MSNetViews: Geographically Distributed Management of Enterprise Network Security Policy

Iffat Anjum
NC State University
Raleigh, NC, USA

Jessica Sokal
Northeastern University
Boston, MA, USA

Hafiza Ramzah Rehman
NC State University
Raleigh, NC, USA

Ben Weintraub
Northeastern University
Boston, MA, USA

Ethan Leba
Northeastern University
Boston, MA, USA

William Enck
NC State University
Raleigh, NC, USA

Cristina Nita-Rotaru
Northeastern University
Boston, MA, USA

Bradley Reaves
NC State University
Raleigh, NC, USA

ABSTRACT

Commercially-available software defined networking (SDN) technologies will play an important role in protecting the on-premises resources that remain as enterprises transition to zero trust architectures. However, existing solutions assume the entire network resides in a single geographic location, requiring organizations with multiple sites to manually ensure consistency of security policy across all sites. In this paper, we present MSNetViews, which extends a single, globally-defined and managed, enterprise network security policy to many geographically distributed sites. Each site operates independently and enforces a site-specific *policy slice* that is dynamically parameterized with user location as employees roam between sites. We build a prototype of MSNetViews and show that for an enterprise with globally distributed sites, the average time for policy state to settle after a user roams to a new site is well below two seconds. As such, we demonstrate that multisite organizations can efficiently protect their on-premises network-attached devices via a single global perspective.

CCS CONCEPTS

• **Security and privacy → Network security**.

KEYWORDS

enterprise network; zero trust; sdn; access-control; least-privilege

ACM Reference Format:
Iffat Anjum, Jessica Sokal, Hafiza Ramzah Rehman, Ben Weintraub, Ethan Leba, William Enck, Cristina Nita-Rotaru, and Bradley Reaves. 2023. MSNetViews: Geographically Distributed Management of Enterprise Network Security Policy. In *Proceedings of the 28th ACM Symposium on Access Control Models and Technologies (SACMAT '23), June 7–9, 2023, Trento, Italy.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3589608.3593836

1 INTRODUCTION

After decades of criticisms of "moat-and-gate" defenses, enterprise network security is on the verge of a fundamental change. Practitioner interest in Zero Trust [40] has reached a tipping point, recently motivated by requirements stated in US Whitehouse Executive Order EO-14028 [25] and Memo M-22-09 [39]. Zero Trust models assume the attacker has breached perimeter defenses and seek to enforce accurate, least-privilege, per-request access decisions to information systems. The predominant Zero Trust models (e.g., BeyondCorp [46] and BastionZero [50]) require organizations to place critical business applications on cloud servers where web application gateways perform multi-factor authentication, device attestation, and behavioral analytics. However, not all on-premises resources can be relocated to the cloud (e.g., development servers, file servers, and device management interfaces), and workstations remain a high-value target for advanced attacks such as Solorigate [19] and NotPetya [6].

NetViews [4] is the most recent work in a series of research proposals [12, 28, 35] that use reactive software-defined networking (SDN) to enforce least-privilege, per-request connections between hosts within an enterprise network. Reactive SDN configures every SDN switch to consult a logically-central SDN controller whenever it receives a packet that does not match its existing forwarding rules. If carefully managed, these forwarding rules can efficiently implement a reference monitor interface [2] to enforce mandatory controls on network flows. As such, on-premises Zero Trust models can be constructed to protect hosts and resources that cannot practically be moved to the cloud.

All prior reactive SDN-based solutions for network access control assume the enterprise network exists in a single geographic site. However, most enterprises consist of many geographically distributed sites. Some enterprises have a small number of very large sites (e.g., the IBM topology in topology-zoo [38], has 18 sites, each with a massive number of resources and clients), while others have a large number of very small sites (e.g., Well's Fargo has 5300 branches across the USA). Employees also commonly move between sites and need differentiated access based on their specific location context. Globally managing user context across an organization is important. For example, Memo M-22-09 [39] states that "Zero trust architectures require metadata about the user to allow agencies to make risk-based decisions at the policy enforcement point. That metadata is maintained, updated, and supplied by systems that manage user identities, keeping the appropriate metadata associated with the correct user *even if that user leaves the organization or moves to a new position within it.* ...Using *centrally managed* systems to provide enterprise identity and *access management* services reduces the burden on agency staff to manage individual accounts and credentials." (emphasis added).

In this paper, we present MultiSite NetViews (MSNetViews), which extends a single, globally-defined, enterprise network security policy to many geographically distributed sites. MSNetViews extends prior work by managing many independently-operating reactive-SDN networks that dynamically react to employee movement between sites, enforcing both role-and location-based access control policies. MSNetViews also introduces the concept of site-specific "policy slices," which avoid unnecessary policy updates and limits the security policy available at each site on a "need-to-know" basis. Our performance evaluation shows that for an enterprise with sites globally distributed, the average time for policy state to settle after a user roams to a new site is well below two seconds, which is negligible with respect to the overall experience of traveling, often hours, and authenticating to a new network.

We make the following contributions in this paper:

- *We propose the MSNetViews policy model for supporting user roaming between sites.* Location updates do not change the global policy specification, but rather parameterize the policy state at individual sites.
- *We propose* policy slicing *to minimize policy information loss when a site is compromised.* Each site maintains a site-specific policy containing only the policy elements and rules needed to govern access to local resources.
- *We propose a global administrative model that maintains the correctness of administrative updates.* With multiple administrators specifying policy for their individual sites, novel policy checks ensure mistakes do not cause interference between the role- and location-based policies.

Throughout the paper, we assume all enterprise network sites are fully provisioned with reactive SDN technology such as OpenFlow. However, hybrid designs can achieve similar protection by using legacy VLAN-capable switches to shunt traffic from individual hosts to SDN-capable distribution switches [1, 31]. While both approaches require changes to networking infrastructure, transition plans are needed for resources that cannot practically be moved to cloud servers, particularly for US government organizations needing to meet the fiscal year 2024 deadline set by Memo M-22-09 [39].

Finally, similar to NetViews, MSNetViews is not a complete zero trust solution for on-premises devices. The goal of MSNetViews is to extend NetViews to a multisite setting. A complete zero trust solution should include device attestation and the behavioral analytics. We discuss MSNetViews and zero trust in Section 8.

The remainder of this paper proceeds as follows. Section 2 provides necessary background. Section 3 overviews our approach. Sections 4, 5, and 6 describe our design. Section 7 evaluates performance. Section 8 discusses MSNetViews and zero trust. Section 9 overviews related work. Section 10 concludes.

Availability: The source code for our NetViews implementation is available at https://github.com/netviews/ms-netviews.

2 BACKGROUND AND THREAT MODEL

NetViews and SDN: In contrast to proactive SDN (statically installing rules in switches), reactive SDN provides a flexible reference monitor interface [2] for limiting attackers' movement once they gain access to a host within an enterprise network. Several prior works have explored using reactive SDN for access control [4, 12, 28, 35]. MSNetViews builds on NetViews [4], which uses NIST's recent Next Generation Access Control (NGAC) policy language to define policies that determine whether and how a given host h_1 can "see" another host h_2. In NetViews, policy "users" are user-device pairs, "objects" are destination hosts, and "rights" are transport layer ports (e.g., tcp/22), ICMP types, and ARP types. NetViews enforces policy using the ONOS SDN controller for OpenFlow. Initially, SDN switches have no forwarding state. When a switch receives a packet that does not match any forwarding rules, it sends a `PacketIn` message to the SDN controller. The logically central controller uses multiple applications (e.g., forwarding, access control) to determine to which physical port the switch should forward the packet. Its response to the switch usually takes the form of a `FlowMod` message, which defines a forwarding rule matching future packets. ONOS enhances traditional OpenFlow environments by providing an Intent abstraction that allows SDN applications to treat the network as "one big switch" with no need to manage `FlowMod` messages to individual switches.

NIST NGAC Policy Language: NGAC is a domain-specific language for writing access control policies. Below we provide a brief introduction to its concepts. For more information we refer the reader to the NGAC whitepaper [15].

NGAC policies are defined in terms of sets of objects and actions on those objects. An NGAC policy $\mathcal{P}$ consists of a set of *policy elements PE*, a set of obligations O, and three sets of *policy relations*. The policy relations are: $\mathcal{R}_e$ (assignments), $\mathcal{R}_a$ (associations), and $\mathcal{R}_p$ (prohibitions). The set of policy elements PE includes the sets of users U, user attributes UA, objects O, object attributes OA, and policy classes PC. All objects are object attributes ($O \subseteq OA$).

Policy elements inherit privileges from other policy elements. In NGAC, the set of assignment relations $\mathcal{R}_e$ is a set of pairs of policy elements. It models the graph-based inheritance between policy elements. For $e_1, e_2 \in PE$ and $(e_1, e_2) \in \mathcal{R}_e$, we denote e_1 inherits privileges associated with e_2 by writing $e_1 \rightarrow e_2$. This relation indicates that any privilege assigned to e_2 will also be held by e_1. The assignment relation defines an upside-down tree structure, of which the policy classes PC are roots. We call a series of pairwise relations connecting two objects a *path*. We denote paths with the $\rightsquigarrow$ relation, e.g., $ua \rightsquigarrow pc$ indicates there is a path from ua to pc. Example assignment relation graphs are shown in Figure 2.

Users are granted access privileges to objects through association relations. The set of association relations $\mathcal{R}_a$ is a set of triplets of user attributes, access rights (privileges), and object attributes. Let AR be a set of privileges, if $ua \in UA$ and $oa \in OA$, then we write $(ua, AR, oa) \in \mathcal{R}_a$ to indicate that all users $u \in U$ with an assignment path $u \rightsquigarrow ua$ can perform rights AR on all objects $o \in O$ with an assignment path $o \rightsquigarrow oa$. NGAC requires association relations *for all* policy classes for an action to be granted. The allowed set of rights is the *intersection* of the maximum sets of

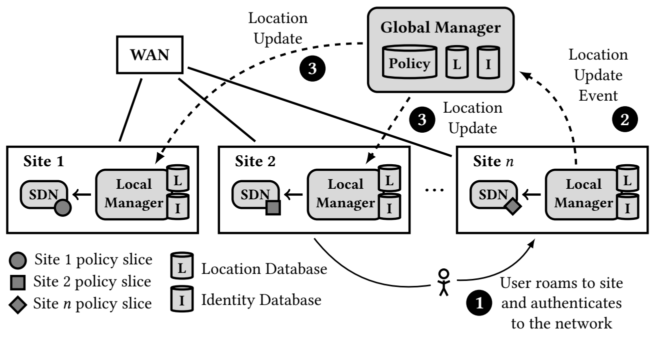


Figure 1: Overview of our approach. Each site runs its own SDN network with a unique subset of the global policy. Users roaming between sites cause location update events that are propagated to other sites as appropriate. Each SDN controller has a different access control policy, as indicated by the different shapes (circle, square, diamond).

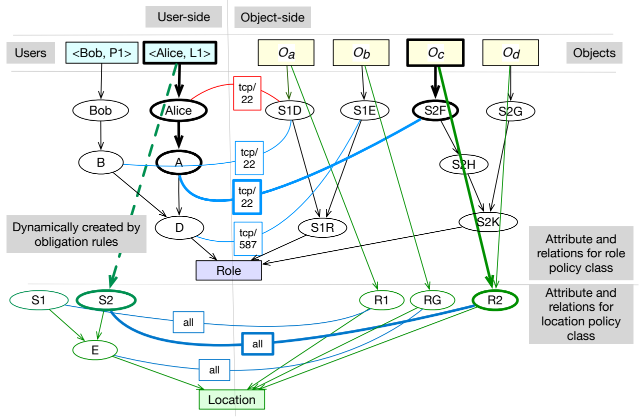


Figure 2: MSNetViews policy example depicting two sites ($S1$ and $S2$). Here, the user $\langle Alice, L1\rangle$ can SSH to object O_c while attached to user attribute $S2$. The bold lined paths through two policy classes ("Role" and "Location") indicates the access control paths needed to be followed for determining this "allow" decision. Downward blue arcs denote associations, each of which is annotated with a set of rights (e.g., `tcp/22`).

rights for each policy class. Association relations are shown as downward blue arcs in Figure 2.

Users are explicitly denied access to resources using prohibition relations. The set of prohibition relations $\mathcal{R}_p$ is a set of triples of user attributes, privileges, and object attributes. For prohibition relations, the existence of multiple policy classes does not matter, and prohibitions *always supersede* association relations. Prohibition relations are shown as upward red arcs in Figure 2.

Finally, NGAC allows *dynamic* updates to policy elements and policy relations using *obligations*. The set of obligations O consists of pairs of event patterns ep and responses r to those patterns denoted $\langle ep, r\rangle$. If an specific event (e.g., change of user location) matches the event pattern, the associated responses are immediately executed, changing the state of the policy. The dashed arrow in Figure 2 depicts an assignment relation created by an obligation.

Threat Model & Assumptions: We assume attackers have control of one or more hosts within the enterprise and are knowledgeable about the environment and defenses. The attacker seeks to compromise devices, exfiltrate data, impersonate users, or disable enterprise services. We assume a worst case scenario where attackers have knowledge of remote code exploits for hosts (e.g., as in NotPetya [6]) and seek to mitigate movement using mandatory controls within the network. Furthermore, for organizations with many small sites, we assume a specific local site can be compromised, leaking enterprise policy and configurations of that site.

Our trusted computing base (TCB) includes the SDN security application, data plane devices, policy engines, and identity management services. We assume administrators securely access the policy configuration interface using TLS-protected communication and multi-factor authentication. We do not consider malicious-but-trusted administrators reconfiguring or updating their authorized system components. The TCB extends to other SDN applications running on the controller not separated from MSNetViews. We assume defenses for known DoS concerns for SDN are deployed.

3 MSNETVIEWS OVERVIEW

Enterprise networks require least-privilege policies that restrict network communication between on-premises hosts. Extending prior solutions to consider multisite environments requires overcoming the following research challenges.

- *Users commonly move between sites, requiring differentiated access based on their location.* The policy and enforcement must dynamically update based on a user's location, and ensure that state is consistent across sites.
- *Compromise of a single site should not leak the global policy.* Security policies are often confidential. An exposed policy at one site should not leak policy details of unrelated sites.
- *Site administrators should only modify policies for their local resources.* Updates to the global policy should be controlled and maintain policy semantics.

We overcame these challenges using a global-local design that simplifies state consistency maintenance. MSNetViews consists of a global manager that serves as a permanent, central leader, which communicates policy and location updates to independent reactive SDN networks. As a result, MSNetViews never requires policies to be merged, eliminating the potential for policy conflicts. Using independent SDN networks also allows internal traffic to continue if the WAN connection fails. In addition to the global-local design, MSNetViews models location policy in a way that allows it to be parameterized, which eliminates full policy updates on each location update event, leading to faster consistency across sites.

Figure 1 shows how MSNetViews operates at both the global and local scales. The global manager coordinates policy management between sites. We envision the global manager residing in a cloud service, though it could be hosted within a site. Site administrators specify their network access control policy directly in the global manager through an administrator console. The global manager needs only to reestablish policy consistency across sites when a policy administrator updates the policy.

At the local scale, each site runs its own independent reactive SDN network with its own SDN controller and local manager to enforce policy. A local manager operates at each site, managing the local policy state, users, and interactions with the global manager. A user's physical movement initiate location and identity update-event, which initiate local policy update in the new-site dynamically. The global manager only propagates the location and identity information to other local-sites, allowing dynamic update in their local policy accordingly.

Modeling Location Policy: Shown in Figure 2, MSNetViews models location-specific policy by creating a separate *policy class* with separate user and object attributes. This separation (a) eases policy management by clarifying the two types of policies and (b) ensures the policy semantics remains intact. To demonstrate the importance for policy semantics, consider the case where both policy classes use object attribute oa, that is, $oa \rightsquigarrow pc_{role}$ and $oa \rightsquigarrow pc_{location}$. Any user that satisfies the location requirement ($u \rightsquigarrow ua$, $ua \rightsquigarrow pc_{location}$, and (ua, AR, oa)) can then access any object o, when $o \rightsquigarrow oa$, regardless of the "Role" policy.

We simplify policy management by not including explicit assignments from users to site-attributes in the global policy. Instead, we use NGAC obligations to create the dynamic assignments between user-device pairs (NGAC users) and user attributes for the "Location" policy class in the local policy. For example, the dashed assignment from $\langle Alice, L1 \rangle$ to S_2 in Figure 2 will be dynamically added in the local sites.

Supporting User Roaming: Users may move between sites (with a travel time ranging from several minutes to days), a process we call *roaming*. A user at a particular site may expect to be able to access local resources (e.g., printers and wireless displays). Conversely, they should *not* be able to access those local resources while not at that site. Other resources (e.g., hardened internal servers) may only be accessible from specific sites.

We present an overview of the roaming process in Figure 1. When the user authenticates to access a new site (e.g., via 802.1x in WiFi enterprise) (step ❶), MSNetViews triggers an event to modify *only* the local policy state via NGAC obligations. The local manager then informs the global manager (step ❷) of the location update event. The global manager, in turn, informs the other sites of the location update (step ❸). This location update event includes mapping the user-device pair to both its new IP address as well as its new site. We discuss roaming in greater detail in Section 4.

Site-Specific Policy Slices: Network administrators commonly consider security policies to be confidential, as they provide valuable intelligence to attackers. A compromise of any individual site should not reveal the global policy. To ensure this, MSNetViews provides per-site policies using a novel policy slicing technique that creates a "need-to-know" policy for each site.

Policy slicing mainly provides two benefits, (1) the attacker only learns the access control policy for the objects (network resources) of the compromised site and (2) it only learns about the users who have access to the compromised site's objects. The reduction in information about users is highly dependent on the type of resources and the policy definition itself. For example, if a site (e.g., the central bank branch) has resources (e.g., an email server) that nearly all employees should access, there will be a minimal reduction in the number of users.

Policy slicing works by identifying resource objects and users of a given site. Our slicing algorithm then traverses the policy graph building marking the subgraph of components connecting the users and resources. That subgraph defines the policy slice that each site will have access to. We discuss policy slicing formally in Section 5.

Policy Administration: Not all administrators can change the entire policy. MSNetViews uses *administrative policies* to limit what each administrator can do. MSNetViews also prevents policy errors using a policy checker that traverses the set of the candidate policies and evaluates then against a policy invariants. Violation of these invariants could lead to conflicts between the multiple policy classes. We discuss policy administration in Section 6.

4 SUPPORTING ROAMING USERS

Geographically distributed enterprises have multiple distinct sites. Business operations frequently allow (or require) users to access local network resources *while physically present at a site.* For example, a policy may permit all users at site S_x to connect to printers and network-connected displays within S_x. Alternatively, a policy may restrict server access to only users at site S_x. This section considers a policy semantics and enforcement system with the flexibility to specify policies for users that move (roam) between multiple sites.[1]

4.1 Dynamic Policy Update

Handling roaming users presents two major policy-related challenges. First, the system must update policies, in real-time, in accordance with changing user locations. Second, those updated policies, and the enforcement of those policies, must be kept consistent across *all* sites in the enterprise. MSNetViews achieves dynamic policy updates using NGAC's concept of *obligations*, which are rules that accept events as input. Based on that input, the policy engine executes a set of pre-defined actions that change the set of enforceable rules. As a result, the administrator-defined, global policy is unchanged, thereby reducing the complexity of maintaining policy consistency across sites.

The MSNetViews policy-specification uses NGAC obligations to manage assignments between user-device pairs (NGAC users) and user attributes for the "Location" policy class. As such, the administrator-defined policy does not include explicit assignments from users to sites. For example, the dashed assignment from $\langle Alice, L1 \rangle$ to S_2 in Figure 2 is dynamically added. By omitting this assignment from the policy specification, MSNetViews needs only reestablish policy consistency across sites when a policy administrator updates the policy.

At each site, external security events are passed to that site's NGAC event processing point (EPP), which causes the policy to execute the actions defined by matching rules. Obligations take the form $\langle ep, r \rangle$ where ep is an event pattern and r is a response. The response, r, is a set of actions. This pair is commonly read as **when** *ep* **do** *r*. NGAC obligations support both internal and external events, where internal events are triggered by policy actions (e.g.,

[1]While we consider location on the granularity of a network site, our policy model is flexible enough to offer finer granularity (e.g., building or room), assuming the network enforcement mechanisms support it (e.g., RoArray [22]).

a policy query allows Alice to read file f) and external events are reported to the policy engine via an outward facing API.

An event pattern ep is a tuple $\langle u, pc, op, pe \rangle$, where a user $u \in U$ is performing an operation $op \in OP$ on a policy element $pe \in PE$ in policy class $pc \in PC$. We model location updates as external events. Each event describes an administrative (u = `admin`) assignment (op = `assign to`) of user-device pairs to the user attribute in the location policy class (pc = `Location`) corresponding to the new site's location (e.g., $pe = S_2$). Upon receipt of a matching event pattern, the policy engine updates the location attribute assignments as appropriate. An environment with n sites requires n obligation rules, which are programmatically generated.

The following example obligation modifies assignments when a user-device pair updates its location to site S_2.

```
when  ⟨admin, Location, assign to, $S_2$⟩
  do  deassign(child_of_assign, $S_1$)
      assign(child_of_assign, $S_2$)
      deassign(child_of_assign, $S_3$)
      ...
      deassign(child_of_assign, $S_n$)
```

Informally, when an administrative event assigns a user-device pair to attribute S_2 in the `Location` policy class, then the policy deassigns that user-device pair from attributes S_1, S_3, …, S_n. `child_of_assign` is a built-in function that returns the user-device pair that was assigned to S_2. We found that external events did not actually create the assignment being matched, therefore we include the assignment in the set of response actions.

4.2 Policy State Consistency Across Sites

Whenever a user roams from one site to another, a location update event is triggered by the local manager of the new site. There is a short post-roaming delay between when the triggering site begins enforcing its new policy rules and when relevant remote sites begin enforcing their new policy rules. This transmission delay is inherent in any distributed system. Fortunately, the resulting policy state inconsistency has limited impact in practice.

Policies can be inconsistent in two generalized scenarios. In the first scenario, a user may be unable to access a resource they should now have access to. As this inconsistency occurs only in the short period after the user attaches to a new network, the impact will be minimal. The local policy engine will shortly catch up with the pending updates, and in worst case, the user will only need to reattempt the connection (e.g., refresh a page in their web browser). In the second scenario, an inconsistency may theoretically allow a user to connect to a network resource they should no longer have access to. In practice, this is not a problem. When a user joins a new site, they will be assigned a different IP address than when in the previous site. Connections from the new IP address will fail. Attempts to forge the old IP address will also fail. If the destination is in the new site, the new site has the updated policy state. If the destination is not in the new site, the packet will not be routed out of the new site. We measure post-roaming stabilization in Section 7.

5 POLICY SLICING

Organizations commonly consider their firewall policies to be confidential [32]. Intuitively, if an attacker has full knowledge of the firewall configuration, they can identify and more easily maneuver towards valuable targets. We mitigate policy exposure by proposing *policy slicing*, a method for distributing policy information based on the need of the site. Therefore, if a single site is compromised, the attacker cannot learn the global policy. It also forces attackers to perform active reconnaissance, which raises the probability of detection. This section presents our policy slicing algorithm and with an explanation of how MSNetViews selectively updates the policy slices for individual sites.

5.1 Policy Slicing Overview

Each site only needs to know how to control access to *local resources*. As such, MSNetViews uses policy slicing to compute and distribute a *site-specific policy* to each site.

Definition 1 (Site-Specific Policy)**.** Given an MSNetViews policy $\mathcal{P}$ and a site S_x, a policy $\mathcal{P}_x$ is a site-specific policy if $\mathcal{P}_x \sqsubseteq \mathcal{P}$ and $\mathcal{P}_x$ only contains the policy elements, assignments, associations, and prohibitions needed to evaluate access control decisions for site S_x. $\sqsubseteq$ denotes the policy elements, assignments, associations, and prohibitions of $\mathcal{P}_x$ are a subset of those in $\mathcal{P}$.

Figure 3 provides an example (excluding location policy class from Figure 2) that depicts the high-level intuition behind policy slicing. Intuitively, the policy slices for site S_1 (Fig. 3b) and site S_2 (Fig. 3c) are created by identifying the objects specific to each site and then finding all of the object attributes, user attributes, and users that refer to those objects by inspecting the assignments, associations, and prohibitions.

As shown in Figure 3b, site S_2 only needs the subset of the policy relevant to resource O_c and O_d. This site-specific policy includes object attributes $S2G$, $S2F$, $S2H$, and $S2K$ are also included for directly or indirectly assigned to resource O_c and O_d. Furthermore, the policy includes user attribute A, because there is an association rule connecting A to $S2F$. Next, the policy includes user $\langle Alice, L1 \rangle$, because $\langle Alice, L1 \rangle \rightsquigarrow A$. The policy slice for site S_1 in Figure 3c is constructed similarly and also considers prohibition relations.

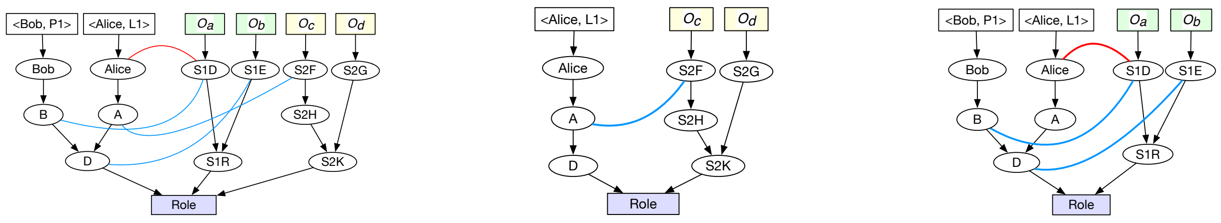
Note that additional policy protection can be achieved by obfuscating the names of user and object attributes. If an administrator wishes to have useful information for debugging, only attributes *without* associations or prohibitions should be obfuscated.

5.2 Policy Slicing Algorithm

The policy slicing algorithm takes as input (1) a global policy, (2) a site S_x, and (3) the set of objects for S_x. It outputs a site-specific policy (Def. 1). The algorithm traverses the global policy to identify all policy elements, assignments, associations, and prohibitions to determine which ones are *relevant* for site S_x.

Definition 2 (Relevant Objects)**.** Let O be the set of objects in the global policy $\mathcal{P}$. The set of relevant objects $O_x \subseteq O$ for site S_x is the set of network resources that reside in site S_x.

The set of *relevant object attributes* are all the object attributes on a path between a relevant object and a policy class. This set is equivalent to all the object attributes with a path to a relevant object. All objects are also object attributes.



(a) Global policy containing sites S_1 and S_2 **(b) Policy slice for site S_2** **(c) Policy slice for site S_1**

Figure 3: Policy slicing example for the policy in Figure 2 (excluding location policy class for simplicity).

Definition 3 (Relevant Object Attributes). Let OA be the set of object attributes in the global policy $\mathcal{P}$ and O_x be the set of relevant objects (Def. 2). The set of relevant objects OA_x is the set $\{oa \mid oa \in OA \land \exists o \in O_x, o \rightsquigarrow oa\} \cup O_x$.

The *relevant associations* and *relevant prohibitions* are those that target relevant object attributes.

Definition 4 (Relevant Associations). Let $\mathcal{R}_a$ be the set of associations in the global policy $\mathcal{P}$ and OA_x be the set of relevant object attributes (Def. 3). The set of relevant associations $\mathcal{R}_{a,x}$ is the set $\{r \mid r \in \mathcal{R}_a \land r = (ua, \_, oa) \land oa \in OA_x\}$.

Definition 5 (Relevant Prohibitions). Let $\mathcal{R}_p$ be the set of associations in the global policy $\mathcal{P}$ and OA_x be the set of relevant object attributes (Def. 3). The set of relevant prohibitions $\mathcal{R}_{p,x}$ is the set $\{r \mid r \in \mathcal{R}_p \land r = (ua, \_, oa) \land oa \in OA_x\}$.

Relevant user attributes require careful consideration. This set includes both the user attributes referenced by a relevant association *and* all of the user attributes on paths between users and policy classes, but only those which traverse through user attributes referenced by a relevant association or prohibition.

Definition 6 (Relevant User Attributes). Let UA be the set of user attributes in the global policy $\mathcal{P}$, $\mathcal{R}_{a,x}$ be the relevant associations (Def. 4), and $\mathcal{R}_{p,x}$ be the relevant prohibitions (Def. 5). Let UA'_x be the set $\{ua \mid ua \in UA \land [\exists(ua, \_, oa) \in \mathcal{R}_{a,x} \lor \exists(ua, \_, oa) \in \mathcal{R}_{p,x}]\}$. The set of relevant user attributes UA_x is $\{ua \mid ua \in UA \land \exists ua' \in UA'_x, (ua \rightsquigarrow ua' \lor ua' \rightsquigarrow ua)\} \cup UA'_x$.

The set of *relevant users* is then straightforward to define. Note that unlike objects, users are not user attributes.

Definition 7 (Relevant Users). Let U be the set of users in the global policy $\mathcal{P}$ and UA_x be the set of relevant user attributes (Def. 6). The set of relevant users U_x is $\{u \mid u \in U \land \exists ua \in UA_x, u \rightsquigarrow ua\}$.

The final information required to create the site-specific policy is the set of *relevant assignments*, which are all of the assignments related to the relevant policy elements.

Definition 8 (Relevant Assignments). Let $\mathcal{R}_e$ be the set of assignments in the global policy and $PE_x = U_x \cup UA_x \cup OA_x \cup PC$ be the set of relevant policy elements (Defs. 3, 6, 7). The set of relevant assignments $\mathcal{R}_{e,x}$ is $\{r \mid r \in \mathcal{R}_e \land \exists e_1, e_2 \in PE_x, r = (e_1, e_2)\}$.

Finally, we do not discuss *relevant obligations*, as we limit our use of obligations to the creation of assignments for managing user roaming between sites (Section 4). We leave the treatment of additional obligation types to future work.

Theorem (Policy Slice Correctness). Let $\mathcal{P}$ be a global policy and $\mathcal{P}_x$ be site S_x's policy slice for objects O_x. Policy slice $\mathcal{P}_x$ is *correct* if there does not exist a user u that has *more* or *less* access rights to an object $o \in O_x$ in $\mathcal{P}_x$ than in $\mathcal{P}$.

Proof. We begin by proving that a user u cannot have *more access rights* in $\mathcal{P}_x$. Access rights are granted by either (a) adding an association that grants access rights; (b) adding an assignment that creates an assignment path from user u to a user attribute that has an association granting access rights; (c) adding an assignment that creates an assignment path from an object o to an object attribute that is the target of an association inherited by user u; or (d) the removal of a prohibition that restricted access rights. Since the policy slicing algorithm does not add any assignments or associations, the first three cases do not apply. Based on the definition of relevant prohibitions (Def. 5), prohibitions are not removed if they reference a relevant object attribute (Def. 3), which are all of the object attributes that have a path from all objects in O_x. Therefore, no prohibitions related to objects in O_x are removed. Hence user u cannot have more access rights in $\mathcal{P}_x$.

Next we prove that a user u cannot have *less access rights* in $\mathcal{P}_x$. Access rights are removed by either (a) adding prohibitions; (b) removing an association that grants access rights; (c) removing an assignment on an assignment path from user u to a user attribute with an association granting access rights; (d) removing an assignment on an assignment path from an object in O_x to an association granting access rights; or (e) removing a user from the policy. Since the policy slicing algorithm does not add any prohibitions, case (a) does cannot occur. Based on the definition of relevant object attributes (Def. 3), case (d) cannot occur. Following this and the definition of relevant associations (Def. 4), case (c) cannot occur. Following this and the definition of relevant user attributes (Def. 6), case (b) cannot occur. Finally, following this and the definition of relevant users (Def. 7), case (e) cannot occur. Therefore, user u cannot have less access rights in $\mathcal{P}_x$. □

5.3 Managing Policy Updates

Network administrators will update the global policy over time. However, not every update will impact all sites. MSNetViews optimizes policy updates by only generating and distributing policy slices for sites impacted by the policy update. Our key intuition is that the set of impacted sites can be easily determined by considering the set of policy elements that are impacted by the policy change. If a site does not contain any impacted policy elements, its policy slice does not need to be regenerated and retransmitted.

Definition 9 (Impacted Policy Elements)**.** Let $\mathcal{P}$ be a global policy and $\mathcal{P}'$ be an update to $\mathcal{P}$. The set of impacted policy elements is determined by the function $\delta(\mathcal{P}, \mathcal{P}')$, which includes: (1) all policy elements added or removed and (2) all policy elements referenced by an assignment, association, or prohibition that has been added, removed, or changed.

We now prove that if the policy $\mathcal{P}_x$ for site S_x does not contain an impacted policy element, then the policy slice for site S_x for $\mathcal{P}$ is identical to its the policy slice for $\mathcal{P}'$.

Theorem (Identical Policy Slice)**.** Let $\mathcal{P}$ be a global policy and $\mathcal{P}'$ be an update to $\mathcal{P}$. Let $\mathcal{P}_x$ and $\mathcal{P}'_x$ be site S_x's policy slices for $\mathcal{P}$ and $\mathcal{P}'$, respectively. Let PE_x be the set of policy elements in $\mathcal{P}_x$. If $\delta(\mathcal{P}, \mathcal{P}') \cap PE_x = \emptyset$, then $\mathcal{P}_x = \mathcal{P}'_x$.

Proof. We provide a proof by contradiction. Assume $\mathcal{P}_x \neq \mathcal{P}'_x$. Let PE_x and PE'_x be the policy elements in $\mathcal{P}_x$ and $\mathcal{P}'_x$, respectively. Recall $\mathcal{P}_x \sqsubseteq \mathcal{P}$ and $\mathcal{P}'_x \sqsubseteq \mathcal{P}'$ by definition. The difference that causes $\mathcal{P}_x \neq \mathcal{P}'_x$ could only result from the following scenarios: (a) adding or removing a policy element, (b) adding or removing an assignment, association, or prohibition, or (c) changing the policy elements referenced by an assignment, association, or prohibition.

If policy element e_1 was added ($e_1 \in PE'_x$ and $e_1 \notin PE_x$), then there must exist an assignment in $\mathcal{P}'_x$ from e_1 to a policy element e_2 where e_2 is in PE_x. This is because all users, objects, and attributes have paths that end in a policy class, and the set of policy classes is fixed. Since e_1 is new, the assignment did not exist in $\mathcal{P}_x$. Therefore $e_2 \in \delta(\mathcal{P}, \mathcal{P}')$. Since e_2 cannot be in both PE_x and $\delta(\mathcal{P}, \mathcal{P}')$, this is a contradiction.

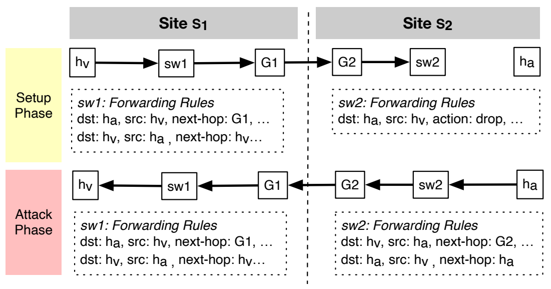
If a policy element e_1 was removed ($e_1 \in PE_x$ and $e_1 \notin PE'_x$), either e_1 was removed from the global policy $\mathcal{P}$ or it was only removed from $\mathcal{P}_x$ but remains in the global policy. If e_1 was removed from the global policy, then e_1 must be in $\delta(\mathcal{P}, \mathcal{P}')$, which is a contradiction. If e_1 was not removed from the global policy, but it was removed from $\mathcal{P}_x$, then there must exist an assignment in $\mathcal{P}_x$ from e_1 to a policy element e_2 where e_2 is in PE_x. This situation can only occur if that assignment was removed, which would cause e_2 to be in $\delta(\mathcal{P}, \mathcal{P}')$. Since e_2 cannot be in both PE_x and $\delta(\mathcal{P}, \mathcal{P}')$, this is a contradiction.

The remaining cases are trivial. If no policy elements are added or removed, and there is an added, removed, or changed assignment, association, or prohibition, it must refer to a policy element e in PE_x. However, e would then also be in $\delta(\mathcal{P}, \mathcal{P}')$, which is a contradiction. Therefore, $\mathcal{P}_x = \mathcal{P}'_x$. □

5.4 Cross-Site Traffic

MSNetViews's site-specific policies ensure that each site only has enough information to enforce access on *local resources*. While this goal minimizes policy disclosure if a site is compromised, it requires an allow-all-outbound policy between sites. However, if not carefully addressed, hosts are vulnerable to unauthorized, cross-site accesses by abusing this allow-all-outbound policy to circumvent the global policy.

Figure 4 depicts a simplified attack scenario where the attacker host h_a is in site S_2 and the victim host h_v is in site S_1. **Setup:** The attacker tricks h_v into making a connection to h_a (e.g., via a resource on a web page). The first packet is allowed to exit S_1 via



Site S1
Site S2
Setup Phase
hv
sw1
G1
G2
sw2
ha
sw1: Forwarding Rules
dst: ha, src: hv, next-hop: G1, …
dst: hv, src: ha , next-hop: hv…
sw2: Forwarding Rules
dst: ha, src: hv, action: drop, …
Attack Phase
hv
sw1
G1
G2
sw2
ha
sw1: Forwarding Rules
dst: ha, src: hv, next-hop: G1, …
dst: hv, src: ha , next-hop: hv…
sw2: Forwarding Rules
dst: hv, src: ha, next-hop: G2, …
dst: ha, src: hv , next-hop: ha

Figure 4: Policy slicing introduces an attack scenario for cross-site traffic. Since sites S_1 and S_2 have incomplete information, and they must allow all outbound traffic.

gateway G_1 but is denied at sw_2 when it enters S_2. At this point, S_1 is configured to allow reply traffic from host h_a. **Attack:** Host h_a now connects to h_v. The allow-all-outbound policy in site S_2 directs S_2's gateway G_2 to forward the packet to site S_1. Switch sw_1 then delivers the packet to h_v, because it appears to be valid return traffic from the original connection.

Note that h_a can connect to *any* port on host h_v, because NetViews [4] optimizes for applications that commonly use multiple connection requests (e.g., HTTP) by allowing any client source port, which avoids redundant flow rules. This attack is not possible in a single-site setting, because the connection to h_a would have been denied and the reply path would not have been set up. Similarly, without a site-specific policy in multisite setting, the global policy would have enforced the policy at the source-site.

The attack is a result of *unverified reverse cross-site traffic*. If site S_2 knows that site S_1 denied the packet, it will not confuse h_a's connection attempt with valid reply traffic. This insight leads us to two possible solutions. The simplest solution is for the destination site to securely inform the originating site that the packet was denied as it entered the site. It will allow the originating site to remove the corresponding forwarding rules. A more elegant solution is for destination sites to include an unforgeable *validation bit* for all reply traffic allowed by the local policy. The gateway could ensure that a designated bit in the IP header is only set for reply traffic. The originating site then will only allow reply-traffic from the trusted destination site if the validation is bit set. The validation bit solution can be efficiently implemented in reactive SDN environments that allow manipulation of headers in addition to forwarding.

6 ADMINISTRATION OF POLICY

Enterprises often have many policy administrators with at least one administrator per site. MSNetViews requires all policy changes to occur within the centralized global manager. We assume a secure admin-console that allows administrators to use their private credentials to access the global policy. This section describes how MSNetViews (1) limits the privileges of individual policy administrators and (2) prevents policy administrators from making errors that unintentionally change the policy semantics.

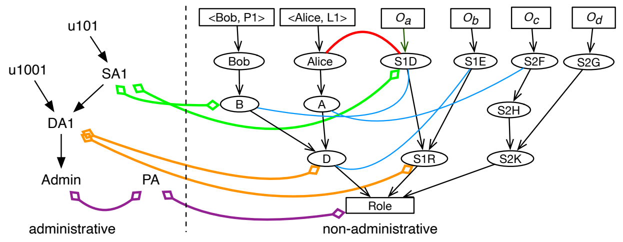


Figure 5: Example administrative policy. The left side defines an administrative policy. The ◇-ended lines show administrative access rights on non-administrative policy (excluded location policy class from Figure 2 for simplicity).

6.1 Administrative Policy

MSNetViews uses NGAC's existing administrative policy model for specifying what actions individual policy administrators can perform. The left side of Figure 5 depicts an example NGAC administrative policy that defines administrative relations to the non-administrative policy.

Administrative Authority: NGAC administrative policies organize administrators into users and user attributes just as with non-administrative policy. By convention, NGAC defines three levels of administrative authority: (1) *Principal Authority* (PA), (2) *Domain Administrator* (DA), and (3) *Subdomain Administrator* (SA). DAs and SAs are subordinate to PAs. A PA is authorized to access the entire global policy. It is the only authority with the ability to create a policy class. The PA can then create and assign subordinate administrators to manage different segments of the global policy. In an enterprise setting, a DA is the appropriate authority for a site administrator, as it manages the policy elements relevant to its own site (as shown in Figure 5).

Administrative Relations: An administrator's ability to update the enterprise policy (i.e., the non-administrative policy) is determined by *administrative relations*, which are analogous to the non-administrative policy relations described in Section 2. Administrative *associations* grant administrative access rights including: (1) addition or deletion of a policy element, (2) addition or deletion of assignments between policy elements, and (3) addition, deletion or update of an association, prohibition, or obligation. Administrative *prohibitions* and *obligations* are the same as their non-administrative counterparts, only with administrative operations.

6.2 Policy Checker

MSNetViews ensures each update from a policy administrator is valid and does not contain errors that unintentionally change the policy semantics. Policy checks are needed for (a) *updating policy elements* (users, objects, object-attributes, or user-attributes) and (b) *updating relations* (assignments, associations, or prohibitions). The MSNetViews uses NGAC policy engine reference implementation (policy-machine-core) [27], which includes checks for all the NGAC-defined syntax and graph-related inconsistencies (e.g., existence of loops). However, we found several limitations with NGAC's existing policy checks.

Missing Checks for Policy Elements: The NGAC documentation [17] states that a policy element should not be declared without a corresponding assignment relation. Such *dangling policy elements* introduce inconsistencies in policy enforcement and mediation of the location-context. We found that the policy-machine-core implementation does not perform this check.

Missing Checks for Policy Relations: The policy-machine-core implementation ensures NGAC's requirement that assignment relations are acyclic, irreflexive, and not defined from an object attribute to an object. However, neither the NGAC documentation or implementation ensure that policy elements have a path to only one policy class, which is important for ensuring *separation between the policy classes.* As discussed in Section 4, sharing user or object attributes between policy classes can unintentionally override rules.

Next, the policy-machine-core performs syntax, redundancy, existence, and similar checks for associations and prohibitions in the NGAC documentation. However, we observe that associations and prohibitions referencing policy elements belonging to multiple policy classes contradict MSNetViews' location-aware policy enforcement. Therefore, additional checks are required.

Table 1: MSNetViews's Additional Policy Check

Rule	Purpose
(1) Dangling PE	Each $pe \in PE$ must lead to at least one $pc \in PC$
(2) Exclusive UA	Each $ua \in UA$ must lead to only one $pc \in PC$
(3) Exclusive OA	Each $oa \in OA$ must lead to only one $pc \in PC$
(4) Exclusive Associations	The source and target attributes of an association relation must lead to same policy class.
(5) Exclusive Prohibitions	The source and target attributes of a prohibition relation must lead to same policy class.

Policy Check Enhancements: Table 1 lists the five policy checks we added the policy-machine-core. Rule (1) ensures the policy does not include any dangling policy elements that do not lead to a policy class. Rules (2) and (3) ensure that user and object attributes can only lead to one policy class. Finally, Rules (4) and (5) ensure that associations and prohibitions do not combine attributes from two different policy classes, which ensures the separation of the two policies. Please refer to our online appendix [3] for more detail.

7 PERFORMANCE EVALUATION

In this section, we evaluate the performance of MSNetViews through the following research questions:

- **Q1** How does MSNetViews enforcement impact system latency and throughput? (Section 7.2)
- **Q2** How long does MSNetViews take to stabilize after a user roams between sites? (Section 7.3)
- **Q3** How expensive are policy checking and update operations, and do they scale? (Section 7.4)

7.1 Network Emulation Methodology

Emulated Network Design: We emulate our enterprise networks using Mininet [44] and ONOS [20]. Because no enterprise has publicly released its network topology, we model a multisite enterprise as two sites with identical topologies. Performing the performance evaluation with only two sites is sufficient, as it represents the case where all sites have direct WAN connections to one another. We emulate this site-to-site connection with a low-overhead GRE

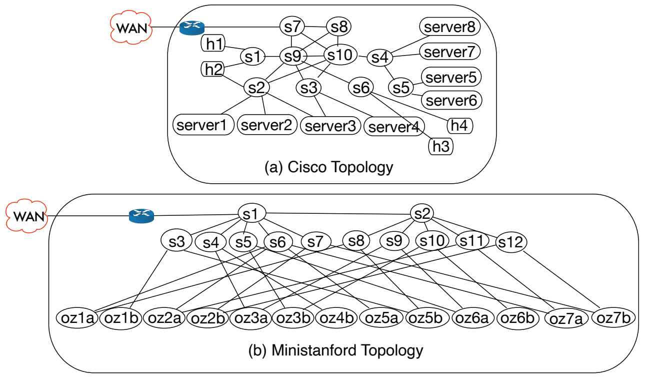


Figure 6: Evaluation topologies for MSNetViews. We replicate the same topology at each site, selecting from either MiniStanford [49] or Cisco [49]. MiniStanford [49] is a Stanford backbone network with 100 devices and 25 switches. Cisco [49] is a network of an enterprise with Cisco PIX firewall with 12 devices and 10 switches.

tunnel over a 10GbE WAN link. We vary the WAN latency to emulate connections between sites in the same city (`WashingtonDC`↔`WashingtonDC`, 1 ms), same region (`WashingtonDC`↔`NY`, 11.2 ms), and across the Atlantic (`WashingtonDC`↔`Copenhagen(CP)`, 105 ms). These latency values come from WonderNetwork's global `ping` stats [47], and the specific cities were chosen so that the latencies were roughly round numbers an order of magnitude apart. Within sites, we use the same topologies used in prior work [4], shown in Figure 6.

Emulated Network Traffic: To measure overhead in cross-site communication, we program hosts to connect to other hosts only at other sites (no intra-site communication). For the Cisco topology, we initiate connections between all multi-site pairs of hosts. For the MiniStanford topology, we randomly select hosts to connect across sites to create a total of 1000 simultaneous flows. 1000 flows was the empirically determined safe limit of our experiment hardware before CPU contention of the host VM affected latencies.

System Configuration: Similar to prior work [10, 21], we use a Docker container for each site. Within each container, an ONOS SDN controller manages an SDN network emulated by Mininet [44]. Our experiments spawned these containers on a VM on a server-class host with a AMD EPYC 7302P processor. The VM ran Ubuntu 20.04 LTS (Linux Kernel 5.4.0) and had 15 cores and 235 GB RAM. In all experiments, we measured latency using ping and throughput using perf3 (version 3.7). Each experiment ran for 60 seconds and was repeated 20 times. We present the average over these runs. Before each run, we clear the ONOS controller of any flow rules to establish a repeatable, "clean-slate" measurement. We start with a single flow, introducing a new flow every 100 ms.

7.2 MSNetViews Overhead

We compare MSNetViews with standard intent forwarding (`ifwd`) and NetViews [4]. `ifwd` serves as a baseline of minimum network performance without the effects of MSNetViews. Our comparison with NetViews demonstrates multisite scalability. For these experiments, there are no roaming or policy update events. Hence, the overhead comes from installing the SDN rules. As MSNetViews uses reactive SDN, forwarding rules are installed only when triggered by first packet of a new flow resulting higher overhead of the first packet than subsequent packets [4]. Our goal is to show that MSNetViews performs similar to NetViews, indicating minimal overhead for adding multisite functionality.

For all studied WAN latencies, the 1^{st}-packet latency overhead of MSNetViews over NetViews remained well under 1.2% for Cisco and 2.5% for Ministanford (see Figure 7a). When compared to `ifwd` on same-city (`DC`↔`DC`) traffic, MSNetViews increased 1^{st}-packet latency by 7 ms (≈5x) on the MiniStanford topology and by 9.6 ms (≈4x) on Cisco (see Figure 7a). In the same region (`DC`↔`NY`) , the measured overhead compared to `ifwd` was 5.8 ms to 7.5 ms (49.1% to 54.5%), and for the global scale (`DC`↔`CP`) , the overhead compared to `ifwd` was 6 ms to 7.3 ms (5.7% to 6.8%) across the two topologies. As the sites move further apart, the overhead decreases proportionately as the WAN latency dominates. Furthermore, latency overhead is limited to the 1^{st}-packet only and does not propagate to subsequent packets (Figure 7b). The n^{th}-packet latencies for NetViews, `ifwd`, and MSNetViews is similar, with a small overhead (< 0.025 ms).

Figure 8 shows throughput results for `DC`↔`DC` and `DC`↔`CP`. For both cases, MSNetViews and NetViews are comparable, with a difference of less than 1% in the median for both Cisco and Ministanford topologies (Figure 8). For `ifwd` in the `DC`↔`DC` case with the Cisco topology, the median MSNetViews throughput falls within the inter-quartile ratio of `ifwd` and the difference in medians is under 1% (Figure 8). For `DC`↔`DC` with the MiniStanford topology, MSNetViews's median aggregate throughput is approximately ≈150 Mbps below `ifwd`, a 5.3% decrease (Figure 8). For `DC`↔`CP`, the median MSNetViews throughput falls within the inter-quartile ratio of `ifwd` for both topologies. The results suggest that MSNetViews does not reduce network throughput compared to NetViews and `ifwd`.

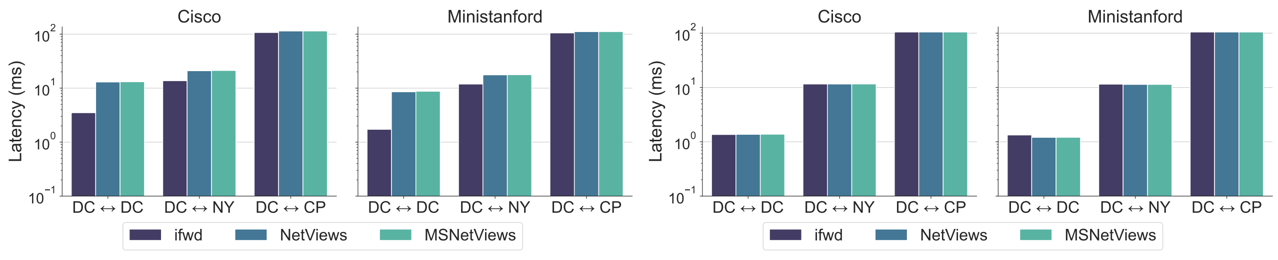
7.3 Post-Roaming Stabilization

When a user roams to a new site, both local and global policy updates occur (Section 4). We measured the time for the policy state in all sites to stabilize to a consistent state after a user location is updated. For a worst-case estimate, we evaluated post-roaming stabilization for the global scale scenario (`DC`↔`CP`), which has the longest inter-site latency. Note that updates use NGAC obligations; hence do not invoke the policy checker or policy slicer.

We define *location update time* as the time difference between when the roaming user $\langle Alice, L1\rangle$ starts the authentication process at a new site, y, and when a location and identity update has been triggered in all n *relevant* sites of the enterprise (steps ❶ and ❸, in Figure 1). This measurement is the worst-case time for the user's location to be consistent at any relevant site.

We studied the impact of two factors on the *location update time*: (1) the number of *relevant* sites in the enterprise, and (2) the number of roaming users concurrently authenticating to a new site. For these experiments, we considered the MiniStanford topology.

Relevant Sites: We emulated moving a single user from one site to another, and varied number of relevant sites in the enterprise exponentially from 2 through 16. As shown in Figure 9a, the time to complete the roaming process remained under 1.5 s across all cases. This time is *only* experienced when the user needs to access non-local resources (resources at other sites) immediately after joining a new site. When accessing local resources, users need not wait for



(a) Average Initial Packet Latency

(b) Average n^{th} Packet Latency

Figure 7: Average end-to-end packet latency for MSNetViews, NetViews [4], and ifwd under three WAN latencies between sites: (1) same city (`WashingtonDC↔WashingtonDC`), same region (`WashingtonDC↔NY`), and global (`WashingtonDC↔Copenhagen(CP)`).

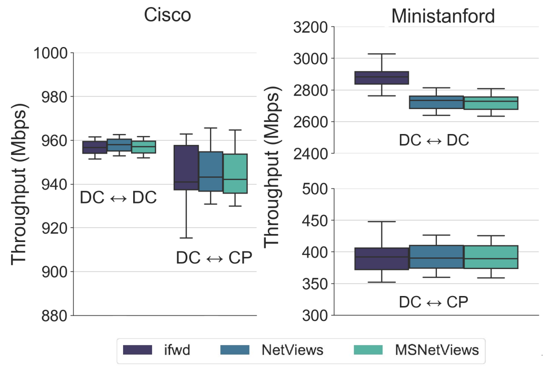
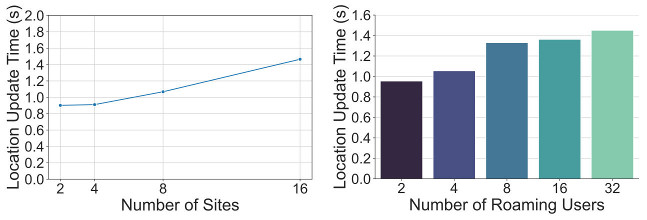


Figure 8: Aggregate throughput for MSNetViews, NetViews, and ifwd under two WAN latencies between sites: (1) same city (`WashingtonDC↔WashingtonDC`), and global (`WashingtonDC↔Copenhagen(CP)`). The scales differ for readability.



(a) Location update time of one roaming user as a function of number of *relevant* sites

(b) Avg. location update time per user as a function of number of users roaming between two sites

Figure 9: Effect of number of roaming users and number of *relevant* sites on average location update time per user for users roaming globally (between `WashingtonDC↔Copenhagen(CP)`). Update events are not batched.

the update to propagate through the enterprise. This time is small and has minimal impact on user experience (impact is similar to that of refreshing a page on a web browser).

Roaming Users: In this experiment, we restricted the number of sites to two, and exponentially varied the number of users moving between sites from 2 through 32. Figure 9b show the average location update time for each set of roaming users. The median update time for all users remained under 1.5 seconds, which is minimal impact for users joining a network after roaming from another site Finally, location updates can also be batched to support many more simultaneous location update events.

7.4 Policy Update Performance

Administrative updates to the global policy trigger two operations that impact performance: (1) checking the policy for syntax errors—an operation performed by the policy checker (Section 6.2)—and (2) computing the new policy slices (Section 5)—an operation performed by the policy slicer. Note that, the above mentioned operations are not initiated for a user roaming events.

Methodology: We evaluate the performance of *global manager in the event of administrative policy update* by experimenting individually on the policy checker and policy slicer components. We used the policy graph of the MiniStanford topology for five component sites, with a host count varied within the range of 100 to 10000. We run this experiment in the same VM host as our previous experiments. To generate self-consistent policies, we rely on the *random policy creation* algorithm as in prior works [4, 33]. This algorithm creates a binary tree-like connectivity among host nodes (user-device pairs and objects) and policy elements. The height of the policy graph (binary tree) and host nodes determines the policy graph complexity (number of policy nodes). The number of policy nodes is determined by calculating $(2^{h+1} - 1) \times n$, where n is the number of hosts and h is the height of the policy graph. The maximum value of the graph height was two for our experiments.

We created a *policy variant generator*, which can create graph variations from a supplied base policy graph using one or more randomly selected update operations. We considered ten types of update operations among possible fourteen types (e.g., add user node, delete association). We avoided the delete operation on attribute nodes and assignment relations for the ease of experiment, as these delete operations can produce a disconnected policy graph. Finally, we generated 50 policy variants of each type of update operation for each run.

Results: As shown in in Table 2, the policy graph complexity is a significant factor in policy checking and slicing delay. This is expected because the policy checking and slicing algorithms traverse the policy at-least twice. Note that this experiment uses the worst-case algorithm (binary tree) for policy generation.

Shown in Table 2, the average delays for both the policy checker and slicer follow similar trends and are marginal even for a significant policy graph with 10,000 host nodes and 30,000 policy nodes

Table 2: Effect of Policy Graph Complexity on Average Policy Checking and Slicing Delay

Host No.	Policy Node No.	Average Delay (ms)	
		Policy Checker	Policy Slicer
100	300	3	6
100	700	6	9
1000	3000	25	38
1000	7000	62	81
4000	12000	151	189
4000	28000	452	516
7000	21000	388	428
7000	49000	1153	1024
10000	30000	654	688
10000	70000	2441	1883

(0.64 seconds and 0.68 seconds, for checking and slicing, respectively). We also analyzed if the operation types affect the policy update overhead. The standard deviation for the 10 operation types varies from 6 to 10 ms, and for policy slicing it varies from 25 to 35 ms for different numbers of host and policy nodes.

For each update, MSNetViews regenerates policy slices; however, it is not necessary to generate new slices for sites unimpacted by the change (Section 5.3). On average, policy slicing occupies 30% to 40% of the total delay; the rest of the time is needed to load the new policy graph and find the differences from the base graph.

8 MSNETVIEWS AND ZERO TRUST

NIST defines ten "Network Requirements to Support ZTA" in Section 3.4.1 of their special article on Zero Trust Architecture [40]. MSNetViews satisfies all but two of the ten requirements. One of these requirements out-of-scope: NIST states enterprise resources should be accessible without needing to traverse enterprise network infrastructure. While this applies to business applications, the on-premises devices protected by MSNetViews cannot be moved off-premises. MSNetViews partially addresses the second requirement it fails to satisfy, which states "the enterprise must be able to distinguish between what assets are owned or managed by the enteprise and the devices' current security posture." MSNetViews does capture which devices are managed by the enterprise; however, it does not incorporate the current security posture of devices. Existing zero trust solutions accomplish this requirement using device attestation and behavioral analytics. Such information can be incorporated into MSNetViews's policy decision and is a straightforward engineering effort for future work.

9 RELATED WORK

Enterprise Access Control: Enterprises seeking to establish *least privilege* access control require well-defined policy definitions. Traditionally, role- and group-based policy definitions have been used to simplify policy management and model user requirements (e.g., Ethane [12], CISCO DNA [14]). However, these prior systems do not capture modern enterprise requirements like dynamic policy definition, multi-context access control, or encoding enterprise configuration [41, 42]. To address these limitations, attribute-based access control (ABAC) [26] and several extensions (e.g., NGAC [15], XACML [18]) were proposed. These enabled granularity and context-awareness, however, most of these approaches consider file-based resources and web services.

Firewalls [13] form the foundation of traditional enterprise network policy enforcement. Firewalls are hard to configure and maintain [48], and even stateful firewalls are prone to vulnerabilities [30]. Without a unified solution, administrators are forced to maintain enterprise security in piecemeal (e.g., VLAN, middle-boxes [49], next generation firewall [36], intrusion detection systems [43]). Some systems combine different state-of-the-art technologies (e.g., SDN) to provide isolation (e.g., PSI [49] or micro-segmentation [23]).

Modern enterprise network security lacks a definitive, context-aware, and granular access control solution. Some solutions provide system support for dynamic policy enforcement with non-existent [12, 35, 49] or non-scalable [28] policy definitions. Several efforts have recognized that the ease of creating, testing, and enforcing access control policies [14, 29] is vital for network security. However, the policy enforcement of these efforts relies primarily on traditional network segmentation or micro-segmentation for firewall placement. Furthermore, other systems address context-aware access control considering different dynamic aspects (e.g., location, identity) of the enterprise [24]. Recently, Anjum et al. [4] proposed a system designed to handle user contexts of enterprise networks; this includes the use of NGAC-supported [15] least-privilege access control by leveraging reactive SDN for policy enforcement.

Zero Trust Architecture (ZTA): ZTA is becoming the gold standard of enterprise security, where no trust between any entities is assumed unless explicitly specified [40]. Google's BeyondCorp [46] has led the industry conversation on Zero Trust, which has inspired subsequent efforts including Software Defined Perimeters (SDP) by the Cloud Security Alliance [34] and BastionZero [50]. Academic literature also leveraging ZTA concepts for secure network environments (e.g., server-to-server access in PagerDuty [16], access control for 5G networks [7]). However, most ZTA efforts concentrate on web applications and require all assets to be moved to the cloud. How ZTA can be incorporated in securing the on-premises entities remains an open question.

Software Defined Networking (SDN): Reactive SDN with a programmable switch architecture [8, 9] and high-level programming languages [45] has the potential to address many enterprise access control issues [31]. Although research has identified attacks against SDN technologies [11, 51], it can replace conventional security components through providing granular flow analysis [37] and defense techniques [5]. SDN has the potential to simplify enterprise policy enforcement [4], and provide software defined perimeters [23, 34].

10 CONCLUSION

Increased interest in zero trust architectures is changing the decades-old "mote-and-gate" approach to enterprise network security. While reactive SDN technologies provide a promising foundation for realizing zero trust goals within on-premises enterprise networks, they are limited to single networks, requiring enterprises with multiple sites to manually ensure consistency of security policy across all sites. This paper presented MSNetViews, an system that extends a single, globally-defined and managed, enterprise network security policy to many geographically distributed sites. Each site operates independently and enforces a site-specific policy slice that is dynamically parameterized with user location as employees roam between sites. In presenting MSNetViews, we demonstrate that

SDN can not only provide an invaluable primitive for achieving zero trust within a single enterprise location, but also across many geographically distributed locations.

ACKNOWLEDGMENTS

This work was supported in part by ONR grant N00014-20-1-2696. Any findings and opinions expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] Rashid Amin, Martin Reisslein, and Nadir Shah. 2018. Hybrid SDN Networks: A Survey of Existing Approaches. *IEEE Communications Surveys & Tutorials* 20, 4 (2018).

[2] J. P. Anderson. 1972. *Computer Security Technology Planning Study.* Technical Report. Air Force Electronic Systems Division.

[3] Iffat Anjum. 2023. *MSNetviews Online Appendix.* https://doi.org/10.5281/zenodo.7871808

[4] Iffat Anjum, Daniel Kostecki, Ethan Leba, Jessica Sokal, Rajit Bharambe, William Enck, Cristina Nita-Rotaru, and Bradley Reaves. 2022. Removing the Reliance on Perimeters for Security using Network Views. In *Proceedings of the ACM Symposium on Access Control Models and Technologies (SACMAT).*

[5] Iffat Anjum, Mu Zhu, Isaac Polinsky, William Enck, Michael K. Reiter, and Munindar P. Singh. 2021. Role-Based Deception in Enterprise Networks. In *Proceedings of the ACM Conference on Data and Application Security and Privacy.*

[6] MITRE ATT&CK. 2019. NotPetya. https://attack.mitre.org/software/S0368/.

[7] Chafika Benzaïd, Tarik Taleb, and Muhammad Zubair Farooqi. 2021. Trust in 5G and Beyond Networks. *IEEE Network* 35, 3 (2021).

[8] Giuseppe Bianchi, Marco Bonola, Antonio Capone, and Carmelo Cascone. 2014. OpenState: Programming Platform-Independent Stateful Openflow Applications inside the Switch. *ACM SIGCOMM Computer Communication* 44, 2 (April 2014).

[9] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-independent Packet Processors. *ACM SIGCOMM Computer Communication Review* 44, 3 (July 2014).

[10] Claudio Calcaterra, Alessio Carmenini, Andrea Marotta, and Dajana Cassioli. 2019. Hadoop performance evaluation in software defined data center networks. In *Proceedings of the International Conference on Computing, Networking and Communications (ICNC).*

[11] Jiahao Cao, Renjie Xie, Kun Sun, Qi Li, Guofei Gu, and Mingwei Xu. 2020. When Match Fields Do Not Need to Match: Buffered Packets Hijacking in SDN. In *Proceedings of the Network and Distributed System Security Symposium (NDSS).*

[12] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. 2007. Ethane: Taking Control of the Enterprise. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM).*

[13] William R. Cheswick, Steven M. Bellovin, and Aviel D. Rubin. 2003. *Firewalls and Internet Security: Repelling the Wily Hacker.* Addison-Wesley Professional.

[14] Cisco. 2022-03-09. Cisco DNA Center - Cisco DNA Center 2.3.2.0 Data Sheet. https://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/dna-center/nb-06-dna-center-data-sheet-cte-en.html.

[15] Wayne Jansen David Ferraiolo, Serban Gavrila. 2015. Policy Machine: Features, Architecture, and Specification. NISTIR 7987 Rev. 1. https://csrc.nist.gov/publications/detail/nistir/7987/rev-1/final.

[16] Doug Barth Evan Gilman. July 2017. *Zero Trust Networks.* O'Reilly Media, Inc.

[17] David Ferraiolo. 2019. Unpacking Next Generation Access Control (NGAC) and Tetrate Q. TETRATE. https://www.tetrate.io/blog/unpacking-next-generation-access-control-ngac-and-tetrate-q/.

[18] David Ferraiolo, Ramaswamy Chandramouli, Rick Kuhn, and Vincent Hu. 2016. Extensible Access Control Markup Language (XACML) and Next Generation Access Control (NGAC). In *Proceedings of the ACM International Workshop on Attribute Based Access Control (ABAC).*

[19] FireEye. 2020. Highly Evasive Attacker Leverages SolarWinds Supply Chain to Compromise Multiple Global Victims With SUNBURST Backdoor. https://www.fireeye.com/blog/threat-research/2020/12/evasive-attacker-leverages-solarwinds-supply-chain-compromises-with-sunburst-backdoor.html.

[20] Open Networking Foundation. 2018. ONOS (Open Network Operating System). https://onosproject.org/.

[21] Dewang Gedia and Levi Perigo. 2018. Performance evaluation of SDN-VNF in virtual machine and container. In *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN).*

[22] Wei Gong and Jiangchuan Liu. 2019. RoArray: Towards More Robust Indoor Localization Using Sparse Recovery with Commodity WiFi. *IEEE Transactions on Mobile Computing* 18, 6 (2019).

[23] Stephen Gutz, Alec Story, Cole Schlesinger, and Nate Foster. 2012. Splendid Isolation: A Slice Abstraction for Software-Defined Networks. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks (HotSDN).*

[24] Sungmin Hong, R. Baykov, Lei Xu, Srinath Nadimpalli, and G. Gu. 2016. Towards SDN-Defined Programmable BYOD (Bring Your Own Device) Security. In *Proceedings of the Network and Distributed System Security Symposium (NDSS).*

[25] The White House. 2021. Executive Order on Improving the Nation's Cybersecurity. https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/.

[26] V. C. Hu, D. R. Kuhn, D. F. Ferraiolo, and J. Voas. 2015. Attribute-Based Access Control. *Computer* 48, 2 (Feb 2015).

[27] Akash Shah Joshua Roberts. 2019. Policy Machine Core. GitHub. https://github.com/PM-Master/policy-machine-core.

[28] N. Kang, O. Rottenstreich, S. G. Rao, and J. Rexford. 2017. Alpaca: Compact Network Policies With Attribute-Encoded Addresses. *IEEE/ACM Transactions on Networking* 25, 3 (June 2017).

[29] Charalampos Katsis, Fabrizio Cicala, Dan Thomsen, Nathan Ringo, and Elisa Bertino. 2021. Can I Reach You? Do I Need To? New Semantics in Security Policy Specification and Testing. In *Proceedings of the ACM Symposium on Access Control Models and Technologies (SACMAT).*

[30] Amit Klein. 2022. Subverting Stateful Firewalls with Protocol States. In *Proceedings of the Network and Distributed System Security Symposium (NDSS).*

[31] Dan Levin, Marco Canini, Stefan Schmid, Fabian Schaffert, and Anja Feldmann. 2014. Panopticon: Reaping the Benefits of Incremental SDN Deployment in Enterprise Networks. In *USENIX Annual Technical Conference (USENIX ATC).*

[32] A. X. Liu. 2008. Formal Verification of Firewall Policies. In *2008 IEEE International Conference on Communications.*

[33] Peter Mell, James M. Shook, and Serban Gavrila. 2016. Restricting Insider Access Through Efficient Implementation of Multi-Policy Access Control Systems. In *Proceedings of the ACM CCS International Workshop on Managing Insider Security Threats (MIST).*

[34] A. Moubayed, A. Refaey, and A. Shami. 2019. Software-Defined Perimeter (SDP): State of the Art Secure Solution for Modern Networks. *IEEE Network* 33, 5 (2019).

[35] Ankur Kumar Nayak, Alex Reimers, Nick Feamster, and Russ Clark. 2009. Resonance: Dynamic Access Control for Enterprise Networks. In *Proceedings of the ACM Workshop on Research on Enterprise Networking (WREN).*

[36] K. Neupane, R. Haddad, and L. Chen. 2018. Next Generation Firewall for Network Security: A Survey. In *Proceedings of the SoutheastCon (SECON).*

[37] Tj OConnor, William Enck, W. Michael Petullo, and Akash Verma. 2018. PivotWall: SDN-Based Information Flow Control. In *Proceedings of the Symposium on SDN Research (SOSR).* ACM.

[38] The University of Adelaide. 2010. The Internet Topology Zoo. http://www.topology-zoo.org/contact.html

[39] Executive Office of the President. 2022. Moving the U.S. Government Toward Zero Trust Cybersecurity Principles. Memorandum. https://www.whitehouse.gov/wp-content/uploads/2022/01/M-22-09.pdf.

[40] Scott Rose, Oliver Borchert, Stu Mitchell, and Sean Connelly. 2019. Zero trust architecture. National Institute of Standards and Technology. https://csrc.nist.gov/publications/detail/sp/800-207/final.

[41] Ravi Sandhu. 1996. Roles versus Groups. In *Proceedings of the ACM Workshop on Role-Based Access Control (RBAC).*

[42] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. 1996. Role-based access control models. *Computer* 29, 2 (1996).

[43] R. Talpade, G. Kim, and S. Khurana. 1999. NOMAD: traffic-based network monitoring framework for anomaly detection. In *Proceedings of the IEEE International Symposium on Computers and Communications.*

[44] Mininet Team. 2018. Mininet An Instant Virtual Network on your Laptop (or other PC). http://mininet.org/.

[45] David Walker. 2014. NetkAT: Semantic foundations for networks. In *Proceedings of the Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL).*

[46] Rory Ward and Betsy Beyer. 2014. BeyondCorp: A New Approach to Enterprise Security. *;login:* 39, 6 (2014).

[47] WonderNetwork. 2022. Global Ping Statistics. https://wondernetwork.com/pings.

[48] A. Wool. 2004. A quantitative study of firewall configuration errors. *Computer* 37, 6 (2004).

[49] Tianlong Yu, Seyed Fayaz, Michael Collins, Vyas Sekar, and Srinivasan Seshan. 2017. PSI: Precise Security Instrumentation for Enterprise Networks. In *Proceedings of the Network and Distributed System Security Symposium (NDSS).*

[50] Bastion Zero. 2021. BastionZero's Multi Root Zero-Trust Access Protocol (MrZAP). https://github.com/bastionzero/whitepapers/blob/5ac531a3a3831a7995bb4319281d5da9e4bc7099/mrzap/README.md.

[51] Menghao Zhang, Guanyu Li, Lei Xu, Jun Bi, Guofei Gu, and Jiasong Bai. 2018. Control Plane Reflection Attacks in SDNs: New Attacks and Countermeasures. In *Research in Attacks, Intrusions, and Defenses.* Springer International Publishing.