromIR`. `ToIR` takes the raw `packet`, a `key` and some metadata (`remote`, `dstZone`) and returns the encrpyted packet with attached tag and an error. The error is `nil` if the call was successful. Inversely, `FromIR` receives an encrypted packet with attached token (`cipher`) and a `key` and then returns the decrypted `packet` together with the token (`additionalData`) and an error. The error is again `nil` if the call to the function was successful.

Listing 6.5: *The Transformer interface used by the authentication module to create and verify authentication tokens.*

```
1   // Transformer transforms IP packets to and from intermediate
2   // representation
3   type Transformer interface {
4       // ToIR transforms packet to intermediate
5       // representation.
6       ToIR(remote string, key, packet []byte, dstZone uint32)
7           ([]byte, error)
8       // FromIR transforms cipher from intermediate representation
9       // back to a regular IP packet
10      FromIR(key, cipher []byte)
11          (additionalData []byte, packet []byte, err error)
12      // ResetState resets the nonce state for remote
13      ResetState(remote string) error
14      // GetZone retrieves the zoneID encoded in token
15      GetZone(token []byte) (uint32, error)
16  }
```

If the token format needs to be changed, only the implementation of the `Transformer` interface needs to be modified. The rest of the module can remain unchanged.

Chapter 7

Evaluation

Table 7.1: *Benchmark results for the transfer policy lookup (ns).*

# of Policies	Cache Hit	Cache Miss
100	307	236
1 K	405	264
10 K	423	293
100 K	497	375

Table 7.2: *Key derivation times for different network sizes (ns).*

# of Branches	1st-level Key	2nd-level Key
100	199	116
1 K	211	136
10 K	238	201
100 K	427	154

Table 7.3: *Processing times for the encryption/decryption for different packet sizes (ns).*

Packet Size (byte)	Encryption	Decryption
100	856	659
500	1082	747
1000	1338	850
1500	1557	950

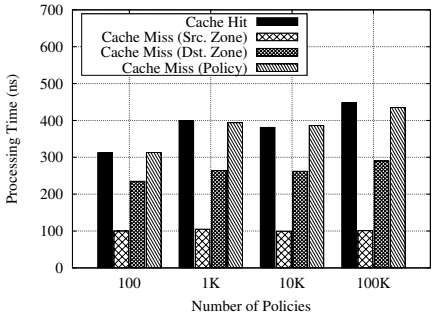

Figure 7.1: *Processing time for intra-domain zone transfer.*

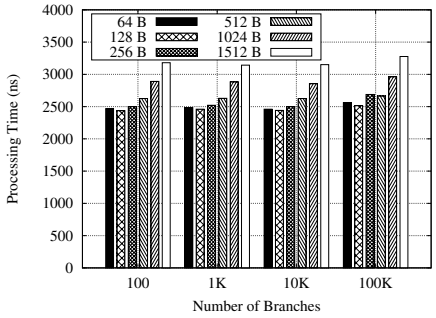

Figure 7.2: *Processing time on TP_S for inter-domain zone transfer.*

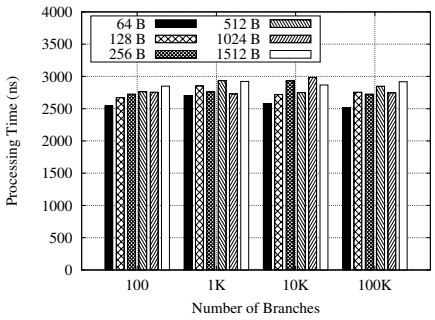

Figure 7.3: *Processing time on TP_R for inter-domain zone transfer.*

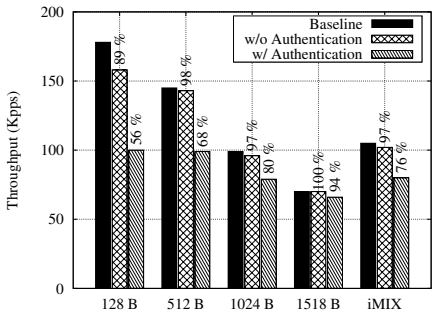

Figure 7.4: *Forwarding performance of TP for various size of packets.*

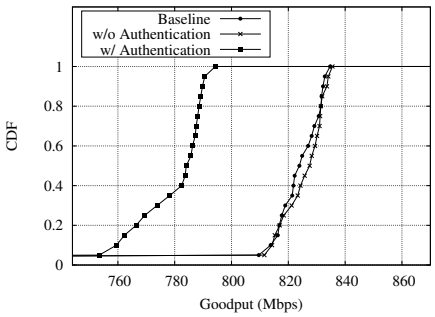

Figure 7.5: *CDF of goodput for 1400-bytes of maximum segment size (MMS).*

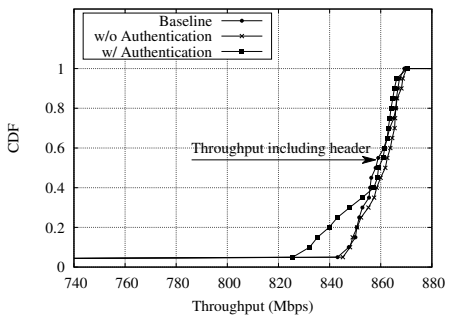

Figure 7.6: *CDF of throughput including extra header fields.*

7.1 System Benchmarks

We first conduct microbenchmark tests to evaluate the performance of our TP, including the derivation, packet authentication, and authorization. For a robust and reproducible evaluation, we leverage the standard benchmarking library `testing` officially supported by golang. The benchmarks are conducted on commodity machines equipped with an Intel i7 2.9 GHz CPU, 16 GB memory, and a 1 GbE NIC.

Authorization From a technical perspective, the zone transfer authorization consists of two tree searches followed by a database lookup; upon receiving the packet metadata from the core module, the transfer module first looks up the corresponding zone identifiers for the source and destination addresses, and then compares them to the zone transfer policies. The authorization performance is therefore dependent on the lookup time of the trees and the policy database.

Table 7.1 shows the benchmark results of database lookups for different numbers of policies. Note that each benchmark ran a couple million iterations and kept the mean value. The authorization check takes approximately 300 to 500 ns per packet, which is a notable result considering: i) a single lookup consists of two tree searches and a database lookup, ii) the result is from a high-level language implementation, and iii) the test set scales 1000 times. As expected, a lookup failure is commonly 24 to 31 % faster than a successful lookup. This implies that abnormal packets with invalid zone transfer requests can be quickly discarded.

Key Derivation We investigate the key derivation performance. Recall that the key derivation proceeds differently for sender and receiver. Since the receiver is capable of directly deriving the second-level key from the local secret which is relatively faster than sender-side key derivation, here we focus primarily on the sender-side key derivation which comprises two steps; fetching the first-level key from the key table (Eq. 5.6) and deriving the second-level key from the first-level key (Eq. 5.7). From a scalability perspective, we vary the number of branches (a TP for each) by increasing the number of stored first-level keys up to 100 K. Table 7.2 depicts the average results of benchmark tests iterated over a million times each.

We observe that the lookup time for the first-level key is independent of the size of the key table, taking between 154 ns and 161 ns. In addition, the size of the key table is also irrelevant for the second-level key derivation because second-level keys are directly derived from first-level keys. The results indicate that the processing time for a key derivation is less than 1 μs even in very large networks with 100 K branch sites, which is negligible considering network latency in today's Internet.

Authentication The additional processing time for packet encryption and decryption is shown in Table 7.3. In summary, it requires approximately 1.5 to 2.5 μs to authenticate various sizes of packets. We note that the processing overhead occurs for the majority of tunneling technologies that provide confidentiality for data transmission. The processing time can be minimized with implementations using the Data Plane Development Kit (DPDK) [40] or by leveraging hardware dedicated to cryptographic operations.

7.2 Network Benchmarks

So far, we evaluated the performance of each instruction newly introduced. Since a different set of instructions needs to be applied depending on the zone transfer use case, it is also important to investigate the overall network performance for handling different types of zone transfer packets. We now benchmark the actual network performance for both intra- and inter-domain zone transfer cases.

Latency Inflation Figure 7.1 illustrates the network benchmark results for the intra-domain zone transfer where the source and destination zones are within the same local network, such that the TP only performs zone transfer authorization. Since no cryptographic operations are involved, the additional latency is negligible (as a single legitimate zone transfer takes $\sim$ 500 ns). There might be an authorization abort due to a lookup failure that could be caused by the following three reasons: no matching source zone ID, destination zone ID, or zone transfer policy. In our prototype, the lookups are performed sequentially and thus there are different processing overheads (100 ns $\sim$ 450 ns) depending on when a lookup failure occurs. Nevertheless, in case there exists no valid zone-transfer policy, the packet will be simply dropped and therefore no additional latency is caused.

For inter-domain zone transfer cases, we benchmark the overall network inflation during TP operations including packet parsing, key derivation, authorization, and authentication. Figure 7.2 and 7.3 depict the processing delay from sender-side TP and receiver-side TP, respectively. From the results, we make the following observations: first, the overall latency inflation that Mondrian introduces is insignificant ($\sim 3\mu$s). Second, Mondrian scales well with the size of the network, i.e., the number of branches. We do not see any notable performance degradation ($\leq$ 200 ns). Third, the size of a packet is the primary factor for the latency increment as expected for all data-plane devices. The packet size has a small latency incremental factor of 1.28 (i.e, 2.4 μs to 3.8 μs). Lastly, we observe no significant bias in network performance between sender-side and receiver-side TPs.

Forwarding Performance We further investigate the actual forwarding performance for various packet sizes (128 B, 256 B, 512 B, 1024 B) including a representative mixture of Internet traffic (iMIX) [41]; we select the minimum packet size of 128 bytes instead of 64 bytes which is commonly considered to be the smallest packet size, because MONDRIAN's tunneling requires an at least 116 bytes long frame, i.e., Outer L3 header (40 bytes), AT header (36 bytes), and EIP (40 bytes). Figure 7.4 shows the results. The baseline is the forwarding performance without TP operations. The other bars represent the forwarding performance for intra-domain zone transfer (with authorization only) and inter-domain zone transfer (with authorization and authentication) respectively.

For 128-byte packets which demonstrates the highest packet rate, and thus requiring the most extreme packet processing, the intra-domain zone transfer exhibits a throughput degradation of only 11%. For other packet sizes we achieve a throughput of 97 $\sim$ 100%. These results are expected because TPs only perform zone authorization for intra-domain zone transfer packets, which increases processing delay by $\leq$ 500 ns. Considering that a typical intra-domain packet transmission usually shows a few milliseconds of latency, the additional delay is negligible.

On the other hand, the inter-domain zone transfer degrades the throughput by 44% for the smallest packets. Although the degradation diminishes as the packet size increases, the performance still degrades by 24% for the iMIX traffic. To investigate the main degradation factor, we compare the amount of transmitted data (goodput) and the total bits transmitted including all the network headers (throughputs) as shown in Figure 7.5 and 7.6. From the comparison, we observe the followings: i) the inter-domain zone transfer achieves a similar throughput to the baseline if the extra headers are considered, ii) the performance degradation is caused by not only the additional processing delay but also transmission delay of the extra headers, and therefore iii) MONDRIAN performs similar to today's tunneling applications while providing security policy enforcement for network zoning.

Chapter 8

Security Analysis

We analyze the security properties that MONDRIAN provides, considering the threat model introduced in §4.3. The attack classes described here are twofold: i) attacks to infiltrate restricted and highly secure zones without proper permission, and ii) disruption attacks that prevent availability.

8.1 Infiltration without Permission

One of the main attack objectives is to access valuable information assets protected by access constraints.

Man-In-The-Middle Attack To become "in the middle", an attacker could initiate independent communication channels with two TPs and relay messages between them. By using the attacker's public key for the channel establishments, the attacker is able to generate valid packets that can bypass the TP's authentication check.

MONDRIAN's PKI design prevents MITM attacks. A MITM attack can succeed only if the attacker convinces each TP that they are talking to each other. In the process of secure channel establishment, however, TPs authenticate each other using the certificates issued by the mutually trusted CA (e.g., enterprise). Since the attacker's public key cannot be certified by a valid certificate issued by that CA, a MITM attack will fail.

Packet Replay Attackers can observe valid MONDRIAN packets and then reuse them to transmit attack traffic. Nevertheless, the validity of the reused packet header will be compromised once the payload is changed—recall that the authentication scope covers the entire packet including AT and EIP as shown in Figure 6.3—and therefore attackers cannot successfully pass the authentication check at the recipient TP.

Brute-force Attack Another approach is to brute-force the key used for authentication or the MAC. As we are using 128-bit cryptographic keys and MACs, such attacks are currently infeasible. To achieve resilience to quantum computers, the key size would need to be doubled, however.

8.2 Denial-of-Service Attack

Exhaustion Attack Flooding the TP's authentication process could be an effective attack vector. Attackers may attempt to forward a large number of packets to the target TP in order to exhaust the TP's resources. Even if the attack packets contain invalid authentication tokens, the TP still needs to verify these tokens which wastes resources and prevents legitimate packets from getting through.

To be resilient to such attacks, we consider operating multiple TPs at the entrypoints of cooperative networks. Multiple TPs enables network operators to load balance and easily switch over to another TP in case of a link (or a TP) failure. In fact, many data centers and large enterprise networks already employ equal-cost multipathing (ECMP) [42, 43] along with multiple gateways and ToRs (Top-of-Rack switches) to provide reliable intra-networking services. In addition, to mitigate possible DoS attacks from the public Internet, Microsoft implemented a global ECMP infrastructure [44]. Path-aware networking along with multipath communication also enables active switching to different entry points, if some fail or are under DDoS attack. [45, 46].

Chapter 9

Practical Considerations

In this chapter, we discuss some practical considerations including functional and management aspects, along with various deployment scenarios describing how MONDRIAN can be realized on today's enterprise infrastructure.

9.1 NAT Devices

Multiple hosts connected through a NAT device appear as a single host to the TP. *Internal* NAT devices hardly pose a problem since the hosts under that NAT device are all subject to the same subnet and therefore belong to a single network zone. Network operators establish a security policy on the translated IP address, such that TPs are able to authenticate the zone transit requests from/to a host under the internal NAT device.

However, an *external* NAT device located in an external network, e.g., carrier grade NAT, could affect the TP's secure tunneling ability. The translated TP's IP address would cause a MAC verification failure—recall that each symmetric key binds to the triplet including TPs' IP addresses as described in §5.3. This, however, can be addressed by enforcing TPs to use their public IP addresses to derive the symmetric keys. The keys are still secure since the first-level key from which the pairwise keys are derived is exchanged with the CA-certified public keys. Discovering the translated TP address is also not a problem thanks to the controller informing senders about the recipient TP's address (see Protocol 3 in Figure 5.1).

A potential operational failure is where multiple TPs are behind the same NAT device. This would lead to TPs having identical keys from the view of remote TPs. One possible solution would be to use a unique TP identifier instead. Since all such TPs would be under one administration domain, assigning unique TP identifiers upon bootstrapping is feasible. Then, the

TPs convey their identifiers in the AT header field alongside the destination zone ID. This might slightly increase the size of the header, but does not degrade the security of the underlying authentication.

9.2 Tunneling Granularity

Secure tunneling can be realized in different granularities: i) site-to-site tunneling, ii) zone-to-zone tunneling, and iii) site-to-zone tunneling (the one used by Mondrian).

Similar to IPSec VPN, site-to-site tunneling provides strong guarantees about communication security and privacy for two tunnel endpoints. However, from a flexibility and manageability standpoint, having a site-to-site tunneling architecture is not ideal. Every tunnel endpoint needs to share a key with every other endpoint with which it wishes to exchange data. This adds state to the endpoints that needs to be kept in sync. Adding a new site requires an update on all the other sites that wish to communicate with the new site. Then, yet another layer of security middleboxes (e.g., firewalls) are required to perform zone transfer authentication since keys do not designate a specific zone.

An alternative way of providing authentication is to use one key per zone pair. In this model, a TP would sign the data on behalf of the zones. In case of a zone transfer, the TP would do the transfer and then use the key of the source/receiver zone pair. This approach has the benefit that sender and receiver get decoupled as in principle any site that contains a given zone is able to decrypt data destined for that zone. Adding a new site would be as easy as fetching the right keys for the zones used in this site. This process is independent of all the other sites. However, a receiver TP needs to be able to fetch the right keys for the zones, which means the zone transfer information must be visible, and thus attacker could potentially learn the zone structure of the observed network. Also, having a separate key per zone pair does not scale since the number of zones get big very quickly for large networks.

Driven by these considerations, we designed the new concept of site-to-zone tunneling, which represents a middle ground combining the advantages of the two approaches, the notion of secure tunneling and zone transfer authentication. The symmetric keys are distinguishable depending on the destination zone, while at the same time the zone-to-zone security policies are not being exposed. Thanks to the flexible and scalable key derivation scheme introduced in [34], the key establishment does not expand state, while still providing unique symmetric keys per zone. Furthermore, this scheme scales linearly with the number of sites, not the number of zones, which ensures scalability.

9.3 Distributed Controllers

Driven by the scalability and reliability issues inherent to a single physical controller, namely *single-point-of-failure*, logically centralized control-planes built on physically distributed instances find wide acceptance in the current practice. The most common approaches realizing distributed controllers can be broadly categorized into horizontal distribution [47, 48] and hierarchical distribution [49, 50]. Independent of which distribution architecture is used, we discuss location, coordination, and migration aspects of distributed controllers.

Location The notion of a logically centralized control-plane offers flexibility in network design and management. A key design choice is placement of the (distributed) controllers, which could impact to performance, reliability, and management scalability of a given network. There is comprehensive research on the controller placement problem considering practical issues from control latency to reliability, from cost-optimization to load balancing, and so on [51–53]. Among those, we are mainly interested in the latency performance indicator; that is the latency between a controller and regional forwarding devices.

The best latency is achieved when each branch site has its own controller. By a placement near local TPs, the controller minimizes the TP-controller latency for the zone transfer authorization protocol, allowing instant feedbacks for packet forwarding—we note that inter-controller communication for global coordination is commonly not latency sensitive. For the sake of control-plane security, the controller resides in a highly restricted zone to which only the local TPs and remote controllers have access. Although the per-site controller offers the best performance regarding policy enforcement for the data-plane, there might be a cost-efficiency problem for a large-scale network with thousands of branches.

Alternatively, we consider a sparse distribution model, e.g., on edge-cloud systems. Similar to today's cloud services, network operators running geographically distributed data centers can instantiate multiple controllers at the central point of regional branches. The control-plane latency overhead would be higher compared to the dense deployment model—if the data center edges are geographically diverse, the overhead could be minimized—but, in terms of cost-optimization and management scalability, it could be a more viable approach.

Coordination It is important to keep consistency in global coordination across the distributed controllers. Indeed, inconsistency in security policy might grant hosts with a low security clearance unauthorized access to highly restricted zones, resulting in unexpected information leakage and

eventually administration failure. With this in mind, we consider a consensus algorithm with strong consistency guarantees [54–56], where the security policy is dynamically shared/replicated across the distributed controller instances, ensuring concurrent policy enforcements toward the data-plane devices. There are numerous open-source projects, such as Consul[1], Apache ZooKeeper[2], and ETCD[3] available.

TP Migration To benefit from the distributed controller environment, a dynamic controller discovery process also becomes important. That is, TPs should be able to search a cluster of best candidates, diagnose the performances of control latency, and seamlessly migrate to the best controller. To this end, we consider a two-step migration process: i) TP-driven control channel initialization and ii) controller-driven TP migration.

TPs are responsible for establishing the first control plane channel with a controller. For example, a new TP (TP_{new}) has been configured to contact an initial controller acting as a first rendezvous point. The initial information contains the controller's IP address (C), the corresponding zone ID (Z_C), and the TP's IP address behind which the controller resides (TP_C). If the controller is located in a remote site (i.e., $TP_{new} \neq TP_C$), TP_{new} should connect with C through TP_C. Otherwise, e.g., C is within the same LAN or in public network, TP_{new} can directly send C a request for control-plane channel establishment.

Once the TP joined the network, the controller then initiates a migration process to find the best controller (C_{best}) for TP_{new}. Upon a migration request broadcasted by C, other controllers measure the possible latency to TP_{new} and reply back the results. Then, C elects C_{best} considering the latency measurements and the current load balance, and sends TP_{new} a `RoleChange()` request containing C_{best}, $Z_{C_{best}}$, and $TP_{C_{best}}$. Finally, TP_{new} swaps the best controller by establishing a new channel with C_{best}. The migration process is also applied when changes in the network are detected.

9.4 Nonce Reset

uncomment sections below

The same nonce must never be used twice with the same key, otherwise the security of the cipher significantly decreases. In theory it is easy to create nonces that fulfill this requirement. One can simply use a counter which is increased for every invocation of the AEAD algorithm. In real systems this is not so easy to achieve since machines can crash and lose their state,

[1] https://github.com/hashicorp/consul
[2] https://zookeeper.apache.org/
[3] https://github.com/etcd-io/etcd

Table 9.1: *Comparsion of different nonce creation strategies.* l *is the length of the nonce,* $s \leq l$ *is the subset of bits reserved for the random part of the nonce,* p *is the number of nonces generated and* r *is the number of unplanned nonce resets.*

	requires randomness	**requires non-volatile memory**	**overlap probability w/o resets**	**overlap probability for** r **resets**	**overlap**
counter only	no	no	0	1	full
randomness only	yes	no	$1 - \frac{(2^l)!}{2^{l*p}(2^l-p)!}$	$1 - \frac{(2^l)!}{2^{l*p}(2^l-p)!}$	partial
counter paired with randomness	yes	no	0	$1 - \frac{(2^s)!}{2^{s*r}(2^s-r)!}$	full
reset points	no	yes	0	0	-

specifically their nonce counter. Outlined below are some techniques to approach this problem.

- *Purely random nonce*: uses all bits of the nonce for randomness. Low probability for overlap but no guarantee even when no resets.
- *Counter paired with random sequence*: divides the nonce into a counter and a randomized part. Initialize the random part after every restart and increase the counter part for every packet. On reset start counter from zero with a fresh randomized part. In case of overlap all packets do overlap.
- *Reset points*: defines specific resets points for the counter which are stored on non-volatile memory (NV-memory). Increment counter in memory and write the next reset point to NV-memory when threshold is crossed. On crash restart counter from reset point on NV-memory.

Table 9.1 shows a comparison of the mentioned approaches.

9.5 Fast Failover

Operating multiple TPs with advanced ECMP-enabled layer 2 protocols (e.g., SPB and TRILL) is a viable network design that provides load balancing and enhanced resiliency against a TP or link failure, as we discussed in §8.2. If a TP is unable to continue data transmission, ECMP engages. It searches an alternative forwarding path considering cost equality and redirects flows through the new path, assuring continuous communication for end hosts [57]. This elastic migration during a transient network outage,

however, does not replicate the forwarding state of each flow, and thus might require another step of repopulating forwarding state. Upon packet arrival, the new TP fetches the zone transfer policy and caches it in the forwarding table. Note that for `established` connections this might cause unexpected packet drops if the first packet is a response. Nevertheless, the communication would continue as soon as the original sender notices the packet drop and requests a retransmission.

9.6 Incremental Deployability

To be incrementally deployable, Mondrian does not require changes from end hosts nor the local network infrastructure. Mondrian can take a supportive role by complementing already installed lines of defense such as firewalls, IPS, and IDS. For instance, traffic can be pre-filtered by TPs before it reaches firewalls located deeper inside the network. This approach favors an incremental deployment strategy. On the other hand, Mondrian can also be used as a single, all-in-one solution providing packet filtering, tunneling, and routing within one device. Such a deployment is especially interesting for small branch sites with much simpler network layouts. Here, Mondrian can drastically reduce the number of devices that need to be maintained.

Chapter 10

Conclusion

10.1 Summary

Network zoning has long been recognized as the cornerstone of secure network operation and management. In the current practice, operators realize network zones with network segmentation technologies and security middleboxes. As information systems become more dynamic from a topological, operational, and functional perspective, however, the conventional network-zoning architectures face new challenges in terms of scalability and flexibility. In this paper, we have shown that lightweight policy enforcement for inter-zone communication is achievable. Following a constructive approach with a cryptographic foundation, it is possible to create a proactive alternative to the mostly reactive systems presently used in network zoning. In conjunction with MONDRIAN, verification based on firewalls becomes simpler because firewalls would only process a limited amount of (filtered) traffic. MONDRIAN consequently reduces the number of management points of distributed networks while retaining a high degree of security.

10.2 Future Work

Focus on privacy

- hide zone id
- pad to full message length
- always send traffic (jondo networks)

Appendix A

Controller Database

```
1  CREATE TABLE Zones(
2  id INTEGER NOT NULL,
3  name TEXT,
4  PRIMARY KEY(id)
5  );
6
7  CREATE TABLE Sites(
8  tp_address TEXT NOT NULL,
9  name TEXT,
10 PRIMARY KEY(tp_address)
11 );
12
13 CREATE TABLE Subnets(
14 net_ip BLOB NOT NULL,
15 net_mask BLOB NOT NULL,
16 zoneID INTEGER NOT NULL,
17 tp_address TEXT NOT NULL,
18 PRIMARY KEY (net_ip, net_mask),
19 FOREIGN KEY (zoneID) REFERENCES Zones(id) ON DELETE CASCADE,
20 FOREIGN KEY (tp_address) REFERENCES Sites(tp_address)
21     ON DELETE CASCADE
22 );
23
24 CREATE TABLE Transfers(
25 src INTEGER NOT NULL,
26 dest INTEGER NOT NULL,
27 PRIMARY KEY (src, dest) ON CONFLICT REPLACE,
28 FOREIGN KEY (src) REFERENCES Zones(id) ON DELETE CASCADE,
29 FOREIGN KEY (dest) REFERENCES Zones(id) ON DELETE CASCADE
30 )
```

The controller database consists of 4 tables: `Zones`, `Sites`, `Subnets`, and `Transfers`. The `Zones` table contains all network zones known to the controller, identified by zone IDs. Additionally, a human readable description is attached. The `Sites` table holds all known branch sites with the addresses

of the corresponding TPs and a textual description. The `Subnets` table describes the configured IP subnets together with their zone membership and the TP behind which they are located. Finally, the `Transfers` table reflects the zone transfer matrix of allowed zone transfers.

Bibliography

[1] IEEE Std. 802.1q-2018: Local and metropolitan area networks-bridges and bridged networks. https://standards.ieee.org/standard/802_1Q-2018.html, 2018.

[2] Alain Mayer, Avishai Wool, and Elisha Ziskind. Fang: A Firewall Analysis Engine. In *Proceeding of IEEE Symposium on Security and Privacy (S&P)*, 2000.

[3] Luciana Obregon. Infrastructure Security Architecture for Effective Security Monitoring. *SANS Institute, Dec*, 2, 2015.

[4] Juan Deng, Hongxin Hu, Hongda Li, Zhizhong Pan, Kuang-Ching Wang, Gail-Joon Ahn, Jun Bi, and Younghee Park. VNGuard: An NFV/SDN Combination Framework for Provisioning and Managing Virtual Firewalls. In *Proceedings of IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, 2015.

[5] Jarrod N Bakker, Ian Welch, and Winston KG Seah. Network-wide Virtual Firewall using SDN/OpenFlow. In *Proceedings of IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2016.

[6] Tom Markham and Charlie Payne. Security at the Network Edge: A Distributed Firewall Architecture. In *Proceedings of DARPA Information Survivability Conference and Exposition II. (DISCEX'01)*, 2001.

[7] Tianlong Yu, Seyed Kaveh Fayaz, Michael P Collins, Vyas Sekar, and Srinivasan Seshan. PSI: Precise Security Instrumentation for Enterprise Networks. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*, 2017.

[8] Yaxuan Qi, Baohua Yang, Bo Xu, and Jun Li. Towards System-level Optimization for High Performance Unified Threat Management. In *Proceedings of the International Conference on Networking and Services (ICNS)*, 2007.

[9] HariGovind V Ramasamy, Cheng-Lin Tsao, Birgit Pfitzmann, Nikolai Joukov, and James W Murray. Towards Automated Identification of Security Zone Classification in Enterprise Networks. In *Hot-ICE*, 2011.

[10] Andrew Gontarczyk, Phil McMillan, and Chris Pavlovski. Blueprint for Cyber Security Zone Modeling. *Information Technology in Industry*, 3(2):38–45, 2015.

[11] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301, 2005.

[12] Su Hua Sun. The Advantages and the Implementation of SSL VPN. In *Proceedings of the 2nd IEEE International Conference on Software Engineering and Service Science*, 2011.

[13] Dennis Felsch, Martin Grothe, Jörg Schwenk, Adam Czubak, and Marcin Szymanek. The Dangers of Key Reuse: Practical Attacks on IPsec IKE. In *Proceedings of the 27th USENIX Security Symposium*, 2018.

[14] Alex X Liu and Fei Chen. Collaborative Enforcement of Firewall Policies in Virtual Private Networks. In *Proceedings of the 27th ACM Symposium on Principles of Distributed Computing*, 2008.

[15] IEEE Std. 802.1aq-2012: Local and metropolitan area networks-shortest path bridging. https://standards.ieee.org/standard/802_1aq-2012.html, 2012.

[16] R. Perlman, D. Eastlake 3rd, D. Dutt, S. Gai, and A. Ghanwani. Routing Bridges (RBridges): Base Protocol Specification. RFC 6325, July 2011. URL https://www.ietf.org/rfc/rfc6325.txt. Updated by RFCs 6327, 6439, 7172, 7177, 7357, 7179, 7180, 7455.

[17] B CheSwick and S Bellovin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley, 1994.

[18] Steven M Bellovin. Distributed Firewalls. *Login Magazine, Special Issue on Security*, 1999.

[19] Juniper. Security Products Comparison Chart. https://www.juniper.net/us/en/local/pdf/datasheets/1000265-en.pdf.

[20] Seyed K Fayaz, Tianlong Yu, Yoshiaki Tobioka, Sagar Chaki, and Vyas Sekar. BUZZ: Testing Context-Dependent Policies in Stateful Networks. In *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2016.

[21] Brendan Tschaen, Ying Zhang, Theo Benson, Sujata Banerjee, Jeongkeun Lee, and Joon-Myung Kang. SFC-Checker: Checking the Correct Forwarding Behavior of Service Function Chaining. In *Proceedings of IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2016.

[22] Yifei Yuan, Soo-Jin Moon, Sahil Uppal, Limin Jia, and Vyas Sekar. NetSMC: A Custom Symbolic Model Checker for Stateful Network Verification. In *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2020.

[23] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright. Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. RFC 7348 (Informational), August 2014. URL https://www.ietf.org/rfc/rfc7348.txt.

[24] D. Eastlake 3rd, T. Senevirathne, A. Ghanwani, D. Dutt, and A. Banerjee. Transparent Interconnection of Lots of Links (TRILL) Use of IS-IS. RFC 7176, May 2014. URL https://www.ietf.org/rfc/rfc7176.txt.

[25] Jonghoon Kwon, Taeho Lee, Claude Hähni, and Adrian Perrig. SVLAN: Secure & Scalable Network Virtualization. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*, 2020.

[26] Ed. Filsfils, C., Ed. Previdi, S., L. Ginsberg, B. Decraene, S. Litkowski, and R. Shakir. Segment Routing Architecture. RFC 8402, 2018.

[27] A. Bashandy, C. Filsfils, S. Previdi, B. Decraene, S. Litkowski, and R. Shakir. Segment Routing with the MPLS Data Plane. RFC 8660, 2019.

[28] R. Bush. Origin Validation Operation Based on the Resource Public Key Infrastructure (RPKI). RFC 7115 (Best Current Practice), January 2014. URL https://www.ietf.org/rfc/rfc7115.txt.

[29] R. Bush and R. Austein. The Resource Public Key Infrastructure (RPKI) to Router Protocol. RFC 6810, 2013.

[30] S. Kent. IP Encapsulating Security Payload (ESP). RFC 4303, December 2005. URL https://www.ietf.org/rfc/rfc4303.txt.

[31] D. Maughan, M. Schertler, M. Schneider, and J. Turner. Internet Security Association and Key Management Protocol (ISAKMP). RFC 2408, November 1998. URL https://www.ietf.org/rfc/rfc2408.txt. Obsoleted by RFC 4306.

[32] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). RFC 2409, November 1998. URL https://www.ietf.org/rfc/rfc2409.txt. Obsoleted by RFC 4306, updated by RFC 4109.

[33] C. Kaufman. Internet Key Exchange (IKEv2) Protocol. RFC 4306, December 2005. URL https://www.ietf.org/rfc/rfc4306.txt. Obsoleted by RFC 5996, updated by RFC 5282.

[34] Benjamin Rothenberger, Dominik Roos, Markus Legner, and Adrian Perrig. PISKES: Pragmatic Internet-Scale Key-Establishment System. In *Proceedings of the ACM Asia Conference on Computer and Communications Security (ASIACCS)*, 2020.

[35] C. Partridge. A Proposed Flow Specification. RFC 1363 (Informational), September 1992. URL https://www.ietf.org/rfc/rfc1363.txt.

[36] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, and T. Kivinen. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 7296, 2014.

[37] Statista. Leading banks in the United States in 2019, by number of branches. https://www.statista.com/statistics/935643/banks-with-the-most-branches-usa/.

[38] Adrian Perrig, Pawel Szalachowski, Raphael M. Reischuk, and Laurent Chuat. *SCION: A Secure Internet Architecture*. Springer Verlag, 2017.

[39] Morris Dworkin. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. NIST Special Publication 800-38D, 2007.

[40] DPDK Project. Data Plane Development Kit. https://dpdk.org.

[41] A. Morton. IMIX Genome: Specification of Variable Packet Sizes for Additional Testing. RFC 6985 (Informational), July 2013. URL https://www.ietf.org/rfc/rfc6985.txt.

[42] D. Thaler and C. Hopps. Multipath Issues in Unicast and Multicast Next-Hop Selection. RFC 2991 (Informational), November 2000. URL https://www.ietf.org/rfc/rfc2991.txt.

[43] C. Hopps. Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992 (Informational), November 2000. URL https://www.ietf.org/rfc/rfc2992.txt.

[44] Microsoft. Microsoft 365 Denial-of-Service Defense Strategy. https://docs.microsoft.com/en-us/office365/enterprise/office-365-microsoft-dos-defense-strategy.

[45] Spencer Dawkins. Path Aware Networking: A Bestiary of Roads Not Taken. IETF Internet Draft, https://datatracker.ietf.org/doc/draft-ietf-sidr-bgpsec-protocol/, Jun. (2018).

[46] Brian Trammell, Jean-Pierre Smith, and Adrian Perrig. Adding Path Awareness to the Internet Architecture. *IEEE Internet Computing*, 22(2), Mar. 2018.

[47] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, et al. Onos: towards an open, distributed sdn os. In *Proceedings of the 3rd Workshop on Hot Topics in Software Defined Networking (HOTSDN)*, 2014.

[48] Jan Medved, Robert Varga, Anton Tkacik, and Ken Gray. Opendaylight:

Towards a Model-driven SDN Aontroller Architecture. In *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, 2014.

[49] Soheil Hassas Yeganeh and Yashar Ganjali. Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications. In *Proceedings of the 1st workshop on Hot Topics in Software Defined Networks (HOTSDN)*, 2012.

[50] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, et al. Taking the Edge off with Espresso: Scale, Reliability and Programmability for Flobal Internet Peering. In *Proceedings of the ACM Conference on SIGCOMM*, 2017.

[51] Tamal Das, Vignesh Sridharan, and Mohan Gurusamy. A Survey on Controller Placement in SDN. *IEEE Communications Surveys & Tutorials*, 22(1):472–503, 2019.

[52] Tianzhu Zhang, Paolo Giaccone, Andrea Bianco, and Samuele De Domenico. The Role of the Inter-controller Consensus in the Placement of Distributed SDN Controllers. *Computer Communications*, 113:1–13, 2017.

[53] Mu He, Amir Varasteh, and Wolfgang Kellerer. Toward a Flexible Design of SDN Dynamic Control Plane: An Online Optimization Approach. *IEEE Transactions on Network and Service Management*, 16(4):1694–1708, 2019.

[54] Aurojit Panda, Colin Scott, Ali Ghodsi, Teemu Koponen, and Scott Shenker. Cap for Networks. In *Proceedings of the ACM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2013.

[55] Kevin Phemius, Mathieu Bouet, and Jérémie Leguay. Disco: Distributed Multi-domain SDN Controllers. In *Proceedings of the IEEE Network Operations and Management Symposium (NOMS)*, 2014.

[56] Xuanhua Shi, Haohong Lin, Hai Jin, Bing Bing Zhou, Zuoning Yin, Sheng Di, and Song Wu. GIRAFFE: A Scalable Distributed Coordination Service for Large-scale Systems. In *Proceedings of IEEE International Conference on Cluster Computing (CLUSTER)*, 2014.

[57] Y. Cai, L. Wei, H. Ou, V. Arya, and S. Jethwani. Protocol Independent Multicast Equal-Cost Multipath (ECMP) Redirect. RFC 6754, October 2012. URL https://www.ietf.org/rfc/rfc6754.txt.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.