Karlsruhe Institute of Technology

A Model for Centralized Management of Decentralized and Distributed Firewalls

Master's Thesis of

B.Sc. Janis Streib

at the Department of Informatics
Intitute of Telematics
Research Group Management of Complex IT-Systems

Reviewer:	Prof. Dr. Bernhard Neumair
Second reviewer:	Prof. Dr. Achim Streit
Advisor:	M.Sc. Julian Schuh
Second advisor:	Dipl.-Math. Klara Mall

15. February 2023 – 15. August 2023

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, 2023-08-11

. . . . . .

(B.Sc. Janis Streib)

Abstract

In a university context, large combined data center and campus networks often consist of different layers of components: From centrally managed routers, layer 3 switches and firewalls in its core to decentralized routers, servers and clients managed and operated by research institutes and other organizational units (OUs). To enforce global (organization-wide) security policies as well as local policies (e.g. defined by decentralized administrators) and requirements in the operation of those structures, large centralized firewalls are deployed. A policy in the context of this work defines how specific traffic flows between hosts or networks are handled by policy enforcement points like firewalls.

Centrally firewalling large networks requires high performance and expensive appliances capable of securing the whole network from outside of the network as well as enforcing local requirements and policies. Despite most modern components like routers, servers or many other network attached devices being able to firewall themselves ("host firewalls"), most local operators tend to rely on the centralized firewall instead of managing and configuring each individual device with their vendor or operating system specific interface, as this is the least labor-intensive approach from the device operator's perspective. In addition to that there might still be end systems which cannot be used for firewall rule enforcement for technical or organizational reasons.

In order to utilize this firewalling capacity in the end systems, this thesis proposes a model, which allows to define a network topology and a policy, which gets distributed among members of the network. To be able to integrate global as well as local policies defined by e.g. system administrators, this model includes trust level of hosts and trust requirements for policies. This allows the network operator(s) to assign trust levels to hosts which define, which firewall rule can be implemented on which systems. A firewall rule distribution algorithm which is proposed in this thesis then assigns the rules to the network in way, which allows the implementation of the policy on end systems if possible. It is shown experimentally, that this approach is able to be used to optimize networks regarding their firewall utilization and can be used to reduce the need of large, central firewall appliances. In addition to that, the architecture of the software created is able to be used as a centralized management system for firewall rules which can be used by the network's administrators as well as administrators of decentralized systems.

Zusammenfassung

Im universitären Kontext bestehen große kombinierte Rechenzentrums- und Campusnetze oft aus verschiedenen Komponenten: Von zentral verwalteten Routern, Layer-3-Switches und Firewalls im Kern des Netzes bis hin zu dezentralen Routern, Servern und Clients, die von Forschungsinstituten und anderen Organisationseinheiten (OEs) verwaltet und betrieben werden. Zur Durchsetzung globaler (organisationsweiter) Sicherheits-Policies sowie lokaler (z. B. von dezentralen Administratoren festgelegter) Policies und Anforderungen beim Betrieb dieser Strukturen werden große, zentralisierte Firewalls eingesetzt. Eine Policy im Kontext dieser Arbeit definiert, wie bestimmte Datenflüsse zwischen Hosts oder Netzwerken von Policy-Durchsetzungsstellen wie Firewalls behandelt werden.

Eine zentrale Firewall für große Netze erfordert leistungsstarke und teure Geräte, die in der Lage sind, das gesamte Netz von außen abzusichern und gleichzeitig lokale Anforderungen und Policies durchzusetzen. Obwohl die meisten modernen Komponenten wie Router, Server oder viele andere an das Netzwerk angeschlossene Geräte in der Lage sind, sich selbst mit einer eigenen Firewall zu schützen ("Host-Firewalls"), tendieren die meisten lokalen BetreiberInnen dazu, sich auf die zentralisierte Firewall zu verlassen, anstatt jedes einzelne Gerät mit seiner hersteller- oder betriebssystemspezifischen Schnittstelle zu verwalten und zu konfigurieren, da dies aus Sicht des Gerätebetreibers der am wenigsten arbeitsintensive Ansatz ist. Genauso kann es weiterhin Szenarien geben, in denen Durchsetzung von Firewallregeln auf Endsystemen technisch oder organisatorisch nicht möglich ist.

Um die im Netzwerk vorhandene Firewall-Kapazität in den Endsystemen besser zu nutzen, schlägt diese Arbeit ein Modell vor, das es erlaubt, eine Netzwerktopologie und eine Policy zu definieren, die auf das Netzwerk verteilt wird. Um sowohl globale als auch lokale Policies, die z. B. von SystemadministratorInnen definiert werden, integrieren zu können, beinhaltet dieses Modell eine Vertrauensstufe von Hosts und Vertrauensanforderungen für Policies. Dies ermöglicht es den NetzbetreiberInnen, den Hosts Vertrauensstufen zuzuweisen, die festlegen, welche Firewall-Regel auf welchen Systemen implementiert werden kann. Ein in dieser Arbeit vorgeschlagener Algorithmus zur Verteilung von Firewall-Regeln weist dann die Regeln teilnehmenden Systemen des Netzwerks in einer Weise zu, die die Verwirklichung der Regeln auf Endsystemen – sofern möglich – bevorzugt. Es wird experimentell gezeigt, dass dieser Ansatz dazu geeignet ist, Netzwerke hinsichtlich ihrer Firewall-Auslastung zu optimieren und den Bedarf an großen, zentralen Firewall-Appliances zu reduzieren. Darüber hinaus ist die Architektur der erstellten Software in der Lage, als zentrales Management-System für Firewall-Regeln eingesetzt zu werden, das sowohl von NetzwerkadministratorInnen als auch von AdministratorInnen dezentraler Systeme genutzt werden kann.

Contents

List of Figures

List of Tables

1. Introduction

Computer networks, especially on large sites like university campuses, consist of a large number of different systems, starting from end systems by users to servers, routers and dedicated firewalls. To protect systems from traffic or accesses from or to network resources which might pose a security risk, firewalls are used. Firewalls are components in a network, which are able to filter traffic based on policies defined by their operators. The network is often operated by a single, central entity. Modern operating systems like Linux include firewall software implementations in their kernel as well. This allows operators of end systems to implement firewall rules for their end systems directly on the machine itself. Most relevant end systems are servers but may also include other types of devices like domain specific appliances.

In theory, most servers could protect themselves by implementing firewall rules using this capability, if the desired policy is known by the operator. This could reduce the required capacity of centrally deployed firewalls in certain network architectures, if those hosts can be attached to the network in a way that bypass the firewalls. Such an approach allows the network to scale fast, as not much central firewalling infrastructure is required in order to cope with the additional traffic. While this increases flexibility in the distribution and design of the network's firewall architecture, more firewalls on different systems with different firewall implementations significantly increase the management effort required to maintain the security requirements in the network. In addition to that, servers and other systems in the network are not necessarily administrated by the network administrators but by other teams within or outside the network administrator's organization or organizational unit. Those systems are referred to as "decentrally operated".

Some network operators may also want to define and enforce policies globally to meet security objectives. One example of such a policy is the limitation of e-mail related traffic in order to prevent spam messages to originate from the network. Decentrally operated systems should not be able to bypass this kind of policy.

As networks grow in size, managing all firewalls and ensuring the correct enforcement of the desired policies can become an unmanageable and error-prone task if done manually. This is why operators of both centrally and decentrally operated systems tend to use monolithic central firewalls instead. Those systems provide a unified interface to manage all firewall rules centrally. The downside of this approach is high costs for those systems, especially with increasing bandwidth requirements, as the central firewall is the most likely bottleneck of the network. Additionally this structure is less flexible, if the structure or size of the network changes or if additional sites are added to the network. Such firewalling appliances need to be overprovisioned in the point of time of the acquisition of the hardware in order to accommodate future growth of the network.

This work aims to make a contribution to solve this management problem (firewall distribution problem) by creating and evaluating a model and accompanying algorithm to

distribute firewall policies in a network which contains multiple firewalls on both centrally and decentrally operated systems while still fulfilling security objectives defined by global and local firewall policies.

This thesis is structured as follows: First, preliminaries and terminology required for this work are outlined in chapter 2. Subsequently, the previously outlined problem is further introduced and analyzed in chapter 3. Chapter 4 outlines a set of use cases, that is networks and policies on which the model and algorithm may be applied. Afterwards the models created for this work are introduced in chapter 5 and the algorithm using that model is defined in chapter 6. The algorithm is then evaluated in chapter 8 using the use cases defined in chapter 4. After that, work related to this thesis is summarized in chapter 9. Finally the findings of this work are summarized and future work is outlined in chapter 10.

2. Preliminaries

Computer networks historically evolved from the need of sharing and accessing resources between computers [CK74]. The American Defence Advanced Research Projects Agency (DARPA) therefore developed mechanisms to achieve this goal. Those mechanisms are still mostly present in today's network and internet architecture. One basic concept is that networks are packet switched networks. This means, that data is put into a packet with an defined length which is labeled with its destination address. The packet is then sent into the network, where it is delivered to its destination (in comparison to circuit switched networks where a communication line between two hosts is established which is used exclusively by the source/destination until the line gets terminated). In general, this allows a more flexible network design, as the path of packets through the network can change dynamically and packet-switching components do not need to hold any state in order to forward a packet [Cla88]. This process requires a protocol, which standardizes the way computers in the network are addressed as well as mechanisms which allow the packet-based inter-network communication with other computer networks via gateways. This protocol is called Internet Protocol (IP) which was proposed in 1974 [CK74; Cla88]. IP is designed in a way which deliberately leaves the implementation of the physical network as well as the content of those packets open. This allows on one hand the implementation of IP upon different network types (for example Ethernet) and on the other hand the implementation of more advanced protocols on top of IP (for example the Transmission Control Protocol, TCP).

This layered architecture was formalized by ISO 7498 (Open Systems Interconnection model, OSI model) [IEC7498-1] in 7 abstract layers. For the context of this work, only the first (lower) four layers are relevant:

1. **Physical Layer**: "The Physical Layer provides the mechanical, electrical, functional and procedural means to activate, maintain, and deactivate physical connections for bit transmission between data-link-entities. A physical-connection may involve intermediate open systems, each relaying bit transmission within the Physical Layer. Physical Layer entities are interconnected by means of a physical medium." [IEC7498-1, section 7.7.2]

   This covers standards for physical communication on wires, fibers or wireless communications.

2. **Data Link Layer**: "The Data Link Layer provides functional and procedural means for connectionless-mode among network-entities, and for connection-mode for the establishment, maintenance, and release data-link-connections among network-entities and for the transfer of data-link-service-data-units. A data-link-connection is built upon one or several physical-connections." [IEC7498-1, section 7.6.2.1]

Protocols like Ethernet are connectionless OSI-Layer-2.

3. **Network Layer**: "The basic service of the Network Layer is to provide the transparent transfer of data between transport-entities. This service allows the structure and detailed content of submitted data to be determined exclusively by layers above the Network Layer." [IEC7498-1, section 7.5.3.1.1]

   By this definition, IP is an OSI-Layer-3 protocol.

4. **Transport Layer**: "The transport service provides transparent transfer of data between session-entities and relieves them from any concern with the detailed way in which reliable and cost effective transfer of data is achieved." [IEC7498-1, section 7.4.2.1]

   By this definition, TCP or UDP are an OSI-Layer-4 protocol.

Layer-3 network members (i.e. network members with an IP address) which are connected in the same Layer-2 domain can communicate directly with each other by utilizing protocols which allow the resolution of IP addresses to Layer-2 addresses. In order to designate, which part of an address belongs to the same subnetwork (e.g. within a Layer-2 network), Classless Inter-domain Routing (CIDR, [RFC4632]) is used. With CIDR, addresses are split into a prefix (the "network address") and a host part. If both the sender's and destination's addresses are within the same network, a packet can potentially be delivered directly via its link network after resolving the destination's Layer-2 address. If the destination address is not within the Layer-2 domain, other networks can be reached by sending the packet to a gateway which must also be reachable through the link network.

A gateway (from a sender's perspective or "router" from a global perspective) is connected to multiple Layer-2 domains and provides an IP interconnect between those networks. This process is called forwarding. The coordination of this process through one or multiple gateways to interconnect all desired networks is called routing [RFC791]. Each host which wants to communicate with hosts outside its networks needs to keep a table of network-gateway mappings which defines, which gateway should be used for which network(s). This table is commonly called routing table.

While the exact routing decision process is an implementation-specific detail of hardware or software routers, the following basic convention on routing decisions on incoming packets can be found in most implementations [IEN30; RFC823]:

1. Upon an incoming packet, inspect its destination address.

2. Consult the router's routing table for the longest (i.e. most specific) prefix matching the packet's destination.

3. Forward to the destination associated to the prefix in the routing table.

The longest prefix matching can be implemented efficiently in hardware for example by using Ternary Content Addressable Memory (TCAM) [RM04]. A routing table can contain other hosts (which must be reachable by the router) or one of the router's interfaces as destination.

The Internet Protocol is used in two versions in today's internet: Internet Protocol version 4 (IPv4) as defined in [RFC791] and Internet Protocol version 6 (IPv6) as defined in [RFC8200]. One of the main differences of those IP versions is the address length: Whereas IPv4 addresses contain 32 bits (which equals 2^{32} possible addresses), IPv6 addresses contain 128 bits (equals 2^{128} possible addresses). This allows to accommodate the massively growing number of devices in today's internet without the use of network address translation (NAT). In IPv6, conventionally addresses are written using hexadecimal numbers which are grouped in 4-digit nibbles, separated by colons. Continuous zero-nibbles can be compacted exactly one time by using a double colon.

An IPv6 address for a network interface in CIDR notation can look like this:

```
2001:0DB8:1:12::1/64
```

In this case, the prefix length is 64 bits. To find the prefix (network address) of the subnet, the first 64 bits of the address is taken and then right-padded with zeros. For the previous example, the prefix is:

```
2001:0DB8:1:12::
```

This means that all hosts within this prefix should be reached directly via their Layer-2 link from other hosts within this prefix.

In this work, only IPv6 addresses are used in examples, although they can be interchanged on an abstract level as implementation details of IP protocols are not relevant for the scope of this work.

One very common example of a higher-layer protocol "on top of" (OSI Layer 4) IP is TCP as historically introduced in [CK74] and defined in [RFC9293]. TCP does provide packet ordering, retransmissions in case of packet loss and error correction, which is not provided by IP or by the User Datagram Protocol (UDP) [RFC768] which is defined as a Layer-4 protocol as well. These features are achieved by first establishing a connection using a three-way handshake and afterwards by explicit acknowledgements of each segment. In order to establish a connection, the initializing host first sends a packet with the SYN-flag set. The communication partner answers this packet by sending a packet with SYN and ACK-flags set in order to establish the connection in both directions and acknowledging the receipt of the SYN packet. Finally, the initializing host acknowledges the receipt of the SYN packet by sending an ACK. During the initialization, the general state of the connection is called "related", afterwards its "established". When using firewalls, the firewall needs to track the state of those connections in order to selectively allow bidirectional communication as a result of a connection establishment. This state tracking is also required for selectively allowing bidirectional UDP traffic, although not based on connections but on other attributes of the packets.

Firewalls are (system) components used to police network traffic. In this work, firewalls are considered to handle IP (Layer 3) traffic with optional (depending on the firewall's capabilities) additional properties and attributes from Layer-4 protocols like TCP, although other types of firewall exists which can handle Layer-2 traffic or application-level traffic. They enforce policies by applying rules upon the traffic. Conventionally, firewalls are a part of IP routers. They apply firewall rules on traffic through them and decide the fate

of the packet, for example to accept the packet and forward it or to drop it. A simple way to use firewalls is to create rules which contain conditions (matchers) and associated actions (like accept or drop). Upon packet forwarding, the firewall inspects packets and can match conditions based on direct attributes of the packet currently inspected. This kind of firewalling does not require additional knowledge beyond the information provided by the packet itself – they are stateless. This works for simple policies. A typical scenario in current networks is to protect hosts from access originated outside the network. When implementing this using stateless firewalls, all packets from the outside are dropped. This does not account for bidirectional communication for example when using TCP though. In order to allow bidirectional communication without allowing access from outside in general, the firewall needs to track the establishment of connections and selectively allow packets which are related to the connections towards inside hosts. This makes the firewall stateful, as it needs to keep this state for a longer time during the communication of the hosts in order to be able to identify packets as part of a connection previously established. As most modern networks contain such scenarios, e.g. to protect primitive end systems by users from the internet while allowing to access resources in the internet, stateful firewalls are often a requirement in modern networks. If this state gets lost, the connection cannot continue and is terminated in a timeout after for example TCP segments don't get acknowledged anymore because packets may be dropped by the firewall as it does not recognize the packets as part of a connection anymore.

Most routers are not able to track state though. Especially so called "layer 2.5" or "layer 3 switches" are able to act as a router, but only support simple access control lists without stateful connection state tracking (for example the Cisco Nexus 9000 series[1]). To achieve stateful firewalling, often additional appliances are required. Host firewalls are usually able to track "their" connections and are therefore stateful firewalls.

As stateless firewalls are just defined by their configuration, redundancy is easily achieved by placing multiple firewalls with the same configuration in the network. In case of an outage, the traffic can get dynamically routed to the other firewall. This can be achieved by using dynamic routing protocols. When building redundancies for stateful firewalls, the design needs to deal with trade-offs, as the CAP theorem applies. The CAP theorem described in [Bre00] and proven in [GL02] states, that any stateful distributed system can only fully achieve two of the following three properties:

- **C**onsistency: The data on all member of the distributed system is consistent.
- **A**vailability: The system is fully available, even one member fails.
- Network **P**artition Tolerance: The distributed system is able to cope with loss of connectivity between members which divides the system into multiple partitions.

One variant is to have one primary firewall which handles the traffic under normal conditions and one backup firewall with the same configuration. In case of an outage of the primary firewall, the traffic gets rerouted to the secondary firewall. This causes established or related connection states (and thus the connections themselves) established

[1] `https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/datasheet-c78-736967.html`

through the primary firewall to get lost, as the connection states are only stored on the primary firewall (no consistency, full availability and partition tolerance). Another variant is to constantly synchronize the connection states between the two (or more) firewalls, rendering each firewall able to handle all existing connections at any time (high consistency, full availability, no partition tolerance). This increases the complexity of the firewalls though, especially if high bandwidths, high connection counts and low latencies during packet processing are required which causes that those components need to be implemented in hardware to reduce time overhead as far as possible. The higher complexity causes the cost of large stateful firewalls to increase in comparison to stateless firewalls.

3. Problem Analysis

In this chapter the basic issues posed by the firewall distribution problem are described in section 3.1 and analyzed in section 3.2.

3.1. Problem

As introduced in chapter 1, this work aims to provide a solution for the management problem posed by applying firewalling policies to a network, which contains a multitude of devices that are theoretically capable of implementing firewall rules (policy enforcement point, PEP). As assigning firewall rules to each PEP manually may be error-prone, especially if the network grows in size, an automatic distribution is desired. The management complexity is dependent on the number of hosts contributing to the fulfillment of a policy: Whereas a fully centralized approach with one firewall covering all policies requires only the maintenance of rules on one device, a (theoretical) fully distributed approach where rules are implemented on all end systems require the rules to be maintained on all end devices. The first approach simplifies the management problem and maximizes the required central firewalling capacity whereas the second approach complicates the management problem and minimizes the required central firewalling capacity.

In order to be able to minimize the required central firewalling capacity, the definition of data structures, which allow to model the network as well as firewall policies, is required. Those must be suitable to allow the automatic distribution of rules among PEPs in a way which

1. ensures the fulfillment of the (abstract) policies in the network, and
2. does fulfill the policies in a way which allows as much of the rules as possible to be implemented on end systems in order to facilitate as much of already available firewalling capacity as possible. This may allow network administrators to reduce the need to operate large and costly central firewalls.

In a theoretical "perfect" network, every end device could firewall itself and thus no central firewall would be required at all once the distribution problem has been solved. Real networks introduce several further problems:

1. It is not possible to implement firewalls on all end systems. This can have several reasons: For example a device may lack technical abilities to implement packet filters in their hard- or software like internet of things (IoT) devices, which are reduced to minimal networking functionalities in order to reduce cost or to save power. Other devices might be outside the network operator's administrative domain like private devices by users.

This requires firewalling in the device's network's routers.

2. Not all PEPs capable of implementing firewalls support all features required to implement some policies. For example some routers only support stateless firewalling, see chapter 2.

3. Not all PEPs can be fully trusted to implement all rules. This is especially true if PEPs in the network are operated by third parties. Those decentral PEPs may not be trusted to implement some global security-relevant policies which must be implemented in all cases. Also some infrastructure components like end systems may be more prone to compromise, either by external or internal attacks. Some policies are required to be tamper-proof.

4. Operators of decentral infrastructure might need to facilitate central firewalls. This is required e.g. if decentral operated devices which are not able to implement firewall rules are placed into the network without a dedicated firewall operated by the decentral administrator who demands certain policies to protect those devices. This makes a centralized firewall management necessary in order to allow decentral operators to "contribute" their policies to the central firewall.

Therefore, the full implementation of firewall rules in end systems is not realistic given the current network requirements and intermediate, centrally managed firewalls are still required. Thus, a model for centralized management of decentralized and distributed firewalls is required.

The firewall distribution problem is defined as follows:

Definition 3.1 (The Firewall Distribution Problem) The firewall distribution problem *is defined by a Policy $\mathcal{P}$ and a network $\mathcal{N}$. The goal is to find a distribution $\mathcal{D}$ of rules r derivable from $\mathcal{P}$ among the nodes $n \in \mathcal{N}$ such that*

- *All rules relevant for hosts within $\mathcal{N}$ are fulfilled*
- *rules are implemented on end systems where possible*

3.2. Analysis

When deploying policy definitions as rules on policy enforcement points (either manually or automated), the rules for the firewall need to be phrased from the perspective of the firewall itself. For example if network `A` and network `B` are both routed on Firewall `F` and `A` must not communicate with `B`, `F` needs to implement the following rule (see figure 3.1):

> For IP forwarding, drop packets where the source address is in network `A` and the destination address in network `B`.

The same goal can also be reached by implementing rules on *each* host's firewall in network `B`:

> Drop all incoming packets whose source address is from network `A`.

Yet another approach would be to block all outgoing packets on *all* hosts in network A whose destination address lies within network B:

> Drop all outgoing packets whose destination address is in network B.

This approach is usually very uncommon, as the implementation of this kind of rules requires a lot more management and tends to be more error-prone than implementing the rules on the "path" towards the communication target or on the destination itself, especially with growing complexity of policies. In addition to that, all senders need to be fully trusted by the network operator in order to allow this kind of implementation, as they can be easily bypassed by senders.

In general it is assumed that measures against address spoofing are in place on all routers.

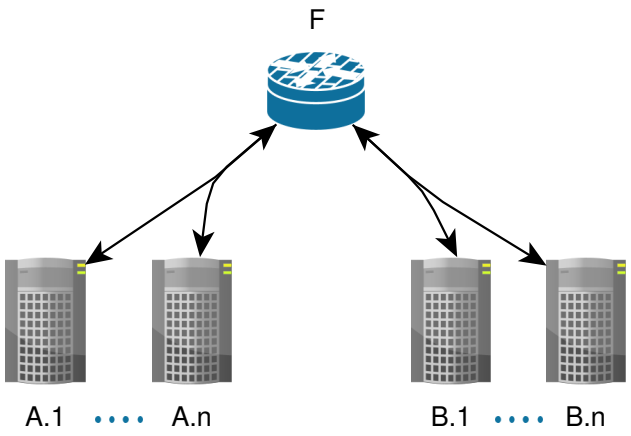


Figure 3.1.: Sample network with two networks A and B, both containing *n* hosts routed on firewall F.

All those previous scenarios are possible *functional indistinguishable* implementations of the same rule, but result into different realizations of the policy.

Definition 3.2 (Functional Indistinguishability) *Two firewall implementations in a network are* functionally indistinguishable *if every data flow from all nodes and to all nodes covered by the policy definition(s) result in the same firewalling actions.*

If two firewall implementation on a network result in the same firewalling actions but can be distinguished by other means (e.g. timing), they are still *functional indistinguishable.*

As this work aims to distribute firewall rules among the network and could be used to apply the same rules on different network topologies, all rules need to be phrased topology-independent. This includes that all policy definitions cannot include assumptions for incoming or outgoing interfaces of individual hosts or the ability of one host to reach another via the network.

Assumption 3.1 (Policy-Topology Independence) *All firewall rules need to be phrased independently of the network topology.*

Many firewall definition languages like Linux iptables are position dependent. This means that rules to match a packet are evaluated in a defined order. The first matching rule gets executed and the evaluation stops. This allows to add exceptions to broad rules without modifying the base rule. For example if there is one specific host `A.1` in network `A` from the previous example which must be able to communicate with hosts in `B` but all other hosts in `A` must not, firewall `F` can implement the policy using following rules:

1. If the packet's source address matches the address of `A.1` forward the traffic.
2. Drop packets where the source address is in network `A` and the destination address in network `B`.

This causes a positional dependency of the rules. As the first rule is evaluated first, all packets from `A` but `A.1`'s are dropped. If the order is changed (i.e. the second rule is evaluated first), all packets from `A` including those from `A.1` are dropped despite the existence of the second rule. As the policy's rules may get distributed by the algorithm on different policy enforcement points depending on arbitrary aspects like their firewalling capability, positional dependent firewall rules may be executed in an incorrect order and thus the policies may be implemented differently than intended, causing ambiguous results (same traffic behaves differently on different deployments of the same policy). For example if `F` implements the second rule and all hosts in `B` the first rule, all traffic from `A` including `A.1` is still blocked, as network `A` is routed on `F` and thus packets from `A` necessarily need to pass though `F`. The previous example policy could be changed into an unambiguous policy:

- Drop packets where the source address is in the upper-exclusive address range from the first address of network `A` up to `A.1`.
- Drop packets where the source address is in the lower-exclusive range from `A.1` to the last address of network `A`.

This phrasing of the policy allows the implementation on all $3^2 = 9$ combinations of hosts and rules on the path (assuming every host is capable of implementing each rule) like in the first example.

Assumption 3.2 (No Interdependent Rules) *In order to be able to distribute rules in a way which does not create a deviation from the intended policy, firewall rules must not depend on other firewall rules.*

A path for a rule is defined by the hosts matched by the source addresses as starting points and hosts matched by the rule's destination addresses as destinations. Usually firewalls are an integrated component of a router and in this case packet filters can be applied before, during or after routing decisions. Therefore the devices in between source and destination are defined by the possible routes a packet can take while being routed through the network (see chapter 2). It is possible that a system has multiple gateways. Those may be set dynamically using routing protocols or statically using metrics. As a result, a network model needs to include all *possible* paths through a network.

The naive way of distributing firewall rules in a network would be to implement all rules on all hosts. This is not possible, because not all PEPs may be able to implement (all) rules and this may also cause significant overhead during packet processing in the existing firewalls. When calculating a firewall distribution which does not require to implement all rules on all hosts in the network, a suitable minimization needs to be found which checks if the desired policy is fully enforced. Otherwise, a policy could not be fully enforced which could create security risks. In this work it is assumed that a policy always gets implemented correctly on all firewalls as intended, such that the fate of each packet doesn't change when the policy is applied on every element of each path respectively. A host-specific rule implementation on actual firewalls is considered out of scope for this work. If all possible paths a packet matching a policy between entities in the network model can take are known, it is sufficient to implement a deny rule on every path at least once (including the end systems of the path) in order to fully implement the rules despite the existence of multiple paths. Accept rules need to be implemented on *all* nodes on *all* paths.

Lemma 3.3 (Path Behaviour of Rules) *Two variants of path behaviour of firewall rules can be observed:*

1. *On each existing path between a rule's source and destination, a firewall deny rule needs to be implemented* at least once *to reach* correctness.

2. *On each existing path between a rule's source and destination, a firewall accept rule needs to be implemented* on all nodes *to reach* correctness.

Not all routers in a network may be able to implement a firewall at all or may only support stateless firewalling. In case a router is only able to route without implementing any packet filters the router just forwards *all* packets.

Lemma 3.4 *Routers with no firewalling capability at all are equivalent to firewalls which always accept all packets.*

As the algorithm which distributes the rules needs to cope with such members of the network, the explicit declaration of accept rules is not required as this needs to be seen as default behaviour of the components. Therefore, only deny rules are considered in this work's algorithm.

Assumption 3.5 (Only Deny Rules) *In this work, accept rules must be omitted during policy definitions.*

4. Use Cases

In order to understand the problem domain where the approach developed in this work could be applied and evaluate the effectiveness of the algorithm later in chapter 8, some realistic network models and policies are required. In this chapter, the approach followed to define the use cases is explained in section 4.1. After that a collection of realistic network models along with target policies are presented in sections 4.2 and 4.3.

4.1. Approach

Use cases should both be able to demonstrate the algorithm's basic functionality (i.e. that it produces correct results) and also demonstrate cases in which the algorithm produces results which for example might allow network operators to optimize their network. Use cases should demonstrate the limits of the algorithm as well. As the purpose of the algorithm is not the optimization of network designs per se, but to distribute firewall rules, the algorithm needs to be executed on different variants of a network model in order to evaluate these variants in regard to optimizations and test, if all firewall rules can still be distributed in another network configuration.

Therefore, the use cases presented in the following sections are separated into two categories: Basic Use Cases (section 4.2) and Network Optimization Use Cases (section 4.3). Basic Network Use Cases are use cases, where the algorithm is executed upon a fixed network representation, whereas Network Optimization Use Cases introduce realistic variations into the network architecture in order to test optimizations a network operator might want to try in order for example to reduce cost in network components or enhance the network's architecture. When testing different network architectures within a use case, the policies are kept constant.

In the following, all nodes labeled `"Clients"` are not centrally managed and do not implement their own firewalls or are firewalls which cannot be managed by a central entity. They are used as a placeholder for networks which contain for example mobile clients like laptops, smartphones and other systems which are outside the network operator's management domain and which may be added and removed form the network at any time. In general, `Clients` are considered untrusted devices.

To visualize the different networks, Cisco's Network Topology Icons[1] are used to represent routers and firewalls. Data flows which must be blocked are drawn using red arrows from its source to its destinations, covering all possible paths. Labels at data flows further specify the conditions for block rules. The condition must evaluate to true for a packet to be blocked.

[1] `https://www.cisco.com/c/en/us/about/brand-center/network-topology-icons.html`

As routers which do not have any firewalling capabilities can be considered as firewalls which always accept flows (see 3.4), only block rules are considered in the use cases.

4.2. Basic Use Cases

This type of use cases aims to demonstrate experimentally the correctness and limits of the algorithm.

4.2.1. Router with Redundant Upstream Gateways

Routers can have multiple upstream routers (i.e. towards the internet), for example to achieve redundancy. How its upstream routers are used depends on the routing configuration of the downstream router. In this use case, depicted in figure 4.1, a network containing `Clients` is routed on firewall `Firewall 1` which has two uplink routers with firewalling capabilities `Firewall 2` and `Firewall 3`. Each of those uplink routers are connected to the internet independently. All firewalls are fully trusted.

The clients must not be reachable from the internet but for related and established connections (stateful connection tracking). This means that only incoming packets towards clients are allowed which are an result of a previously initiated communication by the clients.

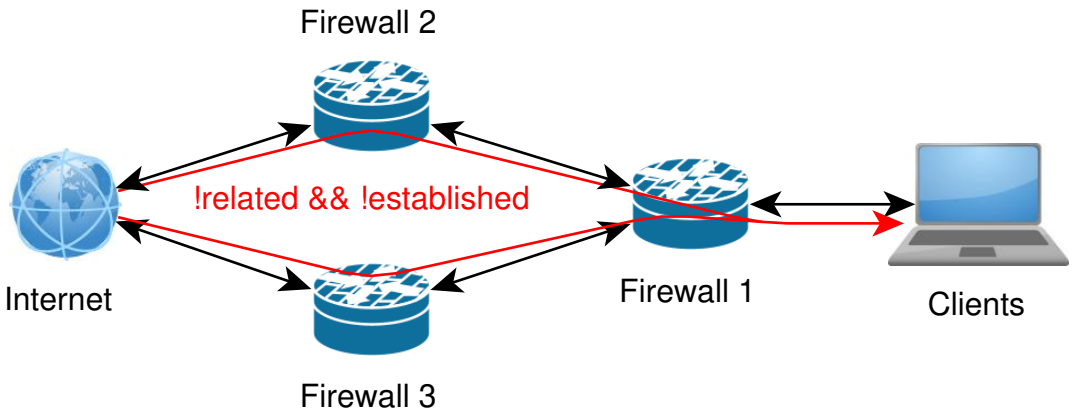


Figure 4.1.: Network with Full Trust, Router with Redundant Upstream Gateways

4.2.2. Network without Full Trust, Router with Redundant Upstream Gateways

This use case is structured similarly to the previous use case in subsection 4.2.1 but places less trust into `Firewall 1`.

As an additional policy, clients must not in any case communicate with an adversary host `A`, which is positioned in the internet. This must still hold true, if `Firewall 1` fails to implement policies and accepts all flows. The use case is depicted in figure 4.2.

This simulates a network, where one part of the network (`Firewall 1` and `Clients`) is operated in a decentralized manner in combination with central policies which must be fulfilled in any case.

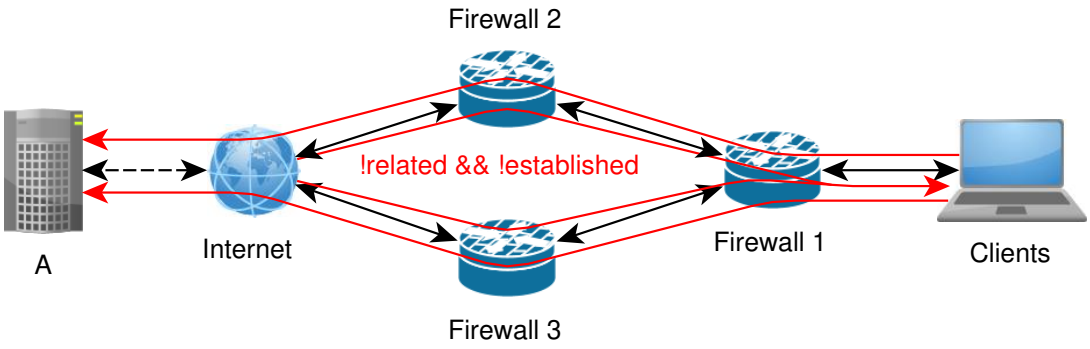


Figure 4.2.: Network without Full Trust, Router with Redundant Upstream Gateways

4.2.3. Router with Redundant Upstream Gateways and Hosts in same L2 Domain

This use case is again structured similarly to the use case introduces in subsection 4.2.1 but adds a fully trusted `Server 1`. `Server 1` and `Clients` are placed within the same OSI-Layer 2 domain and can thus reach themselves directly without traversing through `Firewall 1` (see figure 4.3). As an additional policy, `Clients` must not communicate with `Server 1`.

This use case aims to test a scenario where rules *must* be implemented on a end system, as `Server 1` is directly reachable by `Clients`, which are by themselves unable to implement firewall rules.

4.2.4. Router with Redundant Upstream Gateways and Decentral Hosts in same L2 Domain

This use case adds another variation to the previous use cases (see figure 4.3). In this case, `Server1` is considered as decentrally operated and thus receives a lower trust level, while the policy which prohibits communication from `Clients` towards `Server 1` needs to be implemented as a global policy, which must not be implemented by firewalls with a lower trust level like decentralized servers.

This use case poses a scenario which cannot be implemented without breaking trust requirements. The algorithm is expected to fail when executed on this network and policy combination.

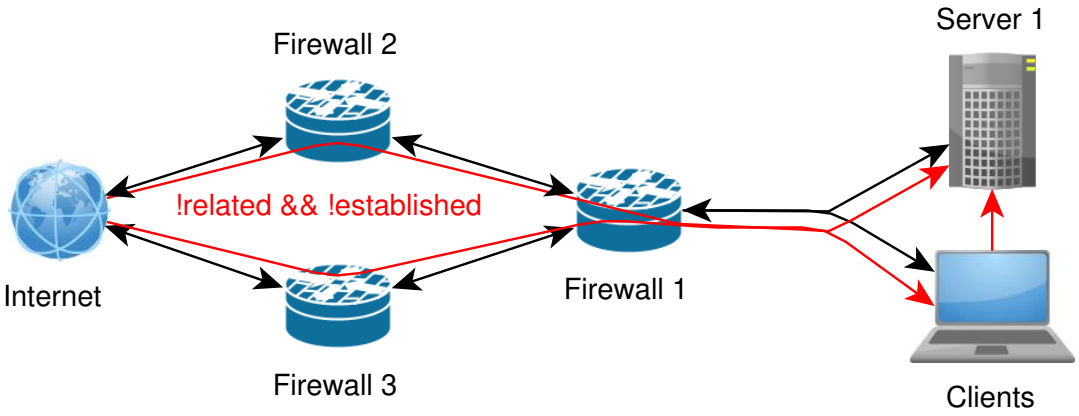


Figure 4.3.: Router with Redundant Upstream Gateways and Hosts in same L2 Domain

4.2.5. Router with Redundant Upstream Gateways and Asymmetric Paths

As shown in figure 4.4, this use case consists of `Clients` and `IoT Devices`, which are all routed on `Firewall 1`. Similar to the use case defined in subsection 4.2.1, `Firewall 1` has a redundant upstream. In this case, the upstream consists of one primary path via `Firewall 2` and a backup path through a series of routers: The firewalls `Backup 1` and `Backup 3` which are able to do stateful firewalling and `Backup 2` which is only capable of stateless firewalling.

In this case, IoT devices have a higher need for protection against attacks from outside as well as inside of the network, `Clients` are not allowed to communicate with the `IoT Devices`. Additionally, both `Clients` and `IoT Devices` must not be reached from the internet without previously initiated by them.

This use case is created because it allows a large number of possible realizations of the given policies.

4.3. Network Optimization Use Cases

These types of use cases consists of at least two networks in order to test whether the algorithm can be used as a tool to evaluate, if a simplification or improvement in the network design is feasible from the distributed firewall's perspective.

4.3.1. Network with Full Trust, Central Router & Border Router

In this use case three servers and a client network are routed on a central firewall (`Firewall 1`). The central firewall is connected via a border router (`Border 1`) to the internet. `Border 1` can only handle stateless firewall rules, while `Firewall 1` and `Server 1-3` can handle both, stateful and stateless rules. All devices besides the clients are fully trusted. This network is visualized in figure 4.5a.

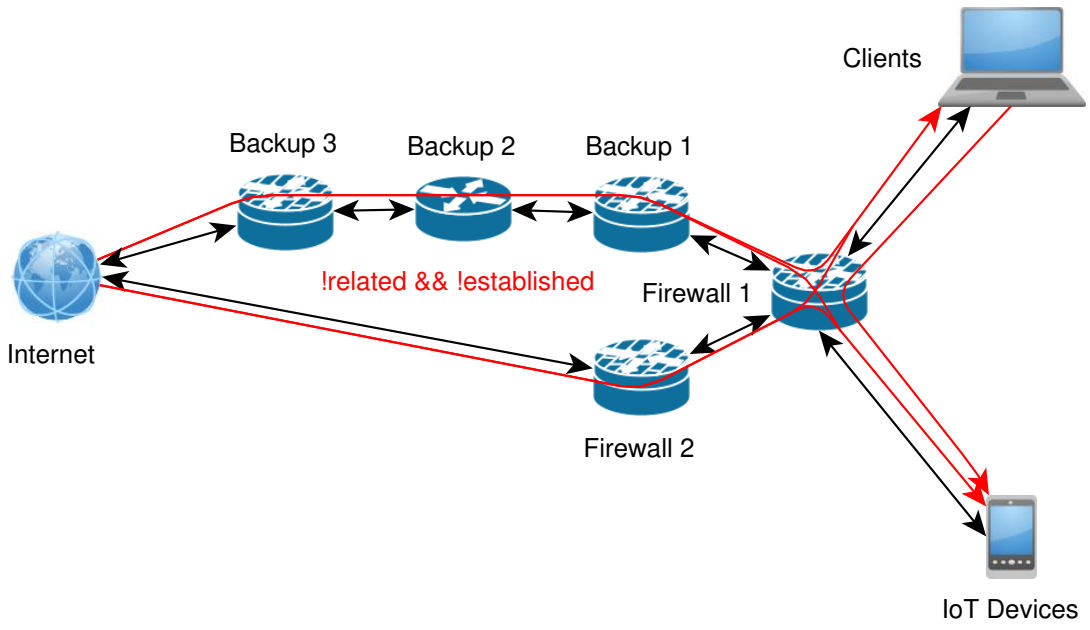


Figure 4.4.: Router with Redundant Upstream Gateways and Asymmetric Paths

The clients must not be reachable from the internet but for related and established connections (stateful connection tracking) whereas the servers must be reachable from all networks. Server 1 and 2 must not be reached on TCP port 548 from the internet.

To reduce the load on `Firewall 1`, a variant of the network is introduced where `Server 1-3` are connected directly to the border router which reduces the potential load of `Firewall 1` as only traffic from and to the `Clients` needs to be processed by the firewall as visualized in figure 4.5b.

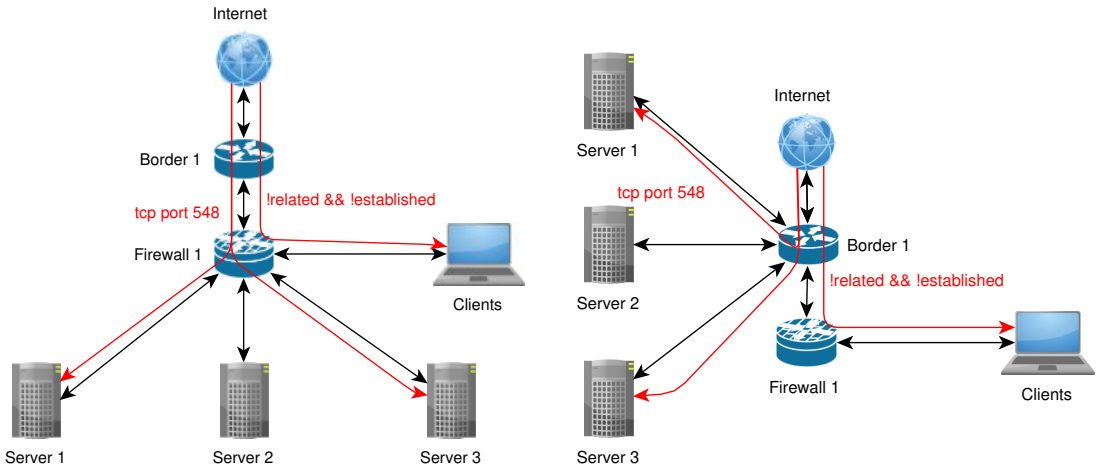
4.3.2. Network with Full Trust, Central Router & Border Router, Decentralized Server

This use case is structured equally to the use case previously defined in subsection 4.3.1, but `Server 1` is operated in a decentralized manner.

The policy to block TCP port 548 must be guaranteed in any case and must not be bypassed by any system which is affected by this rule.

4.3.3. Multi Stage Firewall

This use case aims to simulate a simplified multi staged firewall setup. Such a setup can be found at KIT. The network is protected by the `Border` firewall towards the internet. Behind this firewall, the core router interconnects all subnetworks within the network. This router is only able to handle stateless firewall rules. To be able to protect certain networks within the network from accesses from other subnetworks within the network, a second level firewall `Stage 2` is deployed.



(a) Centralized Variant

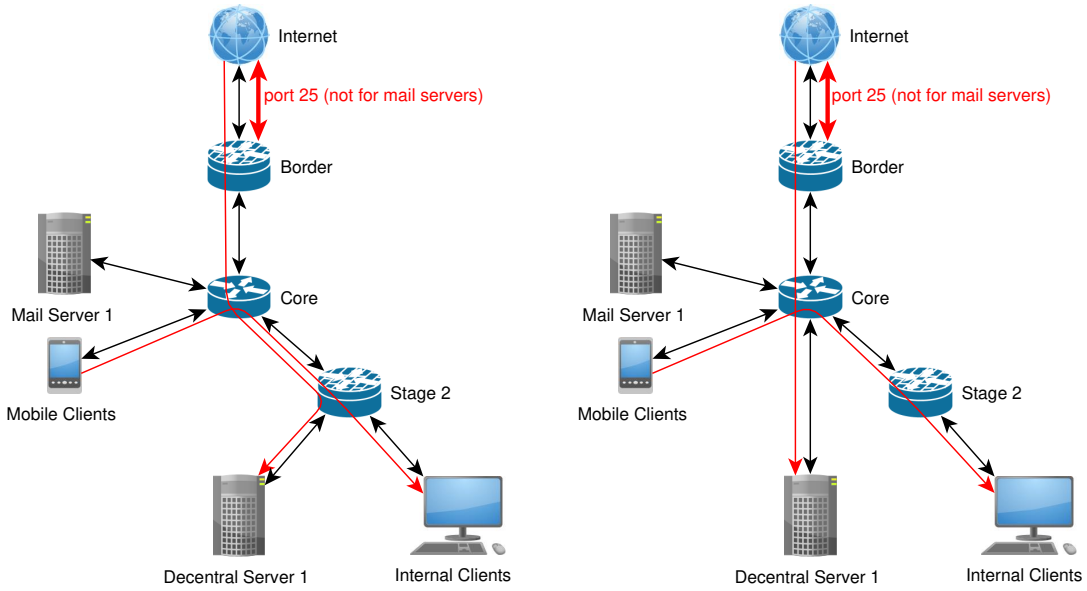
(b) Distributed Variant

Figure 4.5.: Network with Full Trust, Central Router & Border Router in two variants.

In the network, `Border`, `Core` and `Stage 2` as well as `Mail Server 1` are centrally operated and fully trusted. `Mobile Clients` is used as a representant for BYOD network members like devices in a campus wireless network. `Decentral Server 1` and `Internal Clients` are operated by an decentralized entity and are not trusted.

Like in the previous use cases, all clients must not be reached from the internet besides for related and established connections. In addition to that, `Intenal Clients` must not be reached by `Mobile Clients`. `Decentral Server 1` must not be reached from the internet. As a global policy, the network operator requires that TCP port 25 is blocked for all systems from and towards the internet besides for mail servers to prevent spam.

In the centralized variant (figure 4.6a), `Decentral Server 1` is placed behind `Stage 2` while in the distributed variant (figure 4.6b) `Decentral Server 1` is attached to the core directly.



(a) Centralized Variant (b) Distributed Variant

Figure 4.6.: Network with multi stage firewalls, clients with different protection levels within the network, a mail server and a decentrally managed server in two variants. Client rules for related and established connection states omitted for clarity.

5. Model

In order to implement the algorithm, a model of the relevant aspects of the system and its components needs to be created which meets the requirements defined in chapter 3. A network topology model needs to include all system types commonly used in a network. To reduce complexity the network model should be condensed to the aspects relevant for firewalling. As the goal of the rule distribution algorithm is to distribute policies to applicable components in the network, the policy definition should not contain direct references to the network model itself but should only use attributes related to packets instead (see assumption 3.1). In this chapter, the model and abstract data structures created for this work are defined.

5.1. Firewall

A firewall is a component of a device (mostly routers) participating in a network as a packet filter. Therefore more specialized firewalls used to filter packets on layers below routers and routing endpoints are not considered in this work.

On a more abstract view, a firewall can also be seen as a configurable mapping from packet matchers (defined by firewall rules which are a representation of policies) to corresponding actions. Actions include – depending on the firewall implementation – at least one of the following two actions:

- `ACCEPT`: The packet is accepted and forwarded to its destination.
- `DROP`: The packet is silently discarded (dropped).

Firewall implementations like Linux iptables offer more actions which are not included into this work.

Depending on the implementation, a matcher can match on any attribute of a packet like – among many others – the protocol field, source address (`src_addr`) and destination address (`dst_addr`) but also protocol or even application-specific attributes. If a packet matches the expression defined by a matcher, the corresponding action is executed for the packet.

To visualize this high-dimensional matcher space, a model of multidimensional hypercubes is introduced. Hypercubes (or n-cubes) are a geometrical construct, which describe higher-dimensional (n-dimensional) cubes. In this visualization, each discrete dimension of the cube represents an attribute on which a packet can be matched and the discrete fields represent the action taken for each combination of attributes. Figure 5.1 illustrates the dimensions `src_addr`, `dst_addr` and `protocol` of a uninitialized cube. In figures, undefined (uninitialized) actions are colored white, `ACCEPT` actions colored green and `DROP` actions red.

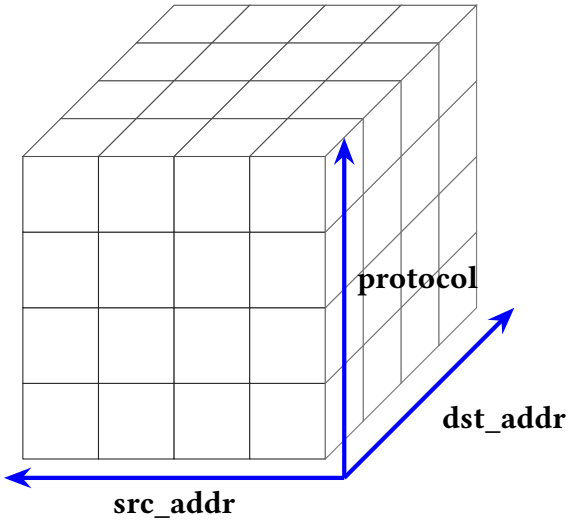


Figure 5.1.: Three dimensions (source address, destination address, protocol) of the uninitialized firewall hypercube.

As IP packets consist of a finite, countable and discrete number of protocol-specific units, each subdivision along the axis of the (hyper-) cube corresponds to one specific discrete unit. Each matcher can be represented as a hypercube which is by itself a subcube of the complete matcher space hypercube.

In order to interact with the hypercubes within the context of this work, a collection of operations on this data structure is defined:

- **`Combine`** The `combine` operation combines two hypercubes into one. In case two hypercubes intersect, the overlapping actions become undefined (white units in figures). Intersecting hypercubes happen, when two rules are constructed interdependent as described in chapter 3 and assumption 3.2. If for example two rules overlap each other's space and define different actions, the order in which the cubes are `combined` is relevant for the output of the function. As ambiguous firewall definitions most likely cause unintended results, overlapping input hypercubes should be avoided. This is why the `combine` operation is commutative.

  Figure 5.2 shows an example of a combination of two three-dimensional (hyper-) cubes.

- **`Select`** The `select` operation selects a part of the cube based on arbitrary criteria like address ranges or protocols. Figure 5.3 illustrates the selection of a sub cube matching a selection criteria. Criteria are Boolean expressions, which can match any conjunction or disjunction in all dimensions of the hypercube. Every part of the hypercube which matches the defined criteria is returned as a new (sub) hypercube.

  Those expressions can consist from specific addresses or address ranges (for example in the form of "addr is contained in `src_addr`") to high level fields like the connection state (for example "packet's `connection_state` is `related`").

- **`Cut`** The `cut` operation removes a sub cube from a target cube. If the sub cube is not included in the target cube, the unmodified target cube is returned. Figure 5.4 illustrates the removal of a sub cube from a target cube.

  This is used to remove a sub-hypercube from the complete policy hypercube when distributing rules to firewalls. It can be seen (together with `select`) as the "reverse" operation to `combine` but with specific criteria.

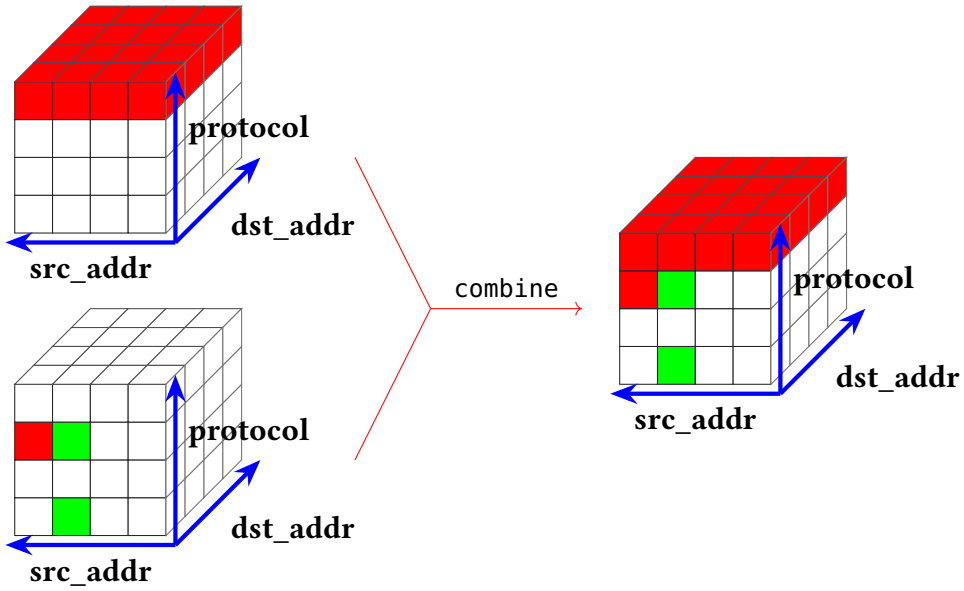


Figure 5.2.: Combination of two policies into one policy. Policy one (top left) drops packets with protocol 3 from and to all addresses, policy two (bottom left) drops packets with protocol 2 from source address 3 to destination address 0 and accepts packets with protocol 2 and 0 from source address 2 to destination address 0.

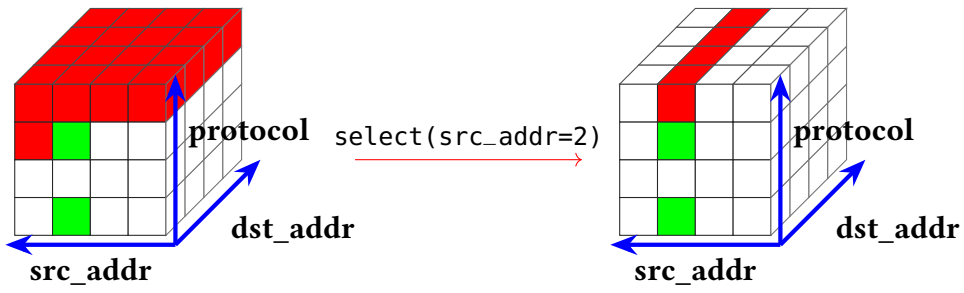


Figure 5.3.: Selection of all actions for `src_addr` 2.

5.2. Network Topology

In order to implement the policies into a network, the definition of a network topology model is required. A network can be viewed as directed graph with devices as nodes and

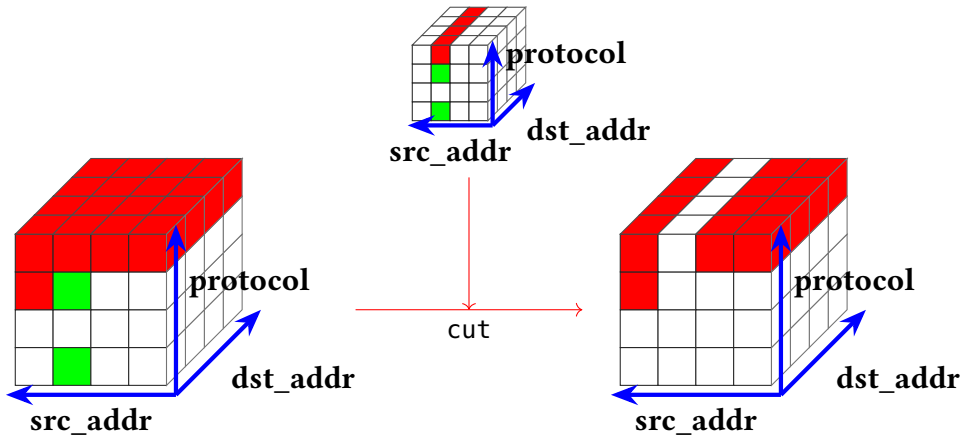


Figure 5.4.: Cut a hypercube (which was previously `selected` in figure 5.3) out of a target hypercube.

links as paths. A node can represent different types of network participants. As firewalls in this work are only considered on end systems and routers, only routers and end systems (OSI-Layer 3 devices [IEC7498-1]) need to be included into the topology model as well as their logical layer 3 connectivity.

Each node may have different capabilities in relation to its firewall. For example, some nodes are capable of full stateful firewalling, while others may only be capable to implement stateless access control lists (ACLs) as well as nodes which are completely incapable of firewalling or nodes, where the implementation of host firewalls (that is a firewall directly on an end device) is organizationally or technically impossible (from now on called "clients"). Such devices can be for example mobile devices, private devices ("Bring your own Device", BYOD) or Internet of Things (IoT) devices such as network attached sensors. Because of the wide variety of possible nodes, the model needs to be flexible concerning the node types.

To define logical links between nodes, interface lists are introduced. Each node contains a list of addresses with subnet definition in the form of `<interface address>/`n with n as prefix length (Classless Inter Domain Routing notation as defined by [RFC4632]). This notation allows to extract the network address from the interface, by splitting the address at the n-th bit and right-padding the address with zeros. All hosts possessing interfaces with same network address are in the same OSI-Layer 2 [IEC7498-1] domain and are therefore able to reach each other directly without a router in between. Interface lists therefore implicitly define layer 2 rechability lists for each node (link reachabilities). In graphical representations like in figure 5.5, link reachabilities from each node's perspective are depicted using black arrows. In order to simplify the inclusion of clients into the model, one proxy interface instead of an explicit interface list can be defined in a node, which allows the implicit creation of interface lists by declaring an address range (containing n consecutive interfaces from the beginning to the end of the range) within the subnet. This acts like n instances of the same node in the same subnet.

The network model is not required to include every component of the network but only those which are capable of routing packages and are used as a router or are able to receive

or send packets into the network. A router which is composed of multiple devices but topologically act as one device can be added to the model as one node.

As not every node may be capable of implementing all relevant rules, nodes need a way to detect, if they could be a firewall for "subordinated" nodes as they may need to implement rules "for them". In order to be able to build this relation, all possible L3 paths of packets through the network needs to be added to the model (see Lemma 3.3). This is achieved by adding gateway relations into to model of the nodes. A gateway relation contains the route (route property) which is routed from its origin to its target node and must be accommodated by a link reachability between the origin and target nodes in the same direction. As routing is not a part of the model itself as actual routing information from the network and thus the actual path selection is not taken into account, all *possible* paths of packets need to be included into the model. For example if a node obtains its routing information using dynamic routing protocols, the model needs to include all possible nodes which could become gateway for each subnet. This is required to cover all possible paths of packets towards a node for the policy enforcement. Otherwise, nodes which must implement a policy in order to fully cover it through all L3 paths might be missed by the algorithm. Nodes with no incoming gateway relations are called "leafs" while nodes with at least one incoming gateway relation are called "routers". A graphical representation of an example network model is depicted in figure 5.5. In this example, `server2` is connected to two routers called `firewall1` and `firewall2` which could both act as their gateways. Hence, both firewalls have an interface within a subnet, which corresponds to interfaces with an individual subnet each on `server2` which is used to connect to a gateway (transfer networks). The gateway relations' `route` is `::/0` which means that `firewall1` and `firewall2` might both be used as gateway for all networks outside `server1`'s link reachabilities (see chapter 2).

A node must provide the following operations:

- **`Get Matching Rules`** Return a subcube of a given input policy which matches the node, i.e. the node has an address within the policy's source or destination address.
- **`Can Implement Rules`** Check, if a subcube can be implemented on the node (i.e. are the cube's trust requirements fulfilled and does the node support the attributes used in the matcher).
- **`Implement Rules`** Mark a subcube as implemented (i.e. covered by the node).
- **`Get Neighbors(`*`destination address`*`)`** Return a list of neighbors where a packet with a given *destination address* would be forwarded to.

To model networks outside of the modeled network e.g. to be able to control flow of packets outside of the network, a special internet node is introduced. The internet node is defined by declaring the CIDR subnet of the modeled network and the internet-side interface addresses of the transfer network towards/from the internet. This node cannot match or implement any firewall rules as it is considered outside the network's administrative domain.

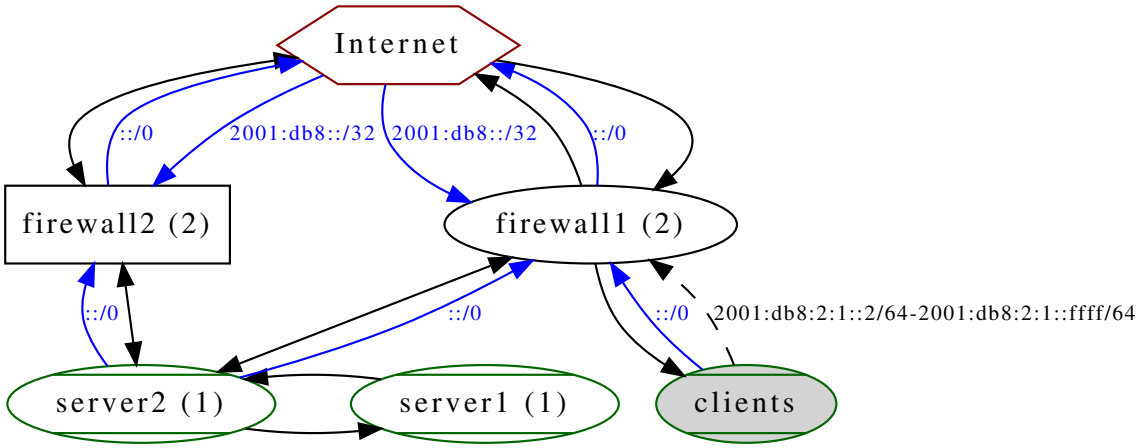


Figure 5.5.: Visualization of a network model. Leaf nodes are marked with green outlines and horizontal bars. Nodes with gray infill do not have any firewalling capabilities. `firewall1` is a firewall capable of handling stateful rules, `firewall2` can only handle stateless firewall rules (depicted with a box node). The number in parenthesis next to node names represent their trust level. Black arrows represent direct reachable nodes from each node's perspective, while dashed arrows represent proxy interfaces. Interface addresses of links are omitted for better readability. Proxy links are labeled with their address range. Gateway edges are represented with blue arrows and are labeled with their `route` attribute. The "Internet" node represents a special node, which cannot match or implement anything and acts as a source and sink for packets from or towards sources and destinations outside the network.

5.3. Trust

As nodes may be decentralized, a model of trust towards nodes is required. Decentralized nodes cannot be trusted to implement rules correctly at any given time, for example due to intervention or misconfiguration by their administrators. Therefore a trust model needs to insure that "important" policies cannot be bypassed by nodes with lower trust.

To add a trust dimension to the network model (section 5.2), each node receives a numeric trust value. In the firewall model (section 5.1), a trust requirement is added to each action in the hypercube. The `Get Matching Policy` operation must not return subcubes containing actions with a trust requirement which is smaller than the node the operation is called on. By defining trust levels in a numerical representation, a hierarchy of trust levels can be modeled, allowing a more fine-grained concept of "trust". This allows the algorithm to leave policies with higher trust requirements untouched by nodes with lower trust levels. An example of such a scenario can be found in use cases 4.3.2 and 4.2.4 where nodes have different trust levels and some policies have higher trust requirements (policies which need to be implemented in any case even if a decentralized node e.g. has a misconfigured firewall).

6. Algorithm

In this chapter the algorithm to distribute firewall rules on policy enforcement points is described.

As firewall policies and the network model needs to be provided upfront, the algorithm is an offline algorithm which calculates a complete solution (i.e. firewall rule distribution) for a given network at once. It is assumed, that the policy definition is provided in an organized manner which allows to query policies efficiently (policy definition database). The algorithm needs to check, whether the desired policy defined in the policy definition database can be implemented on a given network and calculate, which member of the network needs to implement which rules – the algorithm transforms the policy definition database into a policy distribution database. Policy-preserving rule construction from the policy definition on the individual firewalls is *not* part of this algorithm. Changes in the network need to be added to the model and the algorithm needs to be re-executed upon the new network in order to produce a solution for the new network.

The algorithm consists of two stages: In the first stage (subsection 6.1) firewall rules which can be enforced directly by hosts affected by the rules are implemented. In the second stage (subsection 6.2), remaining rules which must be enforced by intermediate firewalls are implemented.

6.1. Host Rule Assignment

This stage is the first round of the algorithm (see algorithm 1, lines 3-7). In this stage, all rules which can be enforced directly on nodes are implemented. This is done by iterating over all nodes in no particular order and all rules returned by the node function `Get Matching Nodes` (see section 5.2) are implemented.

To implement the node function `Get Matching Nodes`, all rules are considered which match the following conditions:

- The rule is not marked as implemented.
- The rule's trust requirement is smaller or equal the node's trust level.
- Any "hard" condition which does not allow the implementation of these rules at all based on the node's firewalling capability. One example of such a hard condition would be an unsupported action. A "soft" condition would be something, which could be still implemented after a cut along an axis is performed (to "cut out" the unsupported part of the rule).

In the first step, all interfaces of the current node are iterated. For each interface there are two ways in which the node can be affected by a rule:

1. The node's interface address is contained in the rule's `src_addr`. This means, that the rule must be implemented as outgoing rule from the node's perspective.

2. The node's interface address is contained in the rule's `dst_addr`. This means, that the rule must be implemented as incoming rules from the node's perspective.

Each rule which affects the current node is `cut` out of the policy and marked as implemented by the current node in the policy distribution database (algorithm 1 lines 5-6).

In the second step, all outgoing gateway edges are iterated. If the interface address on the link edge on which the gateway is reached is in the rule's `src_addr` and the intersection between the gateway edge's `route` and the rule's `dst_addr` is not empty, the rule must be implemented as outgoing rule from the node's perspective. This means, that the node reaches the rule's destination address via its gateway and can therefore `cut` the rule on the node's address and mark the cut itself as implemented by the node (algorithm 1 lines 5-6).

After those two steps, all parts of the rules which could be implemented on the hosts are implemented. The remaining rules cannot be fulfilled on the hosts and must be implemented by intermediate firewalls (section 6.2).

6.2. Intermediate Firewalls

In the second stage of the algorithm, the remaining rules which are not marked as implemented by the previous step are implemented on the intermediate firewalls in the network. As of lemma 3.3, all paths between a rule's source and destination need to be cut at least once in order to reach full network coverage of a rule. To reach this, all unimplemented remaining rules are considered. For each rule, all paths between matching nodes need to be found using a path finding algorithm. In order to find paths, each node needs to know its neighbors which can be used to reach the destination. A neighbor is not considered a neighbour if the rule is already implemented on the node. If there are multiple matching gateway edges with the same prefix length an undefined one is chosen.

Once a path is found, the rule needs to be implemented on one of the nodes on the path. Which node is determined by the firewalling capabilities of the nodes. If there are more than one node capable of implementing the rule, the following heuristic used to decide the node:

- By adding a weight attribute to the nodes the rule can be assigned to the node with the highest weight. This can be used to add the capacity of the firewalls to the model. If the network contains large firewall appliances this metric can be used to concentrate intermediate rules on hosts with higher capacity.

- In case of no weights or equal weights on the path nodes, assign to an undefined node.

The implementation of the rule on the chosen node is marked in the policy distribution database. If no node on the path is able to implement the rule, the algorithm fails.

This path finding algorithm and rule assignment is repeated for the rule until no path is found, resulting in rule coverage on all paths.

6.3. Finalization

After both steps of the algorithm are finished, the policy distribution database is evaluated. If there are any rules which are neither implemented on any host nor on intermediate hosts *and* there are nodes which have interfaces with IP addresses within the source or destination address of the remaining rules, not all rules could be implemented on the network and the algorithm fails. Otherwise, the rules must be materialized in the nodes mentioned in the policy distribution database. Materialization means, that the rules as phrased in the policy definition are translated in each node's native firewall language. This should reach full coverage of the policy in the network.

6.4. Pseudo Code

In this section, the algorithm is outlined as pseudo code in algorithm 1.

Algorithm 1 Policy Distribution Algorithm

```
1:  N ← Nodes
2:  P ← Policy
3:  for n ∈ N do                                          ▷ Host Rule Assignment
4:      rules ← n.get_matching_rules(P)
5:      n.implement_rules(rules)
6:      P ← P.cut(rules)
7:  end for
8:
9:  for r ∈ P do                                          ▷ Intermediate Firewalls
10:     src_nodes ← all nodes with an interface address ∈ r.src_addr
11:     dst_nodes ← all nodes with an interface address ∈ r.dst_addr
12:     while ∃ path without rule implementation p between any src_node and any
    dst_node do
13:         implemented ← False
14:         p ← sort_by_node_weight(p)
15:         for n ∈ p do
16:             if n.can_implement(r) then
17:                 n.implement_rules(r)
18:                 implemented ← True
19:                 break
20:             end if
21:         end for
22:         if ¬implemented then
23:             fail                                  ▷ Not all paths could be covered
24:         end if
25:     end while
26: end for
```

7. Implementation

In this chapter, overall design considerations of the software are outlined in section 7.1. Afterwards, the implementation of the data structures (based on the model introduced in chapter 5) and algorithm (see chapter 7) is outlined in sections 7.2 and 7.3.

The implementation which acts as a proof of concept (PoC) for this work's model and algorithm is written in Python and uses PostgreSQL[1] as data store for dynamic data. Implementation resources can be obtained from this work's complimentary material and the setup process is documented in appendix section A.1.

7.1. Application Design and Components

In order to integrate the model and algorithm into an usable system, several components are required. In figure 7.1 the proposed architecture of the software is illustrated. The central component is the policy distribution algorithm, which uses information provided by the policy definition database and the network model to generate a policy distribution, which is stored in the policy distribution database. Network model, policy definition database, policy distribution database and a visualizer for network models and policy distributions within the network are implemented in this work as described in the following sections.

In order to realize the abstract policy definitions on the actual firewalls, a compiler which converts the abstract policy definition into firewall rules is required (platform-specific firewall language output). This can for example be a translation into Linux iptables syntax or into access control lists of routers. To deploy those rules on the hosts, a host agent is required. This either runs on the firewall controller – which also executes the distribution algorithm and holds the databases – or on another machine in order to deploy the rules via a vendor specific interface or or on the PEP itself. The latter might be the case for servers with firewalling abilities. In order to reduce manual management of the inventory, host agents could contribute "their" definitions of hosts they manage into the network model. For example a Linux PEP host agent could contribute its hostname, interface IP addresses and routing definitions into the model. This allows a flexible deployment of the distributed firewall.

As seen in the "block traffic from the internet" example in chapter 3, some policies might consist of multiple entries in the policy definition database due to non-contiguous addresses ranges. This example shows, that a high-level policy language is needed to increase the ease of use and reduce potential configuration errors due to inconsistently updated policies in case changes are required to the policy. Such a language might be

[1] `https://www.postgresql.org`

developed or an existing one might be adapted in the future. The language must be topology independent (see assumption 3.2) and must not allow policy interdependencies (see assumption 3.1). Languages like Mignis [Adã+14] could provide a good baseline to achieve those features.

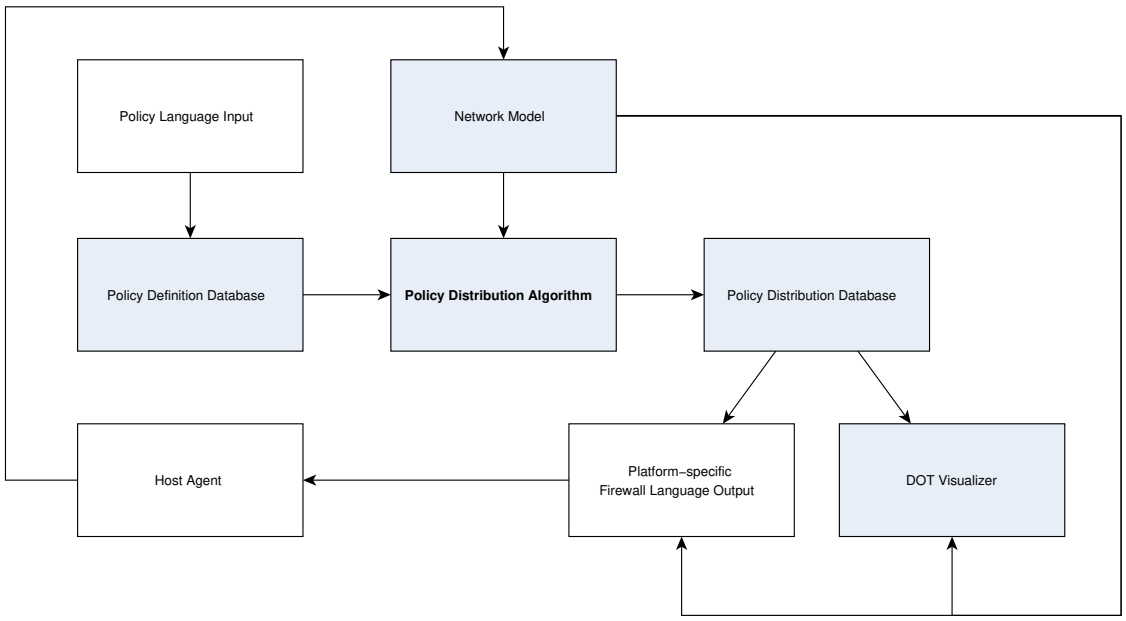


Figure 7.1.: Components of the software. Highlighted: Implemented in this work

7.2. Data Structures

In this section, the implementation of the data structures defined in chapter 5 is described.

7.2.1. Firewall Policies

To implement the firewall policy model described in section 5.1, PostgreSQL is used. The dimensions of firewall hypercubes are implemented using range types. Range types represent an interval on any datatype which has a total order. In this implementation, all ranges are handled as closed intervals. As IP addresses, port ranges, protocol identifiers and other properties of IP packets are represented by discrete numbers, range types can be used. Ranges can be grouped into a multirange, containing a ordered list of non-empty ranges[2]. PostgreSQL's range and multirange types allow a multitude of operations. The following PostgreSQL range operations[3] are used in the algorithm:

- `range1 @> range2` and `range1 <@ range2`: Checks whether `range1` contains `range2` or if `range1` is contained in `range2`. The containment check can also be done with a single element of the range's base type and a range (`element @> range1`).

[2] `https://www.postgresql.org/docs/current/rangetypes.html`
[3] `https://www.postgresql.org/docs/15/functions-range.html`

- `range1 && range2`: Checks whether `range1` and `range2` overlap, i.e. if they have elements in common

- `multirange1 - multirange2`: Calculates the difference between two multiranges. Returns a multirange which contains the remaining ranges.

  Example: Given `multirange1` as `{[a,z]}` and `multirange2` as `{[c,c]}`, the result of `multirange1 - multirange2` would be `{[a,b], [d,z]}`.

The whole policy is stored in a SQL table (`policy_definition`) which represents a sparse policy hypercube. This table represents the policy definition database as a sparse matrix. The table contains the source and destination address ranges (`src_addr` and `dst_addr`) as well as the trust requirement (`trust_requirement`), the `action` (such as `DROP`) and all other matchers like `protocol` and `port` as columns.

On this data structure, the policy model with its operations (see section 5.1) is implemented:

Combine The `combine` function is implemented by adding rows to `policy_definition`. The entity calling the `combine` function is responsible for checking, that the rules do not overlap (see assumption 3.1).

Select The `select` operation is implemented using PostgreSQL's native `SELECT` operation. For example, if all rules (subcubes) matching a specific source address `addr` (i.e. if `addr` is contained in the `src_addr` column) should be selected, the query

```
SELECT * FROM policy_definition WHERE addr <@ src_addr;
```

returns all rows which match this condition.

Cut In order to `cut` the match out of the matching row, the previously defined `select` operation can be expanded by calculating the difference between the original range and the match:

```
SELECT *, ((src_addr::inet_multirange) - (addr::inet::inet_multirange))
AS cut FROM policy_definition WHERE src_addr @> addr;
```

The column *cut* returned by this query contains the original *src_addr* column without *addr*. If *addr* was not the first or last element of the range, *cut* is a multirange containing two elements: The range from the first source address of the rules until *addr* and from $addr + 1$ until the last source address. By duplicating the same row but multiple times with all ranges contained in the *cut* multirange as new *src_addr* value and deleting the original row, the `cut` operation along the *src_addr* is finalized. This can be performed along all columns in order to implement only rules or parts of rules matching the policy enforcement point's firewalling capabilities.

7.2.2. Network

The network model is implemented as python classes. In this PoC, policy Enforcement Points (PEPs) with their types, interfaces and gateway relations are defined statically by defining a so called "inventory definition", which is represented by a YAML file which gets deserialized into the aforementioned python classes. YAML is a human readable data serialization format, which allows the definition of several scalar data types like numbers and strings as well as lists and objects (key-value mappings)[4]. To accommodate different firewall implementations and capabilities, each PEP needs its type defined in the inventory file, which could be extended with data provided by hosts agents in the future. In the PoC, the only implemented PEP types are a stateful and stateless generic firewall and `clients`, which represent devices with no firewalling capability.

The special node `Internet` is added automatically and is defined by specifying the network which is the subject of the distribution as CIDR.

To visualize the inventory, the PoC contains a DOT visualizer, which renders the network model as a graph. DOT[5] is a graph description language, which allows the definition of nodes and edges. An additional external compiler like GraphViz's `dot` transforms the DOT description into different image types. An example of this visualisation generated by the PoC using the example inventory definition mentioned in appendix section A.1 can be found in chapter 5 in figure 5.5.

7.3. Algorithm

In the following, functions declared in the models defined in chapter 5 are used. To ensure that a policy is fully implemented in the network, the implementation of rules on nodes is tracked in a PostgreSQL table called `policy_work`. This table is a copy of the `policy_definition` (see 7.2.1) table but additionally contains a *implemented_by* and *path_implemented* column to track the implementation status of each rule. As `cut` operations are performed during the execution of the algorithm, a *origin* column is introduced to reference the originating rule out of the `policy_definition` table from which the resulting rules where created. Upon the execution of `Implement Rules` on a node, the affected subcube is `cut` out of the policy definition (see section 7.2) and marked as implemented in the `policy_work` table.

The host rule assignment stage is implemented as described in chapter 6.1: For all hosts the algorithm checks, if the rule can be directly implemented on the host firewalls. To track host implementations, the *implemented_by* column is used to mark, by which node the rule is implemented. If *implemented_by* is `NULL`, the rule is not implemented on any host.

In the next stage (intermediate firewalls), calls of `Implement Rules` append the implementing node to the *path_implemented* column. The column is of the type `ARRAY`, which allows the addition of multiple entries into the column. This is necessary, as there may be multiple paths (and by extent multiple nodes) on which a rule needs to be realized.

[4] `https://yaml.org/`

[5] `https://www.graphviz.org/documentation/`

To find paths, Dijkstra's Algorithm [Dij22] is used to return the paths between two nodes, sorted by path length in ascending order This is done to make results more reproducible as paths are always inspected from shortest to longest as each path is "disrupted" by the realization of a rule on the path. For each unimplemented rule, all nodes covered by the rule's `src_addr` are used as starting nodes and all nodes covered by the rule's `dst_addr` are used as destination nodes for Dijkstra's Algorithm. Because Dijkstra's Algorithm requires each node to know its neighbors, `Get Neighbors` is called on each traversed node which ignores neighbours, where the rule currently inspected is already implemented. As this depends on the node's routing information from the network model and routing decisions can vary from implementation to implementation of each node type, this function is placed into the node class within the network model's implementation. The basic implementation of the generic firewall supplied in this work uses the same rules as commonly used by routing decisions in routers: If the destination address is within one interface's network of the node, use the interface. Otherwise, use the longest matching route prefix first, considering the node's outgoing gateway edges. In case a different heuristic is required, another node type can be implemented.

After both stages of the algorithm have completed, the software terminates and the `policy_work` table becomes the policy distribution database. The network model's DOT visualizer is extended to allow the visualization of the distribution in the model's DOT graphs, as shown in the evaluation in chapter 8.

8. Evaluation

In this chapter, the use cases previously defined in chapter 4 are transferred into the model defined in chapter 5 and the algorithm described in chapter 6 and implemented in 7 is executed upon this input and the results are discussed in section 8.1 to show by experiment, if the model is useful for realistic examples. Afterwards, the running time of the algorithm is evaluated in section 8.2.

8.1. Use Cases

In general, the documentation IPv6 prefix `2001:db8::/32` in accordance to [RFC3849] is used to represent addresses in the network modeled. End system networks are assigned `/64` networks within `2001:db8:a::/48`. Intermediate systems (i.e. routers and firewalls) are assigned `/64` networks within `2001:db8:1::/48`. Hosts used as gateways in networks are assigned the address `<prefix>::1/64`. To simplify the evaluation, networks within `2001:db8:1::/48` are never routed via gateway edges and gateway addresses are excluded from firewall rules to avoid implementation of input rules on gateways. This is based on the assumption that in general no services are offered on gateway interfaces or get utilized by the gateways directly.

To improve readability, repeating IPv6 nibbles containing only `ffff` are shortened using three dots ("`...`") and ranges containing only one element are shortened to "`<element>`" instead of "`[<element>,<element>]`".

Each rule in `policy_definition` (see chapter 7.2) is assigned a unique identifier (`id`). The network model and the firewall rule distribution results are visualized using graphs like in figure 5.5 with rule `ids` assigned to the node in the text line below each node's label.

In order to classify the results of the algorithm, some definitions to define a correct outcome of the rule distribution algorithm are required:

Definition 8.1 (Correctness of a Distribution) *A distribution of firewall rules on a network is* correct *if the behaviour of the network for all possible data flows is* functional indistinguishable *from a theoretical network where* every *member implements* all *rules (ignoring node capabilities).*

Definition 8.2 (Redundancy-Free Solution) *A* correct *distribution of firewall rules on a network is* not redundant/locally minimal *if no implementation of a rule can be removed without creating an incorrect distribution.*

In order to quantitatively evaluate, whether the algorithm is in general able to reach the goal of distributing rules on end systems whenever possible, the following metric is introduced:

Definition 8.3 (End System Distribution Factor (ESDF)) *The* end system distribution factor d_{rule} *in a* correct *distribution of rules in a network is defined by*

$$d_{rule} = \begin{cases} \frac{leaf\_implementations(rule)}{matching\_leafs(rule)} & if\ matching\_leafs(rule) > 0 \\ 0 & otherwise \end{cases}$$

With $matching\_leafs(rule)$ *being the number of leafs affected by* rule *(i.e. at least one of a node's interface addresses matches either the* rule*'s source or destination address) and* $leaf\_implementations(rule)$ *the number of leaf nodes which implemented* rule*.*

This metric becomes zero for a rule, if the distribution is fully centralized from the perspective of end systems and one, if the rule is fully implemented on end systems. If there are *any* rules in *any* solution for the firewall distribution problem (see 3.1) with an $ESDF > 0$, the algorithm fulfills the aforementioned goal of distributing rules on end systems. In order to compare distributions among different use cases, $ESDF_{\max}$ is defined:

Definition 8.4 ($ESDF_{\max}$) *The* maximum end system distribution factor $ESDF_{\max}$ *in a* correct *distribution of rules in a network is defined by*

$$ESDF_{\max} = \max(d_1, \ldots, d_n)$$

With d_n *being the ESDF for rule n.*

The following results have been created using a reference implementation of the algorithm and datastructures written in Python, which can be obtained this work's complementary material. A documentation on how to reproduce the results can be found in section A.2.

8.1.1. Basic Use Cases

8.1.1.1. Router with Redundant Upstream Gateways

In this subsection, the use case defined in subsection 4.2.1 is evaluated.

To block not related or established connections from the internet towards `Clients`, the policy is phrased using two rules to keep them dependency-free (assumption 3.1):

- Drop packets with not state established or state related with source addresses from address 0 (`::`) to the exclusive beginning of the network (`2001:db7:ffff:...`)
- Drop packets with not state established or state related with source addresses from the first address outside the network (`2001:db9::`) to the maximum IPv6 address (`ffff:...:ffff`)

The policy definition can be obtained from table 8.1 and the network model including the final distribution of firewall rules in figure 8.1a.

As there are two paths affected by the rules (`Internet` → `Firewall2` → `Firewall1` → `Clients` and `Internet` → `Firewall3` → `Firewall1` → `Clients`), each rule needs to be implemented on both paths at least once (see Lemma 3.3). Because there is no weight

assigned to the `Firewall` nodes and all three `Firewall` nodes are capable of implementing the rules in the policy definition, the rules have been implemented on an undefined node on the path (see chapter 6). The reference implementation has chosen `Firewall1` and `Firewall2` as they happen to be the first ones in the implementation's data structure (see figure 8.1a). As `Firewall1` needs to handle the traffic for `Clients` in any cases, a higher weight in comparison to the other firewalls can be assigned to `Firewall1` in order to concentrate the rules there. The result of this weighted variant can be seen in figure 8.1b.

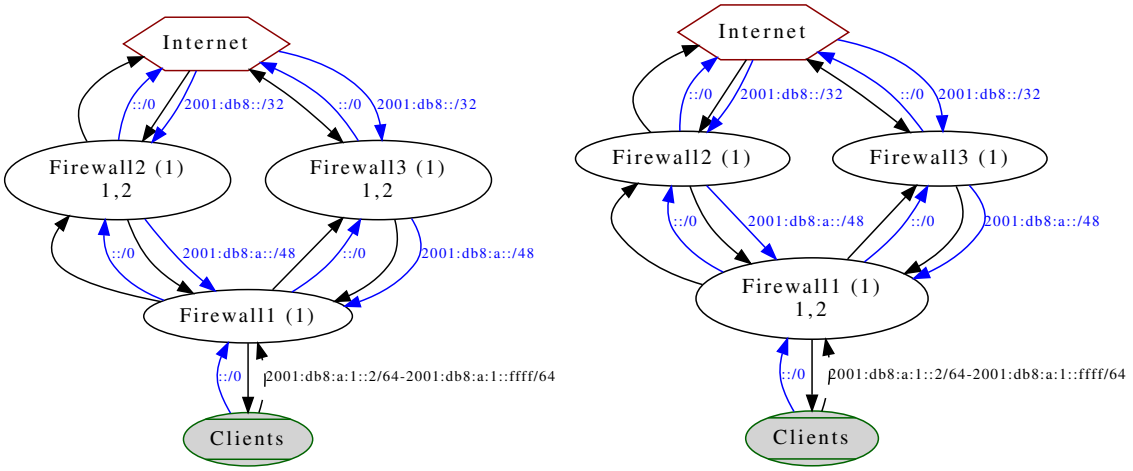
Both solutions meet the *correctness* definition (see definition 8.1) and are a *minimal* solution for the firewall distribution problem (see definition 8.2). In all solutions, the ESDFs d_1 and d_2 are zero, as no end system is capable of implementing firewall rules.

id	*src_addr*	*dst_addr*	*action*
1	[::, 2001:db7:ffff:ffff:…:ffff:ffff]	[2001:db8:a:1::2, 2001:db8:a:1::ffff]	DROP
2	[2001:db9::, ffff:ffff:…:ffff:ffff]	[2001:db8:a:1::2, 2001:db8:a:1::ffff]	DROP

id	*trust_requirement*	*tcp_state_established*	*tcp_state_related*	*protocol*	*port*
1	1	false	false	NULL	NULL
2	1	false	false	NULL	NULL

(2 rows)

Table 8.1.: Policy definition for use case 4.2.1 "Router with Redundant Upstream Gateways"



(a) Default Variant (b) `Firewall1`-weighted Variant

Figure 8.1.: Network model and rule distribution for use case 4.2.1 "Router with Redundant Upstream Gateways"

8.1.1.2. Network without Full Trust, Router with Redundant Upstream Gateways

In this subsection, the use case defined in subsection 4.2.2 (a variation of the previous use case in subsection 4.2.1) is evaluated.

As in table 8.2 visualized, the additional rule with *id* = 3 is added with *trust_requirement* = 2 blocking traffic from `Clients` towards the address `2a00:1398:b::8d03:81e2` which is acting as adversary `A`. The trust level of `Firewall1` and `Firewall2` is increased to two, while `Firewall1` has a trust level of one. Similarly to the previous use case, a variant with default weights is depicted in figure 8.2a. Figure 8.2b shows a variant, where like in the previous use case `Firewall1` received a higher weight. As the rule with *id* = 3 has a higher trust requirement which is only fulfilled by `Firewall2` and `Firewall3`, the rule can only be implemented on them.

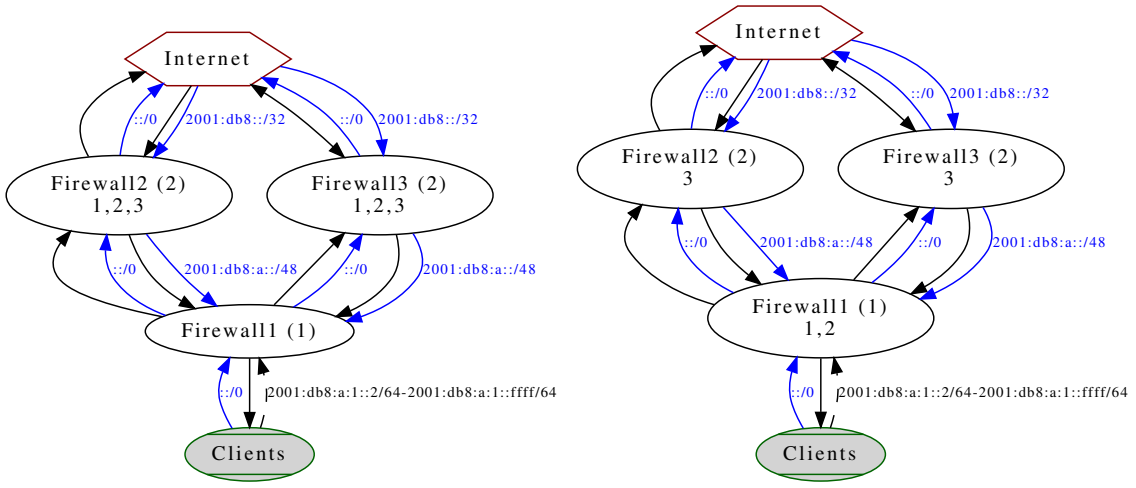
Both solutions meet the *correctness* definition (see definition 8.1) and are a minimal solution for the firewall distribution problem (see definition 8.2). In all solutions, the ESDFs d_1 and d_2 are zero, as no end system is capable of implementing firewall rules.

id	*src_addr*	*dst_addr*	*action*
1	[::, 2001:db7:ffff:ffff:...:ffff:ffff]	[2001:db8:a:1::2, 2001:db8:a:1::ffff]	DROP
2	[2001:db9::, ffff:ffff:...:ffff:ffff]	[2001:db8:a:1::2, 2001:db8:a:1::ffff]	DROP
3	[2001:db8:a:1::2, 2001:db8:a:1::ffff]	2a00:1398:b::8d03:81e2	DROP

id	*trust_requirement*	*tcp_state_established*	*tcp_state_related*	*protocol*	*port*
1	1	false	false	NULL	NULL
2	1	false	false	NULL	NULL
3	2	NULL	NULL	NULL	NULL

(3 rows)

Table 8.2.: Policy definition for use case 4.2.2 "Network without Full Trust, Router with Redundant Upstream Gateways"



(a) Default Variant

(b) `Firewall1`-weighted Variant

Figure 8.2.: Network model and rule distribution for use case 4.2.2 "Network without Full Trust, Router with Redundant Upstream Gateways"

8.1.1.3. Router with Redundant Upstream Gateways and Hosts in same L2 Domain

In this subsection, the use case defined in subsection 4.2.3 (a variation of the previous use case in subsection 4.2.1) is evaluated.

The use case adds a decentralized host `Server1` in the same L2 domain as the clients as indicated by the black reachability arrows between the nodes in figure 8.3. `Clients` must not communicate with `Server1` with the IPv6 address `2001:db8:a:1::a:1`. This rule with $id = 3$ is shown in table 8.3.

Like in the previous use cases, a distribution variant with default weights and one with a higher weight on `Firewall1` is shown in figure 8.3. As `Clients` and `Server1` can reach each other directly, rule three *must* be implemented on `Server1`. Because the node is able to implement the rule, it gets implemented on the node.

Both solutions meet the *correctness* definition (see definition 8.1) and are a minimal solution for the firewall distribution problem (see definition 8.2). In all solutions, the ESDFs d_1 and d_2 are zero, as no matching end system is capable of implementing firewall rules. The ESDF d_3 is 0.5, as the rule gets fully covered by `Server1` which is a leaf node. As the rule matches exactly two nodes in the network and no other nodes including the internet, the maximum value for d_3 in a *minimal* solution is 0.5. An ESDF of one would require both `Clients` and `Server1` to implement the rule which would create a *non-minimal* solution, as one of those rule implementations could be removed without creating an *incorrect* solution.

id	*src_addr*	*dst_addr*	*action*
1	[::, 2001:db7:ffff:ffff:…:ffff:ffff]	[2001:db8:a:1::2, 2001:db8:a:1::ffff]	DROP
2	[2001:db9::, ffff:ffff:…:ffff:ffff]	[2001:db8:a:1::2, 2001:db8:a:1::ffff]	DROP
3	[2001:db8:a:1::2, 2001:db8:a:1::ffff]	2001:db8:a:1::a:1	DROP

id	*trust_requirement*	*tcp_state_established*	*tcp_state_related*	*protocol*	*port*
1	1	false	false	NULL	NULL
2	1	false	false	NULL	NULL
3	1	NULL	NULL	NULL	NULL

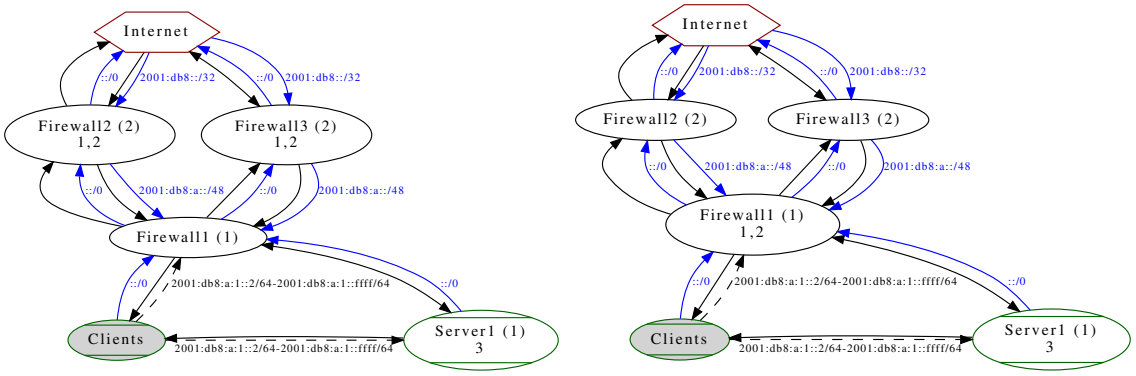
(3 rows)

Table 8.3.: Policy definition for use case 4.2.3 "Router with Redundant Upstream Gateways and Hosts in same L2 Domain"

8.1.1.4. Router with Redundant Upstream Gateways and Decentral Hosts in same L2 Domain

In this subsection, the use case defined in subsection 4.2.4 (a variation of the previous use case defined in subsection 4.2.3) is evaluated.

This variant changes the *trust_requirement* of the rule with $id = 3$ to two while leaving all trust levels of all leaf nodes at one as shown in table 8.4.



(a) Default Variant (b) `Firewall1`-weighted Variant

Figure 8.3.: Network model and rule distribution for use case 4.2.3 "Router with Redundant Upstream Gateways and Hosts in same L2 Domain"

The algorithm fails as `Server1` cannot implement the rule as of its trust level and the direct path between `Clients` and `Server1` is not covered by any implementation besides rule *3* matching these nodes, which is the expected behaviour.

id	*src_addr*	*dst_addr*	*action*
1	[::, 2001:db7:ffff:ffff:...:ffff:ffff]	[2001:db8:a:1::2, 2001:db8:a:1::ffff]	DROP
2	[2001:db9::, ffff:ffff:...:ffff:ffff]	[2001:db8:a:1::2, 2001:db8:a:1::ffff]	DROP
3	[2001:db8:a:1::2, 2001:db8:a:1::ffff]	2001:db8:a:1::a:1	DROP

id	*trust_requirement*	*tcp_state_established*	*tcp_state_related*	*protocol*	*port*
1	1	false	false	NULL	NULL
2	1	false	false	NULL	NULL
3	2	NULL	NULL	NULL	NULL

(3 rows)

Table 8.4.: Policy definition for use case 4.2.4 "Router with Redundant Upstream Gateways and Decentral Hosts in same L2 Domain"

8.1.1.5. Router with Redundant Upstream Gateways and Asymmetric Paths

In this subsection, the use case defined in subsection 4.2.5 is evaluated.

The policies regarding connection states for `Clients` and `IoT` devices are implemented similarly to the previous use cases as rules with the *id* one to four as shown in table 8.1.1.5. To prevent communication of `Clients` with `IoT` devices, the rule with $id = 6$ is defined, which `DROP`s *all* packets originated in the `Client` IPv6 address range and destinations in the `IoT` IPv6 address range range.

The resulting distribution with default weights can be seen in figure 8.4a. As `Firewall1` needs to handle all the traffic from `Clients` and `IoT` anyway, a variant with higher weight on `Firewall1` has been executed as well and is depicted in figure 8.4b.

As the order of path traversals is defined by the shortest path first by using Dijkstra's algorithm to find matching paths for rules, the path `Internet` → `Firewall2` → `Firewall1` → `Clients/IoT` is evaluated first for the rule implementation on intermediate firewalls. This caused the implementation of rules 1-4 to be implemented in `Firewall2` first in the unweighted variant, as it happened to come first in the algorithm's data structure. In the next iteration, Dijkstra's algorithm yields the next shortest path `Internet` → `Backup3-1` → `Firewall1` → `Clients/IoT` causing the implementation of rules one to four on one PEP in this path, in this case `Firewall1`. Now, the implementation on `Firewall1` could be removed without causing an *incorrect* solution, thus making the non-weighted solution not *minimal*. The `Firewall1`-weighted solution however meets the `minimal` definition.

In all solutions, the ESDFs of all rules are zero, as no end system is capable of implementing firewall rules.

id	*src_addr*	*dst_addr*	*action*
1	[::, 2001:db7:ffff:ffff:...:ffff:ffff]	[2001:db8:a:1::2, 2001:db8:a:1::ffff]	DROP
2	[2001:db9::, ffff:ffff:...:ffff:ffff]	[2001:db8:a:1::2, 2001:db8:a:1::ffff]	DROP
3	[::, 2001:db7:ffff:ffff:...:ffff:ffff]	[2001:db8:a:2::2, 2001:db8:a:2::ffff]	DROP
4	[2001:db9::, ffff:ffff:...:ffff:ffff]	[2001:db8:a:2::2, 2001:db8:a:2::ffff]	DROP
5	[2001:db8:a:1::2, 2001:db8:a:1::ffff]	[2001:db8:a:2::2, 2001:db8:a:2::ffff]	DROP

id	*trust_requirement*	*tcp_state_established*	*tcp_state_related*	*protocol*	*port*
1	1	false	false	NULL	NULL
2	1	false	false	NULL	NULL
3	1	false	false	NULL	NULL
4	1	false	false	NULL	NULL
5	1	NULL	NULL	NULL	NULL

(5 rows)

Table 8.5.: Policy definition for use case 4.2.5 "Router with Redundant Upstream Gateways and Asymmetric Paths"

8.1.2. Network Optimization Use Cases

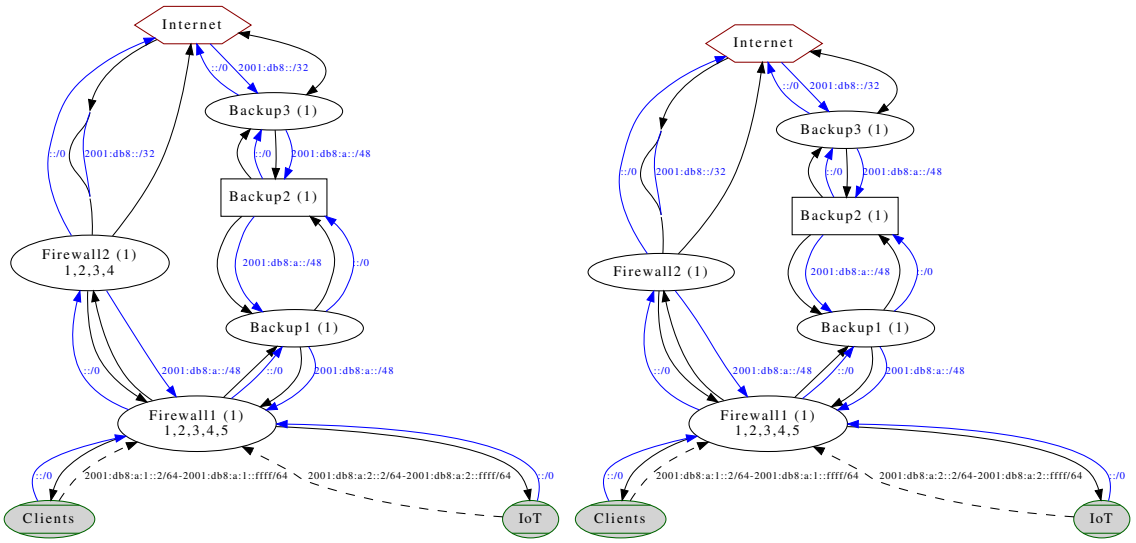
8.1.2.1. Network with Full Trust, Central Router & Border Router

In this subsection, the use case defined in subsection 4.3.1 is evaluated.

The policies regarding connection states for `Clients` are implemented similarly to the previous use cases as rules with the *id* one and two as shown in table 8.6. To block TCP port 548 to `Server1` and `Server3` four rules are defined:

- Drop packets with *protocol* = 6 (TCP according to IP protocol numbers defined by the Internet Assigned Numbers Authority, IANA[1]) and *port* = 548 with source addresses from address 0 (`::`) to the exclusive beginning of the network (`2001:db7:ffff:...`)

[1] `https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml`



(a) Default Variant

(b) `Firewall1`-weighted Variant

Figure 8.4.: Network model and rule distribution for use case 4.2.5 "Router with Redundant Upstream Gateways and Asymmetric Path"

and destination addresses of `Server1` (`2001:db8:a:2::2`) and `Server2` (`2001:db8:a:4::2`), represented by the rules with *id*s 4 and 6.

- Drop packets with *protocol* = 6 and *port* = 548 with source addresses from the first address outside the network (`2001:db9::`) to the maximum IPv6 address (`ffff:...`) and destination addresses of `Server1` and `Server2`, represented by the rules with *id*s 3 and 5.

The distributions for the two network variants are shown in figure 8.5. As both solutions are *correct*, both networks can be implemented and thus the distributed variant in figure 8.5b allows a load reduction of `Firewall1` while still fulfilling the policies. Both distributions are *minimal* as well.

In all solutions, the ESDFs d_1 and d_2 are zero, as no matching end system is capable of implementing firewall rules. Rules $3 - 6$ reach an ESDF of one in both solutions and thus are fully distributed according the definition 8.3.

8.1.2.2. Network with Full Trust, Central Router & Border Router, Decentralized Server

In this subsection, the use case defined in subsection 4.3.2 (a variant of the previous use case 4.3.1) is evaluated.

To implement the changed trust definitions in comparison to the previous use case, the trust levels of `Border1`, `Firewall1` and `Server3` are increased to 2 and the *trust_requirement* of rules 3-6 is increased to 2 as well as shown in table 8.7.

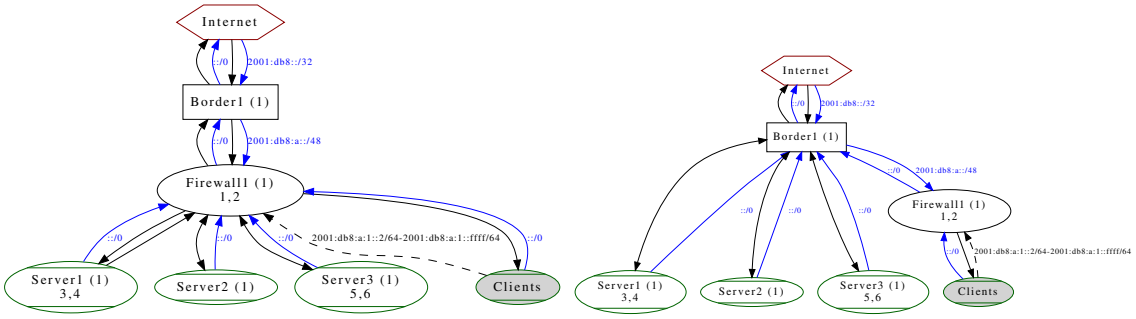
The distributions for the two network variants are shown in figure 8.6. In the centralized variant (figure 8.6a), the changed trust values cause rules 3 and 4 to be implemented in

id	*src_addr*	*dst_addr*	*action*
1	[::, 2001:db7:ffff:ffff:...:ffff:ffff]	[2001:db8:a:1::2, 2001:db8:a:1::ffff]	DROP
2	[2001:db9::, ffff:ffff:...:ffff:ffff]	[2001:db8:a:1::2, 2001:db8:a:1::ffff]	DROP
3	[::, 2001:db7:ffff:ffff:...:ffff:ffff]	2001:db8:a:2::2	DROP
4	[2001:db9::, ffff:ffff:...:ffff:ffff]	2001:db8:a:2::2	DROP
5	[::, 2001:db7:ffff:ffff:...:ffff:ffff]	2001:db8:a:4::2	DROP
6	[2001:db9::, ffff:ffff:...:ffff:ffff]	2001:db8:a:4::2	DROP

id	*trust_requirement*	*tcp_state_established*	*tcp_state_related*	*protocol*	*port*
1	1	false	false	NULL	NULL
2	1	false	false	NULL	NULL
3	1	NULL	NULL	6	548
4	1	NULL	NULL	6	548
5	1	NULL	NULL	6	548
6	1	NULL	NULL	6	548

(6 rows)

Table 8.6.: Policy definition for use case 4.3.1 "Network with Full Trust, Central Router & Border Router"



(a) Centralized Variant

(b) Distributed Variant

Figure 8.5.: Network model and rule distribution for use case 4.3.1 "Network with Full Trust, Central Router & Border Router"

`Firewall1` instead directly in `Server1` as the rules' *trust_requirement* is not met. In the distributed variant (figure 8.6b), the change in trust causes rules 3 and 4 to be implemented in `Border1`. `Border1` can implement these rules as they do not require stateful firewalling and it meets the rules' *trust_requirement* as well.

Both distributions meet the *correctness* definition and are *minimal*. Compared to the previous use case, the ESDFs d_3 and d_4 are reduced to zero while rules 5 and 6 still reach an ESDF of one.

8.1.2.3. Multi Stage Firewall

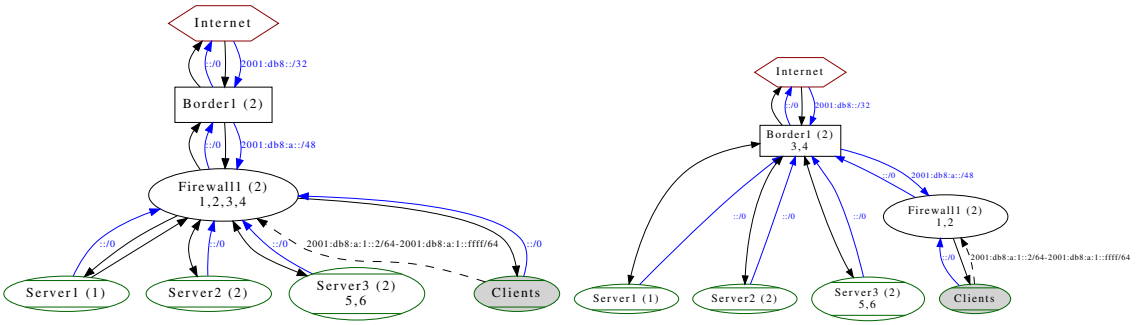
In this subsection, the use case defined in subsection 4.3.3 is evaluated.

id	*src_addr*	*dst_addr*	*action*
1	[::,2001:db7:ffff:ffff:...:ffff:ffff]	[2001:db8:a:1::2,2001:db8:a:1::ffff]	DROP
2	[2001:db9::,ffff:ffff:...:ffff:ffff]	[2001:db8:a:1::2,2001:db8:a:1::ffff]	DROP
3	[2001:db9::,ffff:ffff:...:ffff:ffff]	2001:db8:a:2::2	DROP
4	[::,2001:db7:ffff:ffff:...:ffff:ffff]	2001:db8:a:2::2	DROP
5	[2001:db9::,ffff:ffff:...:ffff:ffff]	2001:db8:a:4::2	DROP
6	[::,2001:db7:ffff:ffff:...:ffff:ffff]	2001:db8:a:4::2	DROP

id	*trust_requirement*	*tcp_state_established*	*tcp_state_related*	*protocol*	*port*
1	1	false	false	NULL	NULL
2	1	false	false	NULL	NULL
3	2	NULL	NULL	6	548
4	2	NULL	NULL	6	548
5	2	NULL	NULL	6	548
6	2	NULL	NULL	6	548

(6 rows)

Table 8.7.: Policy definition for use case 4.3.2 "Network with Full Trust, Central Router & Border Router, Decentralized Server"



(a) Centralized Variant (b) Distributed Variant

Figure 8.6.: Network model and rule distribution for use case 4.3.2 "Network with Full Trust, Central Router & Border Router, Decentralized Server"

Firewall rules handling connection state towards `Clients` and `MobileClients` are handled equally like in all other use cases and are handled by the rules with the *id*s 1-4 as shown in table 8.8. To enforce the global policy to `DROP` TCP port 25 from the internet for all servers but mail servers, all end system ranges *but* the mail server's (`2001:0DB8:a:3::/64`) receive rules which `DROP` TCP packets on port 25 from the internet in a similar pattern as in previous use cases. As those rules (*id*s 5-10) must not be bypassed by decentralized systems, their *trust_requirement* is set to 2 as well as the trust levels of all systems but the `DecentralServer1`. To block all remaining traffic from the internet to `DecentralServer1`, rules 11 and 12 are added. As this is not a global policy, the *trust_requiremnent* for this rule is set to 1. Finally, to block traffic from `MobileClients` to `Clients` rule 13 is added.

The distributions for the two network variants are shown in figure 8.7. Both distributions are *correct* and *minimal*. Compared to the centralized variant (figure 8.7a), `Stage2` needs to implement 2 rules less in the distributed variant and `Stage2` only need to firewall `Clients` so the distributed variant can be seen as an optimization of the network compared to the centralized variant.

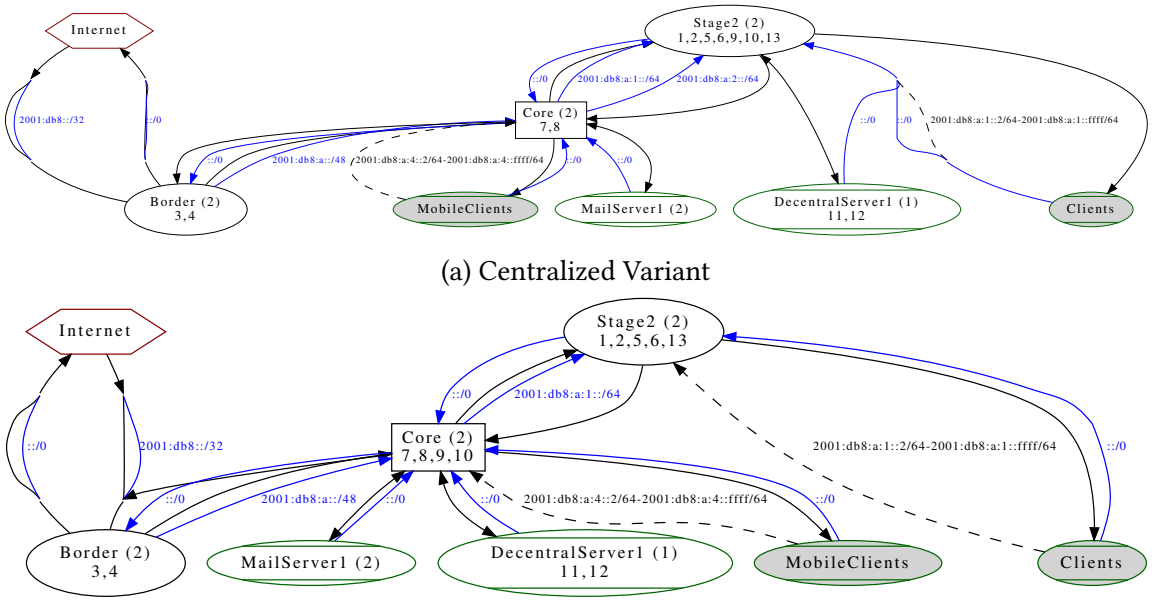
In all solutions, the ESDF of rule 1-8 and 13 is zero, as no matching end system is capable of implementing firewalls. The ESDF of rules 9 and 10 is always zero as well, as no matching end systems meets the rules' *trust_requirement*. In both solutions, rules 9 and 10 reach an ESDF of 1 as all matching end systems (`DecentralServer1`) implement the rules.

id	*src_addr*	*dst_addr*	*action*
1	[::, 2001:db7:ffff:ffff:...:ffff:ffff]	[2001:db8:a:1::2, 2001:db8:a:1::ffff]	DROP
2	[2001:db9::, ffff:ffff:...:ffff:ffff]	[2001:db8:a:1::2, 2001:db8:a:1::ffff]	DROP
3	[::, 2001:db7:ffff:ffff:...:ffff:ffff]	[2001:db8:a:4::2, 2001:db8:a:4::ffff]	DROP
4	[2001:db9::, ffff:ffff:...:ffff:ffff]	[2001:db8:a:4::2, 2001:db8:a:4::ffff]	DROP
5	[::, 2001:db7:ffff:ffff:...:ffff:ffff]	[2001:db8:a:1::2, 2001:db8:a:1::ffff]	DROP
6	[2001:db9::, ffff:ffff:...:ffff:ffff]	[2001:db8:a:1::2, 2001:db8:a:1::ffff]	DROP
7	[::, 2001:db7:ffff:ffff:...:ffff:ffff]	[2001:db8:a:4::2, 2001:db8:a:4::ffff]	DROP
8	[2001:db9::, ffff:ffff:...:ffff:ffff]	[2001:db8:a:4::2, 2001:db8:a:4::ffff]	DROP
9	[::, 2001:db7:ffff:ffff:...:ffff:ffff]	[2001:db8:a:2::2, 2001:db8:a:2:ffff:...]	DROP
10	[2001:db9::, ffff:ffff:...:ffff:ffff]	[2001:db8:a:2::2, 2001:db8:a:2:ffff:...]	DROP
11	[::, 2001:db7:ffff:ffff:...:ffff:ffff]	[2001:db8:a:2::2, 2001:db8:a:2:ffff:...]	DROP
12	[2001:db9::, ffff:ffff:...:ffff:ffff]	[2001:db8:a:2::2, 2001:db8:a:2:ffff:...]	DROP
13	[2001:db8:a:4::2, 2001:db8:a:4::ffff]	[2001:db8:a:1::2, 2001:db8:a:1::ffff]	DROP

id	*trust_requirement*	*tcp_state_established*	*tcp_state_related*	*protocol*	*port*
1	1	false	false	NULL	NULL
2	1	false	false	NULL	NULL
3	1	false	false	NULL	NULL
4	1	false	false	NULL	NULL
5	2	NULL	NULL	6	25
6	2	NULL	NULL	6	25
7	2	NULL	NULL	6	25
8	2	NULL	NULL	6	25
9	2	NULL	NULL	6	25
10	2	NULL	NULL	6	25
11	1	NULL	NULL	NULL	NULL
12	1	NULL	NULL	NULL	NULL
13	1	NULL	NULL	NULL	NULL

(11 rows)

Table 8.8.: Policy definition for use case 4.3.3 "Multi Stage Firewall"



(a) Centralized Variant

(b) Distributed Variant

Figure 8.7.: Network model and rule distribution for use case 4.3.3 "Multi Stage Firewall"

8.1.3. Summary

In this subsection, all use case evaluation results are summarized and shown in table 8.9.

While the algorithm is able to produce *minimal* results it is not guaranteed to do so, as the evaluation of "Router with Redundant Upstream Gateways and Asymmetric Paths" (subsection 8.1.1.5) with default weights shows. For all given use cases, the algorithm produces *correct* results.

The existence of $ESDF_{\max} > 0$ shows, that the algorithm is able to reach the goal of utilizing existing firewalling capabilities on end systems where possible.

8.2. Running Time

In this section, the worst case running time of the unoptimized algorithm as in chapter 6 defined and implemented in chapter 7 is briefly analyzed.

In the first stage, the algorithm iterates once over all nodes N in order to distribute rules on nodes which can be directly implemented. The lookup processes for query rules and nodes are considered constant. This yields the following running time for the first stage:

$$O(|N|) \tag{8.1}$$

In the second stage, the algorithm needs to iterate over all remaining rules, which are worst case all rules R if the first stage couldn't implement any rules. Within each iteration, the current implementation needs to traverse all paths between all nodes in case of rules, which match the whole network including the internet. This yields the following running

Use Case	Correct?	Loc. Minimal?	$ESDF_{\max}$
Router with Redundant Upstream Gateways[†♡]	✓	✓	0
Network without Full Trust, Router with Redundant Upstream Gateway[†♡]	✓	✓	0
Router with Redundant Upstream Gateways and Hosts in same L2 Domain[†♡]	✓	✓	0.5
Router with Redundant Upstream Gateways and Decentral Hosts in same L2 Domain	(✓)[a]	N/A	N/A
Router with Redundant Upstream Gateways and Asymmetric Path[†♡]	✓	✗[†]/✓[♡]	0
Network with Full Trust, Central Router & Border Router[†]	✓	✓	1
Network with Full Trust, Central Router & Border Router, Decentralized Server[†]	✓	✓	1
Multi Stage Firewall[†]	✓	✓	1

[†]With default weights
[♡]Weighted
[a]Algorithm fails which is correct in this case.

Table 8.9.: Comparison of evaluation results.

time for the second stage:

$$O(|R| * O(\text{Dijkstra})^2) \tag{8.2}$$

The worst case running time of Dijkstra's algorithm according to [Cor01] when assuming linear search in adjacency lists is

$$O(|N|^2) \tag{8.3}$$

In combination with 8.1, this yields a total worst case running time of

$$O(|N| + |R| * O(|N|^2)^2) = O(|N| + |R| * |N|^4) = O(|R| * |N|^4) \tag{8.4}$$

The algorithm is thus within polynomial running time.

9. Related Work

As firewalls are essential security-relevant tools for networks, research covers a lot of different aspects of firewalls. Because this work implements a distributed firewall by distributing rules on firewall-capable devices in a defined network topology, the first category of related work summarizes work regarding distributed firewall architectures in section 9.1. As the security-wise most important goal of the distribution algorithm implemented in this work is the distribution of the rules among the firewalls in a way which completely fulfills the policies, research regarding the verification of (distributed) firewalls is related to this thesis as well. Thus, the second category of related work is firewall modelling and verification, summarized in section 9.2.

9.1. Distributed Firewall Architectures

Steven Bellovin described the advantages of distributed firewalls in a 1999 USENIX article [Bel99]: Whereas conventional central firewalls separate the "trusted" local networks from the "untrusted" internet, a distributed approach allows more flexibility regarding the location of network members inside and outside the network. Bellovin proposes a centrally managed distributed architecture, which means that the firewall rules are managed in one central place which get distributed on the network members. The article assumes that not every system is centrally managed (i.e. systems might be administrated by their users) but the system administrators allow the management of firewall rules on their systems by a central "security administrator". This thesis implements this notion of a "security administrator", by creating a model of the network, which also allows local administrators to create rules centrally and automates the distribution based on trust levels of nodes and trust requirements for rules.

A way which addresses trust issues explicitly is proposed in [PM04]. The paper implements "trustworthy" firewalls, by expanding the software (firmware) running on the network interface cards (NIC) in a way that allows the remote management and implementation of the firewall directly on the NIC itself. This allows system operators to run any operating system and software they want on the systems, while the "local" firewall is being cared of by a central management controller and by the NIC itself. While this approach effectively solves the trust issues into the members of the system itself, this kind of platform is not necessarily tamper-proof. Manipulation on hardware level and on network level might still be possible. In addition to that, this approach requires specialized network hardware, which needs to interoperate with the controller infrastructure. Those solutions also reduce flexibility in regards to specialized applications, which might need direct hardware access and as less indirections in the packet path from the operation system towards the network as possible, for example if low latencies are required. Especially in a research

environment like in universities, this kind of approach is not applicable universally and still requires a higher-level model which includes "conventional" firewalls as well. This is due to for example specialized research equipment, IoT devices or BYOD clients, which does not support expansion slots required for specialized NICs. Specialized NICs may be not economically viable in mass as well. The concept of expanding the capabilities of NICs has found some adoption in the industry in recent years by the introduction of so called "Smart NICs" like for example in Nvidia's ConnectX Smart NICs[1], Intel's FPGA Smart NICs[2] or Cisco Nexus Smart NICs[3] (just to name a few), which contain dynamically programmable hardware in order to allow offloading of tasks related to networking (like low-level data processing) into the NIC. This work's model and algorithm is capable of including such a platform as well in case the adoption of those devices becomes necessary in the future.

A different use of distributed firewalls is presented in [GH01]. This paper proposes the use of distributed firewalls in end systems (so called "micro firewalls"). Here, the trust issues are solved by splitting the network apart into security domains, each controlled by a policy manager. Each domain's policy manager is responsible for the distribution and management of local policies within its policy boundary. Global policies are enforced by gateway firewalls between networks and towards the internet. While the proposed architecture strongly encourages a distributed firewalls into end systems and considers that internal system might get compromised, the proposed system relies solely on dynamic intrusion detection and tight coordination between policy controllers to react to such situations. The model in this thesis includes trust assumptions directly into the model, which causes "sensitive" policies to be implemented on more "trustworthy" infrastructure which increases the basic security of the network, but may also reduces the general flexibility in regards to the network topology because this might require the deployment of additional central firewalls.

In the context of software defined networking (SDN) a multitude of approaches for distributed firewalls have been discussed in literature [PY14; Kau+15; Cho+18; TA15]. The purpose of SDN is to increase flexibility in networks in regards to support for existing or new applications, by dynamically adapting functions of network devices (mostly SDN switches) to the required applications in software. This is achieved by decoupling the data and control plane of network devices such as switches or routers. While the data plane (i.e. the handling of traffic, e.g. packet forwarding) is handled by the network devices, the control plane (i.e. decisions on what should happen with packets) is handled in a centralized SDN controller. The controller itself receives its instructions from SDN applications, which contain the application specific logic. The interfaces towards the SDN application layer is commonly called "northbound" and the interface towards the data plane "southbound". [PY14; Kau+15] proposed a distributed firewall as a "northbound" application, which installs all rules on all networking devices. [Cho+18] added the capability of stateful firewalling in such a kind of SDN applications. [TA15] introduced dynamic rule deployment by supplying topology information to the SDN application, which increased the efficiency of

[1] `https://www.nvidia.com/en-us/networking/ethernet-adapters/`

[2] `https://www.intel.com/content/www/us/en/products/details/fpga/platforms/smartnic.html`

[3] `https://www.cisco.com/c/en/us/products/interfaces-modules/nexus-smartnic/index.html`

the firewalling process, as not all rules needed to be evaluated on every device. This is done by selectively installing firewall rules on switches upon the connection of a matching device to the switch. Whereas those approaches follow a similar goal as this work, SDN approaches are limited to SDN capable network devices. In addition to that, the SDN approaches do not consider trust toward network participants into their model. The model in this work is capable of modeling SDN firewalls as well though as this work's model provides an abstract mapping of different device types and host agents (see section 7.1).

The network hardware vendor HPE Aruba Networks implemented dynamic policy enforcement on distributed firewalls in the context of so called "Distributed Services Architectures" as described in a white paper [LM21]. The goal of this kind of architecture is to achieve scalability in fast growing data center architectures. When data centers grow fastly, centralized infrastructure (like central firewalls) need to be resized more often – the scalability is dependent on the capacity of the central infrastructure. To counter this inflexibility, Aruba proposes the implementation of firewalls (among other services) directly in the networking infrastructure – in the context of a data center this would be the top-of-rack switch (i.e. the router supplying network connectivity to the servers or appliances in a rack). The network topology and configurations of the switches is configured centrally as well as firewalling policies. As the controller is topology-aware, it is able to distribute policies to the matching switches in order to achieve a distributed firewall without the need of centralized firewalls. This requires switches with specialized programmable hardware though, which increases the cost per device. Especially in (historically) diverse and hybrid campus and datacenter networks like in a university setting, this approach is not universally applicable as the specialized hardware is not available on all sites and upgrading to such hardware may not be viable in all areas. Like with SDN, those architectures can be used as a firewall policy implementing component controlled by an host agent (see section 7.1) in this thesis's model, which allows a more abstract view of the network and policies.

Overall, different approaches to strongly distribute firewalls and their advantages have been discussed in literature. However, if this approach is implemented in existing and large networks which contain a large variety of different devices with different capabilities, a more abstract approach is required to create a (more) distributed firewall (see chapter 3). In a way, this thesis can be seen as a compromise between a strongly distributed and a "conventional" centralized approach.

9.2. Firewall Modelling, Analysis & Verification

As needed for this work, creating a model of firewalls (i.e. policies, rules and in some cases network topology) is also required when verifying firewalls. The goal of firewall verification is to ensure correctness of the firewall, i.e. that the firewall implements rules as intended. There are different categories of approaches to reach this goal [AG10]:

1. **Firewall Testing**: The firewall is tested experimentally by generating different packets and evaluating the fate of the packet. If the firewall makes the expected decisions for all generated packets, the firewall is *considered* correct. An approach for this is given in [AES09].

2. **Firewall Analysis**: The firewall is analyzed for defects or anomalies, e.g. caused by vulnerabilities or conflicts. A classification of different anomalies in the context of distributed firewalling is created in [Al-+05]. The first anomaly described by the aforementioned paper is the *Shadowing Anomaly*. This anomaly is caused by the implementation of two rules where one of them "covers" other rules, which make other rules ineffective. Another anomaly described is the *Correlation Anomaly*. Those are caused by two intersecting matchers which results in different actions. The two aforementioned anomalies has been identified in this work's problem analysis (chapter 3) as well which resulted in assumption 3.1. In the context of hypercubes, shadowing is a special case of correlation. The *Irrelevance Anomaly* is caused by rules which can never have any effects on packets on a firewall. This cannot happen in this work's implementation, as only hosts are considered for rule enforcement which are in some way related to the rule – either directly or by being on the path along the route through the network. As the evaluation (chapter 8) showed though, the *Redundancy Anomaly*, where rules are more often enforced than necessary can be observed in this work's algorithm in form of not minimal results. This anomaly is not influential on pure security considerations though as long as the firewalls are still able to enforce the rules. It will however affect the performance of the distributed firewall.

3. **Firewall Verification**: Firewall verification algorithms are used to verify that a firewall fulfills a certain property. One example of such an algorithm is given in [AG10], which verifies a firewall in $O(n^d)$ time with n being the number of rules and d the number of fields checked by a firewall. Whereas this specific verification algorithm is defined on single firewalls, it could be adapted for the use in distributed firewalls. Such an approach for the validation of distributed firewalls is presented in [GLJ08]. In that paper, the two properties validated upon the distributed firewalls are the accept property and the discard property, i.e. whether a firewall accepts or discards packets as expected in all cases. In order to achieve this the firewalls are modeled in a tree containing all paths between network nodes ("domain nodes") and intermediate firewalls. To verify the discard property it is sufficient, that one firewall on each path between two nodes affected by a discard rule fully discards a packet. This is consistent with this work's Lemma 3.3 (on each existing path between a rule's source and destination, a firewall rule needs to be implemented at least once). Because of assumption 3.5, it would be sufficient for a firewall verification in this work to only check the discard property. As of the similar construction of this thesis's distribution algorithm and firewall model compared to the verification steps executed in [GLJ08], it is very likely, that firewall validations using the paper's validation algorithm executed upon distributions created by this work's algorithm yield a successful result, given the firewalls implement the rules correctly and the rules are transformed correctly from the policy definition into the firewalls. Frameworks to query existing firewalls implementations for their decisions, which is required to verify some properties of firewalls is discussed in [LG08b] and [MWZ00].

   As shown by [AG11], the verification and redundancy checking (see *Redundancy Anomaly* in firewall analysis) of stateless firewalls are believed to be equivalent

problems and that every redundancy checking algorithm could also be used to validate a firewall and vice versa.

4. **Firewall Design**: Firewalls could be created and designed in a way to fulfill certain properties by construction. Algorithms to create single firewalls from a specification are proposed in [LG08a; GL07]. This work's algorithm and firewall model can be considered as a firewall design tool for distributed firewalls, but this work only focuses on the distribution, not on the on the (rule) construction of the individual firewalls themselves.

10. Conclusion & Future Work

In this work, the firewall distribution problem (see definition 3.1) has been analyzed in chapter 3, a model created (chapter 5) and a distribution algorithm developed (chapter 7). Use cases which on one hand aim to test the algorithm and its limits in general (basic use cases) and on the other hand aim to test, whether network optimizations can be evaluated (network optimization use cases) using real-world scenarios have been defined in chapter 4. Based on those use cases, the algorithm has been evaluated in chapter 8.

The created firewall and network model is sufficient to create a firewall rule distribution. As demonstrated by the successful rule distributions, the model provides sufficient information to algorithmically create a firewall distribution. The model is constructed with the correct level of abstraction to be able to model various real-world firewalls like presented in section 9.1. As shown by comparison with related work in the field of firewall verification in section 9.2, the model created is in line with models created to verify firewalls but adds trust which allows to create an hybrid firewall distribution including all possible firewalls.

The firewall distribution problem can be solved algorithmically. It has been shown by the design and prototype of an algorithm, that the problem can be solved in polynomial time ($\mathcal{O}(|R| * |N|^4)$ with N being nodes and R being rules, see section 8.2).

Further research into optimizations, especially regarding the second stage of the algorithm for the distribution of rules on intermediate firewalls, is necessary to reduce the worst case running time.

The algorithm is capable of utilizing end system firewalls where possible. As the evaluation showed, the algorithm's prioritization of rule enforcement on end systems is effective, if all constraints can be met. This is demonstrated by the existence of *End System Distribution Factors* > 0 in the evaluation (see chapter 8.1.3).

To materialize the abstract firewall rule representations on actual firewall implementations like `iptables`, a compiler needs to be implemented in the future which provides an interface between the abstract policy definition database and the local firewall language.

The algorithm can be used as a tool to evaluate network topology optimizations regrading firewalling capacity. The evaluation of network optimization use cases (section 4.3) showed, that the algorithm can be used to evaluate network optimizations. For example if a given network has end systems behind an internal firewall but the end systems are able to implement the firewall by themselves, it can be evaluated if the end systems can bypass the internal firewall (like evaluated in section 8.1.2.3). If the algorithm still yields a solution, this optimization can be done. This evaluation considers the trust level and all firewalling capabilities of each system in the network. Through the optimizations, reductions in infrastructure cost can be possible in some cases.

The proposed algorithm does not provide redundancy-free results. Whereas the algorithm was able to produce correct (definition 8.1) solutions for all given use cases, in general results are not redundancy-free (see definition 8.2 and evaluation in subsection 8.1.1.5). This means, that results may contain redundant rule implementations, which are not necessary to fully enforce a policy.

Additional research into effective optimizations, for example actively utilizing common nodes in paths by algorithmically assigning them higher weights, is required.

Further research into a formal verification of the distribution algorithm is required. In this work the correctness and other properties have been evaluated based on realistic use cases, but not formal proof was conducted, that distributed firewalls created by this work's algorithm fully cover all policies. Further research into the validation of the algorithm should be carried out.

A policy definition language needs to be added in order to create a user interface. As policy definitions can be only provided directly in the policy definition database, a suitable policy definition language may be defined or adapted in the future, which translates a policy definition into the policy definition database. This can be used to simplify rule definitions, for example to allow the definition of of rules like "block traffic from another external network" without the need to explicitly declare two rules, which "cut out" the local network from the complete address space (see examples in the evaluation of use cases in section 8.1). The manual definition of this type of rule might be more error-prone, as changes in the policy require at least two redundant changes in this case.

Bibliography

[Adã+14] P. Adão et al. “Mignis: A Semantic Based Tool for Firewall Configuration”. In: *2014 IEEE 27th Computer Security Foundations Symposium*. 2014, pp. 351–365. DOI: 10.1109/CSF.2014.32.

[AES09] Ehab Al-Shaer, Adel El-Atawy, and Taghrid Samak. “Automated pseudo-live testing of firewall configuration enforcement”. In: *IEEE Journal on Selected Areas in Communications* 27.3 (2009), pp. 302–314. DOI: 10.1109/JSAC.2009.090406.

[AG10] H.B. Acharya and M.G. Gouda. “Projection and Division: Linear-Space Verification of Firewalls”. In: *2010 IEEE 30th International Conference on Distributed Computing Systems*. 2010, pp. 736–743. DOI: 10.1109/ICDCS.2010.68.

[AG11] H. B. Acharya and M. G. Gouda. “Firewall verification and redundancy checking are equivalent”. In: *2011 Proceedings IEEE INFOCOM*. 2011, pp. 2123–2128. DOI: 10.1109/INFCOM.2011.5935023.

[Al-+05] E. Al-Shaer et al. “Conflict classification and analysis of distributed firewall policies”. In: *IEEE Journal on Selected Areas in Communications* 23.10 (2005), pp. 2069–2084. DOI: 10.1109/JSAC.2005.854119.

[Bel99] Steven M Bellovin. *Distributed firewalls*. Nov. 1999. URL: https://www.usenix.org/legacy/publications/login/1999-11/features/firewalls.html.

[Bre00] Eric A Brewer. “Towards robust distributed systems”. In: *PODC*. Vol. 7. 10.1145. Portland, OR. 2000, pp. 343477–343502.

[Cho+18] Ankur Chowdhary et al. “SDFW: sdn-based stateful distributed firewall”. In: *arXiv preprint arXiv:1811.00634* (2018).

[CK74] V. Cerf and R. Kahn. “A Protocol for Packet Network Intercommunication”. In: *IEEE Transactions on Communications* 22.5 (1974), pp. 637–648. DOI: 10.1109/TCOM.1974.1092259.

[Cla88] D. Clark. “The Design Philosophy of the DARPA Internet Protocols”. In: *Symposium Proceedings on Communications Architectures and Protocols*. SIGCOMM ’88. Stanford, California, USA: Association for Computing Machinery, 1988, pp. 106–114. ISBN: 0897912799. DOI: 10.1145/52324.52336. URL: https://doi.org/10.1145/52324.52336.

[Cor01] Thomas H Cormen. “Section 24.3: Dijkstra’s algorithm”. In: *Introduction to algorithms* (2001), pp. 595–601.

[Dij22] Edsger W Dijkstra. "A note on two problems in connexion with graphs". In: *Edsger Wybe Dijkstra: His Life, Work, and Legacy*. 2022, pp. 287–290.

[GH01] M. Gangadharan and Kai Hwang. "Intranet security with micro-firewalls and mobile agents for proactive intrusion response". In: *Proceedings 2001 International Conference on Computer Networks and Mobile Computing*. 2001, pp. 325–332. DOI: 10.1109/ICCNMC.2001.962615.

[GL02] Seth Gilbert and Nancy Lynch. "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services". In: *SIGACT News* 33.2 (June 2002), pp. 51–59. ISSN: 0163-5700. DOI: 10.1145/564585.564601. URL: https://doi.org/10.1145/564585.564601.

[GL07] Mohamed G Gouda and Alex X Liu. "Structured firewall design". In: *Computer networks* 51.4 (2007), pp. 1106–1120.

[GLJ08] Mohamed G. Gouda, Alex X. Liu, and Mansoor Jafry. "Verification of Distributed Firewalls". In: *IEEE GLOBECOM 2008 - 2008 IEEE Global Telecommunications Conference*. 2008, pp. 1–5. DOI: 10.1109/GLOCOM.2008.ECP.388.

[IEC7498-1] ISO/IEC. "IEC 7498-1: 1994 information technology – open systems interconnection – basic reference model: The basic model". In: *ISO standard ISO/IEC* (1994), pp. 7498–1. URL: https://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip.

[IEN30] V Strazisar and R Perlman. "Gateway Routing, An Implementation Specification". In: *IEN-30, Bolt Berenak and Newman* (1978). URL: https://www.rfc-editor.org/ien/ien30.pdf.

[Kau+15] Karamjeet Kaur et al. "Programmable firewall using Software Defined Networking". In: *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*. 2015, pp. 2125–2129.

[LG08a] Alex X Liu and Mohamed G Gouda. "Diverse firewall design". In: *IEEE Transactions on Parallel and Distributed Systems* 19.9 (2008), pp. 1237–1251.

[LG08b] Alex X Liu and Mohamed G Gouda. "Firewall policy queries". In: *IEEE Transactions on Parallel and Distributed Systems* 20.6 (2008), pp. 766–777.

[LM21] Bob Laliberte and Leah Matuson. *Creating a Distributed Services Architecture in Existing Data Center Environments*. White Paper commissioned by HPE Aruba. Oct. 2021. URL: https://www.arubanetworks.com/assets/wp/ESG-WP_Distributed-Service-Switches.pdf.

[MWZ00] Alain Mayer, Avishai Wool, and Elisha Ziskind. "Fang: A firewall analysis engine". In: *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*. IEEE. 2000, pp. 177–187.

[PM04] Charles Payne and Tom Markham. "Architecture and applications for a distributed embedded firewall". In: *Statistical Methods in Computer Security*. CRC Press, 2004, pp. 133–154.

[PY14] Justin Gregory V. Pena and William Emmanuel Yu. “Development of a distributed firewall using software defined networking technology”. In: *2014 4th IEEE International Conference on Information Science and Technology*. 2014, pp. 449–452. DOI: 10.1109/ICIST.2014.6920514.

[RFC3849] Geoff Huston, Anne Lord, and Dr. Philip F. Smith. *IPv6 Address Prefix Reserved for Documentation*. RFC 3849. July 2004. DOI: 10.17487/RFC3849. URL: https://www.rfc-editor.org/info/rfc3849.

[RFC4632] Vince Fuller and Tony Li. *Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan*. RFC 4632. Aug. 2006. DOI: 10.17487/RFC4632. URL: https://www.rfc-editor.org/info/rfc4632.

[RFC768] *User Datagram Protocol*. RFC 768. Aug. 1980. DOI: 10.17487/RFC0768. URL: https://www.rfc-editor.org/info/rfc768.

[RFC791] Jon Postel. *Internet Protocol*. RFC 791. Sept. 1981. DOI: 10.17487/RFC0791. URL: https://www.rfc-editor.org/info/rfc791.

[RFC8200] Dr. Steve E. Deering and Bob Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 8200. July 2017. DOI: 10.17487/RFC8200. URL: https://www.rfc-editor.org/info/rfc8200.

[RFC823] *DARPA Internet gateway*. RFC 823. Sept. 1982. DOI: 10.17487/RFC0823. URL: https://www.rfc-editor.org/info/rfc823.

[RFC9293] Wesley Eddy. *Transmission Control Protocol (TCP)*. RFC 9293. Aug. 2022. DOI: 10.17487/RFC9293. URL: https://www.rfc-editor.org/info/rfc9293.

[RM04] V.C. Ravikumar and R.N. Mahapatra. “TCAM architecture for IP lookup using prefix properties”. In: *IEEE Micro* 24.2 (2004), pp. 60–69. DOI: 10.1109/MM.2004.1289292.

[TA15] Thuy Vinh Tran and Heejune Ahn. “A network topology-aware selectively distributed firewall control in SDN”. In: *2015 International Conference on Information and Communication Technology Convergence (ICTC)*. 2015, pp. 89–94. DOI: 10.1109/ICTC.2015.7354501.

A. Appendix

A.1. Implementation Setup

The reference implementation in this work's complimentary material is implemented in Python and requires at least `Python 3.6`. Additional dependencies are documented in the `pyproject.toml`. The main script `fwtool` and all its dependencies can be installed for example running by `pip install .` in the project's root directory.

The schema of the PostgreSQL database is stored in the `pgschema.sql` in the project's root directory. An example inventory file (network model definition) can be found in `model.example.yml`.

A.2. Reproducing Evaluation Results

The evaluation files for the use cases are placed in the `/eval` directory. Each directory within this directory contains at least one inventory definition (network model) and one `policy.sql` (policy definition). Each `policy.sql` contains each use case's policy definition as shown in the tables in chapter 8 in PostgreSQL syntax.

The policy distributions including graphics and simple LaTeX representations of the tables can be created for each inventory can be created by running Python DoIt[1] (`doit`) in the `eval` folder.

This requires Python DoIt, Graphviz `dot`, `psql` (the PostgreSQL command line client), `ps2pdf` and `fwtool` (see section A.1) to be installed.

[1] `https://pydoit.org`