Enforcing Multilevel Security Policies in Unstable Networks

Quinn Burke, *Student Member, IEEE*, Fidan Mehmeti, *Member, IEEE*, Rahul George, Kyle Ostrowski, Trent Jaeger, *Member, IEEE*, Thomas F. La Porta, *Fellow, IEEE*, and Patrick McDaniel, *Fellow, IEEE*

***Abstract*—Multilevel security (MLS) systems control access to data by formalizing permissible and impermissible information flows between data sources and destinations (e.g., database servers and clients) fixed with distinct security labels. In computer networks, MLS systems have been used to prevent unauthorized data disclosure in shared-infrastructure settings where network hosts and devices may fall within different trust domains (e.g., in multi-tenant cloud networks or wireless mesh networks). However, current MLS systems assume static network behavior—thus preventing the network from being practically usable in the presence of dynamic network events that frequent unstable network environments, including sudden changes in traffic patterns, link failures, and topology changes as a result of device movement or intermittent device connectivity. In this paper, we introduce `MLS-Enforcer`, a software-defined networking (SDN) controller application that can efficiently deploy network-level MLS policies while retaining the ability to securely relabel network nodes under changing topology state and network traffic demands. We model network adaptivity as an integer linear programming problem that reflects a given security policy. We then introduce heuristic relabeling algorithms that achieve near-optimal performance and are more tractable and efficient for larger networks. We validate `MLS-Enforcer` on several network topologies and traffic loads, demonstrating that it can relabel the network to route 90%+ of flows under normal conditions and quickly converge (on the order of seconds for the heuristic algorithms) under changing needs—from small network structure changes to catastrophic failures. This shows that formally secured networks can feasibly be deployed in diverse, changing, and unpredictable environments.**

***Index Terms*—Software-defined networking, SDN, security services, security management, wireless network security, multilevel security, optimization.**

Manuscript received 1 October 2021; revised 12 March 2022; accepted 16 May 2022. Date of publication 23 May 2022; date of current version 12 October 2022. This research was sponsored by the Combat Capabilities Development Command Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA). This work was also supported in part by the National Science Foundation under award CNS-1946022. The associate editor coordinating the review of this article and approving it for publication was F. Valenza. *(Corresponding author: Quinn Burke.)*

Quinn Burke, Rahul George, Kyle Ostrowski, Trent Jaeger, Thomas F. La Porta, and Patrick McDaniel are with the Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16802 USA (e-mail: qkb5007@psu.edu; rtg64@psu.edu; kto5055@psu.edu; trj1@psu.edu; tfl12@psu.edu; mcdaniel@cse.psu.edu).

Fidan Mehmeti is with the Chair of Communication Networks, Technical University of Munich, 80333 Munich, Germany (e-mail: fidan.mehmeti@tum.de).

Digital Object Identifier 10.1109/TNSM.2022.3176820

I. Introduction

MULTILEVEL security (MLS) systems control access to data through a reference monitor that governs access requests made on data sources. The reference monitor uses security labels and a security policy to formalize permissible and impermissible information flows between data sources and destinations (e.g., database servers and clients). The formalization is particularly useful in computer networks operating under a shared-infrastructure model where tenants share the underlying physical hosts and network devices, but fall within different trust domains— for example, in multi-tenant cloud/enterprise networks [1] or multi-organization wireless mesh networks [2]. As such, MLS systems have become essential components in network routing [3] to protect data between network-service endpoints [4], [5] and to isolate the traffic between different cloud tenants [6].

Software-defined networks (SDNs) have eased the implementation of MLS-based network routing systems by allowing them to run as SDN controller applications. Here, the controller application computes secure network-flow routes and manages network-switch flow tables via a northbound interface to the SDN controller (typically, a REST API). Yet, the current design of MLS systems is limited in that it assumes a fixed set of security labels on network hosts and devices, which leads to under-utilization and sometimes (under inflexible security policies) a failure to route flows—i.e., it achieves low flow *coverage*. This prevents the network from being practically usable in the presence of dynamic network events that frequent unstable network environments (e.g., wireless mesh networks), including sudden changes in traffic patterns, link failures, and topology changes as a result of device movement or intermittent device connectivity [2].

In this paper, we introduce `MLS-Enforcer`, an SDN controller application that routes flows securely under MLS policies and dynamically adjusts network-switch security labels when necessary to improve flow coverage. It therefore allows security policies to be fluidly configured and network-flow routes to be changed in response to evolving traffic and topology profiles—all while providing the service transparently to the entire network. We approach the problem by formulating integer linear-programs (ILP) that reflect MLS security policies that preserve the confidentiality of information flows. We then introduce heuristic relabeling algorithms that achieve near-optimal performance and are tractable and efficient.

As relabeling network switches is an online problem, several unique challenges arise. First, relabeling may affect flows with

already-established routes, and thus relabeling should impose minimal disruption on them. Additionally, changing switch labels requires a secure procedure for wiping and rebooting a switch (to reset the device into a known/trusted/attestable state), which takes the time that is otherwise used for packet forwarding; therefore, the relabeling algorithms must manage invoking switch reboots and distributing flow rules efficiently.[1] Further, policies that allow flows of different labels to traverse common switches (i.e., *flow mingling*) introduce the potential for side-channel attacks (e.g., traffic analysis) by hosts using the switch; therefore, it is essential to assess the tradeoff between increased flow coverage and allowing flow mingling (along with any risk that it introduces). Lastly, it is important that the network maintains near-optimal or optimal coverage of flows (i.e., the number of flows that can be routed) in the presence of dynamic network events such as sudden changes in traffic patterns, link failures, and topology changes as a result of device movement or intermittent device connectivity.

As link failures and device movement are conventionally wireless network problems, we focus our evaluation of `MLS-Enforcer` on common wireless network topologies [7]: mesh and star networks. We also measure the performance of the labeling process in more traditional wired (fat-tree) networks for juxtaposition. We focus on three metrics: coverage, agility, and disruption. A coverage analysis shows that we can achieve high flow coverage for all tested networks and policies: experiments in a mesh network show that the system can achieve 99% coverage using an optimization solver, and 95% using a heuristic algorithm. With respect to agility and disruption, experiments show that the heuristic algorithms are responsive to significant network events, achieving near-optimal coverage within ≈5 mins. The relabeling process completes with minimal and controllable disruption to ongoing flows (where up to 30% of flows may be queued temporarily due to switch reboots). Lastly, computing routes with the optimization solver requires up to 5 minutes to relabel networks of just 48 switches, and thus, we resort to heuristics that achieve 90% of the optimal flow coverage with a 300× reduction in computational overhead.

We contribute the following:

- We formalize dynamic relabeling as an integer linear program, parameterized by the access-control constraints of a given security policy.
- We design heuristic algorithms for dynamic relabeling that achieve near-optimal performance and scale more efficiently with larger networks.
- We demonstrate the feasibility in deploying MLS policies in unstable networks through a comprehensive experimental evaluation.

II. Background

In this section, we discuss background on software-defined networking and formalize multilevel security as it relates to computer networks. We then highlight the gaps in prior work that motivate the design of `MLS-Enforcer`.

A. Multilevel Security

Multilevel security (MLS) systems provide access control over data by assigning *security labels* to subjects (e.g., network hosts/IP addresses) and objects (e.g., database tables) and validating that access constraints are satisfied whenever an access to an object is requested by a subject [8]. A security label consists of both a *security level* and one or more *security categories*. A security level is an hierarchical attribute that indicates the relative authorization power (resp. sensitivity) of a subject (resp. object)—for example, public, confidential, secret, or top secret clearance. Security categories are non-ordered attributes that identify classes of data—for example, financial, medical, or personal files. A subject's label can then be defined, for example, as $L_{subject} = \{secret, \{financial, medical\}\}$. The hierarchy of labels formed by all combinations of levels and categories form a lattice structure called the security lattice [9].

A label is lesser (or greater) than another if the former is a lesser level (or a greater level) and/or its categories are a proper subset (or a proper superset) of the latter's—otherwise labels are incomparable (and an access is denied). We will associate the comparators "lesser" and "greater" with the $<$ and $>$ symbols, respectively. The security lattice ordering thus describes the access constraints that must be satisfied to maintain confidentiality. As an example, an MLS system may require that data only flow between equivalent labels: for an object of label $L_o = \{secret, \{financial\}\}$ and subject of label $L_s = \{public, \{financial\}\}$, all data flows $L_o \rightarrow L_s$ are denied since *public* $<$ *secret*. Here, the less-than sign indicates that the *Public* subject has a lower security level than the *Secret* object.

B. Role of MLS Policies in Networks

Traditionally, multilevel security systems were used to control access to databases and operating systems, by making different data available or presenting data differently to users of different clearances [10]. For example, a database server in a military or industrial organization may be shared among users in both the accounting and engineering departments with complete mediation over accesses to prevent unauthorized data disclosure between users in each department [11]. However, MLS policies have also be deployed in computer networks to provide access controls between network service endpoints that produce/consume data for each other [4], [5] and to isolate traffic between different cloud tenants [6].

The difficulty in deploying an MLS system in a network stems from the fact that there are multiple hops between the source of data and its recipient, and thus information is inherently exposed to intermediate subjects (e.g., an ethernet switch, or a forwarding node in a wireless mesh network), which may become compromised [12]. Here, security constraints must also be satisfied across the entire path of each network flow (i.e., at each intermediate switch) to meet the security policy. For example, such is the case for isolated, wireless military

[1]Note that other methods for attesting the state of the running switch software may be used (perhaps without rebooting). However, any choice of which will inevitably induce some delay (which, in our evaluation, we simulate with sleeps to temporarily suspend switch processing).

networks where resource-limited network nodes are particularly susceptible to attacks [13]. This task has made feasible by leveraging a software-defined networking (SDN) architecture [14], [15]: the decoupling of the network control and data planes [16] provides an opportunity to run an MLS system as a network application on the SDN controller with complete visibility over the network topology and traffic profile. In this setting, the MLS service can be provided transparently to the entire network; whenever a new flow arrives into the network, by default, if there is no route to forward it, that flow is forwarded to the SDN controller be route it (assuming a reactive approach to flow rule installation). There, the access constraints can be evaluated over the flow source and destination to determine if the flow is permitted and, if so, find an appropriate (secure) path through the network.

MLS offers two benefits unique to network security: (1) fine-grained isolation of network traffic flows between different trust domains (i.e., security levels) all using a shared infrastructure (e.g., in a wired datacenter network or wireless mesh network), and (2) a reduced threat surface for adversaries within a particular trust domain. Isolation is enabled by ensuring network flows are routed through paths in the network deemed secure (i.e., satisfy security constraints)—thus ensuring adversaries cannot probe, eavesdrop, or otherwise interact with network hosts or devices outside of their trust domain [17]. A reduced threat surface is achieved by leveraging security categories: they further enforce the principle of least privilege on access to data, thus preventing unrestricted lateral movement (e.g., network scanning and traffic analysis) by potential adversaries [18]. In effect, MLS policies can prevent entire classes of reconnaissance techniques: inter-domain (between trust domains) host scans, port scans, and vulnerability scans [17] can all be immediately dropped at access switches if the source host is not of appropriate security level. This simultaneously prevents data exfiltration, even if some nodes along a flow path were to become compromised, since data will not be leaked from uncompromised network hosts or devices of greater security levels to lower ones (e.g., from Top-secret hosts to Public hosts). Moreover, MLS policies can mitigate intra-domain scanning by restricting access with security categories, and can mitigate denial-of-service attacks as priorities can be given to certain security labels to ensure those flows have sufficient bandwidth.

As an illustrative example, consider the simple lattice, composed of just security levels, shown at the top of Fig. 1. An MLS policy can enforce that a *Secret* flow, originating from the *Secret* laptop-user, only traverses *Secret* switches toward a *Secret* server, preventing any *Public* nodes from being able to eavesdrop on the flow. Without an MLS policy, encryption alone may suffice to prevent a *Public* node from directly accessing the *Secret*-flow data, but cannot prevent traffic analysis by the (less-secure) node were it to become compromised [19]. While end-to-end encryption used together with an MLS policy can provide greater security, device resource constraints may limit when encryption can be an available option, whereas the MLS policy provides strong security guarantees alone.

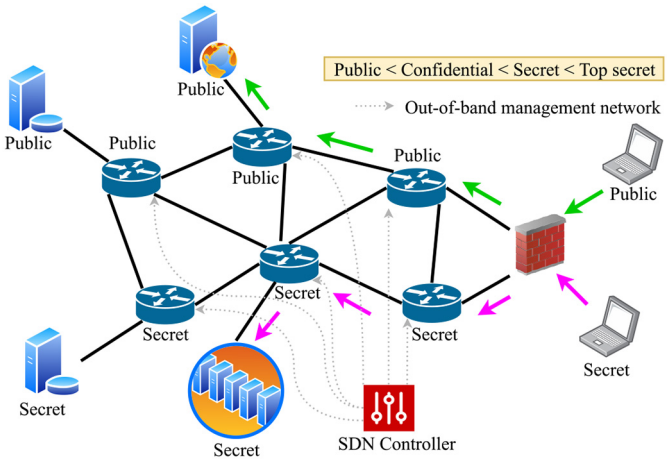


Fig. 1. Network scenario using a 4-level security lattice, with security labels (logically) given to network switches, servers, and users by the SDN controller.

III. MLS-Enforcer Overview

In this section, we detail the network and security model that `MLS-Enforcer` operates under, discuss the two MLS policies that we consider in our design and evaluation, and provide an illustrative example of the relabeling process.

A. Network and Threat Model

As shown in Fig. 1, the system is composed of nodes (e.g., user device, server, or network switch), links, and an SDN controller that orchestrates the network. In `MLS-Enforcer`, nodes (with security label L_{Node}) are subjects and network flows (with security label L_{Flow}) are data objects. A given flow is labeled with the security label of the source host/server, i.e., $L_{Flow} = L_{Source}$. Moreover, links between nodes can be wired or wireless; however, as link failures and device movement are conventionally wireless network problems, `MLS-Enforcer` may have larger benefits in wireless environments.

We leverage an SDN architecture to allow a network administrator to logically assign labels to each network node: since the SDN controller is topology-aware, we maintain the label assignments at the controller and thus provide the service transparently to the entire network. We assume that a set of labels for *endpoints* deemed appropriate per the needs of the organization is given as input to `MLS-Enforcer`, and the initial *switch* labels can be random (as they may be changed). In other words, we assume that a network administrator assigns the endpoint security labels based on a relative security assessment of each device and what traffic classes the device is intended to send [14]. For example, wireless devices (e.g., laptops) with unpatched software may be considered relatively insecure and assigned a lesser security label when they connect to the network, while wired workstations with up-to-date software and used within an office building for secure tasks may be considered relatively secure and given higher security labels when they are connected to the network. A method for choosing device labels is outside the scope of our work.

Then, for simplicity we consider all of the network flows being emitted from the host endpoints as having the same

security label as the endpoint. Note that in future work we will consider host endpoints being able to emit traffic with different security labels (e.g., to differentiate between Web browsing and secure file transfer traffic).

In SDNs, as new flows arrive into the network, by default if there is no route to forward them, they are forwarded to the controller's routing application to be routed (assuming a reactive approach to flow rule installation). `MLS-Enforcer` is then designated as the routing application (typically Java- or Python-based under the widely used OpenDaylight [20] and Frenetic [21] SDN controllers), and it computes secure network-flow routes and manages network-switch flow tables via a northbound interface to the SDN controller (typically, a REST API). Upon receipt of a new flow, `MLS-Enforcer` therefore intercepts the request and: (1) identifies the security label of the flow based on the source IP address, (2) checks if the data source is permitted to send data to the destination, and (3) potentially relabels some switches before computing a path for the flow where the MLS policy constraints are satisfied between the flow's label and the label of each node along the path.

In our threat model, we assume a trusted SDN controller (and trusted administrator of the shared infrastructure) that makes labeling and routing decisions and monitors for conditions that require relabeling. We allow for compromised network hosts or switches in different trust domains: different tenants in a cloud network, or different organizations sharing a wireless mesh infrastructure, may attempt to probe other hosts and switches, eavesdrop on communication, or engage in isolated or coordinated link cutting attacks [22].

B. Relabeling Process

`MLS-Enforcer` can route flows in one of two modes: with relabeling enabled, or with relabeling disabled (by configuring an algorithm parameter, as discussed later). In this way, relabeling can be manually or periodically enabled, or enabled in response to administrator-specified trigger conditions (e.g., a link-failure event) to reduce disruption imposed on the network.

Over time, events that change the traffic profile or the structure of the network (such as link failures) may interrupt flows and require new routing paths to be found for them. Upon detection by the SDN controller, `MLS-Enforcer`'s routing algorithms adapt by (1) potentially changing some switch labels (if relabeling is enabled), (2) invoking switch reboots to reset the switch software into a known/trusted/attestable state, and (3) recomputing flow routes that meet the security policy constraints under the new set of labels. This can be seen in Fig. 2 by observing the label changes from the top part of the figure to the bottom as a result of a detected link failure. The new routes are then distributed to switches as flow rules to realize the new routing configuration.

C. MLS Policies

As the relabeling process must align with the security policies, we focus our formulations on two security policies that maintain confidentiality for network flows: *Strict* and *Relaxed Bell-LaPadula*. The strict security policy enforces total isolation among labels which is typical of current MLS networks. Here, information may only flow between subjects and objects of equivalent security labels. As described below and explored experimentally, the strict policy may overly constrain the routing path options, leading to under-utilization of network switches and an inability to route some classes of flows.

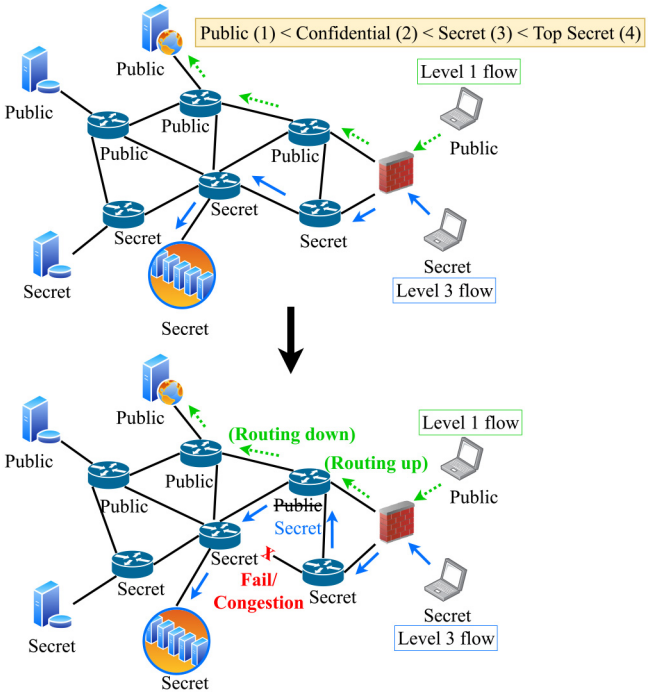


Fig. 2. Network scenario describing how relabeling operates.

The *Relaxed Bell-LaPadula* (R-BLP) enforces the canonical BLP policy [8] between the end-points of a flow (e.g., between source and destination hosts). In the canonical BLP model, a source cannot send information to a destination with a lesser security label than it has. However, BLP allows a source to send information to a destination with a greater security label. R-BLP retains the spirit of the BLP model but extends the model to networks in which there are intermediate nodes between the source and destination. In the R-BLP model, a source cannot send information to or *through* nodes with a lesser security label than it has, but a source can send information to or *through* nodes with a greater security label.

More formally, for all flows, a flow may only be routed through a switch if $L_{Switch} \geq L_{Flow}$, such that L_{Flow} serves as the *floor* for the security labels of switches through which a flow may be routed. However, aficionados of BLP may recognize that R-BLP technically allows a switch to route some flow to another switch of a lesser security label as long as that switch's label is greater than or equal to the flow's label. Thus, there exists the potential for leakage into potential side channels (e.g., the switch of lower security level may perform traffic analysis). In BLP, such switches would need to be trusted or such an action would be considered a violation of BLP. In R-BLP, we address this in two ways. First, the labeling scheme described below constrains the number of violations allowed in the network (to reduce potential leakage into

side channels). Second, there are techniques available to protect and validate the switches integrity (i.e., trustworthiness) and protect the secrecy of communications through encryption (see Section VI-F). We measure this effect in our evaluation, however, we leave an assessment of side-channel information leakage to future work.

D. Network Example

Consider the example network scenario in Fig. 2 using the security public/confidential/secret/top secret security lattice (top of figure) and enforcing a Relaxed Bell-LaPadula policy. The flows have a *Public* user sending a *Public* flow (green) toward a *Public* server and a Secret user sending a Secret flow (blue) toward a Secret server. Here, the routing algorithms can find secure paths that completely isolate the flows from one another (i.e., incidentally satisfying the strict policy). For example, any *Public* flows coming from the *Public* user can reach the *Public* servers using only the *Public* switches.

Now consider a link failure (bottom of Fig. 2). Under the initial labeling the Secret flows would be blocked since there would be no alternative secure path. Recognizing this condition, the controller (or some watchdog service) can enable relabeling. Then, a possible solution identified by the algorithm would be to *relabel* (and securely reboot) a single *Public* switch to the *Secret* level, providing a secure path for the Secret flows. However, the *Public* user would no longer have a route to a *Public* switch. To accommodate both flows, an option allowed by the R-BLP policy is to permit the *Public* flow to be routed through Secret switches (*"routing up"*). But, for the flow to be delivered to a *Public* server, it must be returned to its original level, i.e., the *Public* flow must eventually be delivered back down to a *Public* switch (*routing down*).

R-BLP utilizes such routing *up* and *down* as long as the switches all have a label greater than or equal to the flow's label. Our formulation restricts routing down to a limited number of switches to manage risks.

IV. Optimization Formulations

In this section, we formulate two integer-linear programs (ILPs) that reflect the two security policies introduced previously to preserve the confidentiality of information flows: the *strict* policy and the *Relaxed Bell-LaPadula (R-BLP)* policy. Note that for simplicity we consider security labels just in terms of security levels; the formulations can be easily extended to labels with both levels and categories by adding a category-based term to the access constraints. Further, the proposed framework can easily be extended with similar constraints to support organization-specific security policies. We elaborate on these points in Section IV-C.

We first define several variables used in both policies, and in Table I we provide the notation used throughout.

Definition 1: A link between switches k and l for a flow j is called a *feasible link* (denoted by $X^j_{k,l}$) and can be used to route the flow if those two switches are first-hop neighbors and the level relationship between the flow j and both switches satisfies the security policy.

TABLE I
Definitions and Notation

C_j	Capacity demand of flow j
α_j	Routing indicator of flow j
L_j	Security level of flow j
$f(L_j)$	importance function of flow j
$\mathcal{I}$	Set of all switches
$\mathcal{J}$	Set of all flows
$\mathcal{E}$	Set of all links
$\mathcal{S}$	Set of all flow sources
$\mathcal{D}$	Set of all flow destinations
$I_{k,l}$	Indicator of the neighborhood of switches k and l
$X^j_{k,l}$	flow j link indicator between switches k and l
$Y^j_{k,l}$	flow j decision variable between switches k and l
$C_{S,i}$	Capacity of switch i
$C_{k,l}$	Link capacity between switches k and l
B	Maximum number of route-downs allowed for any flow
X_i	Security level of switch i
m	Number of security levels
ΔX_i	Number of levels switch i is changed by
I_{Δ_i}	Indicator of changing the security level of switch i
M	Number of switches whose labels can be changed
Δ_j	Route-down degree limit allowed for flow j

The variable $I_{k,l}$ denotes whether two switches are neighbors:

$$I_{k,l} = \begin{cases} 1, & \text{if there is a direct link between } k \text{ and } l \\ 0, & \text{otherwise} \end{cases}$$

The security levels of flows (denoted by L_j) or switches (denoted by X_i) form a totally ordered set defined by:

$$L_j, X_i \in \{1, \ldots, m\}.$$

The updated security level of switch *i* is

$$X'_i = X_i + \Delta X_i.$$

Hence, the set of possible values of the change in switch levels from relabeling is

$$\Delta X_i \in \{-X_i + 1, \ldots, m - X_i\}.$$

The indicator variable denoting whether a switch's initial level was changed is

$$I_{\Delta_i} = \begin{cases} 1, & \text{if } |\Delta X_i| > 0, \\ 0, & \text{otherwise} \end{cases}$$

We introduce a tunable parameter, *M*, to allow a network administrator to limit the number of switches that may have their labels changed during a single run of the relabeling algorithm; this in turn can reduce potential disruption caused by waiting for switch reboots.

A. Labeling for the Strict Policy

In this policy, a flow can neither be routed through a switch of higher-security level nor through a switch of lower-security level. Hence, given this assumption, the network is partitioned into groups comprising the flows of the same security level. Then, the feasibility of a link (where the switches may have updated labels) being used for routing a flow is described by extending the traditional MLS constraint:

$$X^j_{k,l} = \begin{cases} 1, & \text{if } I_{k,l} = 1 \text{ and } L_j = X'_k = X'_l \\ 0, & \text{otherwise} \end{cases}$$

In other words, a link is feasible for a flow *iff* the two switches are of exactly the same security level as the flow. The optimization problem formulation is given by (1)–(12) below.

The objective (1) is to maximize the total capacity of the served flows in the network, weighing each flow according to its security level with $f(L_j)$, which can be an arbitrary function, such as a linear function, quadratic function, etc.[2] Constraint (2) ensures only a *feasible link* can be chosen for a flow. Basically, $Y^j_{k,l}$ can be 1 only if $X^j_{k,l} = 1$ (the link is feasible). Initiating $(\alpha_j = 1)$ or not initiating a flow $(\alpha_j = 0)$ from the source is described by (3), whereas (4) denotes the last link of a routed flow $(\alpha_j = 1)$ or the flow not being routed $(\alpha_j = 0)$. The flow preservation property (the flow can leave a node only if it has entered it) is captured by (5). Constraint (6) ensures there are no loops. The left-hand side of (7) denotes the total capacity demand of all flows going through switch *i*. It cannot be larger than the total link capacity (the right-hand side term of (7)). Similarly, the left-hand side term of (8) is the total capacity demand of flows traversing the link between switches *k* and *l*, which cannot be larger than the capacity of that link $(C_{k,l})$. Constraint (9) captures the finite number of switches whose security labels can be changed. Finally, constraints (10)-(12), denoting whether a flow is routed, whether it is routed through the link between switches *k* and *l*, and whether a switch level is changed, respectively, define the decision variables, whose values can be either 0 or 1.

$$\max \sum_{j=1}^{J} C_j \alpha_j f(L_j) \tag{1}$$

$$\text{s.t.} \quad Y^j_{k,l} \le X^j_{k,l}, \quad \forall k \in \mathcal{I} \cup \mathcal{S}, \forall l \in \mathcal{I} \cup \mathcal{D}, \tag{2}$$

$$\sum_k Y^j_{s_j,k} = \alpha_j, \quad \forall j \in \mathcal{J}, \forall k \in \mathcal{I}, \tag{3}$$

$$\sum_k Y^j_{k,d_j} = \alpha_j, \quad \forall j \in \mathcal{J}, \forall k \in \mathcal{I}, \tag{4}$$

$$\sum_k Y^j_{k,l} = \sum_m Y^j_{l,m}, \quad \forall k \in \mathcal{I} \cup \mathcal{S}, \forall l \in \mathcal{I}, \forall m \in \mathcal{I} \cup \mathcal{D}, \forall j \in \mathcal{J}, \tag{5}$$

$$\sum_k Y^j_{k,l} \le 1, \quad \forall k, l \in \mathcal{I}, \forall j \in \mathcal{J}, \tag{6}$$

$$\sum_k \sum_j C_j Y^j_{k,i} \le C_{S,i}, \quad \forall i \in \mathcal{I}, \forall k \in \mathcal{I} \cup \mathcal{S}, j \in \mathcal{J}, \tag{7}$$

$$\sum_j C_j Y^j_{k,l} \le C_{k,l}, \quad \forall k, l \in \mathcal{I}, \tag{8}$$

$$\sum_i I_{\Delta_i} \le M, \quad \forall i \in \mathcal{I}, \tag{9}$$

$$\alpha_j \in \{0,1\}, \quad \forall j \in \mathcal{J}, \tag{10}$$

$$Y^j_{k,l} \in \{0,1\}, \quad \forall k \in \mathcal{I} \cup \mathcal{S}, \forall l \in \mathcal{I} \cup \mathcal{D}, \forall j \in \mathcal{J}, \tag{11}$$

$$I_{\Delta_i} \in \{0,1\}, \quad \forall i \in \mathcal{I}. \tag{12}$$

[2]Note that whether or not a flow is served is controlled by the decision variable α_j, whose value is 1 only if flow *j* is served and otherwise 0.

Complexity: This optimization problem belongs to the class of integer linear programs, which are known to be NP-hard [23]. The problem structure does not allow for an algorithm with a performance guarantee. Hence, we resort to a heuristic algorithm suitable for large networks. Nevertheless, the heuristic algorithms that we present in Section V are demonstrated to provide near-optimal performance (close to that obtained by the optimization solver).

B. Labeling for the Relaxed Bell-LaPadula (R-BLP) Policy

With this policy, a flow may be destined toward higher-security level hosts or be routed up to higher-security level switches, but not to hosts/switches of a lower-security level than the flow itself. Therefore, we extend the above formulation to realize the relaxation on the canonical BLP policy.

Definition 2: A flow traversing any link for which the next-hop host (destination) or switch is of higher-security level than the flow is denoted as being *routed up* (see bottom of Fig. 2).

In allowing routing up, the policy must also potentially allow the flow to be routed down to lower-level switches again in order to reach the destination as long as $L_{Switch} \ge L_{Flow}$. In allowing routing down, we introduce two parameters that dictate to what degree routing down is permitted. The number of routing-downs along a flow route is restricted by the routing-down limit *B*.

The formulation for R-BLP is identical to the strict policy except for the link feasibility constraint which is expressed as:

$$X^j_{k,l} = \begin{cases} 1, & \text{if } I_{k,l} = 1 \text{ and} L_j \le \min\{X'_k, X'_l\} \\ & \text{and } X'_k - X'_l \le \Delta_j \\ 0, & \text{otherwise} \end{cases}$$

Essentially, besides requiring the two switches being neighbors, the flow level should be lesser or equal to both switch labels and the degree of a routing-down should be within the limit. From the definition of a feasible link, we constrain the number of security levels a flow can be routed down, which in turn limits the number of security levels it can be routed up.

Further, we introduce an additional constraint to realize the routing-down limit *B*:

$$\sum_k \sum_{l | X'_k > X'_l} Y^j_{k,l} \le B, \quad \forall k, l \in \mathcal{I}, \forall j \in \mathcal{J}. \tag{13}$$

Complexity: Since the R-BLP formulation (constrained optimization problem) has the same objective function as the strict policy and near-equivalent constraints, it is impossible to provide an approximation algorithm with performance guarantees in this case neither. Therefore, for this problem as well we propose low-complexity heuristic algorithms that achieve near-optimal performance (Section V).

C. Extending the `MLS-Enforcer` Framework

As noted, our formulations provide a framework for realizing dynamic deployment of the strict and R-BLP policies under a standard security lattice, but can be extended to support organization-specific security levels, categories, and policies. Here, we elaborate on how to achieve this.

For simplicity, in the optimization formulations we consider security labels just in terms of the security levels. The formulations can be extended to support security categories by *adding a category-based term to the link feasibility indicator variable* $X^j_{k,l}$. First, the security labels of flows (denoted by L_j) and switches (with updated levels denoted by X'_i) should be extended to represent both a level and set of categories (e.g., $L_j = \{secret, \{financial\}\}$). Then, for the strict policy, a link is considered feasible (i.e., may be used to route the flow) under the following condition: $X^j_{k,l} = 1 \iff (I_{k,l} = 1)$ and $(L_{j,level} = X'_{k,level} = X'_{l,level})$ and $(L_{j,categories} = X'_{k,categories} = X'_{l,categories})$. A similar extension can be applied for the R-BLP policy.

Note that our use of security levels as a representation of the entire label is just a special case under the use of categories, where labels with the same level also simply have the same set of categories. Naturally, both the space of levels and categories can quickly render the space of possible labels very sparse in certain organizations and thus make it difficult to find routes for flows (i.e., it may become difficult to relabel adequately to satisfy the access constraint for most flows). Therefore, we defer a more comprehensive investigation of security categories and the limitations they impose therein to future work.

Just as the formulations can be extended to support categories, any other organization-specific security policies can be implemented by similarly *extending the link feasibility indicator variable*. For example, a security policy may require rejecting flows being emitted after a certain time-of-day. To support this policy, the network administrator may introduce an additional indicator variable T_j for a flow timestamp (e.g., the time it was received at the controller application to be routed). Then, it may check newly arriving flows against a set threshold before permitting an access and installing the necessary flow rules to switches: $X^j_{k,l} = 1 \iff (I_{k,l} = 1)$ and $(L_j = X'_k = X'_l)$ and $(T_j < threshold)$.

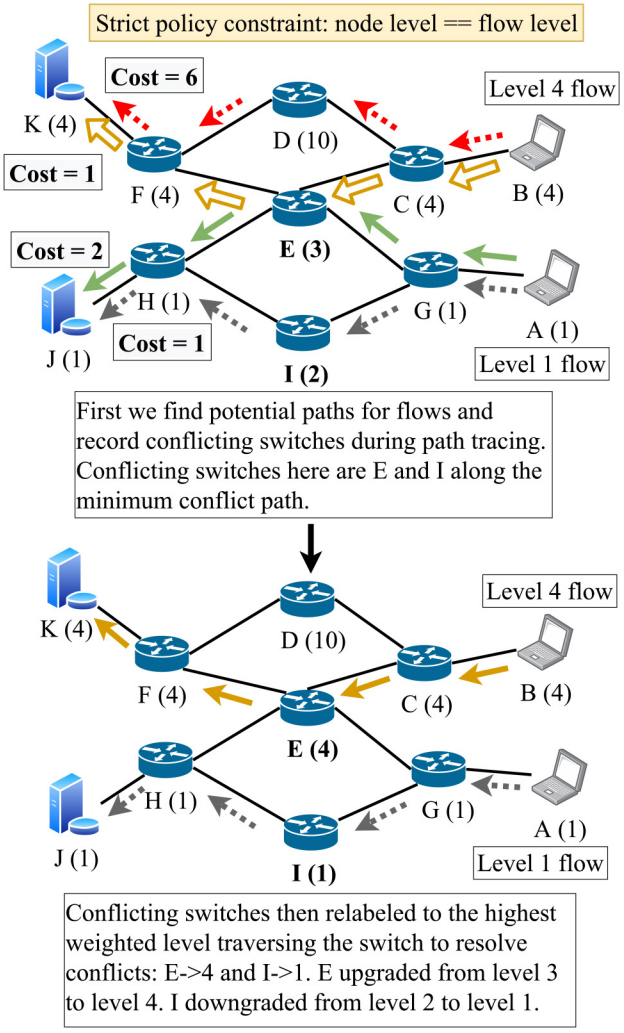


Fig. 3. Example execution of the heuristic for the strict policy. Minimum conflict paths are found for flows, and conflicting switches are recorded, and per-level weights are computed for flows traversing them. Then, conflicting-switch levels are set to the highest-weighting level to unblock flow paths.

V. Heuristic Algorithms

As solving the optimization problems for the given policies is NP-hard, we propose heuristic methods for each policy that are easily deployed in real networks. We approach the problem by dividing the relabeling process into two subroutines: (1) finding potential flow paths and recording conflicts on the paths, and (2) relabeling selected switches based on the information collected about their conflicts. In the first subroutine, we extend Dijkstra's shortest-path algorithm to find potential flow paths in the network and record per-flow *conflicting switches* during path tracing (i.e., those along a potential flow path that cause a policy violation). In the second subroutine, we use all of the conflict information to decide which switches (up to *M* switches) to relabel to resolve conflicts.

To accomplish this, we distinguish between two types of conflicts that a switch level may have with flow levels. *Then, the key idea is to invoke the subroutines iteratively through multiple phases to resolve each conflict type successively*. The strict policy consists of two phases and the R-BLP policy consists of three phases. For each successive phase, we enforce constraints on links that prevent the controller from continuously finding new paths containing conflict types already resolved. We first introduce the conflict types and then describe the relabeling process for both policies.

A. Conflict Types

Type (1) conflicts: This type of conflict may arise when a flow path contains a switch with a different security level than the flow. Marking switches differing in level as conflicting informs the relabeling subroutine of the potential need to upgrade or downgrade the switch level to reduce the switch's threat surface. Marking all of these cases as conflicts allows us to isolate different level flows as much as possible, reducing the possibility of flows of different levels mingling, and hence the threat of leakage into potential side-channels. Moreover, this method will encourage the controller to find similar

paths for similar-level flows to reduce the number of conflicting switches and limit any disruption caused by invoking relabels/reboots.

Type (2) conflicts: This type of conflict only applies to R-BLP and may arise when the flow path contains a link whose routing-down degree (after being routed up) between the ends is larger than that allowed by policy (Δ_j). In this scenario, either end of the link can be relabeled. However, downgrading the head-end of the link may cause additional type (1) conflicts. Therefore, we mark the tail-end of the link as a conflict so that the relabeling subroutine will consider it for upgrade and resolve the level difference.

B. Strict: Two-Phase Relabeling

In the strict policy, the process of relabeling runs in two phases, each consisting of two subroutines. In the first phase, minimum-conflicts paths are found using Dijkstra's shortest-path algorithm (with distance array d and next-hop array p), extended with a custom link metric that is the absolute-value difference between the next-hop switch security level and the flow security level (Algorithm 1, lines 11 and 14). Since rebooting a switch imposes a delay, we impose an additional cost on links containing still-rebooting switches to temporarily reduce the number of flows that would be preempted or queued (i.e., disrupted) by traversing such switches (Algorithm 1, line 11). Type (1) conflicting switches are then recorded during path tracing by checking if each switch has the same level as the flow.

After the potential paths are found, conflicting switches of type (1) are relabeled (up to M switches) to unblock flow paths. While there are several ways to choose which conflicting switch to relabel first, the objective is to eventually relabel all conflicting switches to a converging state. We take a greedy approach by weighting all the conflicting switches based on the flows traversing them, and then relabeling the highest-weighted switches first. We apply the same weighting function $f(L_j)$ from Equation (1) to flow levels during path tracing to find the sum per-level and overall weights for each conflicting switch. We then iteratively select the highest overall-weighted conflicting switches to relabel first. Similarly, for type (1) conflicts, the new security level chosen for a conflicting switch is the security level of the flows that is heaviest at the switch (index 0 of the sorted per-level weights list in Algorithm 2, line 16).

Finally, in phase 2, an additional link constraint is enforced to prevent paths from encountering additional type (1) conflicts (Algorithm 1, line 31), and the resulting paths (if valid) can be distributed to the switches as flow rules.

C. R-BLP: Three-Phase Relabeling

In R-BLP, conflicts of type (1) or (2) may be present since flows may be routed up over appropriate-level switches. Therefore, the first phase operates as above by running the shortest-path algorithm, recording type (1) conflicts by comparing the switch and flow levels, and relabeling the heaviest conflicting switches to the heaviest-level at the switch (Algorithm 2, line 8). Then in the second phase, we run the shortest-path algorithm again and enforce a constraint preventing additional type (1) conflicts (Algorithm 1, line 20). Conflicting switches of type (2) are then recorded during path tracing. As above, the switch at the end of a conflicting segment of a path is chosen to be relabeled and set to the minimum possible level that is still within the bounds of the security policy: the head-end level minus the maximum allowable routing-down degree Δj (Algorithm 2, line 11). Note that there might be multiple conflicting links with the tail-end switch in question, so we choose the link with the highest weighted head-end switch as the one to resolve.

Algorithm 1: `MinConflictPath()` Algorithm

Input: distance array d, previous-hop array p, flow j
Output: updated distance and previous-hop arrays

```
1  Q = {j.source}
2  d[j.source] = 0, p[j.source] = null
3  for v ∈ V \ {j.source} do
4  |  d[v] = ∞, p[v] = null, Q.add(v)
5  end
6  while Q not empty do
7  |  u = node in Q with smallest d[·]
8  |  pop u from Q
9  |  foreach neighbor v of u do
10 |  |  if v ∈ rebooting_switches then
11 |  |  |  w = abs(j.lvl − v.lvl) + reboot_cost
12 |  |  end
13 |  |  else
14 |  |  |  w = abs(j.lvl − v.lvl)
15 |  |  end
16 |  |  if RBLP then
17 |  |  |  if (phase1) and (d[u] + w < d[v]) then
18 |  |  |  |  d[v] = d[u] + w, p[v] = u
19 |  |  |  end
20 |  |  |  else if (phase2) and (d[u] + w <
   |  |  |  d[v]) and (j.lvl ≤ v.lvl) then
21 |  |  |  |  d[v] = d[u] + w, p[v] = u
22 |  |  |  end
23 |  |  |  else if
   |  |  |  (phase3) and (d[u] + w < d[v]) and (j.lvl ≤
   |  |  |  v.lvl) and (u.lvl − v.lvl ≤ Δj) then
24 |  |  |  |  d[v] = d[u] + w, p[v] = u
25 |  |  |  end
26 |  |  end
27 |  |  else if strict then
28 |  |  |  if (phase1) and (d[u] + w < d[v]) then
29 |  |  |  |  d[v] = d[u] + w, p[v] = u
30 |  |  |  end
31 |  |  |  else if (phase2) and (d[u] + w <
   |  |  |  d[v]) and (j.lvl == v.lvl) then
32 |  |  |  |  d[v] = d[u] + w, p[v] = u
33 |  |  |  end
34 |  |  end
35 |  end
36 end
37 return d, p
```

Algorithm 2: `RelabelConflictSw()` Algorithm

Input: sum per-level and overall switch weight array $sw\_weights$
Output: none

```
sorted = sort_per_lvl_weights(sw_weights)
for i = 0 → len(sorted) do
    if num_relabeled == M then
        break
    end
    if RBLP then
        if phase1 then
            sorted[i].lvl = sorted[i].weights[0]
        end
        else if phase2 then
            sorted[i].lvl = max_head − Δj
        end
    end
    else if strict then
        if phase1 then
            sorted[i].lvl = sorted[i].weights[0]
        end
    end
    num_relabeled + +
end
```

Finally, in phase 3, another link constraint is enforced to prevent paths from encountering either type (1) or type (2) conflicts (Algorithm 1, line 23), and the resulting paths (if valid) can be distributed to the switches as flow rules.

D. Complexity

The heuristic algorithms have three steps: (1) execution of the shortest-path algorithm, (2) relabeling conflicting switches, and (3) post-processing to distribute flow rules to switches.

Step (1): Dijkstra's shortest-path algorithm, followed by path tracing, is executed once for each flow in $\mathcal{J}$. If $\mathcal{E}$ denotes the set of network links and $\mathcal{I}$ the set of network switches, the running time for this step is:

$$O(|\mathcal{J}|(|\mathcal{E}| + |\mathcal{I}| \log |\mathcal{I}| + |\mathcal{I}|)) = O(|\mathcal{J}|(|\mathcal{E}| + |\mathcal{I}| \log |\mathcal{I}|)),$$

where path tracing is upper-bounded by $|\mathcal{I}|$.

Step (2): We first sort switches by their per-level weights (across the m levels). Then, we sort switches by their highest weight (i.e., index 0 in the per-level sorted list) and proceed with relabeling in that order. At most M switches are relabeled. Therefore, the running time for this step is:

$$O(|\mathcal{I}| m \log m + |\mathcal{I}| \log |\mathcal{I}| + M) = O(|\mathcal{I}| m \log m + |\mathcal{I}| \log |\mathcal{I}|).$$

Step (3): Finally, the controller performs post-processing on each flow path to determine which flows can be routed and which cannot (i.e., violates the security policy or is queued). The running time for this step is $O(|\mathcal{J}||\mathcal{I}|)$.

The total running time of both algorithms is:

$$O(|\mathcal{J}||\mathcal{E}| + |\mathcal{J}||\mathcal{I}| \log |\mathcal{I}| + |\mathcal{I}| m \log m),$$

which is polynomial. In Section VI, we show that these algorithms are efficient even in large networks. Note that *Step (3)* is only executed when relabeling occurs, and the steps are executed $2\times$ each for the strict policy and $3\times$ for R-BLP.

VI. Evaluation

In the following, we evaluate the performance and security properties of `MLS-Enforcer`. We capture three metrics to assess network performance: (*a*) *flow coverage*, which is the percentage of flows routed under a given labeling policy, (*b*) *agility*, which is the effort required by the network to adapt to destabilizing network events (also called convergence time), and (*c*) *disruption*, the percentage of flows queued or preempted (i.e., not routed immediately or at all). Note that we will refer to *relabeling invocations* as the number of time instants at which relabeling was enabled in `MLS-Enforcer`'s routing algorithms (by setting an $M > 0$). We then measure the risk associated with routing-up and routing-down under R-BLP.

A. Experimental Setup

For simulation-based experiments, we evaluate the performance of `MLS-Enforcer` on two widely used network topologies: mesh and star networks. We also measure the performance of the labeling process in fat-tree networks (used more traditionally in wired networks) for juxtaposition. For the latter, we generate fat-tree networks with a switch port density of $k = 6$ using techniques described in prior work [24]. We use the AT&T North America WAN dataset from Topology Zoo [25] to generate the star topology, and we generate generic full-mesh topologies with 20 switches. We then connect hosts to each switch in the network and assign random levels hosts, with hosts connected to the same edge switch having the same level (e.g., Public hosts all connecting to a Public access point). Initial switch levels are also randomly assigned. We use the standard security lattice from above: *Public (1)* $<$ *Confidential (2)* $<$ *Secret (3)* $<$ *Top-Secret (4)*.

For the end-to-end (E2E) traffic, ≈ 50 flows per second are randomly generated and have source and destination endpoints selected by appropriate levels (i.e., only create flows that comply with the enforced policy). We assume that the controller does not queue flows if they cannot be routed (i.e., that the source may initiate retries of the flow instead). Flow durations are set to an average of 1 s, to assess performance in the worst-case scenario of consistently changing flow patterns. However, we experimented with longer flow durations and found that there was no detectable effect on performance. We then perform relabeling every 100 s and assign a switch reboot time of 10 s (verified experimentally on virtual switches [26][3]).

We also implement the proposed relabeling algorithms as a prototype SDN controller application and perform experiments on a virtual SDN testbed with Mininet [29] and the POX SDN controller [30]. As designed, the labels are maintained entirely

[3]Fast-reboot features in commercial switches [27], [28] take ≈ 25 s, which causes disruption but has no effect on coverage (see below).

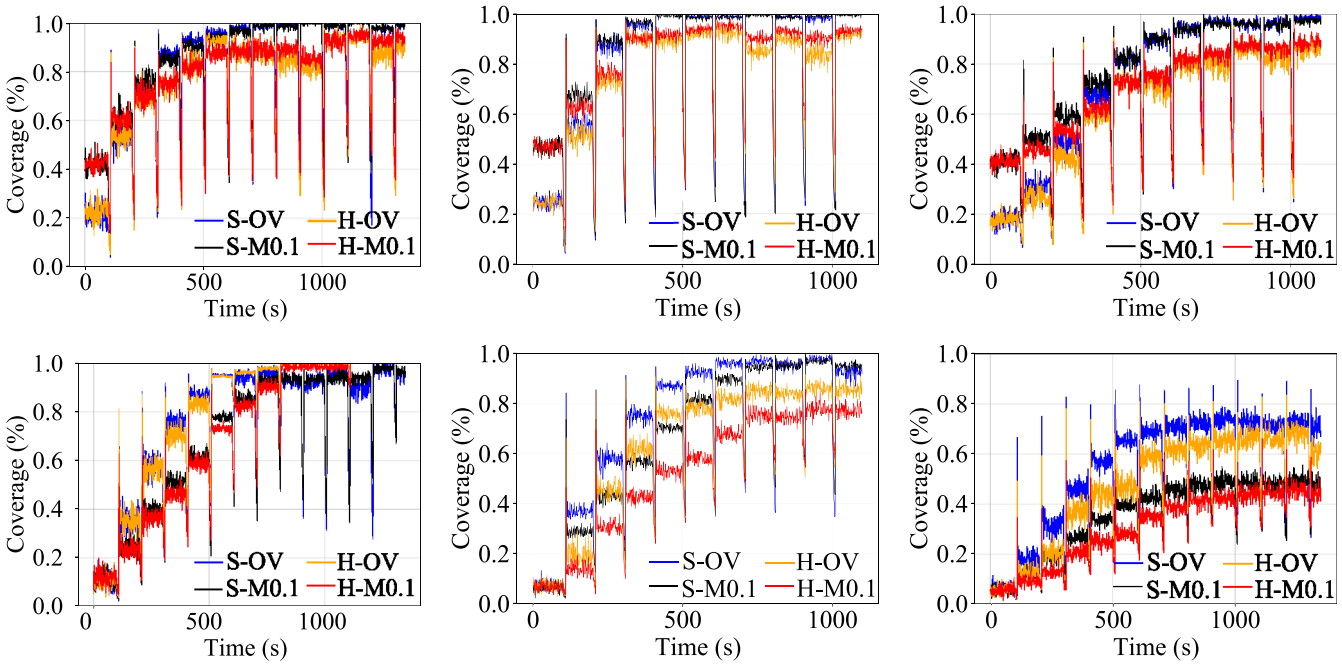


Fig. 4. Coverage in the mesh (left), fat-tree (middle), and star (right) topology for the R-BLP (top) and strict (bottom) MLS policies (S = opt. solver, H = heuristic, OV = obj. value, M0.1 = M is 0.1).

at the controller (where the controller is aware of the given IP-address-to-label mapping), and we evaluate the ability of the application to route flows securely under a fat-tree topology.

We use the Gurobi [31] solver to find optimal solutions, where quadratic flow weighting is used in the optimization formulations and heuristic algorithms, i.e., $f(L_j) = L_j^2$, to give high priority to high-security flows. We also experimented with other flow weighting functions, such as linear (e.g., $f(L_j) = L_j$), cubic (e.g., $f(L_j) = L_j^3$), and other higher degree polynomials, with similar conclusions drawn. Due to space limitations, we do not include figures for those results.

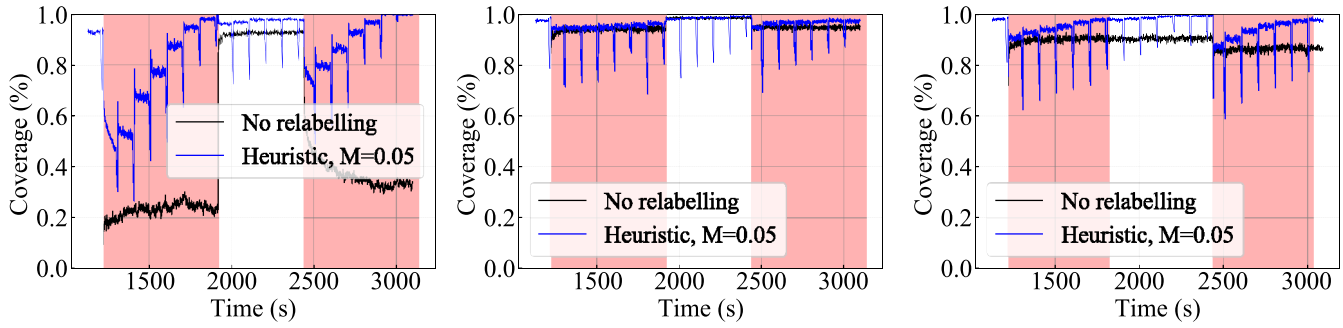
B. Coverage and Running Time (Simulation)

Coverage: Fig. 4 (top left) shows the results of our first set of experiments. The optimization solver for R-BLP achieves ≈99% and the heuristic reaches ≈95% flow coverage for the mesh network within 5 relabeling invocations ($M = 0.1$). The top middle and right graphs show that the solver and heuristic also maintain high coverage in the fat-tree and star networks in steady-state. Note that the visible coverage high/low spikes indicate moments where flows were queued awaiting switch reboots following relabeling. The heuristic with $M = 0.1$ is able to achieve 90% of the maximum objective value (i.e., 90% of the maximum weighted network capacity of routed flows). For comparison, prior MLS networks [14] have only shown to be able to route approximately 60% of network flows under similar network topologies for a 4-level lattice. Here, as expected, we observe that for both the optimization solver and heuristic the convergence is faster with a larger M and shorter relabeling period. We note that experiments with larger networks yielded quantitatively similar coverage results because they only enabled more potential flow paths. In contrast, the limited number of paths in very small networks (e.g., less than 20 switches) already subjects them to low flow coverage (e.g., <50% coverage), and the issue is only exacerbated by destabilizing network events. These networks may require alternative secure-routing approaches.

Moreover, the reboot time had no detectable effect on coverage. More complex security lattices (e.g., composed of several different levels or categories) may induce lower coverage if there are not an adequate number of paths to route flows through (i.e., if the network is too small). Notwithstanding, the heuristic algorithm (red line) maintains nearly the same coverage as the solver (black line), for all three topologies, to within ≈10% of the optimal for R-BLP and within ≈15% for the strict policy (experiments on other topologies yielded similar results).

As to policy, the strict policy performed similar to R-BLP in the mesh network, had a (non-negligible) 10% coverage loss in the fat-tree network, and performed substantially worse (≈20−70% coverage) than R-BLP (≈80−100% coverage) in the star network topology—hence motivating R-BLP. Here, the star network did not have enough redundant switches/interconnectivity to establish a set of non-intersecting paths for all labels. Note that the delay in convergence to the optimal or near-optimal is a reflection of the network startup from a random state, which would occur just once in practice.

Running time: Each invocation of the optimization solver took ≈5 minutes for the fat-tree network of 48 switches and ≈2 minutes for the star (WAN) network of 25 switches and mesh network of 20 switches, where we observed the runtime scaling with a power-law relationship to the number of switches. This renders the optimization solver impractical to use in real-world settings for larger networks (e.g., in large SDN networks, which may manage several thousand switches



(a) Traffic bursts (b) Link failures (c) Device movement

Fig. 5. Adaptive reconfiguration for different network events in the mesh topology.

in datacenter settings). Conversely, the heuristic took ≈1 s on average (in Python) to relabel switches, up to and scaling linearly with networks of several hundred switches and flow arrivals per second. In fact, the heuristic algorithms can compute a relabeling in < 5 s for a mesh network containing 200 switches. The results strongly indicate that the heuristics are both necessary and capable of performing relabeling in complex or large networks with many flows.

C. Network Agility and Recovery (Simulation)

In the next experiments we consider three classes of network events that effect the network performance: traffic bursts, link failures, and device movement. The goal here is to assess how well the network relabeling recovers from (is agile to) to events ranging from small state changes to catastrophic failures.

Traffic bursts: In the experiments depicted in Fig. 5(a), we inject a large burst of top-secret flows for several minutes (pink region). In the mesh topology for the R-BLP policy, the network coverage immediately drops from 90% to 20%. Relabeling is periodically enabled in `MLS-Enforcer` (by setting $M = 0.05$) every 100 seconds to adapt, bringing the coverage back to the steady-state level and even increasing it beyond 90% (the TS-level burst traffic dominates the background traffic) within 2-3 invocations. In contrast, without relabeling (black line) the coverage remains at 20% for the duration of the burst. Note that the network is also re-adjusted within two relabeling invocations once the traffic burst is gone. The network adapts quickly in response to sudden changes in the traffic-level distribution. We observed quantitatively similar convergence times in the fat-tree and star topologies, and slightly longer convergence times under the strict policy.

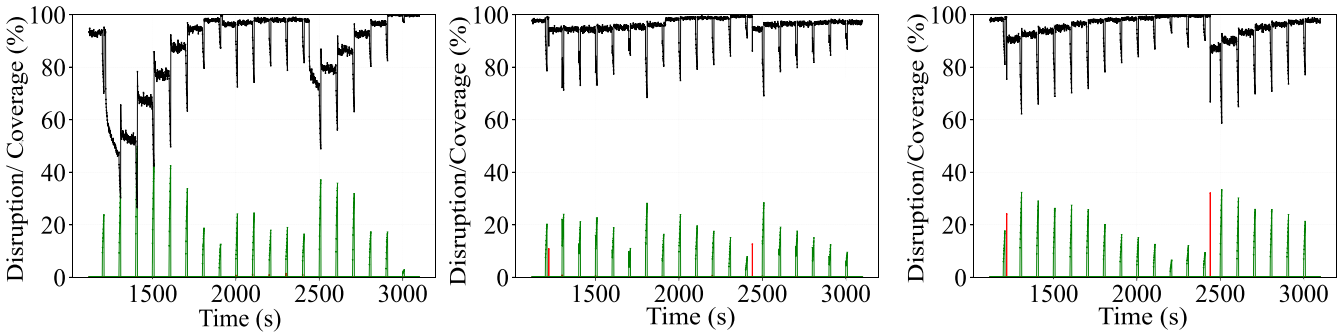
Link failures: Next, we randomly fail 10% of the network links and measure performance in the R-BLP/mesh topology. Shown in Fig. 5(b), coverage drops from ≈98% to around 90%. However, relabeling adapts to these failed links by rerouting flows over other policy-compliant paths, thus incrementally bringing the coverage back to ≈95% over the next three relabeling invocations. Notably, at 10% the magnitude of the coverage drop is manageable, allowing the network to still maintain high coverage. This is partly due to the high degree of connectivity in general in the mesh network. Other experiments for the fat-tree network show that as the volume of links that fail increases, the ability to route traffic drops significantly faster, where we see ≈60% coverage at 20% link failures, 40% coverage at 30%, and 10% coverage at 50%. We observed the same negative effects in the star network topology.

Device movement: In our last set of agility experiments, we relocate 50% of hosts to other parts of the mesh network (i.e., connect through different access networks). This emulates devices being relocated to new physical locations (e.g., a server being moved to a different location inside of a building) or intermittent connectivity of devices. As shown in Fig. 5(c), the coverage drops from the initial 98% to 88%. Further, relabeling enables the network to adapt quickly and reach 95−98% within two invocations. As with the other events, relabeling of the interior switches enables the network to adapt quickly in response to shifts in network structure or host distributions. Note that each relocation event (beginning of the pink zone) causes a random relocation of hosts. However, as we do not revert relocated hosts to their original location, each successive relocation in the non-relabeling network leads to the labeling being further from optimal (and hence lower coverage).

Notably, we conclude that the agility of the system is closely dependent on the security policy; policies that impose more access constraints or use complex security lattices may not converge as quickly under destabilizing events. Moreover, the security policy's flow weighting function $f(L_j)$ also affects the priority of certain flows and must be tuned appropriately to reduce the potential for unfair network partitioning (e.g., relabeling still-alive switches to accommodate a relatively small set of Secret flows but block a larger set of Public flows).

D. Disruption (Simulation)

We define two kinds of disruption: *preempted*, where a flow that was previously routed is then blocked after relabeling, and *queued*, where a flow traverses a path containing a still-rebooting switch. Fig. 6 shows a simulation of each event class for R-BLP on the mesh topology. In general, only ≈ 35% of flows from hosts that were relocated and ≈ 15% of flows from failed links are preempted at the start of the dynamic event, but otherwise the heuristic relabeling algorithms does not preempt flows. The former is due to hosts which had active flows before moving (red line) and the latter due to active flows



(a) Traffic bursts (b) Link failures (c) Device movement

Fig. 6. Time series plots for the mesh topology depicting the disruption caused by switch relabeling in response to dynamic network events (–: coverage, –: preempted, –: queued).

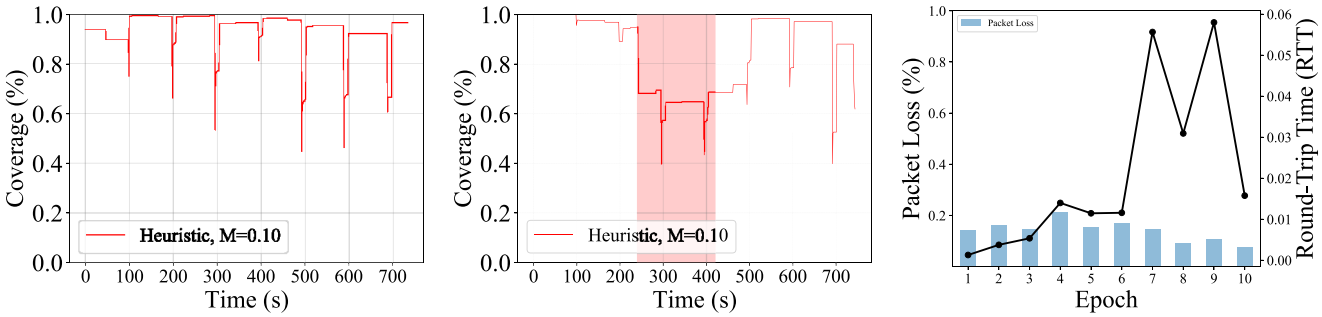


Fig. 7. Experiments with Mininet, POX SDN controller, and `MLS-Enforcer` controller application. From left to right: coverage under normal conditions, agility in response to link failures, and disruption in response to link failures (RTT and packet loss).

using failed links and not having a secure path found at the instant the failure is detected.

On the other hand, switch reboots cause flows to be queued. For link failures and device movement, we observe an average of 30−40% disruption caused by queueing. While a significant proportion of flows, disruption is naturally heavily dependent on switch reboot times, although disruption can be mitigated with lower relabeling frequencies, manually initiating the relabeling algorithm, development of efficient fast-boot features on switches [27], [28], or with a smaller M. A direction for future work is developing *selective* relabeling algorithms that restrict which switch labels may change to provide guarantees about disruption caused to certain flow levels. Note that for the strict policy we observed slightly less disruption. We noticed quantitatively similar results in the fat-tree and star topologies.

E. Coverage, Agility, and Disruption (Mininet)

We then evaluated a prototype SDN controller application (implementing the heuristics) in a Mininet network environment that reflected similar simulation parameters (network size and the R-BLP policy). We generate new flows (as a series of ICMP packets) from hosts every 60-second time epoch. Using a relabeling period of 100s, we then capture measurements by sampling the flow coverage observed by the controller every second. As shown in Fig. 7, under the R-BLP policy in a fat-tree topology, the routing application achieves >90% flow coverage within a single relabeling invocation (at $M = 0.1$), from a random assignment of switch labels. This demonstrates a significant improvement over the coverage granted by prior MLS routing systems [14] (60%) and comparable to the successful packet delivery ratio (>90%) measured in prior works for similar network sizes [32].

We then measured the ability of `MLS-Enforcer` to respond to link failure events by rerouting flows around the failures. The link failure event occurs during the time period highlighted in the red region in the middle plot of Fig. 7 (which corresponds to approximately time 240-420s and time epochs 5-8). We observed in the Mininet network that failure events similarly cause a severe drop in flow coverage that last until relabeling can adjust switch labels to better align with the network traffic profile. Specifically, we observed that the coverage dropped from ≈95% to ≈70% upon detection of the link failure events, where a new route could not be immediately found for many flows that were previously routed across the failed links. However, `MLS-Enforcer` was able to respond to the event to reroute the flows around the failure, achieving near-optimal coverage (≈90%) within 5 relabeling invocations.

From the perspective of hosts, we then measured the disruption on their flows as a result of the link failures (right side of Fig. 7). Of interest are the average RTT (which characterizes the latency of queued flows) and packet loss (which characterizes preempted flows) that flows observe during and after the link failure. Notably, we found under normal conditions that both average flow RTT and packet loss of host ICMP messages are relatively low (at $<0.020s$ and <12%, respectively). However, during the link failure event (which corresponds approximately to time 240-420s and time epochs 5-8), the average RTT and packet loss peak at $\approx 0.06s$ and ≈20%, respectively. Here, some flows are routed around the

failure (incurring higher RTT and a small amount of packet loss), and when the links become live again at epoch 8 (and the controller responds approximately during epoch 9), the average RTT and packet loss begin to drop as `MLS-Enforcer` stabilizes the network.

This supports our observation of high flow coverage and demonstrates the ability of `MLS-Enforcer` to adapt quickly. However, we note that the slight discrepancy in the coverage observed by the controller and packet loss ratios are caused by (1) packets lost before the link event (down/up) is detected, and (2) heavy queueing at the controller that causes some ICMP packets sent from a host to be dropped.

While we present preliminary results under link failures, in future work we plan to refine the implementation to provide a comprehensive analysis of the other network events, as well as the systems challenges introduced when deploying `MLS-Enforcer` in a real network. We refer to Section VII for insights towards integrating `MLS-Enforcer` into commonly used frameworks for deploying various SDN-based network policies. Moreover, to more accurately assess the utility and security in deploying such policies, we defer to future work extending monitoring tools to capture finer-grained routing measurements from different perspectives across the network.

F. Security Analysis

We begin by considering unauthorized flows. Formal MLS policies prevent under-privileged adversaries from capturing network traffic for analysis and mitigate the threat of equally-privileged adversaries (by restricting the potential information flows to only those permitted per the security labels). In this way, `MLS-Enforcer` significantly reduces the capabilities of adversaries performing network scans or attempting to eavesdrop on traffic. Specifically, any unauthorized flows emitted from endpoints or compromised switches are dropped at the nearest uncompromised neighbor—for example, for an unauthorized flow from a malicious endpoint, the controller instructs the access/edge switch to drop the flow, and for an unauthorized flow from a compromised switch, the controller similarly instructs the next-hop switch to drop the flow.

We now consider legitimate flows. In R-BLP a limited number of switches are trusted to "route down" flows, which may represent some risk (see Section III-C). We measure the risk impact in the mesh topology with a limit of five switches per flow ($B = 5$). From the flow perspective, we find in our simulation experiments that 36% of the flows are routed through at least one switch that may route down. However, we find that only 2.52% of the flows received by these switches on average may be routed down below a flow's level (i.e., if the switch is compromised), indicating that routing targets flows that profit from being "routed up."

Nonetheless, risks to the remaining flows can be mitigated using virtual isolation methods. For example, the authors of MLSNet suggest that data routed through devices of lower levels could be encrypted using a level-specific key [14]. The data would be tunneled and encrypted to prevent intermediate switches from eavesdropping on the payload or ascertaining the identities of the endpoints.

Another approach would be to use methods to validate the integrity of the switch run-time environment using remote attestation [33] applied to network equipment [34]. If the network administrators are concerned about traffic analysis, standard techniques such as traffic shaping [35] can be applied.

Evaluation Summary: Our experiments demonstrate that the strict and R-BLP policies could be used to effectively govern a network environment, with associated trade-offs. The relabeling process converges quickly and adapts to changing conditions within a few invocations. Moreover, the relabeling process can be calibrated to be more (faster convergence and recovery) or less aggressive by setting algorithm parameters appropriately.

VII. Discussion

We have demonstrated that `MLS-Enforcer` provides an effective means of constructing and deploying dynamic MLS policies across an entire network infrastructure. The system is designed to integrate into SDNs as a controller application, and therefore has natural extensions into the rich ecosystem of SDN control plane management solutions (e.g., network policy deployment and reconciliation systems [32], [36], [37], [38], [39]). We defer an in-depth analysis of related work to Section VIII, but discuss here avenues for future work in improving the utility of `MLS-Enforcer` and integrating it into other SDN-based policy management systems.

The focus of `MLS-Enforcer` lies in *maximizing flow coverage* under a set of security constraints imposed on all network nodes by the security policy. It therefore requires security labels to be assigned to *all* network nodes, disallows ACL policy violations, and runs the optimization solver or heuristics algorithms to compute an optimal set of switch labels and flow routes through the network. The relabeling feature allows achieving good flow coverage (>90%) compared to the successful packet delivery ratio (>90%) measured in prior works for similar network sizes [32]—but at the trade-off of (potentially) high flow-table usage, since we place no restriction on how many flow rules should be installed on switches to accommodate flows. However, prior works posited that SDN switches provide insufficient flow table capacity, which may lead to performance degradation and network failures [40], [41], [42], and therefore focus on minimizing the number of flow rules deployed on the switches (for implementing ACLs and routing in general) [32], [40], [41], [42].

However, we contend that this argument does not hold in general. For example, in shared infrastructure settings, the infrastructure layer may use virtual (software) switches like Open vSwitch, which has been recently shown to efficiently handle up to several hundred-thousand flow table entries [43]. Therefore, some networks may be able to accept this trade-off at the benefit of enforcing strong security controls across the entire infrastructure. In the future, hardware SDN switches may also be able to support similar flow-table capacities.

As prior works [40], [41], [42] have done, a potential avenue for future work lies in leveraging wildcard flow rules to reduce flow-table usage and reduce controller interaction (e.g., matching subnet prefixes as DIFANE [44] does). With

wildcard rules, as long as end-hosts in the same trust domain are given IP addresses within the same subnets, they could all match against the same flow rules at switches. Using wildcards proactively (as opposed to reactively) and on other flow fields (e.g., protocol numbers) may also help mitigate the impact on flow-table usage. However, wildcarded fields complicate policy enforcement and must be carefully co-designed with the security policy, since matching any flow-field value widens the (inter- and intra-domain) threat surface for adversarial network scanning. See Section II-B for a discussion on using security categories to reduce the threat surface.

Besides constructing dynamic MLS policies, `MLS-Enforcer` can integrate into other systems for policy deployment and reconciliation. For example, prior work introduced the `PrePass-Flow` system [32] for predicting link failures and recomputing the necessary flow rules for enforcing ACLs and routing around the failures. The system uses a K-partite graph technique introduced previously in [45] to find the optimal placement of flow rules (ACLs) onto switches that minimizes the total number of rules deployed. `MLS-Enforcer` could be plugged into `PrePass-Flow` as a replacement for the K-partite graph technique, extending security controls across the entire network infrastructure. Other policy deployment and reconciliation systems, like [38] (especially for hybrid-SDNs), may similarly be extended to support `MLS-Enforcer`, to strengthen the ecosystem of tools available for providing access control in SDNs, particularly those with complex network service chains.

VIII. Related Work

A. Confidentiality in Networks

Several defenses have been proposed to protect confidentiality in networks such as perimeter firewalls, encryption, and routing configuration (e.g., using VLANs). However, these solutions fail to provide comprehensive security guarantees. They only partially address the problem of confidentiality and fail to adapt to dynamic network events. Firewall configuration is complex and error-prone [46]. They are often mis-configured and either violate the user intended security policy or contain inconsistencies and inefficiency among the rules irrespective of the security policy. The inconsistencies could also be among different firewalls (inter-firewall). Furthermore, firewalls fail with regard to insider threats, as attacks that can be staged within the boundary of a perimeter firewall [47].

Similarly, encryption alone cannot ensure confidentiality as adversaries able to capture network traffic may still be able to execute traffic analysis attacks [19], [48], [49]. Traditionally, adversaries have leveraged the packet size of the encrypted traffic as a side channel to infer information about the victim such as which websites were visited. As a result several defenses have been proposed to hide the packet size information, including packet padding and traffic morphing. Traffic analysis attacks based on packet counting [48], [49] were also found to be feasible, whereas defenses such as randomized pipelining over Tor and traffic morphing were found to be insufficient [49] against these classes of attack. Even though the packet counting attacks require identifying the number of packets associated with each Web fetch (which may be challenging in practice), recent work [19] has demonstrated that adversaries can use the packet timing information alone to launch successful traffic analysis attacks. Leveraging MLS security levels and categories, we can prevent under-privileged adversaries from capturing traffic for analysis and mitigate the threat of equally-privileged adversaries (by restricting the potential information flows).

Routing mechanisms such as VLANs offer some degree of isolation: they have been used in cloud settings (including SDNs) [50], [51] with multiple tenants to enforce network traffic isolation by tagging flows in the data plane with a VLAN unique to each tenant. However, VLANs add an additional layer of complexity in providing traffic isolation: they require (1) interacting with switches to manage VLAN assignments on ports, and (2) impose additional network overhead from having to tag every network flow for executing access control checks along a flow path. Thus, they do not scale well for large multi-tenant networks [52]; in contrast, `MLS-Enforcer` ensures an equivalent level of isolation by checking access control constraints at rule installation time, eliminating the need for physical VLAN tags to be attached to each flow.

B. Multilevel Security in Networks

Traditionally, multilevel security systems were used to control access to databases [53] and operating systems [54], by making different data available or presenting data differently to users of different clearances [10]. For example, a database server in a military or industrial organization may be shared among users in both the accounting and engineering departments with complete mediation over accesses to prevent unauthorized data disclosure between users in each department [11]. Furthermore, MLS was also used to secure distributed object oriented systems [55].

Lu and Sundareshan [3] introduced such an MLS system for networks that statically assigned security labels to network switches (based on a relative security analysis of each device) to protect confidentiality in network routing without requiring additional layers of protection, such as encrypted tunnels [5]. This required specialized software to be installed on each network endpoint. While appropriate for the time, the scale and dynamics of modern networks render such a system impractical. The flexibility of SDNs has also been exploited by MLSNet [14], [15] to enforce MLS policies in network routing without requiring specialized software to be installed on each network host and device. Here, the network application at the SDN controller assigns and (logically) maintains security labels for each node (e.g., user device, server, or network switch), and deploys the security policy via flow rules (representing the inter-switch information flow restrictions) that are enforced by the switches. This in turn allows the MLS service to be provided transparently to the network.

Other uses of MLS have been labeling distinct network endpoints that produce/consume data for each other to enforce strong access controls [4], [5] and leveraging hypervisor-level features to isolate network traffic between different tenants in

a cloud network [6]. While these approaches leverage similar MLS techniques as `MLS-Enforcer`, they are limited in that they assume static network behavior and are not designed to adapt to events that alter the network structure or traffic profile. This limitation can lead to significant under-utilization and often a failure to route a large fraction of flows.

C. Deploying and Verifying Network Policies

There is also a large body of work in deploying, verifying, and reconciling SDN-based network policies. For example, constructions and specification languages have been introduced that check for reachability and loop-free forwarding [36], [56], [57], and network-level access controls (ACLs) per-service and per-user-identities [32], [58], [59], [60], [61], among other invariants. However, many of these systems are limited in that they fail to adapt to dynamic network events—the policies are either predefined (static) based on the user identity [60] or service [14], [59], or do not consider the security of intermediate nodes within the underlying shared network infrastructure. They, therefore, cannot meet a security policy such as R-BLP under varying network conditions.

Systems have also been tightly co-designed with SDNs to check for policy compliance in real time [36], [62], building on header-space-analysis [56] (a set of tools to model and check network-wide invariants and identify failure conditions) to incrementally check compliance of state changes such as flow rule installation and removal. The mechanisms have also evolved to reduce controller interaction by providing real-time policy checking entirely within the data plane [32], [45], [63].

More closely related to `MLS-Enforcer`, efficient deployment and reconciliation of SDN-based ACL whitelisting policies have been extensively studied [37], [38], [39], particularly in the presence of network failures [32]. However, the goal of `MLS-Enforcer` lies in providing a framework for *constructing instances of dynamic MLS policies*. Our contributions therefore differ in intent from prior works that focus primarily on *deployment* or *reconciliation* of an *already-defined* set of policies. Moreover, we formulate optimization problems reflecting security policies using formally-defined MLS semantics that protect confidentiality of information flow—a different realization of access control than traditional endpoint-whitelisting/ACLs (which may involve manual composition [38]) that these prior works had not considered.

Besides the functional goal of `MLS-Enforcer`, the design also differs significantly from prior works on dynamic ACL deployment. In particular, recent works have emphasized the increasing threat of the network infrastructure itself becoming compromised, besides potentially malicious network endpoints—from exploiting weakly protected admin Web interfaces to bugs in the switch operating system software and hardware backdoors [64]. These insights motivate our design to extend dynamic ACL deployment beyond endpoint-whitelisting to realize formal (and dynamic) information-flow guarantees across an entire network infrastructure (i.e., across both *endpoints* and *forwarding devices*). As prior works have done [37], `MLS-Enforcer` assigns to network endpoints a security class/group (via a security label) based on a relative security assessment of each device or other labeling scheme for associating devices with particular trust domains. However, `MLS-Enforcer` also assigns security labels to switches, which may change over time to align with network conditions.

Moreover, the optimizations introduced in prior works focus primarily on labeling network endpoints and minimizing "unwanted" traffic in the network, the number of ACL policy violations, and on the number of ACL policies installed on switches [32], [45]. In contrast, we assign security labels to *all* network nodes, disallow ACL policy violations (thereby disallowing any "unwanted" traffic), and focus on maximizing flow coverage (at the tradeoff of more ACL policies being installed; i.e., higher flow-table usage). `MLS-Enforcer` still achieves comparable flow coverage (>90%) to the successful packet delivery ratio (>90%) measured in prior works for similar network sizes [32]. We have already elaborated on the implications of this tradeoff in Section VII.

IX. Conclusion

In this work, we introduced `MLS-Enforcer`, a system that extends network-level MLS capabilities to unstable networks. We envision `MLS-Enforcer` as a network application running on an SDN controller, providing the service transparently to the entire network. The flexibility of SDNs allows the system to relabel network nodes in response to evolving traffic and policy profiles, thus allowing the network to remain agile in the face of instability. We modeled network adaptivity as an integer linear program that enables network administrators to maximize the overall utility capacity of the network under the security constraints of the given information-flow security policy. We then developed polynomial-time heuristic relabeling algorithms that scale more efficiently with larger networks.

We assessed the system performance and security properties by focusing on four evaluation metrics: coverage, agility, disruption, and security risk. Through extensive evaluation, we observed that the system performed well under several network topologies, policies, and destabilizing network events. We showed that `MLS-Enforcer` can optimally relabel the network to support 90%+ of flows under normal conditions and quickly converge under changing needs. Moreover, we showed that the heuristic algorithms can achieve 90% of the optimal flow coverage with a 300× reduction in computational overhead—thus demonstrating that it is feasible for formally secured networks to be deployed in diverse and unpredictable environments. In future work, we will consider different MLS policies and extend the optimization framework to other objective functions, such as minimizing the total number of links on which there are route-down paths over all flows.

Acknowledgment

The authors would like to thank Ryan Sheatsley, Yohan Beugin, Eric Pauley, and Sophia Beyda for their feedback and support on early versions of the paper. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Combat Capabilities Development Command Army Research Laboratory or the

REFERENCES

[1] D. Gonzales, J. M. Kaplan, E. Saltzman, Z. Winkelman, and D. Woods, "Cloud-trust—A security assessment model for infrastructure as a service (IaaS) clouds," *IEEE Trans. Cloud Comput.*, vol. 5, no. 3, pp. 523–536, Jul.–Sep. 2015.

[2] I. F. Akyildiz and X. Wang, "A survey on wireless mesh networks," *IEEE Commun. Mag.*, vol. 43, no. 9, pp. S23–S30, Sep. 2005.

[3] W.-P. Lu and M. K. Sundareshan, "A model for multilevel security in computer networks," *IEEE Trans. Softw. Eng.*, vol. 16, no. 6, pp. 647–659, Jun. 1990.

[4] P. Watson, "A multi-level security model for partitioning workflows over federated clouds," *J. Cloud Comput. Adv. Syst. Appl.*, vol. 1, no. 1, p. 15, 2012.

[5] T. D. Nguyen, M. A. Gondree, D. J. Shifflett, J. Khosalim, T. E. Levin, and C. E. Irvine, "A cloud-oriented cross-domain security architecture," in *Proc. IEEE MILCOM*, 2010, pp. 441–447.

[6] N. Meghanathan, "Review of access control models for cloud computing," *Comput. Sci. Inf. Sci.*, vol. 3, no. 1, pp. 77–85, 2013.

[7] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Comput. Netw.*, vol. 52, no. 12, pp. 2292–2330, 2008.

[8] D. E. Bell and L. J. La Padula, "Secure computer system: Unified exposition and multics interpretation," MITRE Corp., Bedford, MA, USA, Rep. ESD-TR-75-306, 1976.

[9] D. E. Denning, "A lattice model of secure information flow," *Commun. ACM*, vol. 19, no. 5, pp. 236–243, 1976.

[10] G. Pernul, W. Winiwarter, and A. M. Tjoa, "The entity-relationship model for multilevel security," in *Proc. Int. Conf. Conceptual Model.*, 1993, pp. 166–177.

[11] O. S. Saydjari, "Multilevel security: Reprise," *IEEE Security Privacy*, vol. 2, no. 5, pp. 64–67, Sep./Oct. 2004.

[12] A. T. Mizrak, Y.-C. Cheng, K. Marzullo, and S. Savage, "Detecting and isolating malicious routers," *IEEE Trans. Dependable Secure Comput.*, vol. 3, no. 3, pp. 230–244, Jul.–Sep. 2006.

[13] T. Azzabi, H. Farhat, and N. Sahli, "A survey on wireless sensor networks security issues and military specificities," in *Proc. Int. Conf. Adv. Syst. Electr. Technol. (IC ASET)*, 2017, pp. 66–72.

[14] S. Achleitner, Q. Burke, P. McDaniel, T. Jaeger, T. L. Porta, and S. Krishnamurthy, "MLSNet: A policy complying multilevel security framework for software defined networking," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 1, pp. 729–744, Mar. 2021.

[15] V. Varadharajan, K. Karmakar, U. Tupakula, and M. Hitchens, "A policy-based security architecture for software-defined networks," *IEEE Trans. Inf. Forensics Security*, vol. 14, pp. 897–912, 2019.

[16] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov, "Security in software defined networks: A survey," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2317–2346, 4th Quart., 2015.

[17] S. Roy, N. Sharmin, J. C. Acosta, C. Kiekintveld, and A. Laszka, "Survey and taxonomy of adversarial reconnaissance techniques," 2021, *arXiv:2105.04749*.

[18] S. Achleitner, T. La Porta, T. Jaeger, and P. McDaniel, "Adversarial network forensics in software defined networking," in *Proc. Symp. SDN Res.*, 2017, pp. 8–20. [Online]. Available: https://doi.org/10.1145/3050220.3050223

[19] S. Feghhi and D. J. Leith, "A Web traffic analysis attack using only timing information," *IEEE Trans. Inf. Forensics Security*, vol. 11, pp. 1747–1759, 2016.

[20] "OpenDayLight SDN Controller." [Online]. Available: https://www.opendaylight.org/ (Accessed: Apr. 20, 2016).

[21] "The Frenetic Project." [Online]. Available: https://github.com/frenetic-lang/frenetic (Accessed: Feb. 8, 2022).

[22] S. Bellovin and E. Gansner, "Using link cuts to attack Internet routing," in *Proc. 12th USENIX Security Symp.*, 2003, pp. 1–16.

[23] M. Conforti, G. Cornuejols, and G. Zambelli, *Integer Programming*. Cham, Switzerland: Springer, 2014.

[24] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, 2008.

[25] S. Knight, H. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet topology zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.

[26] "OpenVSwitch." [Online]. Available: http://openvswitch.org/ (Accessed: Jun. 4, 2022).

[27] "Loading and Managing System Images Configuration Guide, Cisco IOS Release 15S." [Online]. Available: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/sys-image-mgmt/configuration/15-s/sysimgmgmt-15-s-book.pdf (Accessed: Jun. 4, 2022).

[28] "SONiC Fast-Reboot (Fast-Reload) Design." [Online]. Available: https://github.com/Azure/SONiC/wiki/Fast-Reboot (Accessed: Jun. 4, 2022).

[29] "Mininet—Realistic Virtual SDN Network Emulator." [Online]. Available: http://mininet.org/ (Accessed: Nov. 6, 2017).

[30] "POX—Python based SDN Controller Framework." [Online]. Available: http://www.noxrepo.org/pox/about-pox/ (Accessed: Nov. 6, 2015).

[31] "Gurobi." [Online]. Available: http://gurobi.com (Accessed: Jun. 4, 2022).

[32] M. Ibrar, L. Wang, G.-M. Muntean, A. Akbar, N. Shah, and K. R. Malik, "PrePass-flow: A machine learning based technique to minimize ACL policy violation due to links failure in hybrid SDN," *Comput. Netw.*, vol. 184, Jan. 2021, Art. no. 107706.

[33] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, "Design and implementation of a TCG-based integrity measurement architecture," in *Proc. 13th USENIX Security Symp.*, 2004, pp. 223–238.

[34] (Trusted Computing Group, Beaverton, OR, USA). *TCG Guidance for Securing Network Equipment Using TCG Technology*. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/TCG_Guidance_for_Securing_NetEq_1_0r29.pdf (Accessed: Jun. 4, 2022).

[35] C. V. Wright, S. E. Coull, and F. Monrose, "Traffic Morphing: An efficient defense against statistical traffic analysis," in *Proc. NDSS*, 2009, pp. 1–14. [Online]. Available: https://www.bibsonomy.org/bibtex/265c4a9c5d1a7fc7e9cd55cf6edeef6dc/dblp

[36] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte, "Real time network policy checking using header space analysis," in *Proc. 10th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2013, pp. 99–112.

[37] M. Ali, N. Shah, and M. A. K. Khattak, "DAI: Dynamic ACL policy implementation for software-defined networking," in *Proc. IEEE 17th Int. Conf. Smart Commun. Improving Qual. Life Using ICT IoT AI (HONET)*, 2020, pp. 138–142.

[38] C. Prakash *et al.*, "PGA: Using graphs to express and automatically reconcile network policies," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 29–42, 2015.

[39] S. Pisharody, J. Natarajan, A. Chowdhary, A. Alshalan, and D. Huang, "Brew: A security policy analysis framework for distributed SDN-based cloud environments," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 6, pp. 1011–1025, Nov./Dec. 2019.

[40] J.-P. Sheu, W.-T. Lin, and G.-Y. Chang, "Efficient TCAM rules distribution algorithms in software-defined networking," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 2, pp. 854–865, Jun. 2018.

[41] Y. Guo, H. Luo, Z. Wang, X. Yin, and J. Wu, "Routing optimization with path cardinality constraints in a hybrid SDN," *Comput. Commun.*, vol. 165, pp. 112–121, Jan. 2021.

[42] R. Bauer and M. Zitterbart, "An optimization-based approach for flow table capacity bottleneck mitigation in software-defined networks," 2021, *arXiv:2109.08482*.

[43] B. Pfaff *et al.*, "The design and implementation of open vSwitch," in *Proc. 12th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2015, pp. 117–130.

[44] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 351–362, 2010.

[45] R. Amin, N. Shah, and W. Mehmood, "Enforcing optimal ACL policies using K-partite graph in hybrid SDN," *Electronics*, vol. 8, no. 6, p. 604, 2019.

[46] L. Yuan, H. Chen, J. Mai, C.-N. Chuah, Z. Su, and P. Mohapatra, "FIREMAN: A toolkit for firewall modeling and analysis," in *Proc. IEEE Symp. Security Privacy (SP)*, 2006, p. 15.

[47] L. Spitzner, "Honeypots: Catching the insider threat," in *Proc. 19th Annu. Comput. Security Appl. Conf.*, 2003, pp. 170–179.

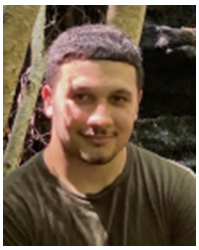
[48] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-boo, I still see you: Why efficient traffic analysis countermeasures fail," in *Proc. IEEE Symp. Security Privacy*, 2012, pp. 332–346.

[49] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching from a distance: Website fingerprinting attacks and defenses," in *Proc. ACM Conf. Comput. Commun. Security*, 2012, pp. 605–616.

[50] S. Gutz, A. Story, C. Schlesinger, and N. Foster, "Splendid isolation: A slice abstraction for software-defined networks," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 1–6.

[51] D. Drutskoy, E. Keller, and J. Rexford, "Scalable network virtualization in software-defined networks," *IEEE Internet Comput.*, vol. 17, no. 2, pp. 20–27, Mar./Apr. 2013.

[52] A. Ranjbar, M. Antikainen, and T. Aura, "Domain isolation in a multi-tenant software-defined network," in *Proc. IEEE/ACM 8th Int. Conf. Utility Cloud Comput. (UCC)*, 2015, pp. 16–25.
[53] X. Qian and T. F. Lunt, "A semantic framework of the multilevel secure relational model," *IEEE Trans. Knowl. Data Eng.*, vol. 9, no. 2, pp. 292–301, Mar./Apr. 1997.
[54] P. Loscocco, "Security-enhanced linux," in *Proc. Linux 2.5 Kernel Summit*, San Jose, CA, USA, 2001.
[55] V. Varadharajan and S. Black, "A multilevel security model for a distributed object-oriented system," in *Proc. 6th Annu. Comput. Security Appl. Conf.*, 1990, pp. 68–78.
[56] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," in *Proc. 9th USENIX Symp. Netw. Syst. Design Implement.*, 2012, pp. 113–126.
[57] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P. B. Godfrey, and S. T. King, "Debugging the data plane with anteater," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 290–301, 2011.
[58] B. Tian *et al.*, "Safely and automatically updating in-network ACL configurations with intent language," in *Proc. ACM Special Interest Group Data Commun.*, 2019, pp. 214–226.
[59] J. Matias, J. Garay, A. Mendiola, N. Toledo, and E. Jacob, "FlowNAC: Flow-based network access control," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014, pp. 79–84.
[60] A. Hesham, F. Sardis, S. Wong, T. Mahmoodi, and M. Tatipamula, "A simplified network access control design and implementation for M2M communication using SDN," in *Proc. IEEE Wireless Commun. Netw. Conf. Workshops (WCNCW)*, 2017, pp. 1–5.
[61] S. T. Yakasai and C. G. Guy, "FlowIdentity: Software-defined network access control," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, 2015, pp. 115–120.
[62] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey, "Veriflow: Verifying network-wide invariants in real time," in *Proc. 10th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2013, pp. 15–27.
[63] J. Liu *et al.*, "Leveraging software-defined networking for security policy enforcement," *Inf. Sci.*, vol. 327, pp. 288–299, Jan. 2016.
[64] K. Thimmaraju, L. Schiff, and S. Schmid, "Outsmarting network security with SDN teleportation," in *Proc. IEEE Eur. Symp. Security Privacy (EuroS P)*, 2017, pp. 563–578.



Quinn Burke (Student Member, IEEE) received the B.S. and M.S. degrees in computer science from the Pennsylvania State University with a focus on computer security, where he is currently pursuing the Ph.D. degree in computer science. His research interests include network and systems security, software-defined networking, and virtualization technologies.

Fidan Mehmeti (Member, IEEE) received the Graduation degree in electrical and computer engineering from the University of Prishtina, Kosovo, in 2009, and the Ph.D. degree from the Institute Eurecom/Telecom ParisTech, France, in 2015. He was a Postdoctoral Scholar with the University of Waterloo, Canada, North Carolina State University, and Penn State University, USA. He is currently working as a Senior Researcher and a Lecturer with the Technical University of Munich, Germany. His research interests lie within the broad area of wireless networks, with an emphasis on performance modeling, analysis, and optimization.

Rahul George received the B.E. degree in information science from the BMS College of Engineering, India, in 2017, and the M.S. degree in computer science from the Pennsylvania State University with a focus on computer security, where he is currently pursuing the Ph.D. degree in computer science. His research interests include network and software security, intrusion detection, and security-performance tradeoffs.

Kyle Ostrowski received the Bachelor of Science degree in computer engineering from The Pennsylvania State University and was a Student Researcher under the supervision of Dr. T. L. Porta. He currently works as a Research and Development Engineer, specializing in firmware and digital circuits. His interests include network security and software-defined networks.

Trent Jaeger (Member, IEEE) is a Professor with the Computer Science and Engineering Department, The Pennsylvania State University. His research interests include systems and software security, on which he has published over 150 journal and conference papers. He has authored the book *Operating Systems Security*, which examines the principles behind secure operating system designs. He has made a variety of contributions to the open-source security community, particularly to the Linux operating system. He serves on the Executive Committee for the *ACM Special Interest Group on Security, Audit, and Control*, is the Steering Committee Chair for the Network and Distributed Systems Security Symposium, and is an Editorial Board Member for the *Communications of the ACM* and IEEE SECURITY AND PRIVACY.

Thomas F. La Porta (Fellow, IEEE) received the B.S.E.E. and M.S.E.E. degrees from The Cooper Union, New York, NY, USA, and the Ph.D. degree in electrical engineering from Columbia University, New York, NY, USA. He is the Director of the School of Electrical Engineering and Computer Science and Penn State University. He is an Evan Pugh Professor and the William E. Leonhard Chair Professor with the Computer Science and Engineering Department and the Electrical Engineering Department. He joined Penn State in 2002, where he was the Founding Director of the Institute of Networking and Security Research. Prior to joining Penn State, he was with Bell Laboratories for 17 years. He was the Director of the Mobile Networking Research Department, Bell Laboratories, Lucent Technologies where he led various projects in wireless and mobile networking. He is a Bell Labs Fellow, received the Bell Labs Distinguished Technical Staff Award, and an Eta Kappa Nu Outstanding Young Electrical Engineer Award. He also won two Thomas Alva Edison Patent Awards. He was the Founding Editor-in-Chief of the IEEE TRANSACTIONS ON MOBILE COMPUTING. He served as the Editor-in-Chief of *IEEE Personal Communications Magazine*. He was the Director of Magazines for the IEEE Communications Society and was on its Board of Governors for three years.

Patrick McDaniel (Fellow, IEEE) is the William L. Weiss Professor of Information and Communications Technology and the Director of the Institute for Networking and Security Research, School of Electrical Engineering and Computer Science, Pennsylvania State University. He also served as the Program Manager and a Lead Scientist of the Army Research Laboratory's Cyber-Security Collaborative Research Alliance from 2013 to 2018. Prior to joining Penn State in 2004, he was a Senior Research Staff Member with AT&T Labs-Research. His research focuses on a wide range of topics in computer and network security and technical public policy. He is a Fellow of ACM and AAAS and the Director of the NSF Frontier Center for Trustworthy Machine Learning.