POLANCO: Enforcing Natural Language Network Policies

Sergio Rivera, Zongming Fei, James Griffioen
Laboratory for Advanced Networking, University of Kentucky
Lexington, Kentucky 40506–0495, USA
Emails: {sergio,fei,griff}@netlab.uky.edu

***Abstract*—Network policies govern the use of an institution's networks, and are usually written in a high-level human-readable natural language. Normally these policies are enforced by low-level, technically detailed network configurations. The translation from network policies into network configurations is a tedious, manual and error-prone process. To address this issue, we propose a new intermediate language called *POlicy LANguage for Campus Operations (POLANCO)*, which is a human-readable network policy definition language intended to approximate natural language. Because POLANCO is a high-level language, the translation from natural language policies to POLANCO is straightforward. Despite being a high-level human readable language, POLANCO can be used to express network policies in a technically precise way so that policies written in POLANCO can be automatically translated into a set of software defined networking (SDN) rules and actions that enforce the policies. Moreover, POLANCO is capable of incorporating information about the current network state, reacting to changes in the network and adjusting SDN rules to ensure network policies continue to be enforced correctly. We present policy examples found on various public university websites and show how they can be written as simplified human-readable statements using POLANCO and how they can be automatically translated into SDN rules that correctly enforce these policies.**

***Index Terms*—network policy, software defined networks, campus network**

I. Introduction

Campus networks have evolved into complex infrastructure consisting of routers, switches, access points, and middleboxes (e.g. firewalls, load balancers, and intrusion detection/prevention systems) interconnecting a plethora of devices, including general-purpose equipment like computer desktops and servers; personal and corporate mobile devices (e.g. phones, laptops, tablets); appliances that provide monitoring and threat detection; and special purpose devices deployed at key places across the physical campus (e.g. copiers, printers, badge readers, motion sensors, IP telephones, surveillance camera, video-conferencing equipment, and payment terminals).

Historically *campus network policies* have largely focused on keeping the campus network secure, blocking forbidden traffic, detecting and preventing intrusions, only allowing authorized users to use the (wireless) network, etc. Because network policies were largely about security, they could often be implemented with relatively little effort using an off-the-shelf network security device such as a firewall or intrusion detection and prevention system (IDS/IDP). These devices often came with pre-configured commonly used security policies (i.e., best-of-breed security rules) that sufficed, with some customization, for most campuses. Consequently, campus network policy documents have historically been rather brief documents (if they exist at all), relying on, and defaulting to, the pre-configured policies that came with the firewall or IDS/IDP device.

However, the growing complexity of campus networks has led to the need for more complex network policies; policies that not only ensure security, but also define how the network is to be used – what is expected, allowable, and acceptable use of the network. For example, the campus network policy may limit access to a research lab's network to authorized users of the lab, or may prevent unallowed or unauthorized traffic from being sent to a particular device such as a printer, surveillance camera, or door lock, or may disallow unencrypted traffic in certain situations such as credit card transaction networks.

Given the growing complexity of campus networks, institutions are increasingly appointing committees – which we will refer to as *Policy Writing Committees (PWCs)* – to make decisions about what the campus network policies should be (at a very high-level), ultimately producing a network policy document. PWCs are often comprised of administrators who understand "who" should be able to access "what", but lack the technical expertise of network sys-admins. They write policies in high-level human-readable natural language formats that must be mapped (by a networking expert) into technically detailed (low-level) network configuration files that enforce the policies. The potential for errors entering into this mapping process – performed by some human network sys-admin – is significant and occurs frequently in practice (see Fig. 1). The process is heavily reliant on the level of expertise of the individual(s) in charge of configuring the network system and the interpretation of the policies. Moreover, modifications to the network often break policies that were being correctly enforced earlier. In short, *the need to translate increasingly complex high-level human-readable campus network policies (written by non-technical policy writing committees) into low-level detailed, complex and error-free, network configurations that enforce those policies is becoming a significant challenge for campuses.*

This work was supported in part by the National Science Foundation under Grants ACI-1541380, ACI-1541426, and ACI-1642134.

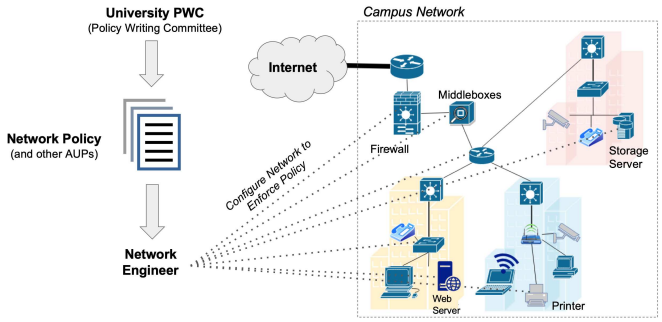

Fig. 1. Network policy enforcement today

To address the growing gap and challenge of translating between high-level natural language network policies written by campus PWCs and the low-level complex network configurations needed to enforce those policies, we propose a new intermediate language called *POlicy LANguage for Campus Operations (POLANCO)*. POLANCO is a human-readable network policy definition language intended to approximate natural language – language that might be used by a PWC. Because it is a high-level language, translation from natural language policies to POLANCO is often a straightforward (less error prone) process than making the jump from natural language to network configuration files. Despite being human-readable, POLANCO can be used to express network policies in a technically precise way – a way that can be automatically translated (via processing) into a set of software defined networking (SDN) rules and actions (or to network configuration files) that enforce the policies, ensuring that the specified policies are being correctly enforced by the network. Moreover, POLANCO is capable of incorporating information about the network (e.g., topology or device changes), reacting to changes in the network and adjusting SDN rules to ensure network policies continue to be enforced correctly.

To demonstrate the power of POLANCO, we present policy examples found on various public university websites and show how they can be written as simplified human-readable statements using POLANCO. We then describe how POLANCO policy statements can be automatically and continuously translated into an SDN controller that is able to push out network configuration rules that enforce the specified policy. (For the purposes of this paper, we assume that the campus network is managed and controlled by a single organization (Campus IT) and that the network is "programmable" in some way – e.g., OpenFlow [1] or NETCONF [2] – via a centralized SDN controller.)

The rest of the paper is organized as follows. Section II begins by discussing the issue of mapping natural language network policies into network configurations. Section III describes the network policy language POLANCO and its design principles. Network policies expressed in POLANCO are translated into Business Rule Management System (BRMS) code that is then translated into network configurations, as described in Section IV. In Section V, we describe an initial prototype and show some example policies that the system is capable of enforcing. In Section VI, we describe related work. Finally, we present our conclusions in Section VII.

II. Network Policy Mapping

Institutions appoint policy writing committees to define (and document) the policies that specify how institutional computional and network resources may be used. Most institutions have a set of *Acceptable Use Policies (AUPs)* that define how campus resources may and may not be used. In some cases, there may be a specific AUP focused on campus network policies, and in other cases the campus network policies may be spread across multiple AUPs, or embedded in a single AUP. For our purposes, we assume the campus network policies are represented by a single AUP devoted to network policy. AUP documents are predominantly intended for non-technical audiences (the users of the systems/network), and are written with imprecise, verbose, and business-like vocabulary. Unfortunately, these documents are oftentimes the only source of truth for network operators to enforce policies. Their lack of precision makes it difficult and error-prone for operators to translate the policies into network equipment.

Unfortunately, PWCs rarely understand the low-level mechanisms (e.g., network configurations) used to enforce the policies, and thus are oftentimes unaware of the challenges of implementing the policies they write. Moreover, given the complexity of today's campus networks consisting of thousands of switches, access points, firewalls, IDS/IDP systems, load balancers, NAT boxes, etc, the potential to introduce errors in the (human/manual) translation process is significant.

Ideally, there should be minimal human intervention in the translation process, relying instead on automated process to transform the PWC's policies (written in imprecise natural language format) to the complex set of low-level network configuration rules that must be distributed to thousands of network devices across the campus. While we want to minimize the human element, it is important that we not abolish it since it is important that the natural language documents produced by PWCs be interpreted by a network expert to ensure the PWC did not unintentionally specify policies that they did not intend. At the same time, the human expert should not be responsible for the entire translation process.

To address this need, we propose an intermediate step in the mapping process that removes the potential for errors and enables feedback between PWCs and network operators. Rather than crafting device configurations themselves, network operators write a set of simplified and *structured* human-readable policy statements – statements that are still understandable by PWCs, but yet can be parsed and automatically converted into the low-level config rules pushed out to network switches.

To minimize the human effort needed to translate PWC network policy documents, we developed a high-level language called *POlicy LANguage for Campus Operations (POLANCO)* that maintains the readability of natural language, but offers

TABLE I
EQUIVALENCE BETWEEN NETWORK POLICY STATEMENTS FOUND IN AUPS AND STATEMENTS WRITTEN IN POLANCO

Id	Network Policy Statements	POLANCO Statements
P1	Configure operating systems to meet system best practices. This includes but is not limited to the following: Enable necessary services and applications; disable all others [3]	when node is connected to a web-server then allow-only web traffic
P2	Applications which transmit sensitive information over the network in clear text like telnet and ftp are prohibited and will be blocked [4]	block "applications which transmit sensitive information"
P3	In no case shall DNS servers (except those maintained by IT department for the express purposes delineated) be connected to the network [5]	when node is connected to a host then allow-only DNS-response traffic from authorized DNS servers.
P4	All inbound traffic to the campus is blocked by default, unless explicitly allowed [6]	when node is a Firewall then block traffic from Internet to campus-network

the structure needed to be automatically translated into the network configurations that implement the network policies.

To demonstrate the relationship between POLANCO and high-level natural language network policies written by PWCs, consider the policies described in Table I. Table I provides example network policy excerpts taken from the AUPs of four different academic institutions along with their equivalent POLANCO statements. Although the statements differ, the mapping between PWC policies and POLANCO maintains much of the same language.

A high-level representation of the pipeline we propose for network policy mapping using POLANCO is shown in Fig. 2. At the start of the pipeline, human intervention (i.e., network engineer) is required to translate PWC policies into high-level POLANCO statments–note this allows both the PWCs and network operators to ensure they are defining and enforcing *the same* network policy. Given a POLANCO specification, the transformation to SDN rules that get pushed into switches can be completely automated.

The first stage of the translation process leverages a *Business Rule Management System (BRMS)* (see Section IV) to combine POLANCO policy statements with information about the current state of the network (obtained from the SDN controller) to determine how best to implement the policy given the current network topology. The output of the BRMS are rules that are then pushed to the appropriate network switches by the SDN controller or possibly to network servers/devices. For example, policies $P1$ and $P3$ would generate an appropriate set of OpenFlow rules to enforce the policy at switches; policy $P4$, on the other hand, would generate a set of iptable rules for a firewall, while policy $P2$ would result in a combination of both since the policy needs to be applied at every node. Should

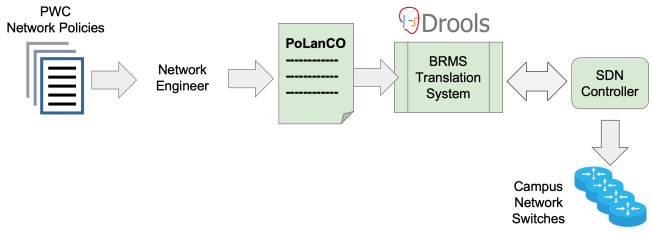

Fig. 2. Proposed network policy translation pipeline

an event change the network state (i.e. a new device is added to the network), previously translated POLANCO statements could be dynamically deactivated and retranslated given the new system conditions. We leverage built-in topology discovery mechanisms found in emerging network architectures such as SDN controllers in order to learn the global view of the network, track network events (e.g. links up, available network paths, hosts joining the network), and abstract out through a RESTful API, connections to data plane devices.

III. A NETWORK POLICY LANGUAGE

As noted earlier, a technically-precise, human-readable language would be useful as a way to provide feedback between PWCs and network operators, and to automate the translation process from policies into low-level configurations. We describe below the set of design goals for POLANCO:

1) **High-Level Network Identifiers:** While network switches ultimately must be configured using low-level identifiers (e.g., MAC addresses, IP addresses, and VLAN numbers), humans write policy using high-level concepts and identifiers. Consequently, we want a language based on high-level identifiers. Instead of using network identifiers, we need the ability to assign each identifier a role and/or trait (e.g. address, device types, type of traffic, group of users) and raise the level of abstraction. Human-readable statements based on such assignments can (1) be easily understood by PWCs, (2) help move towards the documentation of policies that were previously deployed, and (3) be expanded by translation mechanisms into low-level details when needed.
2) **High-level Concepts/Descriptions:** The proposed language must have terms to describe common network abstractions such as file servers, web servers, printers, firewalls, secure channels, blocking, mirroring, etc. The translation mechanisms used to install low-level configurations should derive the types of configurations needed to enforce the policy, what portions of the network should configurations be pushed to, and how the configurations should look.
3) **Event-Aware:** While some of the policies that need to be enforced on networks do not change over long periods of time, some policies are dynamic, periodic, or based on

the current status of the system. The written statements must actively enforce policy and adapt network configurations to cope with events such as the addition of new equipment to the network, detection of a malicious actor or an infected machine, changes in the policy, current time of day, etc.

4) **Exception Specification:** The language should allow for the explicit specification of authorized exceptions. Our past work [7], [8] has demonstrated that exceptions are a common part of a policy ecosystem. The current process to deploy policy exceptions is largely manual and time-consuming and should be specifiable via the language and automatically deployable. The language should act as a middle layer where *both* policies and their exceptions can be written and automatically deployed.

POLANCO was developed following these design goals. Based on gathered network information, the language allows network operators to write simplified and precise human-readable policy statements using the syntax shown in Fig. 3.

TABLE II
LIST OF VALUES FOR EACH POLANCO TOKEN

TOKEN	TOKEN Values
device-type	Firewall, Web Server, Switch, Printer, etc.
Action (param)	Allow, Allow-Only, Block, Send to (Controller, IDS, HoneyPot), Mirror to (Port)
traffic-type	Web, FTP, Video, Print Jobs, etc.
end-point	netlab-network, storage-systems, authorized DNS, etc.

A policy always starts with the keyword `policy` followed by an operator-defined name assigned to it. Policies can be assigned a priority that influences the order of the policy processing by the rule engine that produces network configurations (see Section IV); higher priority policies are evaluated first and are particularly useful for specifying exceptions that need to be evaluated first. POLANCO statements are written using the syntax shown in lines 5-8 of Fig. 3. The syntax is based on business rule management specification (see Section IV) using the keywords `when` and `then`. The syntax in Fig. 3 also shows four types of tokens that can be replaced with the values shown in Table II.

The values of the *device-type* correspond to the labels added to the nodes recorded in the SDN controller (e.g., stored in a graph database) during network information gathering. At present, POLANCO supports actions that include allowing (i.e. forwarding) legitimate traffic, blocking (i.e. dropping) packets of a particular flow, sending packets of the matching flow to an external entity (specified as a parameter) for further processing, and mirroring (i.e. copying packets) to a particular port (if applicable) for out-of-band analysis. As we can see, gathering network information is vital for the definition of POLANCO statements since it provides the context in which policies are applied. In terms of the syntax of the statements themselves, policies should be written following the conditional-body structure of conventional business rules. For policies that have to be applied in **all** the devices in the network, the conditional part of the POLANCO statement may be omitted.

Last but not least, the power of the POLANCO syntax is that with simple combinations of words that describe the types of nodes or their relationships it is possible to identify the appropriate set of devices where network policies can be applied. We present some examples in Section V.

IV. Applying Policies in Context

Given a machine-readable language for expressing network policy, we needed a way to convert the language into network configurations. However, as we mentioned earlier, network policies are dependent on the network state. For example, knowing where to place a firewall to block certain types of traffic requires understanding the current network topology (i.e., network state information); or enabling traffic from certain users to certain printers, requires understanding the types of nodes (e.g., printers) and who is currently logged into the network (e.g., Radius users).

Consequently, we leveraged practices used in business systems to make decisions based on the current state of the system; namely *Business Rule Management System*. A BRMS combines rule definitions with a series of facts and event listeners to represent the realtime context, and is used to determine what set of actions to trigger at any given time. In our implementation, the state of the system is network topology information (e.g. node types, links, paths) and the actions correspond to API calls to the SDN controller that would cause OpenFlow rules enforcing the policy to be installed into routers.

Fig. 4 shows the components of a BRMS. At a high-level, a BRMS consists of:

- A centralized *repository* where rules are stored (e.g. a database, a folder).
- Authoring and maintenance tools used to define rules in terms of system facts (e.g. a GUI, enhanced text editors)..
- A working memory with current system facts (e.g. network information) stored as data objects. Facts may be added on startup or (removed) as a consequence of other rules being activated.
- A runtime environment (called a rule engine) that invokes and/or deactivates rules throughout a continuous execution cycle and may also trigger the execution of external procedures, e.g., code that issues RESTful calls to another system and enforces the decision that has been made.

BRMS authoring tools usually come with syntax restrictions that are commonly bound to the underlying programming language used to develop the actual BRMS. These restrictions make rule definition unnatural for policy administrators, particularly if they do not have a software-development background. Despite their varying levels of complexity, the structure of a business rule is generally the same and POLANCO adheres to such a structure consisting of a conditional (antecedent) and a body (consequent) [9]. The *conditional* (represented by the keyword **`when`**) determines the set of constraints that must be satisfied in order to activate a given rule. Constraints are

PoLanCO Rule Template

```
policy "policy-name"

[policy priority n]

[when
  Node is [connected to] a device-type
then]
  Action [param] [traffic-type] traffic [from end point A] [to end point B]
```

Fig. 3. Syntax of the POlicy LANguage for Campus Operations

always written in terms of the properties of the data objects that are part of the working memory of the BRMS. The *body* (represented by the keyword **then**) contains the set of actions that are executed if the constraints are satisfied. Unlike constraints, actions may include API calls (local or external) or modifications to working memory data objects. The fact that *any* data object can be part of the working memory makes BRMSs flexible mechanisms to enforce policies in different contexts, including campus network policies. The information needed to enforce network policies can be found in the descriptions about nodes and connections of the underlying topology.

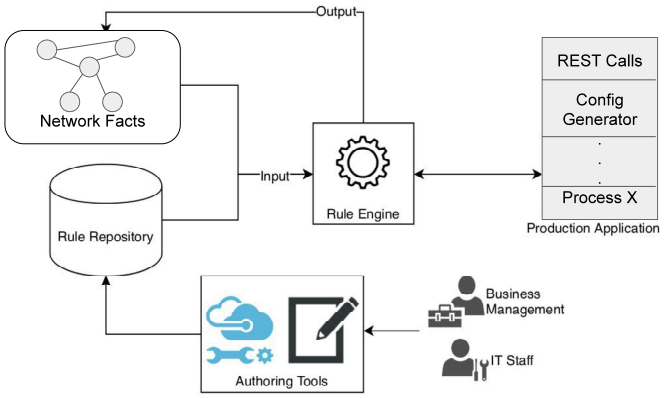

Fig. 4. High-level architecture of a BRMS

For example, recall the policy $P2$ shown in Table I that requires IT administrators to disable insecure protocols from network devices and systems in the network. Since data objects representing *network devices* and *systems* are part of the working memory, it is possible to write BRMS code that would generate calls to a remote system (i.e. the SDN controller) to push OpenFlow DROP rules into the affected nodes. More importantly, the event-awareness of the system makes it ideal to handle events such as a new network device joining the network. In that case, since the policy is still valid and active, the actions of the corresponding BRMS rule would be activated and applied to the newly connected device.

Fig. 5 shows BRMS code implementing the policy $P2$ using Drools. The conditional of the rule (line 3) selects all the nodes where the policy will be enforced (i.e. nodes that are of type `NETDEVICE`). Then, the body of the rule includes configuration details needed to enforce the policy such as: protocol numbers of insecure protocols like FTP (ports 20 and 21) and telnet (port 23); `Config` data objects created using information about each selected node in the conditional, the protocols, and the action representing the policy decision; and the internal function that uses the created `Config` objects to push actual configurations (e.g. OpenFlow rules) into the selected network nodes.

```
1  rule "disable-insecure-protocols"
2    when
3      $n: Node( type == type.NETDEVICE);
4    then
5      protocols = new ArrayList<Integer>(
6                  Arrays.asList(20,21,23));
7      cfg = new Config($n, protocols,
8            PolicyAction.BLOCK);
9      ConfigPusher.push(cfg);
10 end
```

Fig. 5. Example Drools code that generates configurations for network devices in working memory

While the code in Fig. 5 hides a significant portion of what is typically found in software written using traditional programming languages, there are several elements such as symbols, keywords, annotations, data structures, etc. that are never found in human sentences; therefore, readable statements are hard to construct using built-in BRMS syntax. In contrast, POLANCO was intentionally designed to severely restrict the token namespace that network administrators may use to write human-readable policies. Yet, POLANCO is translatable to BRMS-compatible machine code.

Once a network operator writes a POLANCO statement s, s is divided into groups of words $w_0, ..., w_n$ where each w_i is passed as parameter to a translation function $T(w)$ that generates valid BRMS code. Table III shows five w inputs and their corresponding BRMS code. While the first two groups of words (i.e. *policy* and *policy priority*) are simple word replacements, the rest of the groups produce more complex

BRMS code (that is hidden from POLANCO users).

TABLE III
TRANSLATION FUNCTION $T(w)$

POLANCO Grammar w	**Drools Code** $T(w)$
`policy`	`rule`
`policy priority`	`salience`
`[Nn]ode is a {type}`	`$n: NetDevice( $labels: labels contains "{type!uc}");`
`{action} {param} traffic`	`policies.add( CfgGen.fromNode( $n, PolicyAction.{action!uc}, {param},`
`from {src} to {dst}`	`aliasEvaluator.eval("{src}"), aliasEvaluator.eval("{dst}"); CfgPush.push(policies);`

A. Network Information Gathering

BRMS code has the potential to enforce a large number of policies. However its effectiveness largely depends on the information that is fed up into the working memory of the system. For that reason it is important to provide mechanisms that gather relevant network information such as the role a node has in the network (e.g. firewall, L3 router, L2 switch, printer, etc.), the enforcement mechanisms every node supports (e.g. OpenFlow versions, iptables, remote CLI commands, NETCONF), the type of node (e.g. network device, end system), the status (e.g. infected, quarantined, up, down), and the way nodes are connected with each other (e.g. link capacities, VLAN information), in order to guarantee an accurate translation of each POLANCO statement.

We allow for two levels of network information to improve the expressiveness and precision of a POLANCO policy. First, network operators must agree and associate every low-level identifier to a high-level description. The association is what enables the construction of precise human-readable network policy statements and is done in a static file that we refer to as `alias` file (Fig. 6 shows an excerpt of an alias file). For example, MAC and IP addresses can identify individual users, subnets could identify groups of users, VLAN and port numbers could represent types of traffic. Note that these associations have already been made when operating a campus network. However, hardly ever the meaning of low-level identifiers is used in AUPs.

Currently, there are two main components per item, namely, an *alias* (or a *traffic type*) which is the actual word that is to be used in POLANCO statements, and a list of *specifications (specs)* that contain information regarding the low-level identifiers associated with a given alias. The code generated by the transformation function $T(w)$ maps the aliases back to the corresponding low-level identifiers. In addition to the alias file, we rely on state-of-the-art topology discovery capabilites of SDN controllers that use protocols such as LLDP, BDDP, SNMP to gather router/switch information, and packet inspection (e.g. ARP, DHCP, ND) to discover end-host information. The topology discovery features can serve as an *initial topology sketch* comprised of OpenFlow-enabled devices, the connections between them, and the connections to attached devices. Unfortunately, network discovery in SDN oftentimes only distinguishes between two types of nodes, OpenFlow switches and end systems, when networks actually consist of many more systems. The `alias` file described above could be used to add properties to the discovered nodes. For example, hosts discovered in the IP address range $123.100.22.0/27$ can be marked as "netlab" machines. In addition, operators may use traditional network management protocols (e.g. SNMP) to extract information from network devices (both OpenFlow and non-OpenFlow) and add it as properties to specific nodes in the discovered topology. Overall, the network information gathering can be further enhanced by dedicated discovery systems that could edit portions of the alias file (thereby becoming readily available to POLANCO statements). However such systems are out of the scope of this paper.

```
...
alias: netlab-net
specs:
  - ip: 123.100.22.0/27
---
alias: campus network
specs:
  - ip: 123.100.0.0/16
---
traffic: web
specs:
  - port:
    - protocol: tcp
    - number: 80
  - port:
    - protocol: tcp
    - number: 443
---
traffic: ftp
specs:
  - port:
    - protocol: tcp
    - number: 21
  - port:
    - protocol: tcp
    - number: 20
---
traffic: telnet
specs:
  - port:
    - protocol: tcp
    - number: 23
---
traffic: applications which transmit
         information in cleartext
specs:
  - *ftp
  - *telnet
...
```

Fig. 6. Excerpt from an example alias file

V. Prototype Implementation

This section shows examples on how to use POLANCO to write simple, human-readable statements that implement high-level imprecise policies found on several academic institutions websites. All the presented examples assume there is an associated alias file that defines the port numbers of certain types of traffic (e.g. DHCP, DNS, FTP, or HTTP traffic) as well as IP addresses of known end systems (e.g. printers, DHCP servers). Due to space constraints, the alias file is not shown. In most cases, the file is rather straightforward to define (see Fig. 6 for an example alias file).

A. Disabling Insecure Protocols

Recall *P*2 in Table I whose POLANCO statement is "*block applications which transmit information in cleartext*".

Note that the conditional part of both POLANCO statements is omitted because the policy must be enforced in all network devices on the campus network which is the default behavior. Fig. 7 shows a network where the policy is enforced at various places, namely, a firewall, a router, and two switches. The definitions in the alias file we presented in Fig. 6 cause the BRMS to make requests to the SDN controller that would generate OpenFlow rules dropping any incoming packets from any interface whose destination is any of FTP control and data ports (20 and 21) and the telnet port (23).

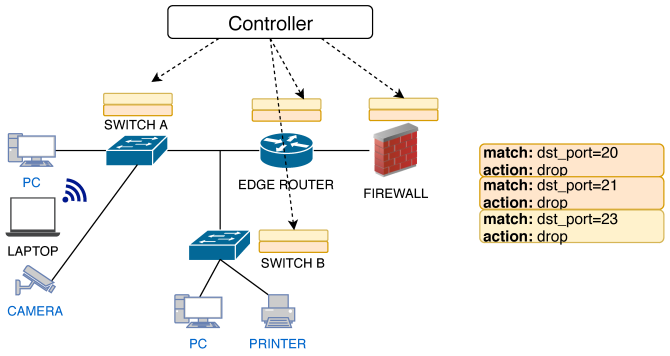

Fig. 7. Enforcing a campus-wide policy that disables insecure protocols

The network configurations shown are in the form of OpenFlow version 1.3 rules. However, during the network information gathering phase each network could have been assigned a different mechanism for policy enforcement (e.g. a different version of OpenFlow, NETCONF/Yang, iptables, remote SSH commands, etc).

B. Securing Network Printers

Most printers come with default configurations allowing users to use them out-of-the-box once plugged into the network. Carelessly plugging network printers into an enterprise network poses various risks because multiple unnecessary services are enabled and printers can be accessed from outside the network if they mistakenly get a public IP assigned.

Fig. 8 shows a printer policy found on the University of California–Berkeley's website [10] addressing these concerns.

```
To secure your printers from unauthorized
access, print configuration alterations,
eavesdropping, and device compromise follow
these printer security best practices:

- Campus printers should not be exposed to
the public Internet.
- Use encrypted connections when accessing
the printers administrative control panel.
- Do not run unnecessary services.
```

Fig. 8. Printer policy of the University of California–Berkeley

Network operators can write POLANCO statements that enforce the practices suggested in the policy in the following way: First, if every end system with a publicly reachable IP is labeled 'PUBLIC' and there is an inventory with the MAC addresses of authorized printers labeled 'PRINTER', then these labels could be included in POLANCO statements to block traffic to/from a misconfigured printer. Second, if IT administrators learn that a printer's control panel is accessible via web, POLANCO statements can use the corresponding alias for HTTPS traffic to represent secure access to control panel. Moreover, we can use the allow-only action keyword to ensure that access to unnecessary services is denied. If the BRMS code cannot directly push a configuration to an end-system, the closest network node connected to the selected device is used as the place to enforce the policy.

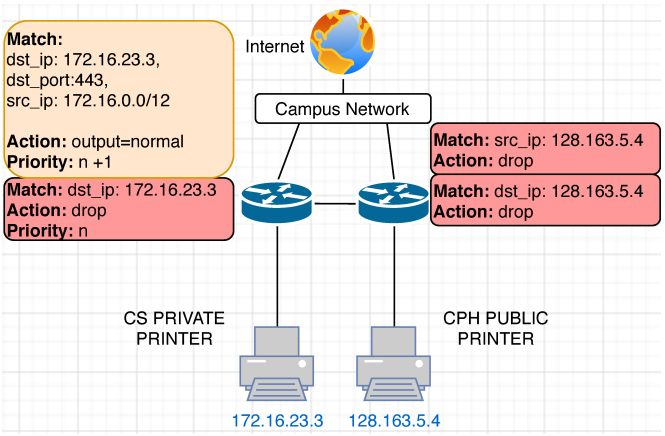

Fig. 9. Three printers in the network with their IP assignments

Fig. 9 shows the topology used in this example and Fig. 10 shows the POLANCO statements used to generate network configs.

The relevant portion of the topology consists of two printers that were (mis)configured and labeled during the topology discovery phase. Each printer has three labels representing the department it belongs to, the type of address assigned and the type of end system. For the Computer Science (CS) printer, two types of rules are added, one that drops traffic to all unnecessary services, and one that explicitly allows

HTTPS traffic to local users. Although not shown, note that an equivalent set of rules should be installed for the reverse direction and any other network range considered 'local' (e.g. 10.0.0.0/8, 192.168.0.0/16). For instance, for the the College of Public Health (CPH) printer that got assigned a public IP, traffic in both directions is blocked.

```
1
2  when node is a PUBLIC PRINTER
3  then block all traffic
4
5  when node is a PRIVATE PRINTER
6  then allow-only secure-web traffic
7         from local addresses
```

Fig. 10. POLANCO statements securing printers from external access

C. Firewall for External Connections

Firewalls are often the first line of defense of any network, including campus networks. It is not uncommon to see policies and guidelines for network traffic that is destined to/from the Internet. Consider an excerpt from a policy involving the perimeter firewall at the University of Missouri-St. Louis [4] shown in Fig. 11.

```
1  All UMSL network traffic to and from the
2  Internet must go through the firewall.
3  Any network traffic going around the
4  firewall must be accounted for and
5  explicitly allowed by the Computer
6  Security Incident Response Team (CSIRT).
```

Fig. 11. UMSL Firewall Policy

Enforcing the core of the policy (line 1) is straightforward in POLANCO. Assume the topology discovered during the network information gathering is the one shown in Fig. 12. There are switches *inside* the network that send traffic out of the network and switches *outside* the campus network that forward data into the network.

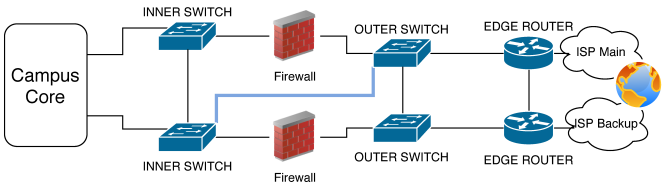

Fig. 12. Example topology discovered at the edge of a campus network

The POLANCO statements that enforce the policy are shown in Fig. 13. Both POLANCO statements would subsequently be translated into the appropriate network configurations for both `INNER` and `OUTER` switches. The translation process identifies information such as the designated interface where packets must be forwarded to, and the IP addresses the rules need to match on (e.g. the campus network address range). Moreover, the blue link connecting the upper `OUTER SWITCH` with the lower `INNER SWITCH` offers an alternative path to traditional routing protocols (e.g. OSPF, BGP) that bypasses the firewalls. The POLANCO statements force all traffic to avoid the alternative route and appropriately send all traffic through the firewall.

Though not shown in the figure, note that network operators can use a *policy priority* in the POLANCO statements to explicitly allow exceptions to the policy and allow the usage of the path that bypasses the firewall. We described the details of an exception system in [7], [8].

D. Rogue Servers

POLANCO can enforce policies that forbid the deployment of rogue servers—a system that is providing services to the network that IT staff is not managing. Take for example policy $P3$ found at the Oberlin College and Conservatory.

The key to enforce the policy is to distinguish between authorized servers and regular hosts using two labels. By distinguishing servers from hosts it is possible to block DNS (or any other traffic that manages IP address like DHCP) traffic destined to the latter. Note that it does not suffice to solely block all DNS packets to enforce the policy because legitimate end systems would be unable to resolve names. Instead, the BRMS should produce configurations that only allow responses issued by authorized servers and block messages issued by any other device (i.e. a rogue server). The POLANCO statements are presented in Fig. 14.

Fig. 15 shows the translation of POLANCO statements into OpenFlow rules. First, the BRMS selects all the network devices that are connected to a `REGULAR-HOST` node (i.e. `SWITCH A` and `SWITCH C`). Then, the `allow-only` action of the POLANCO statements (lines 1 and 4) produces two OpenFlow rules per selected switch. A similar approach could be used for other types of servers such as DHCP. Specifically, a rule with priority n drops *all* DNS response traffic, thereby preventing messages originating from rogue servers from reaching end systems; and another rule (with higher priority, say, $n+1$) that explicitly allows response traffic coming from authorized servers to reach end systems for legitimate name resolutions.

```
1  when node is an INNER SWITCH
2  then send to PERIMETER FIREWALL traffic
3       from campus network to Internet
4
5  when node is an OUTER SWITCH
6  then send to PERIMETER FIREWALL traffic
7       from Internet to campus network
```

Fig. 13. POLANCO statements enforcing a firewall policy

```
when node is connected to a REGULAR-HOST
then allow-only DNS-response traffic from
     authorized DNS server
```

Fig. 14. POLANCO statements prohibiting traffic from rogue servers

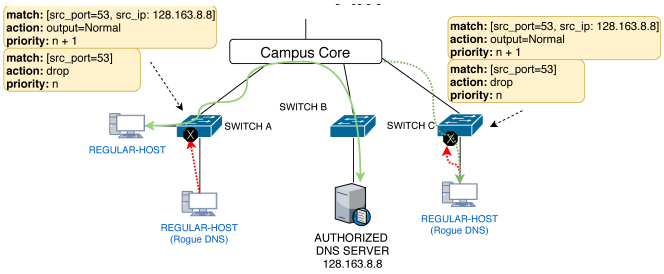

Fig. 15. Rules installed to prevent rogue servers

VI. Related Work

Firewalls are arguably the most well-established technology for enforcing network policies, with a focus on protecting networks from unauthorized access [11]. The types of network policies that can be specified by the firewalls are very limited and typically at low-level (IP addresses, port numbers, etc). On-demand security exceptions [7], [8] takes advantages of programmability provided by SDN networks and allows trusted users to specify security exceptions for trusted flows, such as big data transfer, to improve the throughput of these flows on a campus network. These policies of security exception can be dynamically requested and implemented on demand, and thus greatly improve the flexibility.

Policy Graph Abstraction (PGA) [12] allows network operators from various units in a campus network specify policies simultaneously using network graphs. Although we share PGA's goal to automate the way network operators translate high-level policies into low-level network configuration commands, PGA and our work address the problem from two perspectives. PGA resembles diagrams network operators draw when designing policies, while our work focuses on the definition of human-readable and technically-precise statements derived from AUP that use imprecise language.

Closely related to our work is OpenSec [13], an OpenFlow-based framework where network operators can specify security policies in a *human-readable language*. Although OpenSec's language is more readable than what could normally be written using network programming languages, we argue that POLANCO provides better human-readability because OpenSec's language still uses low-level identifiers (e.g., VLAN numbers, port numbers), does not resemble human-readable sentences found in AUP, and only focus on the packet processing done by middle-boxes and does not consider policies that are embedded in configuration files of end systems.

VII. Conclusions

To bridge the gap between human readable network policies and the enforcement of them via low level network configurations and/or SDN rules, we introduce the POLANCO language. POLANCO approximates natural language, but yet is technically precise, capable of being translated into SDN rules and actions that can automatically enforce the high level policies. POLANCO leverages a BRMS-based translation system that can observe and measure the changes in the networks and dynamically adapt the SDN rules in switches to enforce network policies when the network state changes. We demonstrated the expressiveness of POLANCO by showing that it can be used to expressed a variety of network policies found on University websites.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[2] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network Configuration Protocol (NETCONF)," Internet Requests for Comments, RFC Editor, RFC 6241, June 2011. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6241.txt

[3] Carnegie Mellon University, "Web Server Security Guidelines," https://www.cmu.edu/iso/governance/guidelines/web-server.html, 2014.

[4] University of Missouri-St. Louis, "Information Technology Network Usage Policy," https://www.umsl.edu/technology/networking/networkpolicy.html, 2019.

[5] Oberlin College and Conservatory, "Network Policy," https://www.oberlin.edu/cit/policies/network-policy, 2019.

[6] Weber State University, "Network Security/Firewall Policy," https://www.weber.edu/ppm/Policies/10-3_NetworkSecurity.html, 2007.

[7] J. Griffioen, Z. Fei, P. S. Rivera, J. Chappell, M. Hayashida, P. Shi, C. Carpenter, Y. Song, B. Chitre, H. Nasir, and K. L. Calvert, "Leveraging SDN to Enable Short-Term On-Demand Security Exceptions," *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 13–18, 2019.

[8] J. Griffioen, K. Calvert, Z. Fei, S. Rivera, J. Chappell, M. Hayashida, C. Carpenter, Y. Song, and H. Nasir, "VIP Lanes: High-speed Custom Communication Paths for Authorized Flows," in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2017, pp. 1–9.

[9] H. Barringer, K. Havelund, D. Rydeheard, and A. Groce, "Rule Systems for Runtime Verification: A Short Tutorial," in *Runtime Verification*, S. Bensalem and D. A. Peled, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–24.

[10] Berkeley Information Security Office, "Network Printer Security Best Practices," https://security.berkeley.edu/education-awareness/best-practices-how-articles/system-application-security/network-printer-security, 2019.

[11] Palo Alto Networks, "Palo Alto Networks Next Generation Firewalls," https://https://www.paloalto-firewalls.com/, 2019.

[12] C. Prakash, J. Lee, Y. Turner, J.-M. Kang, A. Akella, S. Banerjee, C. Clark, Y. Ma, P. Sharma, and Y. Zhang, "PGA: Using Graphs to Express and Automatically Reconcile Network Policies," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 29–42.

[13] A. Lara and B. Ramamurthy, "OpenSec: Policy-Based Security Using Software-Defined Networking," *IEEE Transactions on Network and Service Management*, vol. 13, no. 1, pp. 30–42, March 2016.