Agile Network Access Control in the Container Age

Cornelius Diekmann, Johannes Naab, Andreas Korsten, and Georg Carle

Abstract—Linux Containers, such as those managed by Docker, are an increasingly popular way to package and deploy complex applications. However, the fundamental security primitive of network access control for a distributed microservice deployment is often ignored or left to the network operations team. High-level application-specific security requirements are not appropriately enforced by low-level network access control lists. Apart from coarse-grained separation of virtual networks, Docker neither supports the application developer to specify nor the network operators to enforce fine-grained network access control between containers.

In a fictional story, we follow DevOp engineer Alice through the lifecycle of a web application. From the initial design and software engineering through network operations and automation, we show the task expected of Alice and propose tool-support to help. As a full-stack DevOp, Alice is involved in high-level design decisions as well as low-level network troubleshooting. Focusing on network access control, we demonstrate shortcomings in today's policy management and sketch a tool-supported solution. We survey related academic work and show that many existing tools fail to bridge between the different levels of abstractions a full-stack engineer is operating on.

Our toolset is formally verified using Isabell/HOL and is available as Open Source.

Index Terms—Security management, Centralized management, Operations & Administration, Tools, Access control, Policy, Firewall, Formal methods, Isabelle/HOL, Docker, Container

I. INTRODUCTION

NETWORK-LEVEL access control is a fundamental security mechanism, not only in traditional networks, but also in distributed applications, clouds, and microservice architectures. Unfortunately, configuring network-level access control still is a challenging, manual, and thus error-prone task [1]–[3]. It is a known and unsolved problem for over a decade that "corporate firewalls are often enforcing poorly written rule sets" [4]. Also, "access list conflicts dominate the misconfiguration errors made by administrators" [5]. A recent study confirms that this problem persists as a "majority of administrators stated misconfiguration as the most common cause of failure" [6]. In addition, not only is implementing a policy error-prone, but also designing it is challenging, even for experienced administrators [7].

In this article, we tell a fictional story about administrator Alice. Alice is responsible for designing and operating a distributed web application. She uses Linux containers managed by Docker [8]. The story covers both the design phase and operations. Alice is not responsible for the application logic, but she is responsible for helping the application scale and for network security. In modern terminology, Alice can be

C. Diekmann, J. Naab, A. Korsten and G. Carle are with the Department of Informatics, Technical University of Munich (TUM), Garching bei München 85748, Germany (e-mail: {diekmann, naab, korsten, carle}@net.in.tum.de).

called a DevOp or SRE (Site Reliability Engineer) [9]. Alice knows that when using Docker it is best practice to decrease the attack surface by limiting container networking [10] and our story primarily focuses on network-level access control.

We present two tools which help Alice in various situations. First, *topoS* [11] is a constructive, top-down greenfield approach for network security management. *topoS* translates high-level security goals to Linux iptables firewall configurations, which can be installed on a Docker host. The automatic translation steps prevent manual translation errors. Furthermore, *topoS* visualizes the results of all translation steps to help Alice uncover specification errors and allow low-level tuning. The second tool *fffuu* [12] is complementary to *topoS*: *fffuu* digests existing iptables rules and visualizes their filtering behavior. This direction is particularly challenging due to the vast amount of modules and low-level features which can be used in iptables. We chose Docker as a particularly challenging environment as Docker in its early days was known to "thrash and [destroy] you [*sic*] iptables rules, network interfaces hierarchy and routing tables" [13]. An overview of our tools is sketched in Figure 1.

topoS & *fffuu* are not specific to our case study. Both tools are formally verified [14], [15] with Isabelle/HOL [16]. Isabelle is an LCF-style interactive theorem prover; the correctness of all proven facts is based on the correctness of a small mathematical inference kernel. This architecture is very robust and not a single bug which practically affects a user's proof emerged since nearly 20 years. As Isabelle is an interactive proof assistant—in contrast to automated theorem provers—proofs in Isabelle often require a significant amount of work. In return, Isabelle provides a high level of confidence about the correctness of the proven facts. Our tools *topoS* and *fffuu* took several years to be developed and verified. As a result, we contribute formally verified tools which are proven correct for all inputs, can run stand-alone without Isabelle, do not require any manual proof from Alice, nor expose overformalization. Their core functionality can also be reused as a library in further projects. In this article, we will not present the formal background [7], [12], [14], [15], [17], instead, we demonstrate applicability from an operator's point of view; not requiring a single formula.

It would have been possible to carry out the development in a different interactive theorem prover, for example Coq [18]. In contrast, tools such as model checkers, automated theorem provers (atp), or smt solvers are not sufficient for this task. Traditional model checkers are unsuitable since one cannot simply exhaust all the state space of our model (for example, our model includes an arbitrary function to model an oracle for iptables match conditions, thus also supporting match conditions which are not even developed yet [19]). In addition, atps and smt solvers usually fail or time out on the complicated
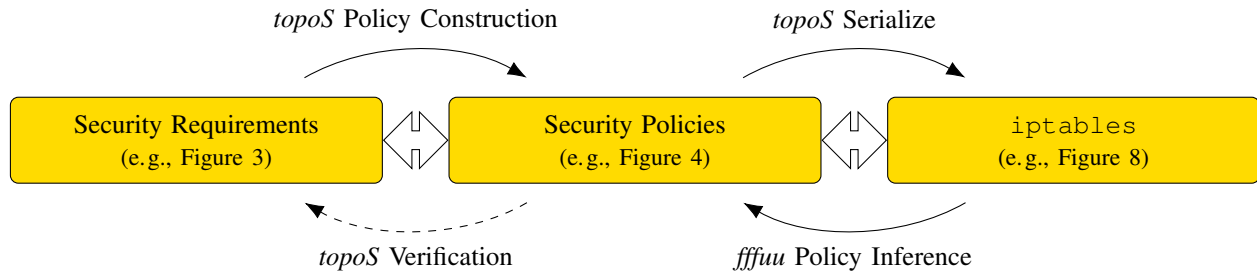
Fig. 1. Overview of the Tools *topoS* and *fffuu* Bridging Between High Abstraction Levels (left) and Low-Level Details (right)

proof obligations. Isabelle employs many state-of-the-art atps and smt solvers to help automatically discharging proof obligations, but very often, the core ideas of a proof or ingenious helping lemmas are discovered by a human.

This article is partly based on our previously published paper [11]. Our previous publication discusses the design phase and provides a formally-verified method to translate security requirements to a security policy (§II). In this article, we use those initial results and show its integration with Docker (§III). Additionally, we use *fffuu* to verify the low-level iptables rules, which is crucial for the non-trivial interaction of the Docker-generated rules and our *topoS*-generated `iptables` rules. Ultimately, this enables the usage of our tools not only in a clean slate design, but also in non-trivial operations (§IV). To the best of our knowledge, this is the first time that formally-verified tools are presented to help operators bridging the gaps between the abstraction level of Figure 1 in both directions. Our key contribution are:

- We apply the formally verified tools *topoS* and *fffuu* for network access control management in container cloud environments.
- We investigate how a network operator can create a formally verified firewall ruleset based on high level security goals.
- We provide a method to easily understand feedback on changes to the low level firewall rules.
- We review the related work to show that this is the first time, that a comprehensive solution to map from security policies to enforcement device implementation and back has been provided.

The rest of this article is structured as follows. We tell how Alice is designing the network in Section II (based on previous publication [11]). Alice deploys her setup with Docker in Section III. In Section IV, the service goes live and we track Alice as operator. We present related Docker work in Section V and related academic work in Section VI (extending on previous publication [11]). Finally, we discuss at the example of the fictional story in Section VII how the two tools presented enhance the state-of-the-art.

II. DESIGNING THE NETWORK WITH *topoS*

The security requirements of distributed applications depend on the usage scenario. Alice utilizes the tool *topoS* to configure and design the network architecture according to the needs

of her specific web application. Alice specifies the high-level security requirements and *topoS* synthesizes the low-level iptables rules for her. *topoS* suggests the following workflow:

A. Formalize high-level security goals
   a) Categorize security goals
   b) Add scenario-specific knowledge
   c) Auto-complete information (⋆)
B. Construct security policy (⋆)
C. Construct stateful policy (⋆)
D. Serialize iptables configurations (⋆)

All steps annotated with an asterisk are automated by *topoS*. As the (⋆)-steps illustrate, once the security goals are specified, the process is completely automatic. Between the automated steps, Alice may manually refine the intermediate result. *topoS* supports re-verification of the manual refinement to prevent the introduction of human errors.

The automated intermediate (⋆)-steps are proven correct for all inputs [14]. The proofs are machine-verified with Isabelle/HOL [16]. Thus, it is guaranteed that *topoS* performs correct transformations. As a side note, since the intermediate transformations are proven correct once and for all for all inputs, Alice does not need to prove anything manually. The final serialization of iptables configurations is not verified since it is merely syntactic rewriting of the result of the previous step. Alice will also later change this rewriting slightly to better accommodate for her Docker environment. To prevent errors in this ad-hoc low-level step, Alice later verifies her resulting configuration with *fffuu*.
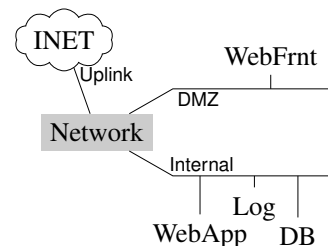

Fig. 2. Network Schematic

Now we consider the actual web application. The scenario was chosen because it has been used previously [11], is minimal and comprehensible for an article, but also realistic and features many important aspects. Alice schematically illustrates the overall architecture in Figure 2. The grey box

represents the Docker network. The setup hosts a news aggregation web application, accessible from the Internet (*INET*). It consists of a web application backend server (*WebApp*) and a frontend server (*WebFrnt*). The *WebApp* is connected to a database (*DB*) and actively retrieves data from the Internet. All servers send their logging data to a central, protected log server (*Log*).

Alice implements the network-related aspects of the scenario with different protocols and technologies. The custom backend, the *WebApp* is written in `python`. The *WebFrnt* runs `lighttpd`. It serves static web pages directly and retrieves dynamic websites from the *WebApp*. All components send their `syslog` messages via UDP (RFC 5426 [20]) to *Log*. Since the implementation details are irrelevant, we prototyped the setup and checked connectivity with `busybox` container images.

For details on the architecture and working principles of *topoS* we refer to the original publication [7].

A. Formalizing High-Level Security Goals

Formalizing the security goals, i.e., step A. in the process of using *topoS*, is the most crucial and manual part. First, Alice collects the entities in her setup: *INET*, *WebApp*, *WebFrnt*, *DB*, and *Log*. Now, *topoS* provides a modular, attribute-based language [7] to specify the security requirements. *topoS* comes with a pre-defined library of security invariant templates as listed in Table I.

Name	Description
Simple BLP	Simplified Bell-LaPadula
Bell-LaPadula	Label-based Information Flow Security with trusted entities
ACL	Simple ACLs (Access Control Lists)
Comm. With	White-listing transitive ACLs
Not Comm. With	Black-listing transitive ACLs
Dependability	Limit dependence on certain hosts
Domain Hierarchy	Hierarchical control structures
NoRefl	Allow/deny reflexive flows. Can lift symbolic policy identifiers to role names (e.g., symbolic host name corresponds to an IP range.)
NonInterference	Transitive non-interference properties
PolEnforcePoint	Central application-level policy enforcement point. Master/Slave relationships.
Sink	Information sink. Hosts (or host groups) must not publish any information
Subnets	Collaborating, protected host groups
SubnetsInGW	Simple, collaborating, protected or accessible host groups
Simple Tainting	Simplified label-based Privacy
Tainting	Label-based Privacy with untainting

TABLE I
SECURITY INVARIANT TEMPLATES DEFINED BY *topoS*

A template formalizes generic, scenario-independent aspects of a security goal and must be formally defined in *topoS* using Isabelle/HOL. Alice only instantiates those templates by adding scenario-specific information. She does so by assigning attributes to entities. *topoS* does not require Alice to assign attributes to all entities. Alice must only provide all security-relevant information and *topoS* auto-completes the missing

values with provably secure default values [7]. For this scenario, Alice instantiates four invariant templates to define her security goals. We now explain the invariants in the language of *topoS*. Figure 3 shows the final specification Alice writes.

Subnets $\{DB \mapsto$ internal,
$\qquad Log \mapsto$ internal,
$\qquad WebApp \mapsto$ internal,
$\qquad WebFrnt \mapsto$ DMZ$\}$

Sink $\{Log \mapsto$ sink$\}$

Bell LaPadula $\{DB \mapsto$ confidential,
$\qquad Log \mapsto$ confidential,
$\qquad WebApp \mapsto$ declassify$\}$

ACL $\{DB \mapsto$ Access allowed by : $WebApp\}$

Fig. 3. Security Invariants

1) First, as illustrated in Figure 2, *DB*, *Log*, and *WebApp* are considered internal hosts. Alice uses a template called *Subnets*. She labels internal hosts with the internal attribute. The *WebFrnt* must be accessible from outside, it is a classical DMZ member and labeled accordingly.
2) Next, Alice wants to ensure that logging data must not leave the log server. Therefore, using a template called *Sink*, she classifies *Log* as information sink.
3) Using a template called *Bell LaPadula*, Alice specifies that *DB* contains confidential information. Since it sends its log data to the log server, she labels *Log* as confidential. Finally, the *WebApp* is allowed to retrieve data from the *DB* and to publish it to the *WebFrnt*. Therefore, the *WebApp* is trusted and allowed to declassify data.
4) A traditional access control list, using the *ACL* template, specifies that only *WebApp* may access the *DB*.

This is all the information *topoS* needs to operate. Based on this, *topoS* can now compute a security policy as access control graph, shown in Figure 4. This policy is much more fine-grained than the simple DMZ architecture Alice initially drafted in Figure 2.

For the sake of brevity, our story omits an important aspect: Alice did not specify a perfect set of security goals at the first attempt. It took her some iteration to arrive at the specification shown in Figure 3. Fortunately, *topoS* has proven extremely helpful in this process. As described in the previous paragraph, once Alice instantiates a set of security invariant templates, *topoS* can compute a policy from them. This provides Alice with feedback about what she is specifying and what her specification actually means. In addition, *topoS* also allows Alice to define her own policy and *topoS* visualizes any flow which contradicts a specified security goal or highlights flows which Alice did not consider but which would be valid w.r.t. specified security goals. Alice iterated this process several times and refined her security goals until she was certain that the specification carries the intended meaning.

Due to this motivating feedback-driven process, Alice has now documented a clear and formal specification of her

security requirements. The specification is modular and split into four invariants. Consequently, in the future, it is easy to add new security requirements or verify whether a proposed change violates the existing invariants.

B. Policy Construction with topoS

Given the specification of the security goals (Figure 3), *topoS* computes the security policy shown in Figure 4. Alice is happy with the policy, but she makes one small change. While the web frontend must be accessible from the Internet, there is currently no need that the web frontend also establishes connections to the Internet by itself. Alice modifies the policy as shown in Figure 5 and *topoS* verifies that her new policy complies with the security goals specified earlier.

A careful reader may already notice the subtle semantics we gave to the arrows in Figure 5: For example, the arrow from the Internet to the web frontend means that the Internet may set up connections to the web frontend, but not vice versa. A connection usually implies that—once it is established—packets may flow in both directions. But the policy does currently not permit packets to flow from the web frontend to the Internet. Therefore, using the specification of the security goals again, *topoS* can convert a security policy into a stateful policy [17]. The result is shown in Figure 6. The orange dashed arrows represent connections which may be stateful, i.e., once established by the corresponding entity (solid black arrow), packets may travel in both directions (solid black + dashed orange arrow).

To compute the stateful policy, *topoS* distinguishes between access control (invariants 1 and 4) and information flow (invariants 2 and 3). This distinction is inherently built into the invariant templates [7] and Alice does not need to configure anything for this step. In this scenario, the two stateful flows can be justified as they do not introduce access control violations since the initiator needs to establish a connection before a bidirectional flow is allowed. However, it can be seen that all flows which send data to the logging server are not allowed to be stateful. This is because of information flow security: a bidirectional channel here would allow the confidential logging information to leak.

III. Deploying to a Docker Host

Now, Alice wants to enforce the stateful policy of Figure 6 on her Docker host. Docker uses the iptables firewall internally. The first question Alice raises is whether she wants to operate custom firewall rules or whether Docker can enforce her policy out of the box.

Since Docker version 1.10, it is possible to create custom internal networks [21]. The `--internal` flag protects a network from accesses from the outside. However, by default, all containers in an internal network can reach each other. It is possible to configure the network such that containers cannot access each other. Unfortunately, a bug in Alice's version of Docker allowed containers to communicate [22]. In addition, there was no possibility to enable fine-grained access control between the containers in a network. A `--link` option exists to connect two containers within a custom network, but

it merely sets environment variables, it does not influence the actual IP connectivity. The Docker design philosophy is to decouple the application developer from networking details [23]. Only a coarse-grained network abstraction in terms of different networks is exposed to the application developer. The network IT team —i.e., Alice— should manage the network. Since one compromised container in an internal network can attack all other containers in its network, Alice desires further network-level access control than the coarse grained separation provided by Docker.

The web application as a whole should be isolated from other containers on the host. Therefore, Alice creates a new Docker network. She disables inter-container communication (icc) for this network. Due to the lack of fine-grained access control, Alice decides to install additional custom firewall rules on her host. In addition, the fact that the fix to the bug mentioned above was initially not considered a security issue by the Docker developers confirms Alice's choice of running her own firewall. As it turns out, many administrators are looking for means to fine-tune their Docker firewall because of certain shortcomings in Docker [24]. Alice chooses to enhance the default Docker-generated rules with rules for fine-grained access control generated by *topoS*. She merges both rulesets manually. Figure 8 shows the final ruleset of her Docker host. Everything which does not mention `MYNET` is part of the default ruleset of docker. Alice added the custom chain `MYNET` and a jump to it. The contents of the chain `MYNET`, i.e., the actual fine-grained access control rules, are generated by *topoS*. For brevity, the name of the bridging interface created by Docker for Alice's new network is `dbr`.

As a side note, Alice is well aware that Docker itself does not provide security out of the box [25], [26]. The security provided by containerization heavily depends on the specific Docker and container setup. For example, if a container has the privilege to access raw sockets, the container can spoof arbitrary IP addresses or perform ARP spoofing [27]. This spoofing cannot be blocked by traditional firewall rules since all containers in Alice's network are connected to the same interface `dbr`. Hence, in a generic case, Docker networking does not provide authenticity. Consequently, since a malicious, privileged container can perform ARP spoofing, no guarantees can be given that the connectivity structure enforced by Alice's firewall actually corresponds to Figure 6. Alice solves this problem by teaching her application developers about Linux capabilities and rejecting containers which require any additional capabilities or are otherwise 'naked' [28].

IV. Operations

The design and development of the firewall policy are only the initial phase of the app's life cycle. After the firewall has been deployed, additional and changing requirements necessitate updates to the previously designed ruleset. In contrast to network design and tool development, operations is usually an unstructured ad-hoc discipline which has to deal with unexpected events and usually is not taught in any course of studies [29]. In this section, we follow Alice while she deals with unexpected events, ad-hoc requests, overlapping issues,
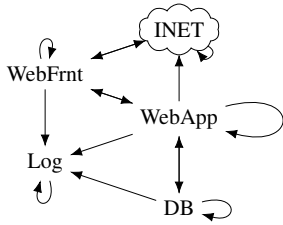
Fig. 4. Security Policy
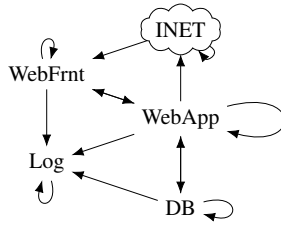(computed by *topoS*)
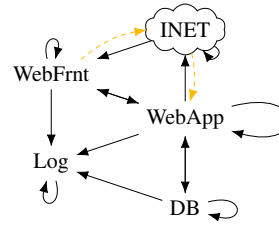

Fig. 5. Security Policy
(manually refined)

Fig. 6. Stateful Policy
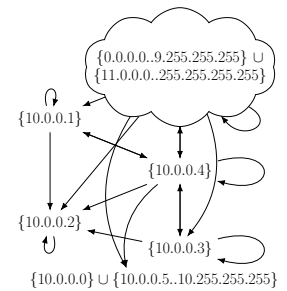(computed by *topoS*)

Fig. 7. Firewall Overview
(reconstructed by *fffuu*)

```
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:DOCKER - [0:0]
:DOCKER-ISOLATION - [0:0]
:MYNET - [0:0]
-A FORWARD -j DOCKER-ISOLATION
-A FORWARD -j MYNET
-A FORWARD -o dbr -j DOCKER
-A FORWARD -o dbr -m conntrack                        ↵
        --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -i dbr ! -o dbr -j ACCEPT
-A FORWARD -o docker0 -j DOCKER
-A FORWARD -o docker0 -m conntrack                    ↵
        --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -i docker0 ! -o docker0 -j ACCEPT
-A FORWARD -i docker0 -o docker0 -j ACCEPT
-A FORWARD -i dbr -o dbr -j DROP
-A DOCKER-ISOLATION -i docker0 -o dbr -j DROP
-A DOCKER-ISOLATION -i dbr -o docker0 -j DROP
-A DOCKER-ISOLATION -j RETURN
-A MYNET -m state --state ESTABLISHED                 ↵
       ! -i dbr -o dbr -d 10.0.0.4 -j ACCEPT
-A MYNET -m state --state ESTABLISHED                 ↵
        -i dbr -s 10.0.0.1 ! -o dbr -j ACCEPT
-A MYNET -i dbr -s 10.0.0.1 -o dbr -d 10.0.0.1 -j ACCEPT
-A MYNET -i dbr -s 10.0.0.1 -o dbr -d 10.0.0.2 -j ACCEPT
-A MYNET -i dbr -s 10.0.0.1 -o dbr -d 10.0.0.4 -j ACCEPT
-A MYNET -i dbr -s 10.0.0.3 -o dbr -d 10.0.0.3 -j ACCEPT
-A MYNET -i dbr -s 10.0.0.3 -o dbr -d 10.0.0.2 -j ACCEPT
-A MYNET -i dbr -s 10.0.0.3 -o dbr -d 10.0.0.4 -j ACCEPT
-A MYNET -i dbr -s 10.0.0.2 -o dbr -d 10.0.0.2 -j ACCEPT
-A MYNET -i dbr -s 10.0.0.4 -o dbr -d 10.0.0.1 -j ACCEPT
-A MYNET -i dbr -s 10.0.0.4 -o dbr -d 10.0.0.3 -j ACCEPT
-A MYNET -i dbr -s 10.0.0.4 -o dbr -d 10.0.0.2 -j ACCEPT
-A MYNET -i dbr -s 10.0.0.4 -o dbr -d 10.0.0.4 -j ACCEPT
-A MYNET -i dbr -s 10.0.0.4 ! -o dbr -j ACCEPT
-A MYNET ! -i dbr -o dbr -d 10.0.0.1 -j ACCEPT
-A MYNET -i dbr -j DROP
COMMIT
```

Fig. 8. Docker `iptables` firewall (only `filter` table shown).

and long-term improvements of her setup within limited time budget.

Alice used *topoS* to help her with *static* configuration management. Consequently, Alice now needs tools to help her in *dynamic* contexts. Alice installs a cronjob which runs `iptables-save` regularly and looks for changes. If a change to the ruleset is discovered, *fffuu* is run to compute an overview of the firewall policy currently enforced. The result is visualized with tikz and emailed to Alice. The result for the initial firewall rules is visualized in Figure 7.

For an deeper look into operating principles and implementation of *fffuu* we refer to the original publication [12].

Since *fffuu* operates on the raw iptables rules, it does not know the hostnames of the containers. Further comparing the

designed policy in Figure 6 with Figure 7, Alice realizes that her firewall policy is already non-optimal. The reason is that the default Docker-generated rules are interacting with her rules. One oddity in *fffuu*'s visualization is the IP range at the bottom which corresponds to all unused IPs in Alice's new network 10.0.0.0/24. Those IPs are potentially accessible from the Internet and the $WebApp$. Since no containers are assigned to these IP addresses, Alice decides that this is acceptable for now. For her, a more concerning observation is that the Internet can access almost all internal containers. But Alice knows that she does not export internal services to the Internet in her Docker configuration, which is not visible in the firewall setup. Due to time pressure, Alice decides that the current firewall rules are good enough and she will add a second line of defense later. This will also protect against attacks from containers running on the same host but in a different Docker network. Alice runs one last check with *fffuu*, verifying that $INET \rightarrow WebFrnt$ (10.0.0.1) is the only stateful flow; as discussed, $WebApp \rightarrow INET$ is already allowed bidirectionally. Due to time constraints, Alice decides not to ask *fffuu* about stateful flows furthermore.

On Friday afternoon, Alice receives a call from a friend at heise.de (193.99.144.80), who is complaining that one of her containers is pinging his webserver excessively. Alice knows that the web backend tests connectivity from time to time by sending one echo request to Heise. She decides to postpone investigating the core of the problem and installs a short-term mitigation by just rate limiting all connections to Heise. We print changes to the ruleset in unified `diff` format. Alice installs the following rules.

```
 :DOCKER-ISOLATION - [0:0]
 :MYNET - [0:0]
 -A FORWARD -j DOCKER-ISOLATION
+-A FORWARD -d 193.99.144.80 -m recent        ↵
    --set --name rateheise --rsource
+-A FORWARD -d 193.99.144.80 -m recent        ↵
    --update --seconds 60 --hitcount 3        ↵
    --name rateheise --rsource -j DROP
 -A FORWARD -j MYNET
 -A FORWARD -o dbr -j DOCKER
 -A FORWARD -o dbr -m conntrack               ↵
    --ctstate RELATED,ESTABLISHED -j ACCEPT
```

While this change to the ruleset is identified by her cronjob, *fffuu* confirms that the overall access control structure of the firewall has not changed: It still corresponds to Figure 7.

While Alice is updating the ruleset, she remembers from Figure 7 that the Internet still has too many direct access rights

to her internal containers. While working on the ruleset, Alice decides to fix this issue right away mow.

```
 -A MYNET -i dbr -s 10.0.0.4 -o dbr -d 10.0.0.3 ↵
    -j ACCEPT
 -A MYNET -i dbr -s 10.0.0.4 -o dbr -d 10.0.0.2 ↵
    -j ACCEPT
 -A MYNET -i dbr -s 10.0.0.4 -o dbr -d 10.0.0.4 ↵
    -j ACCEPT
--A MYNET -i dbr -s 10.0.0.4 ! -o dbr -j ACCEPT
--A MYNET ! -i dbr -o dbr -d 10.0.0.1 -j ACCEPT
+-A MYNET -i dbr -s 10.0.0.4                     ↵
    ! -o dbr ! -d 10.0.0.0/8 -j ACCEPT
+-A MYNET ! -i dbr ! -s 10.0.0.0/8               ↵
    -o dbr -d 10.0.0.1 -j ACCEPT
 -A MYNET -i dbr -j DROP
+-A MYNET -o dbr -j DROP
+-A MYNET -s 10.0.0.0/8 -j DROP
+-A MYNET -d 10.0.0.0/8 -j DROP
 COMMIT
```

After a few seconds, she receives the new firewall overview generated by *fffuu*, shown in Figure 9. Comparing this figure to Figure 7, it can be seen that the Internet is now appropriately constrained. Alice also happily realizes that her firewall ruleset now corresponds to the original design of Figure 6.
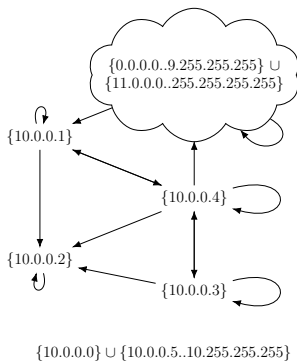

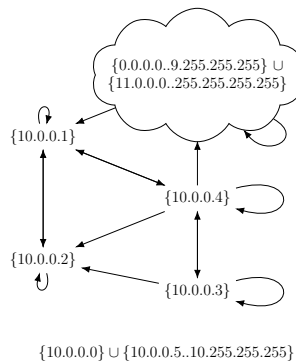
Fig. 9. Overview computed by *fffuu* after call from Heise

Fig. 10. Overview computed by *fffuu* after WebDev call (HTTP)

The next week, Alice receives a call from her web developer. He requests ssh access to all containers. In addition, he requests that the log server (10.0.0.2) may access a status page of the web frontend (10.0.0.1) over HTTP. Both permissions should only be granted temporarily for debugging purposes. Alice sets up the appropriate firewall rules.

```
 -A FORWARD -j DOCKER-ISOLATION
 -A FORWARD -d 193.99.144.80 -m recent          ↵
    --set --name rateheise --rsource
 -A FORWARD -d 193.99.144.80 -m recent          ↵
    --update --seconds 60 --hitcount 3           ↵
    --name rateheise --rsource -j DROP
+-A FORWARD -m state                             ↵
    --state ESTABLISHED,RELATED -j ACCEPT
+-A FORWARD -p tcp --dport 22 -j ACCEPT
+-A FORWARD -s 10.0.0.2 -d 10.0.0.1 -p tcp       ↵
    --dport 80 -j ACCEPT
 -A FORWARD -j MYNET
 -A FORWARD -o dbr -j DOCKER
 -A FORWARD -o dbr -m conntrack                  ↵
    --ctstate RELATED,ESTABLISHED -j ACCEPT
```

While the Docker container connectivity works as intended, *fffuu* now computes two interesting access control overviews.[1]

---

[1]By default, *fffuu* only computes the overview for a fixed service, by default ssh and HTTP.

First, it visualizes that there are no longer any restrictions for ssh, i.e., *fffuu* only shows one node. This single node comprises the complete IPv4 address space and may access itself, i.e., unconstrained ssh access. Second, *fffuu* presents a new overview of the HTTP connectivity, shown in Figure 10. The only difference is that the log server may now access the web frontend. The two overviews underline that Alice implemented her web developer's request correctly.

While the policy overview computed by *fffuu* is still comprehensible for Alice due to the pooling of equivalent IP addresses, the raw firewall ruleset, now comprising 37 rules, is slowly becoming a mess[2]. The ruleset partly contains unused artifacts installed by Docker and Alice's hot fixes are cluttered all over it. While Alice is poring over about how she could clean up the rules, she receives an emergency call. Her manager tells her that the webservice was mentioned on reddit and that the web frontend cannot cope with the increased load. Alice is spawning an additional frontend container with IP address 10.0.0.42.

Alice's firewall adheres to best practices and implements whitelisting. Consequently, the new container does not have any connectivity. Alice now is in the urgent situation to get the firewall rules set up which permit connectivity for the second frontend instance. Just permitting everything is not an acceptable option for security-aware Alice.
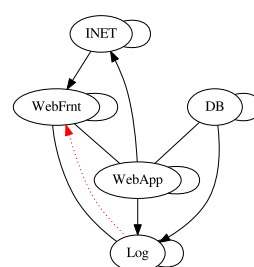

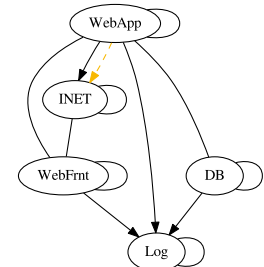Fig. 11. Uncovered violation (screenshot of *topoS*)

Fig. 12. Stateful Policy (screenshot of *topoS*)

Fortunately, Alice remembers that she has previously specified the security requirements with *topoS* (Figure 3). First, Alice loads the policy overview for HTTP computed by *fffuu* (Figure 10) into *topoS*. She immediately gets a visualization (Figure 11) which tells her that her policy has already diverged from the security requirements: The dotted red arrow, visualized by *topoS*, reveals a flow which is violating the security requirements. It corresponds to the flow installed upon the request of her web developer previously. Based on the security requirements, Alice should probably have rejected the request from her web developer in the very beginning. But this is neither the time to cast the blame nor to discuss security requirements. Given the time pressure and the many odd-looking Docker-generated rules in her ruleset, Alice decides that her current ruleset is not salvageable. Alice is not the first

---

[2]In a study of 15 real world iptables rulesets with up to 4946 rules, we observed at most 13 groups of IP addresses for the SSH or HTTP [12]. For a specific group, the behaviour can be determined by the interaction with the remaining groups

person to notice that the Docker daemon may make surprising changes to an iptables configuration [24], [30]. She decides to take over complete control over the ruleset. Therefore, Alice prohibits Docker from making any changes to the firewall by setting `--iptables=false`.

```
-A FORWARD -i dbr -s $WebFrnt -o dbr -d $WebFrnt -j ACCEPT
-A FORWARD -i dbr -s $WebFrnt -o dbr -d $Log -j ACCEPT
-A FORWARD -i dbr -s $WebFrnt -o dbr -d $WebApp -j ACCEPT
-A FORWARD -i dbr -s $WebFrnt -o $INET_iface -d $INET    ↩
   -j ACCEPT
-A FORWARD -i dbr -s $DB -o dbr -d $DB -j ACCEPT
-A FORWARD -i dbr -s $DB -o dbr -d $Log -j ACCEPT
-A FORWARD -i dbr -s $DB -o dbr -d $WebApp -j ACCEPT
-A FORWARD -i dbr -s $Log -o dbr -d $Log -j ACCEPT
-A FORWARD -i dbr -s $WebApp -o dbr -d $WebFrnt -j ACCEPT
-A FORWARD -i dbr -s $WebApp -o dbr -d $DB -j ACCEPT
-A FORWARD -i dbr -s $WebApp -o dbr -d $Log -j ACCEPT
-A FORWARD -i dbr -s $WebApp -o dbr -d $WebApp -j ACCEPT
-A FORWARD -i dbr -s $WebApp -o $INET_iface -d $INET     ↩
   -j ACCEPT
-A FORWARD -i $INET_iface -s $INET -o dbr -d $WebFrnt    ↩
   -j ACCEPT
-A FORWARD -i $INET_iface -s $INET                       ↩
   -o $INET_iface -d $INET -j ACCEPT
-I FORWARD -m state --state ESTABLISHED -i $INET_iface   ↩
   -s $INET -o dbr -d $WebApp -j ACCEPT
```

Fig. 13. Fresh ruleset generated by *topoS*, considering only the requirements

To start over, Alice asks *topoS* to compute a completely new ruleset for her, based only on the requirements specified before. *topoS* computes the stateful policy shown in Figure 12; the dashed orange flow indicates a connection with stateful semantics. The result is serialized to the firewall rules shown in Figure 13. All containers are attached to the same Docker bridge. Alice only needs to fill in the IP addresses of the machines and the Internet-facing interface. Alice sticks to the IP addresses of her containers as before, except for the web frontend which now has two active containers running. She sets $WebFrnt = 10.0.0.1, 10.0.0.42$. This syntax is supported by iptables and iptables will automatically expand this syntax to several rules upon loading. To specify the interface and IP range of the Internet, Alice defines that the Internet is 'everything except the Docker subnet'. Therefore, Alice negates her Docker interface and internal Docker IP range. For example, the second last rule becomes the following:

```
-A FORWARD ! -i dbr ! -s 10.0.0.0/8               ↩
   ! -o dbr ! -d 10.0.0.0/8 -j ACCEPT
```

Yet, for some rules, the iptables command refuses to load her rules and complains that negation is not allowed with multiple source or destination IP addresses. For example in line four, iptables prohibits the use of `! -d 10.0.0.0/8` in combination with the two source addresses `-s 10.0.0.1,10.0.0.42` specified for $WebFrnt$. To work around this iptables limitation, Alice uses the `iprange` module to declare the IP range of the Internet. For example, the fourth rule now becomes

```
-A FORWARD -i dbr -s 10.0.0.1,10.0.42 ! -o dbr    ↩
   -m iprange ! --dst-range                       ↩
   10.0.0.0-10.255.255.255 -j ACCEPT
```

This loads fine. Fortunately, *fffuu* understands those matching modules. The new firewall overview is visualized in Figure 14. It is remarkably similar to Figure 9, the last, old visualization when the ruleset was still in a good state. The main difference is that the web frontend is now represented by two machines and that it may establish connections to the Internet
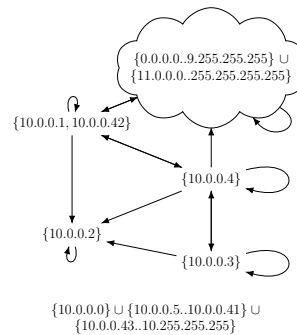

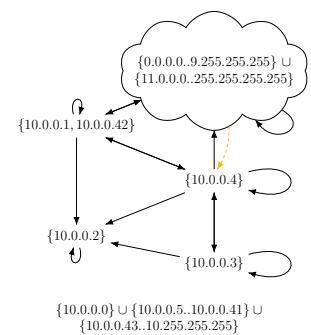Fig. 14. Overview of Figure 13 computed by *fffuu*

Fig. 15. HTTP access control overview with state (by *fffuu*)

itself. This has been prohibited by the old manually-refined policy, but it does not contradict any security requirement. A final test confirms that the container connectivity works as expected and the two frontend instances can cope with the load.

Looking at her todo list, Alice decides to install some old rules again. This time, she designs a clean ruleset and handles all of her temporary rules in a chain she calls `CUSTOM`. After her custom chain, she hands over control to the *topoS*-generated rules. Alice still has not investigated why some container is excessively pinging 193.99.144.80, so she installs the rate limiting again. Alice is more careful about the other temporary rules. *topoS* has shown her that the log server must not communicate with the web frontend, so she is not enabling this rule. However, since her ssh server is securely configured, she does not see a problem with the ssh exception and enables it again. She installs the following rules:

```
:INPUT ACCEPT [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0] +:CUSTOM - [0:0]
+-A FORWARD -j CUSTOM
+-A CUSTOM -d 193.99.144.80 -m recent            ↩
    --set --name rateheise --rsource
+-A CUSTOM -d 193.99.144.80 -m recent            ↩
    --update --seconds 60 --hitcount 3           ↩
    --name rateheise --rsource -j DROP
+-A CUSTOM -m state --state ESTABLISHED -j ACCEPT
+-A CUSTOM -p tcp -m tcp --dport 22 -j ACCEPT
 -A FORWARD -i dbr -s 10.0.0.1,10.0.0.42         ↩
    -o dbr -d 10.0.0.1,10.0.0.42 -j ACCEPT
 -A FORWARD -i dbr -s 10.0.0.1,10.0.0.42         ↩
    -o dbr -d 10.0.0.2 -j ACCEPT
 -A FORWARD -i dbr -s 10.0.0.1,10.0.0.42         ↩
    -o dbr -d 10.0.0.4 -j ACCEPT
```

Alice knows that it is a good practice to have a rule which allows all packets belonging to an established connection [31]. She definitely needs an `ESTABLISHED` rule to make ssh work so she just copies it from a guide. Though, Alice wonders why *topoS* did not generate such a rule. She becomes skeptical about her decision and wants to double check. Asking *fffuu* about the potential packet flows once a connection is initiated, *fffuu* confirms that there are currently no limitations at all, not even for HTTP. She compares this to the stateful implementation computed by *topoS*, shown in Figure 12. The dashed orange line indicates a flow with stateful semantics, i.e., packets may flow in both directions once the connection was initiated by the web application. She realizes that *topoS*

takes great care to enforce unidirectional information flow to the log server. This is due to the information sink security invariant specified in the requirements. Alice knows from recent news that a badly protected log server may leak information which may lead to the compromise of all her machines [32]. Therefore, Alice restricts her ESTABLISHED rule to ssh. She uses the multiport module which conveniently allows matching on either source port or destination port in one rule. She makes the following final adjustment to her ruleset.

```
 -A CUSTOM
 -A CUSTOM -d 193.99.144.80/32 -m recent          ←
     --set --name rateheise --rsource
 -A CUSTOM -d 193.99.144.80/32 -m recent          ←
     --update --seconds 60 --hitcount 3           ←
     --name rateheise --rsource -j DROP
--A CUSTOM -p tcp -m state --state ESTABLISHED    ←
     -j ACCEPT
+-A CUSTOM -p tcp -m state --state ESTABLISHED    ←
     -m multiport --ports 22 -j ACCEPT
 -A CUSTOM -p tcp -m tcp --dport 22 -j ACCEPT
 COMMIT
```

Alice runs one final verification of the implemented policy with *fffuu*, shown in Figure 15. This time, she also includes the stateful flows. *fffuu* identifies only one stateful flow, visualized with an orange dashed line. The direction of the stateful flow is the other way round compared to Figure 12. This is merely an artifact of the visualization, a stateful flow is essentially bidirectional once it is established. Otherwise, Figure 12 and Figure 15 show isomorphic graphs. This verifies that Alice's firewall rules are correct.[3]

V. RELATED DOCKER WORK

Tools to improve firewall management for Docker hosts exist [33], [34]. Docker-fw [33] is a convenient iptables wrapper with Docker-specific features, such as retrieving the IP address of a container using the Docker API. It currently only supports the default Docker bridge, but not custom networks. DFWFW [34] is also a convenient tool to manage the iptables firewall on a Docker host. It runs as daemon and can apply changes dynamically if the Docker setup is modified, e.g., if new containers are instantiated.

Both tools provide features that could help to make the management process with *topoS* and *fffuu* more convenient. At the moment, *topoS* generates raw iptables rules but leaves the actual IP addresses to be set by the user, e.g., $WebFrnt$ in Figure 13. To further automate the setup, *topoS* could generate Docker-fw [33] rules which automatically resolve the correct IP address. To further automate firewall management, *topoS* could directly generate DFWFW [34] configurations. This would mean that no manual configuration is required any longer if multiple instances of the same container are spawned.

Alice tests *topoS* together with DFWFW. For this, Alice simply adapts the *topoS* serialization step to generate rules in the DFWFW configuration format. Since DFWFW is also built to primarily support whitelisting, the translation is straightforward. A rule in this format first matches on the Docker network, then it allows specifying the source and destination container, an arbitrary string which will be added to the iptables match expression, and finally the iptables action. The

match on the container names permits the use of Perl regular expressions. To allow dynamic spawning of multiple instances of a container, Alice writes a regex which matches on the container name and any trailing number, e.g., webfrnt, webfrnt1, webfrnt-1, webfrnt200. The beginning of the configuration file looks as follows:

```
{
  "container_to_container": {
  "rules": [
    {
      "network": "alicewebappnet",
      "src_container": "Name =˜ ^webfrnt-?\\d*$",
      "dst_container": "Name =˜ ^webfrnt-?\\d*$",
      "filter": "",
      "action": "ACCEPT"
    },
    {
      "network": "alicewebappnet",
      "src_container": "Name =˜ ^webfrnt-?\\d*$",
      "dst_container": "Name =˜ ^log-?\\d*$",
      "filter": "",
      "action": "ACCEPT"
    },
    ...
```

Disregarding the JSON formatting, it is similar to Figure 13, but only the first two rules are shown. Alice tests that all containers have the necessary connectivity with this setting. Alice also tests that the firewall gets dynamically updated once she instantiates new containers and that the most obvious attempts to subvert the security policy are successfully blocked. She also verifies the generated iptables rules with *fffuu*. This reveals that the overall setting is indeed good, but two open issues exist: First, Internet connectivity is again unconstrained and the stateless semantics are not enforced correctly, i.e., once a connection with the log server is established, bidirectional communication is permitted. Alice leaves these engineering issues of fine tuning the DFWFW configuration to future work.

Ultimately, Alice already enhances the state-of-the-art of Docker container management by combining a dynamic Docker firewall framework with the static policy management tool *topoS*. While the Docker firewall deploys the security policy, *topoS* generates it. This combination lifts *topoS* to dynamic contexts since it allows dynamic spawning and deletion of containers at runtime while still providing strong guarantees about the enforced security requirements. In addition, *fffuu* can verify the correctness of potentially modified firewall rules according to the policy at runtime.

VI. SURVEY OF RELATED ACADEMIC WORK

We first define four management abstraction layers to subsequently classify related work in the field of network management and security, not limited to Docker. Each abstraction layer is responsible for an individual problem domain. We illustrate the four layers in Figure 16. The layers have well-defined interfaces, thus, it is possible to combine solutions of individual problems.

We propose the following four layers of abstraction.

Security Invariants Defines the high-level security goals. Representable as predicates. For example, Figure 3.

Access Control Abstraction Defines the allowed accesses between policy entities. Representable as global access control matrix. For example, Figure 4.
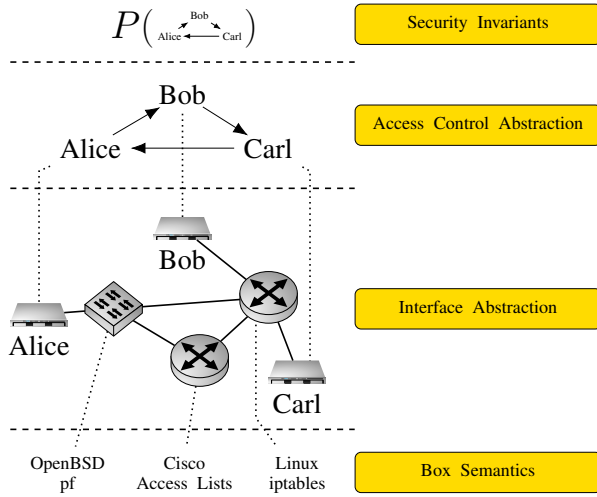
---

[3]For HTTP. Alice disregards ssh.

Fig. 16. Four Layer Abstractions

Fig. 17. Four Layer Abstraction in Related Work

Interface Abstraction Defines a model of the complete network topology. Representable as a graph, packets are forwarded between the network entity's interfaces.

Box Semantics Describes the semantics (i.e., behavior) of individual network boxes. Usually, the semantics are vendor-specific (e.g., iptables, Cisco ACLs, Snort IDS).

The main difference between the interface abstraction and the box semantics is that the latter describes the behavior of only one network entity, whereas the former describes the interconnection of many, possibly different, network boxes. In our story, both coincide since only one Docker host is considered. By separating the box semantics from the interface and access control abstraction, the low level implementation of the enforcement device can be exchanged, if the new devices can provide equal semantics. Only the generation of the target configuration must be adapted. For example, *topoS* can also used to generate an OpenFlow configuration [11].

In Figure 17, we summarize how related work bridges the abstraction layers. With regard to the abstractions, work may be horizontal or vertical: Vertical work bridges abstraction layers, for example, translating security invariants (Figure 3) to the access control abstraction (Figure 4) is a vertical step. Horizontal work adds features or conducts safety checks on the same abstraction level, for example, augmenting the directed policy (Figure 4) to a stateful policy (Figure 6) is a horizontal step. A direct arrow from the access control abstraction to the box semantics (and vice versa) means that the solution only applies to a single enforcement box. Solutions such as Firmato and Fireman achieve more and are thus listed multiple times.

Security Invariants → Access Control Abstraction: Zhao et al. [35] present a policy refinement framework for network services. They state the need to express security requirements in high-level terms and present a high-level logic language to encode them. For this language, an automated translation procedure to the access control abstraction is presented. Also, low-level policies which can be enforced directly by certain security mechanisms are presented (not shown in Fig. 17 since their current prototype implementation only supports one
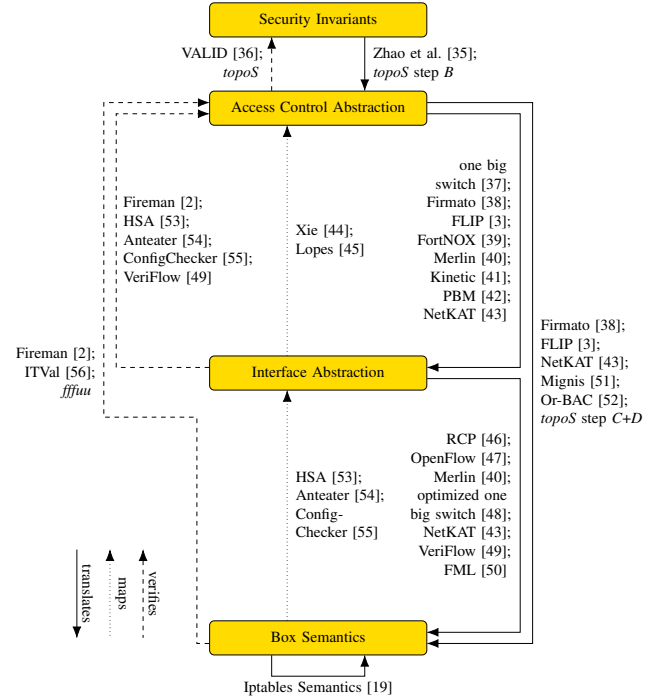
single policy rule). The logic-based, abstract policy language roughly corresponds to first-order logic with set theory, orderings, and relations over an UML class diagram. It allows to express almost unrestricted statements. The downside for a user of such a language may be that it is hard to specify and hard to follow. The system does not provide a reader with feedback about whether the encoded high-level security requirement actually corresponds to the policy author's intention. In contrast, *topoS* comes with a library of pre-specified logic formulas (security invariant templates). A policy author only needs to assign attributes, but is not required to manually encode requirements as formulas. In addition, *topoS* gives immediate feedback to a policy author by visualizing the resulting access control policy as graph. This enables a policy author to verify the requirement specification and perform *"What-if?"* analyses.

Access Control Abstraction --→ Security Invariants: Bleikertz and Groß also state the need to express security requirements in a high-level language and verify policies against them. They propose VALID [36] which is a specification language to express security invariants in cloud infrastructures. VALID has aspects in common with *topoS*, for example, it requires a network's connectivity and information flow structure in the format of a graph as input and allows specifying predicates over it. The language is formally defined in the AVISPA Intermediate Format. It can verify that a given network topology conforms to specified high-level goals, but it cannot translate these goals to a network topology nor give feedback about the meaning of the specified goals.

Access Control Abstraction → Interface Abstraction: From the point of view of our fictional story, Firmato [38] is probably the work most closely related to *topoS*. It defines

an entity relationship model to structure network management and compile firewall rules from it, illustrated in Figure 18. Firmato focuses on roles, which in our model correspond to container names. A role has positive capabilities and is related to other roles, which can be used to derive an access control matrix. Zones, Gateway-Interfaces, and Gateways define the network topology, which corresponds to the interface abstraction. As illustrated in Figure 18, the abstraction layers identified in this work can also be identified in Firmato's model. The Host Groups, Role Groups, and Hosts definitions provide a mapping from policy entities to network entities, which is Firmato's approach to the naming problem. With close correspondence in the underlying concepts to Firmato, Cuppens et al. [57] propose a firewall configuration language based on Or-BAC [58]. Similar to Firmato (with more support for negative capabilities) is FLIP [3], which is a high-level language with focus on *service* management (e.g., allow/deny HTTP). Essentially, both FLIP and Firmato enhance the access control abstraction horizontally by including layer four port management and traverse it vertically by serializing to firewall rules.

FML [50] is a flow-based declarative language to define, among others, access control policies in a DATALOG-like language. Comparably to our directed (stateless) policy, FML operates on unidirectional network flows. FML solves the naming problem by assuming that all entities are authenticated with IEEE 802.1X [59].
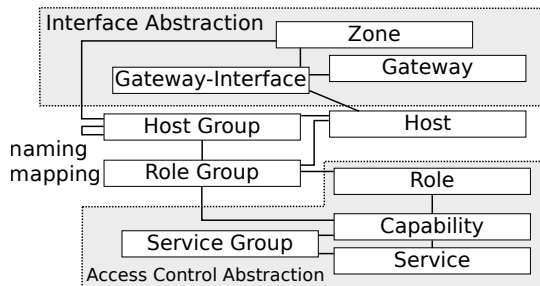

Fig. 18. Firmato ERM

Policy-based management (PBM) [42] was introduced to simplify network administration. Similar to our work, it proposes different levels of abstraction and describes how to translate between them. Policy-based management defines a generic information model [60], [61] which is not limited to access control, however, we focus our discussion solely on access control and security. In a central policy repository, global policy rules are stored. Policy decision points retrieve these rules and interpret them. Using our terminology, this step translates the access control abstraction to the interface abstraction. A policy decision point forwards decisions to policy enforcement points, implementing the translation from the interface abstraction to box semantics. This last step may be very device-specific [62] and is not the core focus of PBM. While PBM was built on the idea of specifying business-level abstractions in terms of requirements [42], the IETF specified a rule-based policy repository [60], [61], which restricts storing high-level requirements that cannot easily be expressed as rules. In strong contrast to *topoS*, which focuses on specifying

higher-level requirements (i.e., security invariants), the IETF Policy Framework working group focused on the specification of lower-level policies [60, §2.1. Policy Scope]. This can also be witnessed in many languages which were developed over the years [63] since, in particular when it comes to security, they usually only provide access control abstractions.

The "One Big Switch" Abstraction [37], [48], [64] allows to manage a network as if it were only one, central big switch. This effectively allows solutions which only support to manage one device to be applied to a complete network, consisting of multiple switches. With regard to Figure 17, any solution which supports translating the access control abstraction to box semantics can also be applied to translate from the access control abstraction to the interface abstraction by translating to the "One Big Switch".

NetKAT [43], [65] is a SDN programming language with well-defined semantics. It features an efficient compiler for local, global, and virtual programs to flow table entries [43]. Among others, it allows implementing the "One Big Switch" Abstraction [43].

Access Control Abstraction → Box Semantics: Mignis [51] proposes a declarative language to manage Netfilter/iptables firewalls. They focus on packet filtering and NAT. Their policy language is restrictive to avoid the usual policy conflicts. In particular, it only allows to write blacklisting-style policies, additionally NAT and filtering cannot arbitrarily be mixed to provide consistent policies.[4] Apart from *fffuu*, it is the only work we are aware of which provides a formal semantics of an iptables firewall. They describe a semantics of the packet filtering and NAT behavior of iptables. Their semantics only models the aspects of iptables which are required for their policy language. While it supports NAT (which the semantics of *fffuu* does not[5]), it does not support user-defined chains or arbitrary match conditions.[6] Though supporting NAT and stateful filtering, advanced iptables features such as `NOTRACK` or `connmark` are not considered.

Craven et al. [66] present a generalized (not network-specific) process to translate access control policies, enhanced with several aspects, to enforceable device-specific policies; the implementation requires a model repository of box semantics and their interplay. Pahl [67] delivers a data-centric, network-specific approach for managing and implementing such a repository, further focusing on things.

Interface Abstraction --→ Access Control Abstraction : As illustrated in Figure 17, Fireman [2] is a counterpart to Firmato. It verifies firewall rules against a global access policy. In addition, Fireman provides verification on the same horizontal layer (i.e., finding shadowed rules or inter-firewall conflicts, which do not affect the resulting end-to-end connectivity but

---

[4]The assumptions for the translation impose many restrictions [51, §5].

[5]Note that Docker utilizes iptables NAT features, but not in the `filter` table, where usually all access control decisions are made. This is the reason that *fffuu* (though it does not support NAT) could be used to analyze Docker firewall filtering.

[6]Mignis supports match conditions but imposes additional assumptions on them to ensure that they are consistent with NAT. For example, the `iprange` module is not allowed. Without inspecting all matching modules manually, there is no generic way to assure whether a filter condition is compatible with Mignis.

are still most likely an implementation error). Abstracting to its uses, one may call rcc [68] the fireman for BGP. *fffuu* is unique as it not only verifies rules, but also translates them back to the access control abstraction.

Header Space Analysis (HSA) [53], Anteater [54], and ConfigChecker [55] verify several horizontal safety properties on the interface abstraction, such as absence of forwarding loops. By analyzing reachability [44], [45], [53]–[55], horizontal consistency of the interface abstraction with an access control matrix can also be verified. Verification of incremental changes to the interface abstraction can be done in real-time with VeriFlow [49] and NetPlumber [69], the former can also prevent installation of violating rules. These models of the interface abstraction have many commonalities: The network boxes in all models are stateless and the network topology is a graph, connecting the entity's interfaces. A function models packet traversal at a network box. These models could be considered as a giant (extended) finite state machine (FSM), where the state of a packet is an (interface×packet) pair and the network topology and forwarding function represent the state transition function [45], [70]. In contrast to arbitrary state machines, it is believed that those derived from networks are comparatively well-behaved [70]. Anteater [54] differs in that interface information is implicit and packet modification is represented by relations over packet histories.

Horizontal Enhancements: Most analysis tools make simplifying assumptions about the underlying network boxes. Diekmann et al. [19] present simplification of iptables firewalls. This makes complex real-world firewalls available for tools which were built with simplifying assumptions about rulesets. This horizontally simplifies the box semantics and *fffuu* is built on top of these simplification procedures [12].

FortNOX [39] horizontally enhances the access control abstraction as it assures that rules by security apps are not overwritten by other apps. Technically, it hooks up at the access control/interface abstraction translation. Kinetic [41], [71] is an SDN language which lifts static policies (as constructed by *topoS*) to dynamic policies. To accomplish this, an administrator can define a simple FSM which dynamically (triggered by network events) switches between static policies. In addition, the FSM can be verified with a model checker.

Features are horizontally added to the interface abstraction: a routing policy allows specifying *paths* of network traffic [48]. Merlin [40], [72] additionally supports bandwidth assignments and network function chaining. Both translate from a global policy to local enforcement and Merlin provides a feature-rich language for interface abstraction policies. Conceptually similar (but with a completely different implementation), RCP [46] allows to logically centralize routing while remaining compatible with existing routers.

VII. COMPARISON TO ACADEMIC STATE-OF-THE-ART

We are not aware of any academic related work about network access control management specifically for docker. To the best of our knowledge, *topoS* and *fffuu* are the only academic works where applicability for a Docker environment has been specifically demonstrated. But Docker is merely an application example, both tools were not specifically designed for it but worked out of the box.[7] We broaden the scope and compare their underlying theory to the general state of the art of tools for helping network management and administration. The comparison is outlined in Figure 19. It is aligned similar to Figure 17, but we omit the Interface Abstraction. The solid single arrows visualize work which is able to translate between the corresponding abstractions. The dashed single arrow represents work which is only capable of verification: Given an access control policy and security invariants as input, their conformance can be verified. But one cannot be derived from the other. In general, a translation from an access control policy to security invariants is not possible without guessing a policy author's intention.
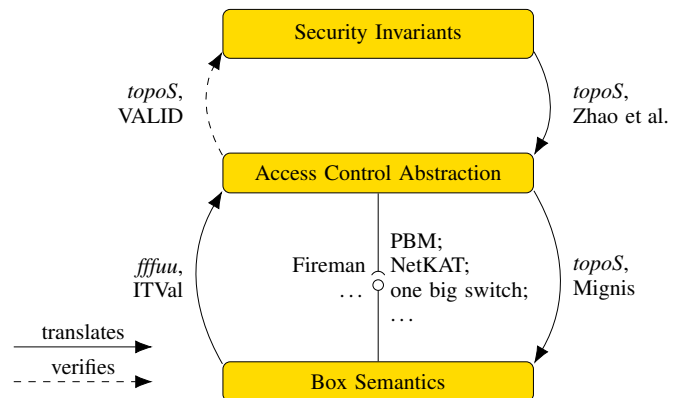

Fig. 19. Overview of Comparison to Related Work (cf. Fig. 17)

Our combination of *topoS* and *fffuu* is the only compatible toolset which is able to bridge all levels of abstractions in both directions out of the box. The fictional story reveals that this back-and-forth is useful in several scenarios. The story also reveals features an administrator may wish for. We now compare *topoS* and *fffuu* with related work specifically for the following use cases, considered in isolation.

Build Networks Based on a Security Requirement Specification: Instead of writing a policy or low-level configuration by hand, the fictional story shows that it is useful to generate working network configurations directly from a scenario-specific security requirement specification. This is useful for the initial design and implementation, as well as for starting over in certain scenarios.

VALID [36] allows to express security requirements, but it cannot derive network configurations from them. Zhao et al. [35] also propose a framework which allows to express security requirements. In contrast to VALID, their framework additionally allows to derive working network configurations. The language proposed by Zhao et al. exposes a lot of formalism to the administrator and almost bears more resemblance to programming than it bears to a specification. Compared to this, *topoS* distinguishes strongly between templates and instantiating a template. While defining new templates also bears resemblance to programming and is only intended for expert users, the common operation to define a specification is

---

[7]The power of formally verified code.

by instantiating those templates, which only requires configurations and exposes very little formalism to the administrator.

Allow Intervention and Low-Level Control for the Administrator: The fictional story revealed several occasions where Alice wanted to fine-tune the low-level policy by hand. One example was the temporary, ad-hoc permission for ssh. Another example was the rate limiting to heise.de. It may also be imaginable that Alice needs to reorder some rules at some point for performance reasons. Both are true low-level operation which should not be achievable on a higher level of abstraction.[8]

The Mignis [51] firewall configuration language allows to specify filtering policies. It gives the administrator the optional possibility to add arbitrary additional low-level iptables match conditions to the high-level rules. These additional match conditions may introduce soundness issues. The language does not permit the administrator to change the generated iptables rules directly, e.g., reordering, restructuring, or ad-hoc changes without recompiling are not allowed. In contrast, *topoS* permits arbitrary changes to the generated iptables rules since *fffuu* can be used to verify correctness of the changes.

Detect Erosion and Drift of the Implemented Policy vs. the Specified Policy: The terms *erosion* and *drift* are usually used for software architectures [73]. However, our fictional story shows that network security policies and the corresponding configurations also decay, become unmaintainable, and violations of the original requirements sneak in. In addition, being able to detect differences between a configuration and a specified policy is an important step towards understanding legacy configurations or verifying manual low-level changes, as discussed in the previous paragraph.

To analyze the current policy enforced by an iptables ruleset, ITVal [56], [74] can be used. Similar to *fffuu*, it allows to partition the complete IPv4 address range into equivalence classes. ITVal computes one set of equivalence classes jointly for all layer 4 ports. In contrast, *fffuu* can only compute them for one selected port. It depends on the scenario which of the two approaches is more suitable: ITVal's overview is very helpful for a first, quick, overview of a firewall. *fffuu*'s service-specific overview provides better granularity, once one knows which ports one is interested in. ITVal only supports IPv4 while *fffuu* supports IPv4 and IPv6. In addition, ITVal is known to have bugs [12] while *fffuu* is formally proven correct [15]. Ironically, ITVal segfaults for some Docker rulesets of this article while *fffuu* processes them without complaint.

Distributed Enforcement?: Our work only focuses on one single, central enforcement device. However, having only one central firewall or only one central Docker host is not a satisfying scenario. This raises the question whether our work is useless for large installations or whether distributed enforcement is an orthogonal issue. The Interface Abstraction—discussed in Section VI and omitted in Figure 19—corresponds to distributed enforcement. Figure 17 lists several related work which takes one centralized policy and enforces it in a distributed fashion. Therefore, distributed enforcement is an orthogonal issue and our tools can help to develop, verify,

and maintain the centralized policy which is then enforced in a distributed fashion. For example, policy-based management [42] systems or the NetKAT compiler [43] for SDN could be used. We explicitly visualize an interface boundary —⊂○— in Figure 19 to highlight that *topoS* produces an access control matrix, which is understood by many technologies for distributed enforcement. Therefore, *topoS* can be used as a module for access control within another system. In addition, algorithms for distributed firewall analysis (as supported by Fireman [2]) can also benefit from the pre-processing and simplification provided by *fffuu*.

VIII. CONCLUSION

We presented our tools *topoS* and *fffuu*, demonstrating design, management, and operations of network-level access control. In a fictional story about an operator in a Docker-based environment, we showed how this toolset helps both for the design of a setup and for daily operations. We demonstrated several situations in which our tools provide useful feedback, uncover bugs, and even help to migrate setups. The duality of *topoS* and *fffuu* in combination with their common policy abstraction makes them a powerful combination and enhances the academic state-of-the-art.

The underlying theory of both tools is formally verified and their code is directly generated by Isabelle/HOL, providing strong correctness guarantees about their results. Having proven the correctness of the tools, no theorem prover is required at runtime; the code is correct for all inputs. Our tools are the first, jointly designed, formally machine-verified, open source, real-world-approved tools which bridge the gaps between high-level security requirements and low-level firewall behavior in both directions. Notably, they are not limited to Docker environments, but also applicable to different scenarios. This becomes explicit since they were never designed for Docker specifically, but worked flawlessly for the Docker scenarios. We surveyed related work, showed how our tools enhance the state-of-the-art, and how different tool may interact given common abstractions.

Availability: *topoS* and *fffuu* can be obtained at

> https://github.com/diekmann/topoS
> https://github.com/diekmann/Iptables_Semantics

A stable version of the theory files can also be obtained from the Archive of Formal Proofs (AFP) [14], [15], [75]–[77]. AFP maintenance policy ensures that our formalization will keep working with future Isabelle releases. The raw data of the Docker story and many other iptables dumps can be found at

> https://github.com/diekmann/net-network

To the best of our knowledge, this is the largest, publicly-available collection of real-world iptables firewall rulesets.

---

[8]tautologically, higher levels of abstraction abstract over low-level details.

REFERENCES

[1] F. Mansmann, T. Göbel, and W. Cheswick, "Visual Analysis of Complex Firewall Configurations," in *9th International Symposium on Visualization for Cyber Security*, ser. VizSec '12. ACM, Oct. 2012, pp. 1–8.

[2] L. Yuan, H. Chen, J. Mai, C.-N. Chuah, Z. Su, and P. Mohapatra, "FIREMAN: A Toolkit for FIREwall Modeling and ANalysis," in *IEEE Symposium on Security and Privacy*, May 2006, pp. 199–213.

[3] B. Zhang, E. Al-Shaer, R. Jagadeesan, J. Riely, and C. Pitcher, "Specifications of a High-level Conflict-free Firewall Policy Language for Multi-domain Networks," in *12th ACM symposium on Access control models and technologies*, ser. SACMAT'07. ACM, Jun. 2007, pp. 185–194.

[4] A. Wool, "A Quantitative Study of Firewall Configuration Errors," *IEEE Computer*, vol. 37, no. 6, pp. 62–67, Jun. 2004.

[5] H. Hamed and E. Al-Shaer, "Taxonomy of Conflicts in Network Security Policies," *IEEE Communications Magazine*, vol. 44, no. 3, pp. 134–141, Mar. 2006.

[6] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making Middleboxes Someone Elses Problem: Network Processing as a Cloud Service," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 13–24, Oct. 2012.

[7] C. Diekmann, S.-A. Posselt, H. Niedermayer, H. Kinkelin, O. Hanka, and G. Carle, "Verifying Security Policies using Host Attributes," in *Formal Techniques for Distributed Objects, Components, and Systems: 34th IFIP WG 6.1 International Conference, FORTE*. Berlin, Germany: Springer Berlin Heidelberg, Jun. 2014, pp. 133–148.

[8] Docker, Inc., "docker," 2016, using Docker version 1.12.1 on Ubuntu 16.04, Accessed on Aug. 18, 2016. [Online]. Available: https://www.docker.com/

[9] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, *Site Reliability Engineering – How Google Runs Production Systems*. O'Reilly Media, Mar. 2016, ISBN 978-1-4919-2909-4.

[10] A. Mouat, *Docker Security – Using Containers Safely in Production*, 1st ed. O'Reilly, 2015, Accessed on Jul. 17, 2017. [Online]. Available: https://www.openshift.com/promotions/docker-security.html

[11] C. Diekmann, A. Korsten, and G. Carle, "Demonstrating topoS: Theorem-prover-based synthesis of secure network configurations," in *11th International Conference on Network and Service Management (CNSM)*, Barcelona, Spain, Nov. 2015, pp. 366–371.

[12] C. Diekmann, J. Michaelis, M. Haslbeck, and G. Carle, "Verified iptables Firewall Analysis," in *IFIP Networking 2016*, Vienna, Austria, May 2016.

[13] "boycott docker," website, Jun. 2016, Accessed on Jun. 20, 2016. [Online]. Available: https://web.archive.org/web/20160620101452/boycottdocker.org/

[14] C. Diekmann, "Network Security Policy Verification," *Archive of Formal Proofs*, Jul. 2016, formal proof development. [Online]. Available: https://www.isa-afp.org/entries/Network_Security_Policy_Verification.shtml

[15] C. Diekmann and L. Hupel, "Iptables Semantics," *Archive of Formal Proofs*, Sep. 2016, formal proof development. [Online]. Available: http://isa-afp.org/entries/Iptables_Semantics.shtml

[16] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, ser. LNCS. Springer, 2002, last updated 2016, vol. 2283. [Online]. Available: http://isabelle.in.tum.de/

[17] C. Diekmann, L. Hupel, and G. Carle, "Directed Security Policies: A Stateful Network Implementation," in *Engineering Safety and Security Systems (ESSS)*, ser. Electronic Proceedings in Theoretical Computer Science, vol. 150. Singapore: Open Publishing Association, May 2014, pp. 20–34.

[18] The Coq development team, *The Coq proof assistant reference manual*, LogiCal Project, 2004, last updated 2016. [Online]. Available: http://coq.inria.fr/

[19] C. Diekmann, L. Hupel, and G. Carle, "Semantics-Preserving Simplification of Real-World Firewall Rule Sets," in *Formal Methods*, Jun. 2015.

[20] A. Okmianski, "Transmission of Syslog Messages over UDP," RFC 5426 (Proposed Standard), Internet Engineering Task Force, Mar. 2009.

[21] Docker Core Engineering, "Docker 1.10: New Compose File, Improved Security, Networking And Much More!" blog, Feb. 2016, Accessed on Dec. 4, 2016. [Online]. Available: https://blog.docker.com/2016/02/docker-1-10/

[22] C. Diekmann, "Issue #29108 – Networking/Security: Custom net with both --internal and --icc=false does not block icc," github, Dec. 2016, Accessed on Dec. 5, 2016. [Online]. Available: https://github.com/docker/docker/issues/29108

[23] J. Radhakrishnan, "Docker Networking Design Philosophy," blog, Mar. 2016, Accessed on Dec. 4, 2016. [Online]. Available: https://blog.docker.com/2016/03/docker-networking-design-philosophy/

[24] B. Meyer, "Issue #22054 – Docker Network bypasses Firewall, no option to disable," github issue and discussion, Apr. 2016, Accessed on Dec. 4, 2016. [Online]. Available: https://github.com/docker/docker/issues/22054

[25] Docker Inc., "Docker security," Accessed on Jul. 17, 2017. [Online]. Available: https://docs.docker.com/engine/security/security/

[26] J. Hertz, "Abusing Privileged and Unprivileged Linux Containers," NCC Group, Jun. 2016. [Online]. Available: https://www.nccgroup.trust/us/our-research/abusing-privileged-and-unprivileged-linux-containers/

[27] T. Bui, "Analysis of Docker Security," *CoRR*, vol. abs/1501.02967, Jan. 2015. [Online]. Available: http://arxiv.org/abs/1501.02967

[28] J. Frazelle (@jessfraz), "Naked Container: A container run without the default AppArmor, default Seccomp, or SELinux. Naked Containers make me sad." Tweet, Apr. 2017, Accessed on Apr. 17, 2017. [Online]. Available: https://twitter.com/jessfraz/status/853952267594207232

[29] A. Goldfuss, "Passing the Console: Fostering the Next Generation of Ops Professionals," in *LISA*. Boston, MA: USENIX Association, 2016.

[30] V. Petersson, "The dangers of UFW + Docker," blog, Nov. 2014, Accessed on Nov. 3, 2016. [Online]. Available: http://blog.viktorpetersson.com/post/101707677489/the-dangers-of-ufw-docker

[31] J. Engelhardt, "Towards the perfect ruleset," May 2011, Accessed on Jul. 17, 2017. [Online]. Available: http://inai.de/documents/Perfect_Ruleset.pdf

[32] I. Duffy, "Azure bug bounty Pwning Red Hat Enterprise Linux," blog, Nov. 2016, Accessed on Nov. 30, 2016. [Online]. Available: http://ianduffy.ie/blog/2016/11/26/azure-bug-bounty-pwning-red-hat-enterprise-linux/

[33] gdm85, "Docker-fw," github, Mar. 2016, commit d335396. [Online]. Available: https://github.com/gdm85/docker-fw

[34] I. Rad, "DFWFW – Docker Firewall Framework," github, Aug. 2016, commit 94d1036. [Online]. Available: https://github.com/irsl/dfwfw

[35] H. Zhao, J. Lobo, A. Roy, and S. M. Bellovin, "Policy Refinement of Network Services for MANETs," in *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011)*, Dublin, Ireland, May 2011.

[36] S. Bleikertz and T. Groß, "A Virtualization Assurance Language for Isolation and Deployment," in *International Symposium on Policies for Distributed Systems and Networks (POLICY)*. IEEE, Jun. 2011, pp. 33–40.

[37] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing Software Defined Networks," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, ser. NSDI'13. Lombard, IL: USENIX Association, Apr. 2013, pp. 1–13.

[38] Y. Bartal, A. Mayer, K. Nissim, and A. Wool, "Firmato: A Novel Firewall Management Toolkit," in *IEEE Symposium on Security and Privacy*. IEEE, May 1999, pp. 17–31.

[39] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A Security Enforcement Kernel for OpenFlow Networks," in *First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. Helsinki, Finland: ACM, Aug. 2012, pp. 121–126.

[40] R. Soulé, S. Basu, P. J. Marandi, F. Pedone, R. Kleinberg, E. G. Sirer, and N. Foster, "Merlin: A Language for Provisioning Network Resources," *CoRR*, vol. abs/1407.1199, 2014. [Online]. Available: http://arxiv.org/abs/1407.1199

[41] H. Kim, J. Reich, A. Gupta, M. Shahbaz, N. Feamster, and R. Clark, "Kinetic: Verifiable Dynamic Network Control," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, ser. NSDI'15. Oakland, CA: USENIX Association, May 2015, pp. 59–72.

[42] D. C. Verma, "Simplifying Network Administration Using Policy-Based Management," *IEEE Network*, vol. 16, no. 2, pp. 20–26, Mar. 2002.

[43] S. Smolka, S. A. Eliopoulos, N. Foster, and A. Guha, "A Fast Compiler for NetKAT," in *International Conference on Functional Programming (ICFP)*. ACM, Sep. 2015, pp. 328–341.

[44] G. G. Xie, J. Zhan, D. A. Maltz, H. Zhang, A. G. Greenberg, G. Hjálmtýsson, and J. Rexford, "On Static Reachability Analysis of IP Networks," in *24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, vol. 3. IEEE, Mar. 2005, pp. 2170–2183.

[45] N. P. Lopes, N. Bjørner, P. Godefroid, and G. Varghese, "Network Verification in the Light of Program Verification," Tech. Rep., Sep. 2013. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=201589

[46] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and Implementation of a Routing Control Platform," in *2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, ser. NSDI'05. Boston, MA: USENIX Association, May 2005, pp. 15–28.

[47] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, Apr. 2008.

[48] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the "One Big Switch" Abstraction in Software-defined Networks," in *9th ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT. ACM, Dec. 2013, pp. 13–24.

[49] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey, "VeriFlow: Verifying Network-Wide Invariants in Real Time," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, ser. NSDI'13. Lombard, IL: USENIX Association, Apr. 2013, pp. 15–27.

[50] T. L. Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, and S. Shenker, "Practical Declarative Network Management," in *1st ACM workshop on Research on enterprise networking*, ser. WREN'09. ACM, Aug. 2009, pp. 1–10.

[51] P. Adão, C. Bozzato, G. D. Rossi, R. Focardi, and F. L. Luccio, "Mignis: A Semantic Based Tool for Firewall Configuration," in *27th Computer Security Foundations Symposium*, ser. CSF. IEEE, Jul. 2014, pp. 351–365.

[52] F. Cuppens, N. Cuppens-Boulahia, T. Sans, and A. Miège, "A Formal Approach to Specify and Deploy a Network Security Policy," in *Formal Aspects of Security and Trust (FAST)*. Springer US, Aug. 2004, pp. 203–218.

[53] P. Kazemian, G. Varghese, and N. McKeown, "Header Space Analysis: Static Checking for Networks," in *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, ser. NSDI'12. San Jose, CA: USENIX Association, Apr. 2012, pp. 113–126.

[54] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P. B. Godfrey, and S. T. King, "Debugging the Data Plane with Anteater," in *ACM SIGCOMM*, Toronto, Ontario, Canada, Aug. 2011, pp. 290–301.

[55] E. Al-Shaer, W. Marrero, A. El-Atawy, and K. Elbadawi, "Network Configuration in A Box: Towards End-to-End Verification of Network Reachability and Security," in *IEEE International Conference on Network Protocols (ICNP)*, Oct. 2009, pp. 123–132.

[56] R. M. Marmorstein, P. Kearns *et al.*, "Firewall Analysis with Policy-based Host Classification," in *20th USENIX Large Installation System Administration Conference (LISA)*, vol. 6. Washington, D.C.: USENIX Association, Dec. 2006.

[57] F. Cuppens, N. Cuppens-Boulahia, T. Sans, and A. Miège, *Formal Aspects in Security and Trust: IFIP TC1 WG1.7 Workshop on Formal Aspects in Security and Trust (FAST), World Computer Congress*. Boston, MA: Springer US, 2005, ch. A Formal Approach to Specify and Deploy a Network Security Policy, pp. 203–218.

[58] A. Abou El Kalam, R. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miege, C. Saurel, and G. Trouessin, "Organization Based Access Control," in *IEEE 4th International Workshop on Policies for Distributed Systems and Networks (POLICY)*, Jun. 2003, pp. 120–131.

[59] *802.1X – Port based Network Access Control*, IEEE Std. 802.1X-2001, Jun. 2001.

[60] B. Moore, E. Ellesson, J. Strassner, and A. Westerinen, "Policy Core Information Model – Version 1 Specification," RFC 3060 (Proposed Standard), Internet Engineering Task Force, Feb. 2001, updated by RFC 3460.

[61] B. Moore, "Policy Core Information Model (PCIM) Extensions," RFC 3460 (Proposed Standard), Internet Engineering Task Force, Jan. 2003.

[62] S. Hinrichs, "Policy-Based Management: Bridging the Gap," in *15th Computer Security Applications Conference (ACSAC)*. IEEE, Dec. 1999, pp. 209–218.

[63] W. Han and C. Lei, "A survey on policy languages in network and security management," *Computer Networks*, vol. 56, no. 1, pp. 477–489, Jan. 2012.

[64] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, "Modular SDN Programming with Pyretic," *;login: The USENIX Magazine*, vol. 38, no. 5, pp. 40–47, Oct. 2013.

[65] C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker, "NetKAT: Semantic Foundations for Networks," in *41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '14. San Diego, California: ACM, Jan. 2014, pp. 113–126.

[66] R. Craven, J. Lobo, E. Lupu, A. Russo, and M. Sloman, "Policy Refinement: Decomposition and Operationalization for Dynamic Domains," in *7th Conference on Network and Service Management (CNSM)*, Oct. 2011, pp. 1–9.

[67] M.-O. Pahl, "Data-Centric Service-Oriented Management of Things," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, Ottawa, Canada, May 2015.

[68] N. Feamster and H. Balakrishnan, "Detecting BGP Configuration Faults with Static Analysis," in *2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, ser. NSDI'05. Boston, MA: USENIX Association, May 2005, pp. 43–56.

[69] P. Kazemian, M. Chan, H. Zeng, G. Varghese, N. McKeown, and S. Whyte, "Real Time Network Policy Checking Using Header Space Analysis," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, ser. NSDI'13. Lombard, IL: USENIX Association, Apr. 2013, pp. 99–111.

[70] S. Zhang, S. Malik, and R. McGeer, "Verification of Computer Switching Networks: An Overview," in *Automated Technology for Verification and Analysis*, ser. LNCS. Springer, 2012, pp. 1–16.

[71] H. Kim, J. Reich, A. Gupta, M. Shahbaz, N. Feamster, and R. Clark, "Kinetic: Verifiable Dynamic Control," 2014, Accessed on Sep. 8, 2016. [Online]. Available: http://resonance.noise.gatech.edu/

[72] R. Soulé, S. Basu, P. J. Marandi, F. Pedone, R. Kleinberg, E. G. Sirer, and N. Foster, "Merlin: A Language for Provisioning Network Resources," in *10th ACM International on Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT. Sydney, Australia: ACM, Dec. 2014, pp. 213–226.

[73] D. E. Perry and A. L. Wolf, "Foundations for the Study of Software Architecture," *SIGSOFT Software Engineering Notes*, vol. 17, no. 4, pp. 40–52, Oct. 1992.

[74] R. M. Marmorstein and P. Kearns, "A Tool for Automated iptables Firewall Analysis," in *USENIX Annual Technical Conference, FREENIX Track*. USENIX Association, Apr. 2005, pp. 71–81.

[75] C. Diekmann, J. Michaelis, and L. Hupel, "IP Addresses," *Archive of Formal Proofs*, Jun. 2016, formal proof development. [Online]. Available: http://isa-afp.org/entries/IP_Addresses.shtml

[76] C. Diekmann, J. Michaelis, and M. Haslbeck, "Simple Firewall," *Archive of Formal Proofs*, Aug. 2016, formal proof development. [Online]. Available: http://isa-afp.org/entries/Simple_Firewall.shtml

[77] J. Michaelis and C. Diekmann, "Routing," *Archive of Formal Proofs*, Aug. 2016, formal proof development. [Online]. Available: http://isa-afp.org/entries/Routing.shtml