This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2020.2974727, IEEE Transactions on Dependable and Secure Computing

Verifiable Policy-Defined Networking using Metagraphs

Dinesha Ranathunga, *Member, IEEE,* Matthew Roughan, *Fellow, IEEE,* and Hung Nguyen, *Member, IEEE*

***Abstract*—Reliable network-policy specification requires abstractions that can naturally model policies together with rigorous formal foundations to reason about these policies. Current specifications satisfy one of these requirements or the other, but not both. A Metagraph is a generalized graph theoretic structure that overcomes this limitation. They are a natural way of expressing high-level end-to-end network policies. The rich formal foundations provided by metagraph algebra help analyze important network-policy properties such as reachability, redundancy and consistency. These features make metagraphs a clear choice for modeling and reasoning about policies in Formally-Verifiable Policy-Defined Networking (FV-PDN): a network-programming paradigm which has verifiability built-in. In this paper, we demonstrate the use of metagraphs in policy specification by modeling and analyzing real policies from a large university network. We show their benefit in FV-PDN by developing a prototype solution which automatically refines metagraph-based high-level policies to device configurations and deploys them to a SDN-based emulated network.**

***Index Terms*—Network autoconfiguration, Provable network security, Policy metagraph, Policy defined networking**

I. Introduction

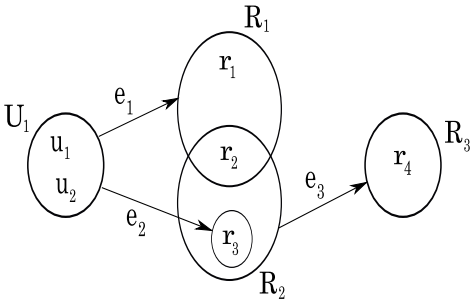
A graph is a central concept in the design of networks and many information-processing systems such as decision-support systems. It is also an important concept in designing communication-network polices. In fact, network admins often design policies by drawing graph diagrams on whiteboards. However, simple graphs commonly associate individual information elements and not sets of elements. Sets of elements (*i.e.,* Endpoint Groups (EPGs)) such as users and IP addresses are often the basis for defining real network-policies [37].

A *metagraph* is a generalized graph theoretic structure that overcomes the limitations of simple graphs. A metagraph is a directed graph between a collection of sets of 'atomic' elements [6]. Each set is a node in the graph and each directed edge represents the relationship between the sets. Figure 1(a) shows an example where a set of users (U_1) are related to sets of network resources (R_1, R_2, R_3) by the edges e_1, e_2 and e_3 describing which user u_i is allowed to access resource r_j.

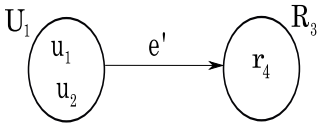
The mathematical operations defined on a metagraph go beyond standard graph operators and help analyze properties such as reachability and consistency. For example, an operation unique to metagraphs is a projection which is a simplified metagraph that highlights particular aspects of the original [6].

In a complex metagraph with many edges, a projection helps visualize the important aspects. For instance, consider the projection over the subset of elements $X = \{u_1, u_2, r_4\}$ of the metagraph in Figure 1(a) shown in Figure 1(b). It consists of one edge describing reachability between this subset of elements. No elements outside of X appear in the projection.

Of the many related works in Policy Defined Networking (PDN) [4], [5], [49], [53], most (*e.g.,* NetKAT [4] Firmato [5])



(a) Metagraph consisting of four sets and three edges.



(b) Projection of (a).

Fig. 1: *A Metagraph example and its projection over the subset of elements $X = \{u_1, u_2, r_4\}$. The projected edge e' represents reachability between the subset of elements.*

operate at the network level; i.e., they intertwine policy with network implementation details. Their users hence, require to specify minute details (*e.g.,* device IP or MAC addresses) with policy which often leads to error-prone policy specification.

We overcome the problem by using metagraphs to decouple policy from the underlying physical infrastructure. This separation of policy from network minutiae (a) helps express high-level, end-to-end network policies without concerning about the topology or other implementation-specific intricacies; and (b) keeps the complexity of the resultant policies low, relative to the base network-device configurations.

Of the many tools available for policy specification [5], [22], [24], [49], most lack mathematical rigor and the ability to prove policy properties. This lack of verifiability (a) hinders the detection of policy inconsistencies such as conflicts, leading to vendor-dependent policy behavior; and (b) prevents providing assurance that a specified policy delivers the expected outcome pre- and post-deployment. Our work fills these gaps.

Our solution uses metagraphs and PDN to enable high-level policies (*e.g.,* using concepts like user/resource groups), and we model these policies using formal constructs to enable a rigorous verification framework. The formalisms allow us to reason about the policies - *e.g.,* is the policy consistent? Once the policies are provably safe, they can be mapped to OpenFlow or traditional Cisco/Juniper/Huawei switches alike using our system's built in policy refinement capabilities.

We demonstrate our system's use in specifying and checking policies by extracting and modeling real network policies from a large university IT network. The network consists

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2020.2974727, IEEE Transactions on Dependable and Secure Computing

of over 30,000 users (students and staff), 43 logical zones ranging in complexity from several IP addresses to over 450 noncontiguous subnets (over 200K IP addresses) and hundreds of switches and routers. The network also encompasses five campuses and nine remote sites and includes many interconnections, for instance, to hospital networks and data center networks. Our study covers a diverse policy set including access-control, QoS, intrusion-detection and reporting policies.

Our case study finds policy inconsistencies (*e.g.,* conflicts), that current network-management tools left undetected. We raised these findings with the university's network operations team, to verify and fix problems. In addition, we resolve these inconsistencies automatically and deploy the validated policies to a SDN-based emulated network using Mininet. Automated, pathological-traffic based tests conducted in this emulated network provide university network administrators assurance of expected policy behavior prior to deployment.

The contributions of our paper are as follows: we first demonstrate how metagraphs are a useful graphical tool for specifying high-level network policies. Secondly, we use metagraph algebras; a matrix algebra defined over metagraph elements and edges, to check policy consistency. Using these algebras, we detect conflicts and redundancies across multiple policy domains such as QoS and intrusion detection. In doing so, we show how these algebras enable a generalized framework for detecting policy inconsistencies. Finally we apply our framework to a large university network consisting of a rich policy set to demonstrate how useful this tool metagraphs is in describing and verifying network policies.

II. Background and Related Work

> "Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?"
>
> *Brian Kernighan [23]*

Past network configuration studies have revealed policy inconsistencies such as conflicts to be a common occurrence [16], [27], [42], [54]. The introductory quote is part of the reason. Current network-policy specifications have evolved by adding features and complexity. But in making configuration cleverer, they have made debugging more difficult.

Inability to debug policy conflicts, leads to unintended consequences. For instance, in a network-security context, such inconsistencies can create an illusion of security and lead to network admins leaving unpatched systems behind their firewalls, confident in the blanket of protection provided. Consequences will only be revealed at runtime (*e.g.,* users unexpectedly lose connectivity or security holes are exploited).

Policy-based network management (PBNM) [7], [8], [48], [51], [53] is a well-known approach that allows top-down configuration where device configurations are derived from high-level policies. Policies are intended to capture how a user wants the network to behave. Through a number of policy refinement steps, these policies are translated (automatically or semi-automatically) to device configurations [7], [53]. There have been more then two decades of research in PBNM with diverse proposals on specification languages [1], [12], [28], [34], [47], conflicts resolution [29], refinement [7], [48] and intents for PBNM. A comprehensive resource for general ideas and solutions for PBNM can be found in [51].

These ideas are recently reinvigorated with the advance of the software defined networking paradigm that uses software to instantaneously and automatically change device behaviors depending on the current status of the network - providing tightly integrated automation and fine-grained control. Research in PDN within an SDN context addresses the questions of policy languages [4], [15], [43], conflicts resolution and refinement [22], [24], [37], [46] and covers issues unique to large scale SDN networks such as policy authoring [31], adaptive policy attack mitigation [44] and hierarchical policies [14].

Lupu et al.'s classification of conflicts into modality conflicts and application-specific conflicts [29] resonates close to our work. Modality conflicts arise when two or more policies which refer to the same subjects have opposite modalities (*e.g.,* access control actions). Application specific conflicts refer to the consistency of the policy contents and external criteria, *e.g.,* the same manager cannot authorize payments and sign the payment cheques. Lupu models these policies using predicates and translates them into assertions in Prolog to detect conflicts. In contrast, we detect modality conflicts using a rigorous formal framework; we use metagraphs to model network policies and leverage metagraph algebras to precisely check properties such as consistency.

The SDN data-plane verification approach proposed in [22] works offline and has been applied to operational networks and multiple real-world bugs were uncovered. Our method offers more flexibility; it can verify device configurations from SDN and traditional networks alike. This flexibility allows to detect device misconfiguration in, for instance, critical infrastructure networks which typically employ legacy equipment which cannot be easily upgraded due to high availability demands.

Related to the work on formal models for network configuration is the work on routing algebra, especially metarouting [17], [18] that allows a high-level declarative language to specify routing protocols so that implementations can be generated automatically. Both metarouting and our metagraph policy model address the generation of network device configurations from high-level policies but in different domains.

Our work complements the existing efforts in PBNM and SDN by providing a novel formal model that can be used to make sure that (a) policy to be formally verified as error-free prior to deployment; and (b) assurance that the expected policy outcome pre- and post-deployment are consistent.

The rich formal foundations readily supported by metagraphs combined with their ability to describe high-level policies make them a clear choice for modeling and reasoning about policies in FV-PDN. The formal structure of a metagraph can be defined as follows:

Definition 1 (Metagraph)**.** *A metagraph* $S{=}\langle X, E\rangle$ *is a graphical construct specified by a generating set* X *and an edge set* E *defined on a set of subsets of* X*. A generating set is a set of variables* $X = \{x_1, x_2, ..., x_n\}$ *and an edge* $e \in E$ *is a pair* $e{=}\langle V_e, W_e\rangle$ *such that* $V_e \subset X$ *is the invertex and* $W_e \subset X$ *is the outvertex.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2020.2974727, IEEE Transactions on Dependable and Secure Computing

TABLE I: *Comparison table of the different network policy analysis approaches. In the supported features,* slicing *refers to the ability to partition a network into independently programmable segments to ensure traffic isolation between them [19].* Ordering *is the ability to compose policies sequentially,* Inconsistencies *refer to policy conflicts and redundancies (FDD- Firewall Decision Diagram, DFD- Diverse Firewall Design, FPA- Firewall Policy Advisor).*

Approach	Underlying abstraction	Analysis method	Supported features						
			Verify consistency	Verify reachability	Verify traffic isolation	Pre-deployment verification	Easy policy visualization	SDN support	Traditional networks
FPA [2]	tree	heuristic conflict detection	✓	✗	✗	✗	✗	✗	✓
DFD [27]	FDD	heuristic policy composition	✓	✗	✗	✗	✗	✗	✓
VeriFlow [24]	graph	heuristic policy verification	✓	✓	✓	✗	✓	✓	✗
Header Space [22]	graph	heuristic policy verification	✓	✓	✓	✗	✓	✗	✓
NetIDE [46]	graph	heuristic policy composition	✓	✗	✗	✗	✗	✓	✓
Frenetic [15]	one big switch	parallel composition operator	✓	✗	✓	✗	✗	✓	✗
Pyretic [43]	one big switch	sequential and parallel composition operators	✓	✗	✓	✗	✗	✓	✗
NetKAT [4]	one big switch	KAT algebras, sequential and parallel composition operators	✓	✓	✓	✗	✗	✓	✗
PGA [37]	graph	heuristic policy composition	✓	✓	✓	✗	✓	✓	✗
PDN	**metagraph**	**metagraph algebras**	✓	✓	✓	✓	✓	✓	✓

This definition is similar to that of a directed hypergraph, but in addition metagraphs have several useful operators and properties. One in particular is the notion of a *metapath* [6] which describes connectivity between sets of elements in a metagraph, but is somewhat different from a path in a graph.

Definition 2 (Metapath)**.** *A metapath from source $B \subset X$ to target $C \subset X$ in a metagraph $S{=}\langle X, E\rangle$ is an edge set E' s.t. every $e' \in E'$ is on a path from an element in B to an element in C. In addition $[\bigcup_{e'} V_{e'} \setminus \bigcup_{e'} W_{e'}] \subseteq B$ and $C \subseteq \bigcup_{e'} W_{e'}$.*

Reachability between a source and a target metagraph node is described by valid metapaths between the two [6] (*e.g.,* the metapath from $\{u_1, u_2\}$ to $\{r_4\}$ in Figure 1(b) is $\{e_1, e_2, e_3\}$).

Metagraphs have a property called dominance which allows determination of redundant components (edges or elements) [6]. A metapath is *input-dominant* if no proper subset of its source connects to the target; *edge-dominant* if no proper subset of its edges is also a metapath from the source to the target; and *dominant* if it is both input- and edge-dominant [6]. Non-dominant metapaths indicate redundancies in a metagraph and hence, redundancies in the policies depicted by the metagraph.

In metagraph theory, the notion of cutsets and bridges allow one to locate edges that are critical [6]. A *cutset* is a set of edges which if removed, eliminates all metapaths between a given source and a target. A singleton cutset is a *bridge*. In an access-control policy context for instance, bridges and cutsets indicate if there exists a critical policy or a policy set that enable access between certain users and resources.

Table 1 summarizes several popular network-policy specification and analysis approaches. We concentrate here on the recent efforts in SDN and refer the traditional PBNM approaches to [51]. SDN languages such as Pyretic [43] and NetKAT [4] enable modular specification of network-wide policies. NetKAT additionally supports formal foundations to reason about these policies rigorously. However, the underlying "one big switch" abstraction does not yield a natural policy representation. This shortfall hinders ease of policy visualization, particularly when policies become complex (which is typical in a large distributed network). Being SDN specific, these languages also cannot help manage traditional networks.

In contrast, our approach using metagraphs, delivers a natural policy representation that users can easily visualize. When policies become complex, a metagraph projection offers a precise, simplified view of the important policy aspects. Metagraph algebras also provide formalisms to detect potentially conflicting sets of policies. The users only need to apply domain specific knowledge (*e.g.,* criteria for an access-control policy conflict) to these subsets to detect actual conflicts.

Table 1 also shows how related works (e.g., PGA [37], VeriFlow [24], Header Space Analysis [22]) also use graph based abstractions for specifying and analyzing network policies. PGA for instance, employs graphs as building blocks to specify access-control and service-chain policies [37]. The approach defines a new type of policy composition to capture the joint intent of two policies. However, the set of validations in PGA are limited to the type of checks that are implemented on the graph. For instance, in PGA is it difficult to determine reachability between two sets of endpoint elements when those elements are distributed across multiple EPGs.

A metagraph's metapath operation trivializes this computation. Our use of a generalized policy-graph model also encapsulates such previous models. For instance, like PGA, our system can create modular, abstract policies and compose them, but in addition, metagraph algebras allow to formally reason about policies- *e.g.,* is the policy consistent? can endpoint group A communicate with group B using HTTP?

VeriFlow [24] and Header Space Analysis [22] uses graph-based abstractions to model information about network topology and forwarding policies. These systems allow to check for network variants violations (*e.g.,* no forwarding loops). But, due to the naturally overlapping nature of policy EPGs, a graph-based representation requires tracking EPG overlaps to conduct such checks accurately. Metagraphs remove this burden by handling EPG overlaps automatically.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2020.2974727, IEEE Transactions on Dependable and Secure Computing

Related works have also investigated the detection of conflicts in firewall access-control policies in traditional networks (*e.g.,* FPA [3] and DFD [27]). FPA represents access-control rules using a single-rooted tree structure and defines algorithms over this policy tree to identify intra- and inter-device conflicts. The method yields a firewall configuration tool that allows network admins to create and edit firewall policies correctly. We go beyond simple firewall access-control policies and detect conflicts in more complex anti-malware and URL-filtering policies as well as QoS and reporting policies.

Despite the use of abstractions for programming network policies, many related works [4], [5], [49] still tightly couple policy to network-minutiae such as device IP or MAC addresses. The resulting policies are as complex as the base network-device configurations. Our use of metagraphs allows us to describe high-level policies decoupled from the underlying physical infrastructure. The approach reduces the burden on network admins in crafting and implementing policies.

In our past work, we have used metagraphs to model IoT device Manufacturer Usage Description (MUD) policies [20]. MUD is an IETF proposal which pushes IoT vendors to develop formal specification of the intended purpose of their IoT devices so that their network behavior can be locked down and verified rigorously. In that work, we checked MUD policies against an organization's local security policy for compliance using metagraphs. Understanding where in a company's network, an IoT device can safely be installed helps to ensure the network is not exposed to cyber attacks. In comparison, our work here uses metagraphs to model more complex network policies from multiple domains (*e.g.,* QoS, intrusion detection) and check their underlying properties.

In order to demonstrate the advantages of metagraphs we use them, in the next section, to model a real network's policy.

III. Policy Extraction

Network-device configurations can be long and complex; *e.g.,* one switch configuration in our case study has 5043 lines. So, we built an automated Parser (Figure 2) with details below:

Switch Config: A layer-3 switch configuration text-file containing Virtual Routing and Forwarding (VRF) instance details. A VRF instance is a logical zone used by administrators to keep users separate on common infrastructure [10].

Firewall Config: An input firewall configuration text-file containing interface configurations and VRF-Zone based (high-level) policy rules. For instance, the case study network we discuss in Section VI includes firewalls with Access Control Lists (ACLs), Quality of Service (QoS) rules, Intrusion Detection and Prevention (IDP) rules and reporting rules.

Route-Table Data: The route-table details from a network router provided as an input text file describing next-hop and outgoing-interface details.

Interface and Route Processing: The processing of firewall-interface configurations and route-table data to extract interface names and IP addresses reachable from them. The step allows construction of an IP address map for the VRF zones, so high-level policy rules can be refined down to network level.

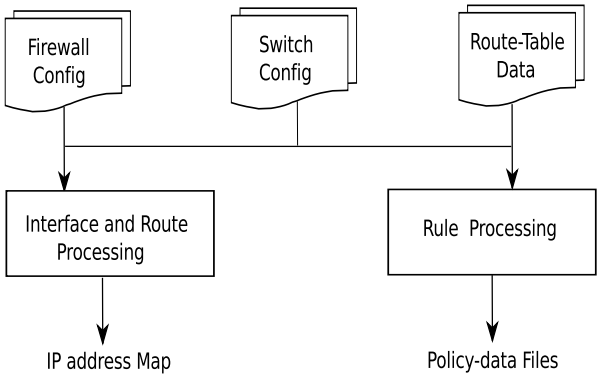


Fig. 2: *Network-device configuration parsing process.*

Rule Processing: The processing of ACLs, QoS rules, IDP rules and reporting rules configured on firewalls. This step allows the construction of policy-data files.

Policy-data Files: The output files containing details of the enforced policies. Each file describes particulars of a single policy class (*e.g.,* QoS) enabling metagraph modeling.

IP address map for VRF zones: The output IP address map describing the allocation of subnets and hosts per VRF zone. This map enables the PDN framework to automatically refine VRF-Zone based policies to network level.

Our Parser currently accepts the following device configurations as input: Juniper SRX 5800 firewalls, Palo Alto 5060 firewalls and Cisco Catalyst 6807 switches. It first processes the firewall-interface configurations and identifies the VRF zone each interface belongs to. Route-table data processing then locates the hosts and subnets per VRF zone. Rule processing then parses the ACLs, QoS rules, IDP rules and reporting rules enabled on the devices. The Parser outputs the policy-data files and an IP address map for the VRF zones. We model this policy data using metagraphs as outlined next.

IV. Policy Modeling

Metagraphs can have attributes associated with their edges. An example is a *conditional metagraph* [6] which includes propositions – statements that may be true or false – assigned to their edges as qualitative attributes [6]. The generating sets of these metagraphs are partitioned into a variable set and a proposition set. We define a conditional metagraph as follows.

Definition 3 (Conditional Metagraph)**.** *A conditional metagraph is a metagraph* $S{=}\langle X_p \cup X_v, E\rangle$ *in which* X_p *is a set of propositions and* X_v *is a set of variables, and:*

1. at least one vertex is not null, i.e., $\forall e' \in E, V_{e'} \cup W_{e'} \neq \phi$

2. the invertex and outvertex of each edge must be disjoint, i.e., $X = X_v \cup X_p$ *with* $X_v \cap X_p = \phi$

3. an outvertex containing propositions cannot contain other elements, i.e., $\forall p \in X_p, \forall e' \in E$, *if* $p \in W_{e'}$, *then* $W_{e'} = p$.

Conditional metagraphs enable the specification of stateful network-policies by allowing a policy (such as permit user u_1 to access resource r_1) to be activated conditionally (*e.g.,* during business hours only). We describe below our approach to modeling the different policy classes using conditional metagraphs. The approach mainly relies on (a) identifying applicable EPGs; and (b) defining suitable policy propositions.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2020.2974727, IEEE Transactions on Dependable and Secure Computing

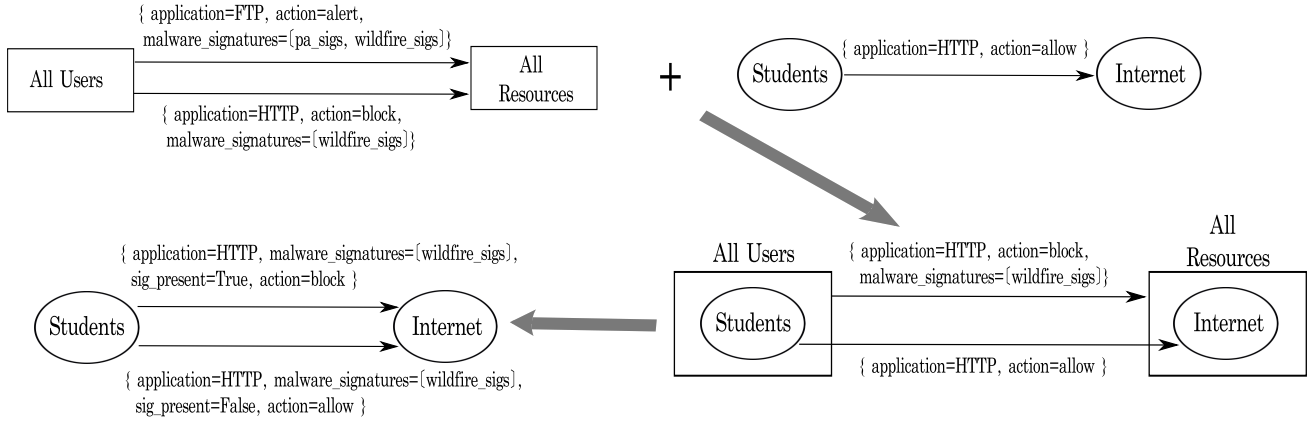


Fig. 3: *Metagraph modeling of an anti-malware policy. An anti-malware profile (top-left) specifying malware signature lists and response actions for FTP and HTTP traffic is associated with an access-control policy (top-right). The intermediate metagraph model (bottom-right) incorporates the applicable edges from this profile to the access-control policy (*i.e., *FTP rule is not applicable to this access control policy and is hence dropped). The resultant anti-malware policy model on the bottom-left describes how malware-infected HTTP traffic is blocked while uninfected traffic is permitted.*

A. Access-control policies

Access-control policies are commonly represented using the five-tuple: source/destination address, protocol, source/destination ports [11], [21], [35]. We construct metagraph models for the VRF-Zone based access-control policies leveraging this idea. A representative example from our case study is shown in Figure 4. Here, the source/destination addresses are represented by the EPGs- *Staff* and *DMZ*. Protocol, ports and time are propositions of the conditional metagraph.

B. Anti-malware/spyware policies

A core component of a company's Intrusion-detection policy is malware and spyware detection. Malware is malicious software that is designed to disrupt operations of computer systems or gain control of computers without consent [50]. Malware detection attempts to identify malicious-software signatures embedded in traffic permitted through a network. Many vendor databases (*e.g.,* Palo Alto [35], Wildfire [36]) maintain up-to-date lists of known malware signatures. Vendors also support multiple actions that can be taken in response to a successful malware signature match (*e.g., block* or *alert*).

Unlike more generic malware, spyware captures and relays information about systems or users without consent [50]. Hence, an anti-spyware policy aims to detect malicious traffic leaving a network from infected clients (*e.g.,* beaconing out to external command-and-control (C2) servers).

Administrators often create anti-malware profiles associating distinct traffic types with malware signature lists (*e.g.,* from a vendor database) and response actions. Likewise, network admins also often create anti-spyware profiles based on spyware-severity levels (*e.g.,* critical) and associate appropriate response actions (*e.g.,* block) [11], [21], [35]. An anti-malware (anti-spyware) policy can then be associated with an access-control policy to enable malware (spyware) detection.

We model anti-malware (anti-spyware) policies using two steps; first we create metagraph model for the anti-malware (anti-spyware) profile, then we model the profile association to the access-control policies. The top-left metagraph in Figure 3 shows an example anti-malware profile in our case study. An anti-malware (anti-spyware) profile is not restricted to a particular set of users or resources. So, the invertex and outvertex of a profile's conditional metagraph have EPGs *All-Users* and *All-Resources*. The propositions of the example model in Figure 3 describes that alerts are enabled upon FTP-based malware (spyware) signature detection (checked against Palo Alto and Wildfire signatures) and that traffic is blocked upon HTTP-based malware detection.

The top-right metagraph model in Figure 3 is the access control policy that associates this anti-malware profile to detect malware-infected HTTP flows. The bottom-right metagraph is the intermediate model that integrates these two policies; it only includes the profile edges applicable to the access-control policy context. This policy integration is done automatically by our system. The end result is given by the bottom-left model of Figure 3, the model propositions reflect the original intent; HTTP traffic without a match against the malware signatures list: *wildfire_sigs*, is permitted while those with a match are blocked. In addition, no FTP traffic is allowed by this access control policy, so that part of the security policy is redundant. This behavior extended to other user groups (*e.g.,* Staff), who were also allowed to access the Internet via HTTP. The anti-malware policy did not apply to access control policies which explicitly disabled user access to the Internet using HTTP.

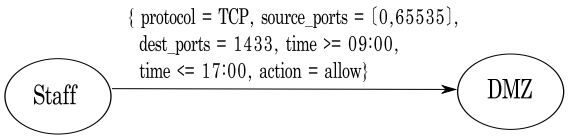


Fig. 4: *An access-control policy metagraph which permits MS-SQL from STAFF zone to the DMZ during business hours.*

C. URL-filtering policies

URL-filtering is another powerful feature network admins use to monitor and control how users access the Web. For instance, to stop students accessing illicit sites through the university network. However, research academics may sometimes require access to web pages that might be seen as illicit

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2020.2974727, IEEE Transactions on Dependable and Secure Computing

in another context, *e.g.,* cyber-security experts may need to examine hackers' websites. Thus, exceptions are needed.

The feature utilizes one or more URL-filtering databases containing millions of websites with each site classified in to one of many categories (*e.g.,* Palo Alto DB has 60 different categories [35]). Network admins usually create URL-filtering profiles containing groups of website categories and associate filtering actions (*e.g.,* block or override) per group [11], [21], [35]. These profiles can then be associated with access-control policies to gain visibility and control of user-access to the Web.

We re-apply our two-stage modeling approach here; URL-filtering profiles are modeled first and then the association of these profiles to the access-control policies is modeled.

Figure 5 shows the conditional metagraph of a URL-filtering profile found in our case study. The profile can apply to any user or resource. The protocol and port propositions have scope of Web traffic. Here, these propositions block access to URLs categorized as *extremism, malware* and *phishing*. In addition, access to URLs classified as *abused-drugs, adult, nudity, gambling* and *weapons* trigger alerts and access to sites classified as *abortion* require restriction override using a temporary password (generally issued by network admins or helpdesk staff). Associating this profile to the access-control policy in Figure 4 yields a policy model which has propositions that collectively block, alert, override or allow through URL-access requests, depending on the URL category.

D. Reporting policies

Network devices can also generate logs, alerts, traps, and provide other information, for instance, via SNMP (Simple Network Management Protocol) polling [39]. We group all these informational content and call them *reporting*.

An integral component of reporting is TCP session-logging which is commonly used to troubleshoot and debug policy problems. Network admins can choose to log a TCP session on transaction initiation (*i.e.,* session start) or end. For access-control policies that permit traffic, the best practice is to log at session end [21], [35]. Doing so, provides a detailed summary of important information including the application utilizing the session traffic (*e.g.,* Facebook), endpoint details (*i.e.,* IP addresses and ports), policy-rule ID applied to the flow and flow statistics (*e.g.,* number of bytes). Logging at session start is recommended for access-control policies that deny traffic as it involves a smaller subset of information overhead.

It's worth noting here that these TCP session logs can be exported to a Syslog server or a NetFlow collector for further analysis (requires translating logs to NetFlows first). Traffic statistics similar to those obtained from TCP session logs can also be derived by directly enabling NetFlow on the Firewall.

We model a TCP session-logging policy by extending the metagraph model of an access-control policy to include an additional *log_event* proposition. This proposition can accomodate the values *session-start* and *session-end*.

E. Quality of Service(QoS) policies

QoS assigns priorities to network devices and services and controls the amount of bandwidth each device/service is allowed to consume. Network admins often manage QoS by

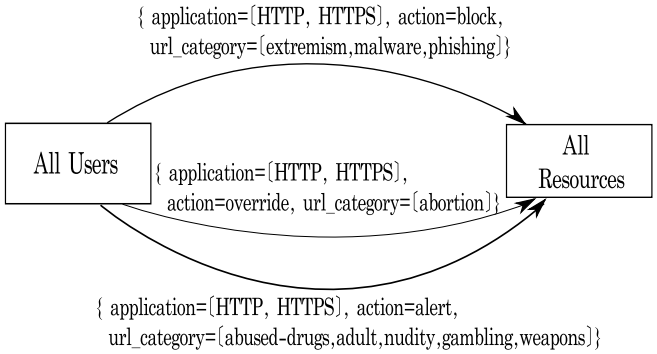


Fig. 5: *URL-filtering policy metagraph specifying URL categories and response actions for HTTP and HTTPS traffic.*

associating a priority level (*e.g.,* high, medium or low) with traffic types (*e.g.,* HTTP), applications (*e.g.,* netflix) or devices (IP or MAC address). We model a QoS policy by extending the metagraph model of an access-control policy to include an additional *priority* proposition.

The policy models derived for the different policy classes above reveal what an intuitive graphical tool, metagraphs actually are for specifying network policies. In the next section, we describe how we instantiate the policy models derived above in code to analyze their underlying properties.

V. Policy Definition and Verification

We wrote *MGtoolkit* [38] – a generic package for implementing metagraphs – to define our policy models. *MGtoolkit* is implemented in Python 2.7. The API allows users to instantiate metagraphs, apply metagraph operations and evaluate results.

The API provides a `Metagraph` class to instantiate a metagraph object consisting of `Node` and `Edge` objects. Each `Node` contains a subset of elements from the metagraph's generating set. An `Edge` has the members: `invertex` and `outvertex`, assigned a `Node` each, and an `attributes` member that returns any edge attributes. The class also supports methods to derive its adjacency matrix, find metapaths, check metapath properties (*e.g.,* `is_dominant_metapath()`) and edge properties (*e.g.,* `is_cutset()`).

The toolkit also provides a `ConditionalMetagraph` class which extends a `Metagraph` and supports proposition attributes in addition to variables. A `ConditionalMetagraph` inherits the base properties and methods of a `Metagraph` and additionally supports methods to check connectivity properties and redundancy properties. We use the `ConditionalMetagraph` class to instantiate the policy models described in Section IV using EPGs as variables, policy propositions and edges. We also extended the API methods in the original *MGtoolkit* package to include network-policy specific logic to accurately check policy consistency (*e.g.,* when detecting policy conflicts).

Our verification of metagraph consistency uses *dominance* [6] which can be introduced constructively as follows:

Definition 4 (Edge-dominant Metapath)**.** *Given a metagraph* $S{=}\langle X, E\rangle$ *for any two sets of elements* B *and* C *in* X*, a metapath* $M(B, C)$ *is edge-dominant if no proper subset of* $M(B, C)$ *is also a metapath from* B *to* C*.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2020.2974727, IEEE Transactions on Dependable and Secure Computing

Definition 5 (Input-dominant Metapath)**.** *Given a metagraph S=$\langle X, E\rangle$ for any two sets of elements B and C in X, a metapath $M(B, C)$ is said to be input-dominant if there is no metapath $M'(B', C)$ such that $B' \subset B$.*

In other words, edge-dominance (input-dominance) ensures that none of the edges (elements) in the metapath is superfluous. Thus, a dominant metapath can be defined as follows:

Definition 6 (Dominant Metapath)**.** *Given a metagraph S=$\langle X, E\rangle$ for any two sets of elements B and C in X, a metapath $M(B, C)$ is said to be dominant if it is both edge dominant and input-dominant.*

A non-dominant metapath indicates redundancy in the policy represented by the metagraph. The property provides a general framework to detect redundancies in multiple policy domains; simply check all feasible metapaths in a policy metagraph for edge and input dominance, if either fails, the policy includes redundancies. Past works [3] also classify policy redundancies based on the level of policy-rule overlap. But, these classifications are only meaningful when the policy-rule order is important (*e.g.,* in a vendor-device implementation). In a policy metagraph, rule order is generally inapplicable.

Algorithm 1 shows our implementation of Definition 6 to identify redundancies in each of our policy classes. An important aspect of the algorithm is how we compute feasible metapaths in the policy metagraph to check dominance. The text in [6] leaves determining this up to the implementor.

We could consider every possible combination of metagraph nodes as a potential source and target to search for feasible metapaths, but that would be too exhaustive. Instead, we begin by first identifying edge sets with overlapping invertices, outvertices and propositions (lines 2-16). A redundancy can only occur between edges with some invertex or outvertex overlap. Using this less exhaustive approach, we narrow down sources and targets (lines 17-22) to base our search for feasible metapaths. Each feasible metapath is then checked for dominance to identify policy redundancies.

The potential 'conflict set' of propositions in a metapath $M(B, C)$ can also be defined as follows:

Definition 7 (Conflict-set of Propositions)**.** *Given a conditional metagraph S=$\langle X_v \cup X_p, E\rangle$ for any two sets of elements B and C in X, a metapath $M(B, C)$ has the potential conflict set of propositions given by $(\bigcup_{e \in M(B,C)} V_e) \cap X_p$.*

Definition 7 provides a general framework for detecting policy conflicts for multiple policy classes. We can use it to compute the potential conflict set of propositions per metapath in a policy metagraph. We then apply domain specific knowledge to determine if the conflict is valid. For instance, with anti-malware policies, a conflict occurs when flows overlap and (a) have identical malware signature lists with different response actions (*e.g., alert* and *block*); or (b) have identical response actions with different malware signature lists (*e.g., pa_sigs* and *wildfire_sigs*). Algorithm 1 also shows the implementation of this policy conflict detection logic.

The code snippet in Listing 1 instantiates an anti-malware policy using *MGtoolkit* and then checks policy consistency.

Algorithm 1 Detects policy redundancies and conflicts using Metagraph algebras. The policy metagraph is pmg.

```
1:  procedure DetectPolicyInconsistencies(pmg)
2:      R1 ← dict()
3:      R2 ← dict()
4:      processed ← []
5:      redundancies ← []
6:      conflicts ← []
7:      for each e1 ∈ pmg.edges do
8:          for each e2 ∈ pmg.edges do
9:              if e1 != e2 and (e2,e1) not in processed then
10:                 i1 ← e1.invertex ∩ e2.invertex
11:                 i2 ← e1.outvertex ∩ e2.outvertex
12:                 i3 ← e1.propositions ∩ e2.propositions
13:                 if len(i1)> 0 and len(i3)> 0 then
14:                     R1[e1].append(e2)
15:                 if len(i2)> 0 and len(i3)> 0 then
16:                     R2[e1].append(e2)
17:             processed.append((e1, e2))
18:     for each e1, v1 ∈ R1 do
19:         src ← e1.invertex
20:         for each e2 ∈ R1[e1] do
21:             src ← src ∪ e2.invertex
22:         for each e3, v3 ∈ R2 do
23:             target ← e3.outvertex
24:             i4 ← e1.propositions ∩ e3.propositions
25:             if len(i4)> 0 then
26:                 mps ← GetMetapaths(src, target)
27:                 for each mp ∈ mps do
28:                     if not IsDominantMetapath(mp) then
29:                         redundancies.append(mp)
30:                     if (PropositionsConflict(mp) and
31:                         IsValidConflict(mp)) then
32:                         conflicts.append(mp)
        return (redundancies, conflicts)
```

The snippet is based on the example in Figure 4 together with another policy. The latter is generated by associating an anti-malware profile – enabling alerts for infected HTTP traffic – to the flow: *Students* → *Internet : protocol=Any, action=allow*. Executing the code snippet returns a conflict (Listing 2). This conflict originates from the different response actions (*i.e., block* vs *alert*) for a malware-signature match against *wildfire_sigs* between the two profiles.

The construction of a formal policy metagraph also allows us to reason about policies. For instance, we could check a specified security policy against an industry best practice policy for compliance. Our past work [40] has investigated this in the context of security policies in critical infrastructure networks using graph-based policy abstractions. The semantics developed there can readily be applied to policy metagraphs given a metagraph is a generalization of a simple graph.

VI. A University Network Case Study

Obtaining real network configurations from enterprise networks is difficult due to the sensitive nature of the data. We were able to obtain such configurations from a large university network. Due to security concerns and non-disclosure agreements, a modified version of the real network analyzed is presented. Effort has been taken to ensure that the implemented network, its policies and underlying issues uncovered remain intact. However, details such as IP addresses are anonymized.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2020.2974727, IEEE Transactions on Dependable and Secure Computing

Listing 1: MGtoolkit *implementation of an anti-malware policy defined using VRF-Zones as endpoint groups (*e.g., *Students).*

```
from mgtoolkit.library import ConditionalMetagraph, Edge

# define policy metagraph
variable_set = {'students','staff','internet','dmz'}
propositions_set = {'protocol=tcp', 'dest_port=80', 'sig_present=true', 'sig_present=false', 'action=permit',
          'action=deny', 'action=alert', 'malware_sigs=[wildfire_sigs]', 'malware_sigs=[pa_sigs, wildfire_sigs]'}
cm = ConditionalMetagraph(variable_set, propositions_set)

cm.add_edges_from([
 Edge({'students'}, {'internet'}, attributes=['protocol=tcp', 'dest_port=80', 'malware_sigs=[wildfire_sigs]', '
     sig_present=true', 'action=deny']),
 Edge({'students'}, {'internet'}, attributes=['protocol=tcp', 'dest_port=80', 'malware_sigs=[wildfire_sigs]', '
     sig_present=false', 'action=permit']),
 Edge({'students'}, {'internet'}, attributes=['protocol=tcp', 'dest_port=80', 'malware_sigs=[pa_sigs, wildfire_sigs
     ]', 'sig_present=true', 'action=alert']),
 Edge({'students'}, {'internet'}, attributes=['protocol=tcp', 'dest_port=80', 'malware_sigs=[pa_sigs, wildfire_sigs
     ]', 'sig_present=false', 'action=permit'])])

# check for conflicts using metagraph algebras
metapaths = cm.get_all_metapaths_from({'students'}, {'internet'})
for mp in metapaths:
   # apply domain-specific knowledge
   if cm.has_conflicts(mp):
      print('conflict detected: \n' \%s'\%(repr(mp.edge_list)))
```

Listing 2: *Partial output from running code in Listing 1. The detected conflict is caused by overlapping HTTP flows having different malware response actions (*i.e., *deny vs alert) for a match against the malware-signature list* wildfire_sigs*.*

```
conflict detected:
[Edge(set(['protocol=tcp', 'students', 'dest_port=80', 'malware_sigs=[wildfire_sigs]', 'action=deny', 'sig_present=
    true']), set(['internet'])),
Edge(set(['protocol=tcp', 'students', 'malware_sigs=[pa_sigs,wildfire_sigs]', 'dest_port=80', 'sig_present=true', '
    action=alert']), set(['internet']))]
```

A. Analyzed Configuration Data

The System Under Consideration (SUC) includes hundreds of switches and routers and thousands of links. Figure 6 shows a high-level view of the physical-network topology. The core devices enforcing bulk of the network policies include a pair of Internet-facing Palo Alto firewalls, a pair of internal Juniper SRX firewalls and nine Cisco Catalyst core/distribution routers. The multiple network devices and links provide redundancy and defense-in-depth. The function of these devices and networks are described below:

NREN: The National Research and Education Network which provides Internet services to the nation's education and research communities and their research partners.

The Cisco Border Routers (BR1, BR2): These perimeter routers connect the university network to the outside world.

The Palo Alto 5060 firewalls (PA1, PA2): These firewalls segregate the internal university networks (*i.e.,* inside) from the NREN (*i.e.,* outside). The firewall configurations are mirror images of each other and they implement access-control policies which restrict traffic flows. Being stateful firewalls, they ensure only solicited connections are allowed. The traffic filtering capabilities of these units extend to application layer filtering (*i.e.,* deep-packet inspection). QoS is also enabled on these firewalls using traffic rate-limiting functionality.

The Juniper SRX 5800 firewalls (SRX1, SRX2): These firewalls restrict traffic flow between internal university networks. Being stateful, they also only allow solicited connections. Their configurations are also mirror images of each other given they provide redundancy in the network.

Data Centers (DC1, DC2): These offsite facilities provide secure, reliable backup and archiving of campus-network user data. Our study scope includes policies that enable traffic to or from these centers but excludes internal data center policies.

Cisco Catalyst 6807 Core Switches (CSW1, CSW2): These switches construct the core layer of the network. They enable a high-speed packet switching backbone and do not perform any packet manipulation such as policy-based filtering.

Cisco Catalyst 6807 Distribution Routers (DR1- DR2): These routers form the distribution layer of the network which is the demarcation point between the core layer and the access-layer.

LANs and VLANs: These form the access layer of the network which comprise of devices enabling workgroups and users to utilize services provided by the distribution and core layers. These networks construct the VRF zones admins use to manage policies. There were 43 such logical zones including *Students, Staff, Internal- and External-DMZ.*

The Palo Alto firewalls have intrusion and threat prevention enabled using anti-malware, anti-spyware, file-blocking and URL-filtering policies. Each firewall has three anti-malware profiles configured and used; these profiles alert on or block different types of malware-infected traffic. Two spyware policies are also configured on each firewall but only one is used. This active policy sends alerts or blocks infected traffic based on the spyware-severity level. Each firewall also has 10 URL-filtering profiles configured and used. The most commonly used profile (*i.e.,* default profile) blocks the URL categories *copyright-infringement* and *extremism* and sends alerts for the categories *abortion* and *alcohol-and-tobacco*. Some profiles

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2020.2974727, IEEE Transactions on Dependable and Secure Computing

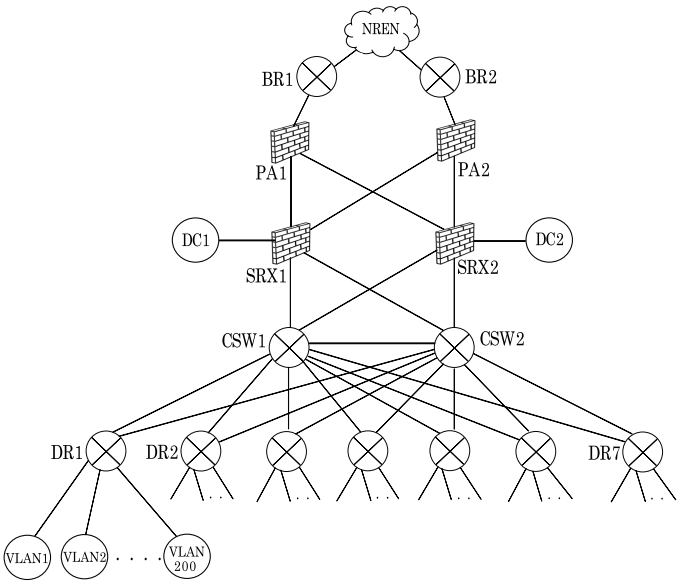
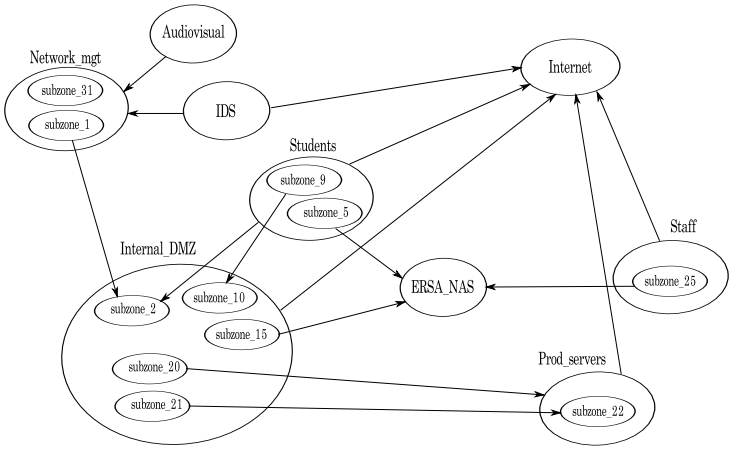
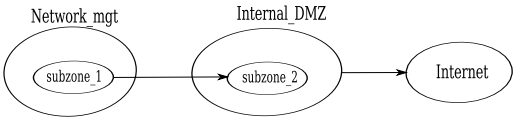


Fig. 6: *A high-level view of the university network studied. It includes border routers (BR), heterogeneous firewalls (Palo Alto and Juniper SRX), core switches (CSW) and distribution routers (DR). These devices collectively enable access between internal networks as well as between internal networks and the NREN (National Research and Education Network).*



(a) Metagraph describing access-control policies in the SUC (partially shown).



(b) Projection of (a).

Fig. 7: *The high-level policy metagraph (partially shown) describing SUC access-control policies and its projection over the subset of elements* $X = \{Network\_mgt, Internal\_DMZ, Internet\}$. *Policies are described using VRF zones and subzones. The projected edges depict reachability between the elements of* X.

create exceptions to this default policy; for instance, one allows selected access to URLs classified as *copyright-infringement*.

There are 931 access-control rules on each Palo Alto firewall enabling traffic flow between internal networks and the Internet. These rules allow, for instance, outbound Web, DNS and SSH application access from *inside*. They also allow SMB and FTP based file sharing, HTTPS and SMTP access inbound from the Internet. Of these access-control rules, 451 rules associate an anti-malware profile to enable malware-detection at a flow level, 436 rules have spyware-detection enabled and 61 have URL-filtering enabled.

Each Palo Alto firewall also has 29 QoS rules configured and active. These rules rate limit for instance, *inside* to *outside* traffic belonging to *netflix, twitch* and *Peer-to-Peer (P2P)* applications during business hours. Some rules create QoS exceptions; for instance, to bypass rate-limiting of P2P applications from *inside* to *outside* for particular subnet addresses.

Each Juniper firewall has 86 access-control rules enabling internal traffic flow. For instance, hosts in *Students* and *Staff* zones are allowed to access the licensing-servers located in the *Internal-DMZ*; and *Staff* and *Management-network* hosts can access the SQL servers in the *External-DMZ*.

B. Policy Visualization

We ran our system on a standard desktop computer (*e.g.,* Intel Core CPU 2.7-GHz computer with 8GB of RAM running Mac OS X). Our Parser extracted 1017 high-level rules written for 43 EPGs for the SUC. The EPG sizes varied from several IP addresses to over 200K IP addresses.

We instantiated a policy metagraph for each policy class in our SUC using *MGtoolkit*. The high-level access-control policy metagraph, for instance, consisted of 714 nodes and 1044 edges. For brevity, a partial view of this metagraph is shown in Figure 7(a). The resemblance of this policy to a network allows us to exploit the pattern recognition capabilities of the human visual cortex to better visualize policies.

Figure 7(b) is a projection over the subset $X = \{Network\_mgt, Internal\_DMZ, Internet\}$, describing end-end reachability between these elements. Each projection edge corresponds to one or more metapaths in the original metagraph. The reduction of size and complexity relative to the original metagraph simplifies understanding policy. For instance, as per the projection, access between $Internal\_DMZ$ and $Internet$ is enabled at a zone level while access between $Network\_mgt$ and $Internal\_DMZ$ is enabled at a subzone level.

This high-level reachability summary provided by a projection is very useful to network administrators. For instance, in the event of a cyber attack involving these zones, the knowledge allows one to quickly and precisely evaluate what access facilitates the ongoing attack. After all, the projection in Figure 7(b) consists of only two edges in comparison to the many in the original metagraph. Administrators can then disable corresponding access to mitigate the cyber attack.

We demonstrated these projections to our university's network operations team and were informed that they were much helpful in comparison to current visualization tools such as the Cisco Adaptive Security Device Manager (ASDM) [9] or Palo Alto Panaroma [33]. The ASDM allows to visualize single policy rules (*e.g.,* access-control rules) in a diagram, to clarify which direction a rule is controlling traffic, but does not allow to generate a summary view between two end points to understand the net effect of rules.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2020.2974727, IEEE Transactions on Dependable and Secure Computing

TABLE III: *Summary of policy inconsistencies found in the SUC (* VRF-Zone based rules, ** IP address/subnet based rules)*

Device	Policy Type	High-level rule count*	Network-level rule count**	# Intra-Device Conflicts	# Inter-Device Conflicts	# Redundancies
SRX Firewall	Access-control	86	61,305	9	179	14
	Reporting	72	29,656	0	0	10
Palo Alto Firewall	Access-control	931	1,591,633	142	179	69
	Anti-malware	451	565,908	309	0	32
	Anti-spyware	436	636,241	0	0	38
	URL-filtering	61	366,720	306	0	809
	Reporting	588	551,662	53	0	89
	QoS	29	563,296	38	0	49

TABLE II: *Summary of execution times of our PDN tool to detect SUC policy inconsistencies (* describes time taken to deploy low-level policy rules to a Mininet emulated network).*

Processing step	Execution time (seconds)
High-level policy extraction	37
Conflict detection (High-level)	223
Policy refinement (High-level to Low-level)	8
Conflict detection (Network-level)	3715
Conflict resolution (Network-level)	4
Policy deployment*	11

C. Formal Verification

We checked policy consistency as per Section V. Table II depicts a summary of execution times of our PDN tool for policy extraction, refinement, conflict detection, conflict resolution and policy deployment on to an emulated network. The conflicts and redundancies found are summarized in Table III.

Using metagraphs we can analyze policy conflicts and redundancies both (i) at a high-level (*i.e.,* independent of network implementation details); and (ii) at a network level (*i.e.,* incorporating implementation details). The high-level policy conflicts (redundancies) involving VRF zones, indicate inconsistencies in network admins' intent. Network-level conflicts (redundancies) additionally include inconsistencies caused by network-implementation errors (*e.g.,* the incorrect assignment of IP addresses to VRF zones which lead to unexpected zone overlaps). We report in Table III the more comprehensive network-level policy conflicts and redundancies.

We classify the policy conflicts found as *intra-device* and *inter-device conflicts* as described below.

1) Intra-Device Policy Conflicts: These are policy rules within a network device that have common packet matching criteria and yield conflicting outcomes. Each policy class in our SUC is disjoint. Hence, we can detect these by creating a metagraph per policy class and device and applying the metagraph algebras over the structures. We found 151 intra-device access-control policy conflicts in the Palo Alto and Juniper firewalls. These conflicts stemmed from overlapping rules with different modalities (*e.g., permit* vs *deny*).

Likewise, there were 38 QoS conflicts in the Palo Alto firewall due to overlapping rules having different traffic-priority levels. An example of such a QoS conflict is depicted in Figure 9. We also found 309 anti-malware policy conflicts in the Palo Alto firewall. These were due to overlapping rules associating profiles with different response actions (*e.g.,* alert vs block) for the same signature match (an example is depicted in Figure 8). There were also 306 URL-filtering conflicts in the Palo Alto firewall due to overlapping rules having different response actions for an identical URL-category match.

2) Inter-Device Policy Conflicts: Even when there are no intra-device conflicts for a particular policy class, there could still be conflicts between policies of different network devices. For instance, an upstream firewall might deny traffic that is allowed by a downstream firewall. We refer to such policy conflicts as inter-device policy conflicts.

The disjoint nature of the policy classes in our SUC allows us to detect these conflicts by considering policy interactions across different network devices for the same policy class. By doing so, we found 179 access-control policy conflicts between Palo Alto and Juniper firewalls. These originated from overlapping rules having different access-control actions.

3) Redundancies: We consider here policy redundancies that stem within a network device. The disjoint nature of the policy classes in our SUC allows us to re-use the metagraphs created for detecting intra-device policy conflicts, to also detect redundancies. We used Algorithm 1 to locate the redundancies in Table III.

We discussed with the university's network operations team to verify and fix the above problems. It is worth noting here that we were able to identify the above policy inconsistencies through the use of the rigorous formal foundations provided by our new tool: metagraphs. Application of metagraphs to the target use case allows us to demonstrate their effectiveness in formally describing and verifying policies in a large network.

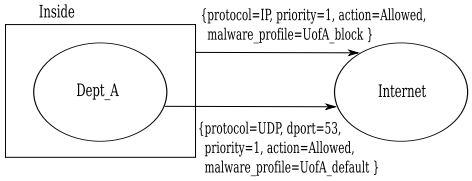


Fig. 8: *Example Anti-malware policy conflict found by our tool. The conflict occurs due to overlapping rules associating anti-malware profiles with different response actions (*e.g., *alert vs block).*

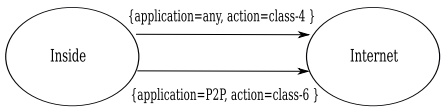


Fig. 9: *Example QoS policy conflict found by our PDN tool.*

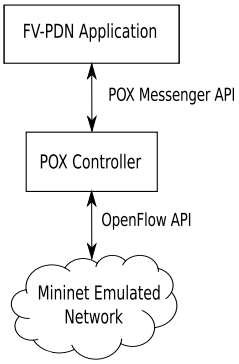


Fig. 10: *SDN test bed used for pre-deployment verification. Network policies are specified via the FV-PDN application and sent through to the POX SDN controller to update on to the mininet-based emulated network.*

D. Pre-deployment Verification

Checking policy consistency helps evaluate policy correctness [41], but configuration problems can also arise due to policy creators oversights as well as bugs in the policy refinement process. A key step in FV-PDN is to debug such configuration problems prior to deployment [41].

Network emulation offers a cost effective way to test configurations [25]. We use an OpenFlow based network emulator – Mininet – for our pre-deployment tests. The emulator is open source and enables virtual devices and interconnections using a singe Linux kernel [32]. Mininet offers flexibility- custom SDN topologies (such as that in our target network in Figure 7) can be created from a single, simple Python API.

We sourced our policies from a traditional network. But, we deploy them to a SDN network, to show the versatility of FV-PDN; users can manage heterogenous networks without being conversant with technology details required to deploy policies. The PDN engine automatically translates the user-specified high-level policies to match the target-network.

We use a POX [30] controller with Mininet in our test bed (Figure 10). POX is well-known for rapid prototyping of SDN controllers. In this setup, the Mininet network communicates directly with the POX controller via OpenFlow, which in turn communicates with our FV-PDN application using a built-in messenger-service API. The application obtains emulated-network topology details via this API to check compatibility of the input policy against the target network - another key verification step in FV-PDN [41].

Our tool resolves detected policy conflicts, automatically using a simple first-match strategy (*i.e.,* selects the most recently configured rule as the desired administrator intent). This strategy works for most cases in our SUC but in future a more complex conflict resolution framework can be incorporated to support more granular, conflict resolution policies.

Our system derives OpenFlow-equivalent statements from the conflict-free policies for the target SDN switches using the shortest-path algorithm. PDN sends these flow statements to the switches via the controller. Currently, we deploy policies that are natively supported by OpenFlow v1.1 switches (*i.e.,* access-control and QoS). We use the Mininet Python API to also create test scripts automatically; *i.e.,* test sources and sinks in the emulated network that match the input policy. These scripts are generated using *Expect* – a UNIX scripting and testing utility – which enables automated interactions with programs that expose a text terminal interface [26]. When the emulation is run, these tests create pathological traffic to verify expected policy-configuration behavior. The test scripts verify that the permit rules in a policy work correctly (*i.e.,* positive vetting), but we also need to ensure that services not explicitly enabled are blocked (*i.e.,* negative vetting). We achieve the latter using *nmap* and *tshark* based exhaustive port-scans.

E. Limitations

Our tool can currently extract and verify policies configured on a few vendor models; Palo Alto 5060, Juniper SRX 5800 and Cisco Catalyst switches. This support needs to be extended to popular vendor models such as Cisco ASA, Firepower firewalls [45] and Huawei USG6600 [52] to allow analysis of their network policies. In addition, the metagraph algorithms, both in mgtoolkit and PDN tool needs performance improvement. For instance, currently the PDN tool takes 62 minutes to find non-dominant metapaths of a policy metagraph containing 200 nodes and 1500 edges! Our PDN tool is also limited in its metagraph visualization capabilities. We could not find any off the shelf graph package that allows to crisply visualize metagraphs. Currently we utilize Graphviz [13] for this task; the package allows clustering of nodes, but creates duplicate nodes when a node is associated with more than one cluster.

VII. Conclusion and Future Work

There are various obstacles that hinder reliable network-policy specification. One most prominent is the lack of policy abstractions that (i) can decouple policy from the underlying physical infrastructure; and (ii) offer rigorous formal foundations to verify and reason about policies.

Metagraphs can help address the shortfall; they allow to express network policies abstractly, independent of implementation details and provide rich algebras to analyze important policy properties such as reachability and consistency. We demonstrate the versatility of metagraphs in network-policy specification by using them to model and analyze real policies from a large university network.

References

[1] DMTF Policy Working Group. CIM Simplified Policy Language (CIM-SPL). International Standard DSP0231,DMTF, 2009.
[2] E. Al-Shaer and H. Hamed. Design and implementation of firewall policy advisor tools. *DePaul University, CTI, Technical Report*, 2002.
[3] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan. Conflict classification and analysis of distributed firewall policies. *IEEE JSAC*, 23(10):2069–2084, 2005.
[4] C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker. NetKAT: Semantic foundations for networks. *ACM SIGPLAN Notices*, 49(1):113–126, 2014.
[5] Y. Bartal, A. Mayer, K. Nissim, and A. Wool. Firmato: A novel firewall management toolkit. *ACM TOCS*, 22(4):381–420, 2004.
[6] A. Basu and R. W. Blanning. *Metagraphs and their applications*, volume 15. Springer Science & Business Media, 2007.
[7] M. S. Beigi, S. Calo, and D. Verma. Policy transformation techniques in policy-based systems management. In *Proceedings. Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, 2004. POLICY 2004.*, pages 13–22, June 2004.
[8] R. Boutaba and I. Aib. Policy-based management: A historical perspective. *JNSM*, 15(4):447–480, Dec 2007.

[9] A. Cisco. Series firewall ASDM configuration guide. *Cisco Systems Inc., updated March*, 31, 2014.

[10] Cisco Systems Inc. *Catalyst 6500 Series Switch and Cisco 7600 Series Router Firewall Services Module Configuration Guide*, 2010.

[11] Cisco Systems Inc. *Cisco ASA Series Configuration Guide, 9.0*, 2013.

[12] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. In *Policies for Distributed Systems and Networks*, pages 18–38. Springer, 2001.

[13] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull. Graphvizopen source graph drawing tools. In *International Symposium on Graph Drawing*, pages 483–484. Springer, 2001.

[14] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi. Hierarchical policies for software defined networks. In *HotSDN*, pages 37–42, New York, NY, USA, 2012. ACM.

[15] N. Foster, M. J. Freedman, R. Harrison, J. Rexford, M. L. Meola, and D. Walker. Frenetic: a high-level language for openflow networks. In *PRESTO*, page 6. ACM, 2010.

[16] M. G. Gouda and A. X. Liu. Structured firewall design. *Computer Networks*, 51(4):1106–1120, 2007.

[17] T. G. Griffin and J. L. Sobrinho. Metarouting. SIGCOMM '05, pages 1–12, New York, NY, USA, 2005. ACM.

[18] A. J. T. Gurney and T. G. Griffin. Lexicographic products in metarouting. In *2007 IEEE ICNP*, pages 113–122, Oct 2007.

[19] S. Gutz, A. Story, C. Schlesinger, and N. Foster. Splendid isolation: A slice abstraction for software-defined networks. In *HotSDN*, pages 79–84. ACM, 2012.

[20] A. Hamza, D. Ranathunga, H. H. Gharakheili, M. Roughan, and V. Sivaraman. Clear as mud: generating, validating and applying iot behavioral profiles. In *IoTS&P*, pages 8–14. ACM, 2018.

[21] Juniper Networks, Inc. *Getting Started Guide for the Branch SRX Series*. 1133 Innovation Way, Sunnyvale, CA 94089, USA, 2016.

[22] P. Kazemian, G. Varghese, and N. McKeown. Header space analysis: Static checking for networks. In *NSDI*, pages 113–126, 2012.

[23] B. W. Kernighan and P. J. Plauger. *The elements of programming style*. McGraw-Hill, PJ New York, 1978.

[24] A. Khurshid, W. Zhou, M. Caesar, and P. Godfrey. Veriflow: Verifying network-wide invariants in real time. *ACM SIGCOMM Computer Communication Review*, 42(4):467–472, 2012.

[25] S. Knight, H. Nguyen, O. Maennel, I. Phillips, N. Falkner, R. Bush, and M. Roughan. An automated system for emulated network experimentation. In *ACM CoNEXT*, pages 235–246, 2013.

[26] D. Libes. *Exploring Expect: A Tcl-based toolkit for automating interactive programs*. O'Reilly, 1995.

[27] A. X. Liu and M. G. Gouda. Diverse firewall design. *IEEE ICDSN*, pages 1237–1251, 2008.

[28] J. Lobo, R. Bhatia, and S. Naqvi. A policy description language. In *Proceedings of AAAI*, pages 291–298, 1999.

[29] E. C. Lupu and M. Sloman. Conflicts in policy-based distributed systems management. *IEEE TSE*, 25(6):852–869, Nov 1999.

[30] M. M. [Online]. POX Available: www.noxrepo.org/.

[31] C. C. Machado, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho. Policy authoring for software-defined networking management. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 216–224, May 2015.

[32] Mininet. [Online]. An Instant Virtual Network on your Laptop (or other PC) Available: www.mininet.org/.

[33] P. A. Networks. Panorama administrator's guide. *Palo Alto Networks Inc., updated July*, 8, 2019.

[34] J. Nicklisch. A rule language for network policies. *Position Paper*, 1999.

[35] Palo Alto Networks, Inc. *PAN-OS Administrator's Guide, 8.0*. 4401 Great America Parkway, Santa Clara, CA 95054, USA, 2017.

[36] Palo Alto Networks, Inc. *WildFire Administrator's Guide, 7.0*. 4401 Great America Parkway, Santa Clara, CA 95054, USA, 2017.

[37] C. Prakash, J. Lee, Y. Turner, J.-M. Kang, A. Akella, C. Clark, Y. Ma, and Y. Zhang. PGA: Using graphs to express and automatically reconcile network policies. In *ACM SIGCOMM, pages 29-42*, 2015.

[38] D. Ranathunga, H. Nguyen, and M. Roughan. Mgtoolkit: A python package for implementing metagraphs. *SoftwareX*, 6:91–93, 2017.

[39] D. Ranathunga, M. Roughan, P. Kernick, and N. Falkner. Towards standardising firewall reporting. In *WOS-CPS*. Springer LNCS, 2015.

[40] D. Ranathunga, M. Roughan, P. Kernick, and N. Falkner. Malachite: Firewall policy comparison. In *IEEE ISCC*, pages 310–317, June 2016.

[41] D. Ranathunga, M. Roughan, P. Kernick, N. Falkner, H. Nguyen, M. Mihailescu, and M. McClintock. Verifiable policy-defined networking for security management. In *IJCET*, 2016.

[42] D. Ranathunga, M. Roughan, H. Nguyen, P. Kernick, and N. Falkner. Case studies of SCADA firewall configurations and the implications for best practices. *IEEE TNSM*, pages 871–884, 2016.

[43] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker. Modular SDN programming with Pyretic. *Technical Report of USENIX*, 2013.

[44] R. Sahay, G. Blanc, Z. Zhang, K. Toumi, and H. Debar. Adaptive policy-driven attack mitigation in sdn. In *XDOMO*, pages 4:1–4:6, New York, NY, USA, 2017. ACM.

[45] O. Santos, P. Kampanakis, and A. Woland. *Cisco Next-Generation Security Solutions: All-in-one Cisco ASA Firepower Services, NGIPS, and AMP*. Cisco Press, 2016.

[46] A. Schwabe, P. A. Aranda-Gutiérrez, and H. Karl. Composition of sdn applications: Options/challenges for real implementations. In *ANRW*, pages 26–31, 2016.

[47] R. Sinnema and E. Wilde. eXtensible Access Control Markup Language (XACML). RFC 7061, 2013.

[48] M. Sloman. Policy driven management for distributed systems. *Journal of network and Systems Management*, 2(4):333–360, 1994.

[49] 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 *CoNEXT*, pages 213–226. ACM, 2014.

[50] K. Stouffer, J. Falco, and K. Scarfone. Guide to Industrial Control Systems (ICS) security. *NIST Special Publication*, 800(82):16–16, 2008.

[51] J. Strassner. *Policy-Based Network Management: Solutions for the Next Generation (The Morgan Kaufmann Series in Networking)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[52] H. Technologies. USG6600 Series Next-Generation Firewall. [Online]. Available: https://e.huawei.com/au/products/enterprise-networking/security/firewall-gateway/usg6600, 2019.

[53] D. C. Verma. Simplifying network administration using policy-based management. *IEEE Network*, 16(2):20–26, 2002.

[54] A. Wool. Trends in firewall configuration errors: Measuring the holes in Swiss cheese. *IEEE Internet Computing*, 14(4):58–65, 2010.

Dinesha Ranathunga is a Postdoctoral research fellow at the Teletraffic Research Centre at University of Adelaide, Australia. He received his Ph.D. for his thesis titled Auto-configuration of critical network infrastructure from the University of Adelaide in 2017. His research interests include SCADA network security, Policy Defined Networking, Software Defined Networking and IoT security.

Prof. Matthew Roughan obtained his PhD in Applied Mathematics from the University of Adelaide in 1994. He has since worked for the Co-operative Research Centre for Sensor Signal and Information Processing (CSSIP), in conjunction with DSTO; at the Software Engineering Research Centre at RMIT and the University of Melbourne, in conjunction with Ericsson; and at AT&T Shannon Research Labs in the United States. Most recently, he works in the School of Mathematical Sciences at the University of Adelaide, in South Australia. His research interests range from stochastic modelling to measurement and management of networks like the Internet. He is author of over a 100 refereed publications, half a dozen patents, and has managed more than a million dollars worth of projects. In addition, his coauthors and he won the 2013 Sigmetrics "Test of Time" award, and his work has featured in New Scientist and other popular press.

Hung Nguyen joined the Teletraffic Research Centre at the University of Adelaide in 2012. He received his PhD in Computer and Communication Sciences from the Swiss Federal Institute of Technology, Lausanne, Switzerland (EPFL). His research interests include Software Defined Networking, 5G, network measurements, tomography, and privacy preserving techniques. He has published more than 40 refereed papers on these topics.