Removing the Reliance on Perimeters for Security using Network Views

Iffat Anjum
NC State University
Raleigh, NC, USA

Daniel Kostecki
Northeastern University
Boston, MA, USA

Ethan Leba
Northeastern University
Boston, MA, USA

Jessica Sokal
Northeastern University
Boston, MA, USA

Rajit Bharambe
NC State University
Raleigh, NC, 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

Traditional enterprise security relies on network perimeters to define and enforce network security policies. Emerging application-focused Zero Trust architectures attempt to address this long-standing challenge by moving business applications to the cloud and performing enhanced identity and access control checks within a web gateway. However, these solutions ignore the security needs of workstations, development servers, and device management interfaces. In this work, we propose *Network Views* (abbrev. NetViews) for least-privilege network access control where each host has a different, limited view of the other hosts and services within a network. We present an SDN-based design and demonstrate that our implementation has network latency and throughput comparable to baseline reactive forwarding. We further provide an optimization for multi-connection flows that significantly reduces both redundant access control checks and forwarding state storage in switches. As such, NetViews provides a practical primitive for removing the reliance on security perimeters within enterprise networks.

CCS CONCEPTS

• **Networks** → **Network manageability**; • **Security and privacy** → **Network security**.

KEYWORDS

enterprise network security, zero trust architectures, software-defined networking, least-privilege

ACM Reference Format:
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 27th ACM Symposium on Access Control Models and Technologies (SACMAT) (SACMAT '22), June 8–10, 2022, New York, NY, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3532105.3535029

1 INTRODUCTION

Enterprise networks traditionally rely on perimeters for defense. Perimeters provide boundaries both at the WAN access edge as well as at critical points within the network. However, this moat-and-gate approach for network access control has not aged well as the functional needs of enterprise networks have evolved. This lack of defense-in-depth enables both ransomware and advanced persistent threat (APT) adversaries to advance towards critical targets inside the network (e.g., Solorigate [20], NotPetya [4]).

Zero Trust architectures (e.g., NIST SP 800-207 [57]) remove the reliance on network perimeters for defense. The Zero Trust model promoted by Google's BeyondCorp [68] and the recent US Whitehouse memo M-22-09 [55] focuses on applications, removing the network from consideration altogether. In this idealized environment, a cloud-based web application gateway authenticates users, the accessing device, and uses behavior analytics to determine if the request should be forwarded to the target web application.

However, even if business applications can be moved into the cloud, on-premeses networks cannot be ignored. Enterprise networks often contain on-premises development servers, file servers, and device management interfaces. Furthermore, both Solorigate and NotPetya spread using network services commonly left open on workstations, motivating stronger defenses *within* networks. Indeed, both SP 800-207 and M-22-09 note the potential need for logical micro-segmentation as an isolation strategy. However, micro-segmentation does not solve the root of the problem. Rather, it exposes the complex communications needs of real networks, requiring network engineers to continually redefine new network boundaries and firewall rules between them. In contrast, we argue that *networks should define defense at the granularity of individual devices,* embracing the communication needs of those devices and extending the existing flexible and dynamic enterprise role- and attribute-based policies into the network.

The technology needed to achieve this vision exists and is commercially available. Software Defined Networking (SDN) technologies such as OpenFlow [42] and P4 [7] provide opportunities to create flexible and reactive policies. However, prior work has focused primarily on enforcement mechanisms, identifying novel ways of using forwarding rules to improve performance (e.g., FlowTags [16], PSI [71], Alpaca [32], Kinetic [35]). Proposals that do consider access control (e.g., Ethane [11], FML [26], Alpaca [32]) do not capture the dynamicity and scalability requirements of enterprise environments.

In this paper, we propose *Network Views* (NetViews for short) as an abstraction and model for access control within enterprise networks that provides a fine-grained least-privilege network access control. In this model, each host has a different "view" of what other hosts and services exist in the network. We designed and built a prototype of NetViews and find that even on a heavily loaded network, both new and established flows have latency and throughput comparable to baseline reactive forwarding. Our security analysis on a concrete reference topology demonstrates the effectiveness of NetViews in reducing reachability, improving overall security.

We make the following contributions in this paper.

- *We propose an access control model supporting the NetViews abstraction.* The model is based on NIST's Next Generation Access Control (NGAC) policy language [19] and allows flexible management of network connectivity.
- *We design and implement NetViews as an SDN application.* NetViews leverages ONOS's Intent [22] framework for efficient management of forwarding rules in switches. Our Mininet-based performance evaluation using three representative topologies shows latency and throughput comparable to baseline reactive forwarding.
- *We provide a multi-connection optimization that significantly reduces redundant access control policy checks and forwarding state in switches.* We show that a 5-tuple flow definition (s_{ip}, s_{port}, d_{ip}, d_{port}, $proto$) of access control is unnecessarily strict, and that allowing any s_{port} does not sacrifice security. This optimization significantly reduces the tertiary content addressable memory (TCAM) requirements for switches.

Note that throughout the paper we assume the enterprise network has been fully provisioned with a reactive SDN technology such as OpenFlow. However, a full deployment is not necessary in practice. Brockelsby and Dutta [8] recently found that most network traffic in access switches in university networks is north-south. In doing so, they show that legacy switches with VLAN capabilities can be used to isolate all traffic in the access switch layer, forcing it to a more capable distribution switch with SDN capabilities. We leave the exploration of such architectures to future work.

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

2 MOTIVATION

The goal of Zero Trust is to "*prevent unauthorized access to data and services* coupled with making the *access control enforcement as granular as possible*" [57]. Emerging application-focused solutions such as BeyondCorp [68] move business applications to the cloud and place them behind web gateways that perform enhanced identity and access control checks. However, such application-focused solutions ignore the workstations, development servers, and device management interfaces that remain in the on-premises network. Figure 1 shows an example enterprise network consisting of several floors of a building. Consider a traditional network architecture where the dashed boxes represent coarse security domains, where firewalls enforce access control at the perimeter of each domain.

Organizations struggle with the growing threats of ransomware and APT attacks, where the attacker initially gains access to a single host. The attacker then is able to move freely within the

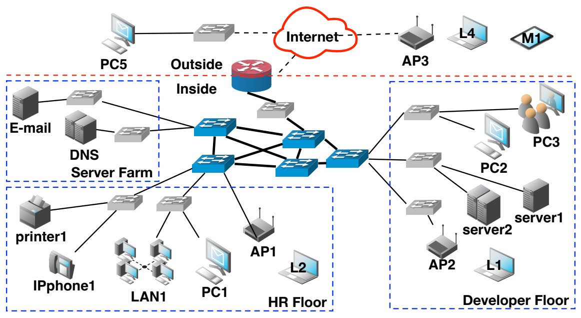


Figure 1: Example Enterprise Network

perimeter until it finds a host that is allowed to connect to a host in a different perimeter. It progressively moves across the network until it achieves its goal of accessing high-value information and resources like customer data or code repositories.

NotPetya, a malware attack that dominated the year 2017 [4, 13], took advantage of the lack of least privilege policy and granular enforcement. NotPetya propagated through a combination of vulnerabilities in the Microsoft SMB service running on specific TCP ports (tcp/139, tcp/445) as well as compromised user credentials. These stolen credentials gave the attacker additional authority to exploit other reachable hosts via the vulnerable file and network information sharing service. Later, in the 2020 Solorigate event [20, 44], attackers used a remote PowerShell (tcp/5985, tcp/5986), which is often enabled on the hosts to allow administrators to scale administration tasks. However, due to of the lack of the least privilege policy within the network, those remote PowerShell commands could originate from anywhere in the network, not necessarily from machines associated with specific network administrators. These examples demonstrate the need for a new network access control abstraction to severely limit or slow down lateral movement. In both cases, the specific network ports on hosts were left open for administrative purposes. While host-based firewalls could theoretically limit access by IP address, managing those in mass deployments is complicated and error-prone. In contrast, a network-centric enforcement of fine-grained, least-privilege policies would significantly reduce the attack surface.

Enforcing fine-grained, least-privilege network connections between hosts requires a corresponding access control policy. Existing methods of specifying firewalls will not scale to the tens of thousands of potential connections between clients and server ports. Furthermore, the policy will constantly change as personnel roles and organizational objectives are updated over time. Consider a developer, Alice, who sits on the developer floor (Figure 1) working from `L2` and needs access to Git on `server2`. Right before a meeting with her manager, who is on the HR floor, Alice needs to print a project report on `printer1`. Alice's meeting went exceptionally well, and she is promoted to a management position with a large office on the HR floor. However, during the transition, Alice still needs access to the Git server on `server2` to finish a project, as well as to be available to fix bugs until she can train a replacement. There can be multiple users like Alice in the enterprise, which makes the overall security maintenance complex, and can lead to inconsistencies among different perimeters.

Threat Model and Assumptions: NetViews seeks to provide an access control framework for enterprise environments (e.g., government, university, military, corporate). We assume both insider (e.g., authenticated users, services) and external (e.g., exploited software, IoT devices) attackers. The goal of attackers is to compromise devices, exfiltrate data, or disrupt enterprise services. Attackers may compromise user passwords and multiple hosts, pivoting through the network to accomplish their goals. Our trusted computing base includes all implementation components of NetViews. We assume the SDN infrastructure, including controller and switches, is securely deployed (e.g., out-of-band or TLS-protected southbound communication) and free of errors. We assume all installed SDN applications are benign and free of errors. We also assume appropriate defenses are deployed to mitigate known DoS concerns for reactive forwarding (e.g., `PacketIn` flooding). Finally, we assume the enterprise network has a secure and robust authentication mechanism for users and services.

3 OVERVIEW

We seek to restrict the lateral movement used by APTs and ransomware by enhancing network environments with granular, least-privilege access control of network flows between hosts. Achieving this goal requires overcoming the following research challenges.

- *Policy must apply to every network packet.* Achieving complete mediation of network traffic requires the ability to inspect every packet sent by and received from every host. Forcing traffic through choke-points induces unnecessary latency and reduces network reliability.
- *Policy enforcement must be fine-grained.* At fine-granularity, not all of the policy can be precomputed (e.g., client source ports). Even if it could be precomputed, the policy size would be too large to store in network switches.
- *There is a semantic gap between networking primitives and an enterprise's organizational structure.* The roles and duties of employees change in response to organizational needs. Network administrators should not be expected to manually translate changes in roles into changes in firewall policy.

Reactive SDN architectures such as OpenFlow provide the ability to perform complete mediation of all packets sent by and received from every host without forcing traffic choke-points. In a reactive setup, switches send `PacketIn` messages to a logically central controller whenever they encounter a packet that does not match an existing forwarding rule. The controller responds to a `PacketIn` message with a `FlowMod` message, which tells the switch which port to forward the packet to as well as a rule for matching future packets. Thus, reactive SDN only incurs additional latency on the first packet for the rule, and subsequent packets are forwarded at line-speed. The need for additional `PacketIn` messages depends on the granularity of the match rule, which can be coarse (e.g., based on a CIDR prefix or MAC address) or fine-grained (e.g., based on a 5-tuple of source IP, source port, destination IP, destination port, transport-layer protocol). Switches have a limited amount of TCAM for storing flow rules, and old rules may be expunged if the TCAM fills. Therefore, care must be taken when using fine-grained forwarding rules.

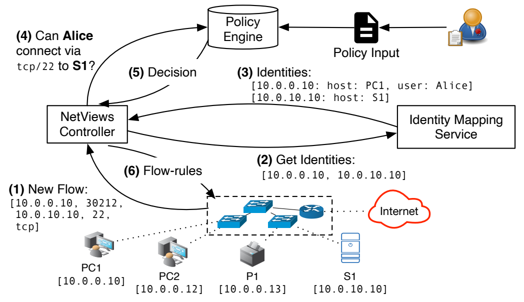


Figure 2: Overview of NetViews design portraying the necessary components and a possible interaction among them.

Prior work [11, 26, 49] has proposed reactive SDN for enforcing access control of network flows between hosts. However, their policy models are either non-existent or based on groups, which cannot reflect the permission hierarchy and dynamic nature of an enterprise [58, 59]. Alpaca [32] incorporates roles into IP address assignments to enable efficient packet enforcement in forwarding devices; however, the number and granularity of roles are limited. In contrast, we adopt NIST's NGAC model [17, 19], which can describe role (RBAC) and attribute (ABAC) based policy models, as well as a range of dynamic policy features. We overcome the performance limitations of prior fine-grained match rules by (1) building on top of the Intent primitive in the ONOS SDN controller to proactively send `FlowMod` messages to all switches on a forwarding path as soon as the `PacketIn` message at the first switch is received; and (2) using a multi-connection optimization that slightly expands the match rule without sacrificing security. In doing so, we envision an enterprise can simply extend their existing organization policy into the network.

Figure 2 shows a high-level overview of the NetViews design depicting the key logical components: (1) the NetViews controller, (2) the identity mapping service, (3) the policy engine, and (4) an SDN data plane with reactive forwarding. When the data plane encounters an unknown flow, the NetViews controller receives the event (Step 1). The NetViews controller then queries the identity mapping service to translate flow IP addresses into the user and host information referenced by the policy (Steps 2 and 3). Next, the NetViews controller queries the policy engine using the derived user and host, as well as the Layer-4 connection information, e.g., `tcp/22`, (Step 4). If the policy decision is *deny*, then the flow is dropped. However, if the policy decision is *allow* (Step 5), the NetViews controller installs forwarding rules in all switches on the determined path from the source to the destination (Step 6). Forwarding rules are defined for the network flows in both directions.

4 NETVIEWS POLICY MODEL

A key contribution of this paper is the integration of a flexible access control policy model into a least-privilege network environment. By building upon NIST's NGAC, NetViews benefits from decades of access control research. NGAC's flexibility also allows NetViews to integrate with enterprises currently using both role and attribute based policies, assuming suitable tools to transform RBAC and ABAC policies into an NGAC policy.

Existing policy models such as NGAC, RBAC, and ABAC assume a traditional operating system environment. This section addresses two key questions: (1) *How should NGAC policy concepts capture network primitives while bridging the semantics?* And (2) *What are the semantics of an* allow *decision and also how should the networking infrastructure respond to an allow decision?*

4.1 Users, Objects, and Access Rights

The NGAC model [19] defines authorized users (U), processes (P), objects (O), user and object attributes (UA and OA), policy classes (PC), operations (Op), and access rights (AR). A key question for our work is how to encode these essential elements with network access control concepts.

While the full NGAC policy model is too complex to describe here, the goal of an NGAC policy is to determine if a *user* ($u \in U$) has sufficient access rights ($ars \in 2_1^{AR}$) for an *object* ($o \in O$). Much of NGAC's flexibility comes from its use of *attributes* (AT), defining both user attributes (UA) and object attributes (OA), where $AT = UA \cup OA$. By definition, every object is considered an object attribute ($O \subseteq OA$). Users, objects, and attributes are *assigned* to other attributes in a hierarchical fashion. For example, if a user $u \in U$ is assigned to a user attribute $ua \in UA$, then ua is said to *contain* u. Semantically, u will inherit all of the rights granted to ua. Access rights are granted via *associations* ($UA \times 2_1^{AR} \times AT$) that specify a user attribute has a specific set of access rights on a target attribute, e.g., $\langle ua, ars, at \rangle$. Finally, NGAC supports two additional types of relations: user-based *prohibitions* can override access granted by an association, and *obligations* are defined as event-response relations. It is possible to have events trigger obligations and dynamically update policy. Furthermore, the policy configuration can be set up or updated manually by an administrator or through an event processing module for a dynamic update.

Users: We primarily focus on user workstations. We assume that each host on the network has at most one user at any given time, and a single user entity can access the network from different hosts. This assumption is realistic, even for shared workstations that require users to login (e.g., via Active Directory), as the domain controller can inform the network infrastructure which user is associated with a host. For servers, we assume the use of per-user virtual machines, in some cases, the "user" for the server may not be a physical person but rather an abstract user to perform a task. Containers could also be handled similarly, assuming proper bookkeeping by the orchestration platform and perspective into the networking primitives (e.g., IP addresses) within the container platform.

NetViews decides whether or not an *IP packet* should be forwarded. This decision must be based on the network flow 5-tuple information available in the packet. Given the expressiveness of NGAC, the assignment of users and user attributes to other user attributes can capture the relationships between IP addresses, hosts, and real users. Therefore, we mapped NGAC's definition of *user* with the real $\langle user, device \rangle$ pair and used a separate identity mapping service to translate the network flow 5-tuple to the $\langle user, device \rangle$ pair. However, changing the device does not change a user's role; it can only introduce some extra capabilities or restrictions. For this reason, we decided to maintain a unique user identifier for each user and consider it as a first-hop user attribute.

Objects and Access Rights: Given the identity mapping service, NGAC objects could be hosts or TCP ports on a host. TCP ports map to server applications, which could be suitable objects. Hosts could then be object attributes that contain the TCP port objects. However, this design is incompatible with aspects of network operation. For example, it does not naturally capture how to control access to ARP, which is a prerequisite for most IPv4 flows. Since our goal is to define a model for *visibility*, it is more natural to map NGAC objects to server hosts and manage TCP/UDP ports, ICMP, and ARP as NGAC access rights.

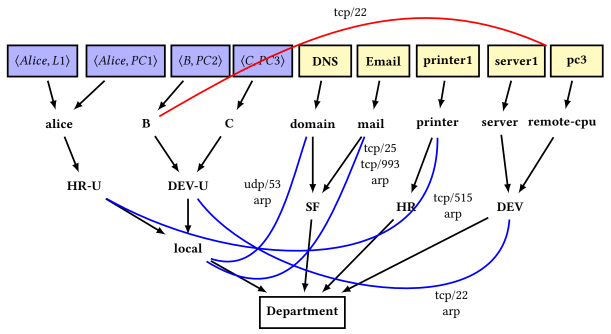


Figure 3: NetViews policy example with four user-device pairs (blue) and five resource or objects (yellow). Downward blue arcs denote associations and their corresponding rights (e.g., `tcp/22`). The upward red arc denotes a prohibition and the restricted rights.

Example NetViews Policy Scenario: Figure 3 depicts a small example NetViews policy for the network topology in Figure 1. The policy includes a single policy class, a set of users U, and a set of objects O. The policy also shows a set of user attributes UA = {u1, u2, u3, HR-U, DEV-U, local} and a set of object attributes $OA = O \cup$ {domain, mail, printer, server, remote-cpu, SF, HR, DEV}. The black lines with arrows depict *assignments* which correspond to the notion of *containment*. For the object side of the policy, attribute containment indicates that access rights are given to a set of objects.

The figure depicts *associations* as downward blue arcs labeled with a set of access rights. Recall that an association $\langle ua, ars, at \rangle$ specifies that all users contained by user attribute ua have access rights ars for all policy elements contained by at. The figure also depicts *prohibitions* as upward red arcs labeled with a set of access rights. There are several different types of NGAC prohibitions that provide flexibility in how the original set of access rights is restricted. Interested readers are referred to the NGAC specification [19] for more details.

NetViews Policy Language and Maintenance: NetViews leverages NGAC's existing language and policy specification tools. The policy configuration can be set up or updated manually by an administrator or through an event processing module for a dynamic update. The events can trigger obligations and in turn dynamically update the policy. NGAC follows an administrative policy model consisting of administrative policy elements (e.g., admin rights, associations, prohibition, obligations, and routines). We exclude these administrative aspects for simplicity. Currently, both JSON and GUI-based graph specification is available. As the NGAC project evolves, NetViews will continue to benefit from new features and usability enhancements. In current setup, an administrator needs to identify subjects, access rights, and objects and write the specifications for generating a meaningful policy.

4.2 NetViews Identity Mapping Service

NetViews requires an external service to map the network packet 5-tuple information to real user-devices and server hosts. Usually the identity mapping is relatively static with respect to the rate of IP packets traversing the network. Therefore, identity mapping updates are not on the critical path for access control decisions.

NetViews's identity mapping service must coordinate with the identity services used by the enterprise. The most straightforward scenario is traditional enterprise network environments that use static DHCP to allocate IP addresses to known hosts. They also rely on MAC address-based allow lists to prevent MAC address spoofing. In environments that use 802.1x [31], the identity mapping service needs to coordinate with the authenticator and authentication server (e.g., RADIUS) to determine the policy user that corresponds with the supplicant. Additional security protection against IP and MAC address spoofing can be provided by enhanced solutions such as SECUREBINDER [29]. For environments with shared hosts that authenticate via a domain controller (e.g., Microsoft Active Directory), the identity mapping service can coordinate with the domain controller to associate hosts with the currently logged-in user.

Finally, when the mapping between IP address and policy identity is dynamic, the NetViews controller should be notified when a mapping is invalidated (e.g., logout). Ideally, logout events should cause the NetViews controller to remove corresponding forwarding rules. However, given sufficiently short forwarding rule idle timeouts (e.g., the ONOS default is 10s), allowing the rules to expire will provide sufficient security in most deployments.

4.3 Access Control Semantics

Unlike in OS access control, where enforcement is needed just in one direction of the communication (for example, enforcing access control on a process reading a file), network access control must allow network flows in *both directions* for successful network communication. Specifically, *stateless* firewalls (also known as stateless firewall filters and access control lists) must define rules that allow flows in both directions. They consider each packet in isolation, using Layer 3 and 4 header flags to differentiate the first packet in a connection from the reply traffic. In contrast, *stateful* firewalls only define rules for connection initiation and then track the connection to allow all subsequent packets in both directions. Stateful firewalls are generally viewed as more secure and have replaced nearly all stateless firewalls. The primary benefit of stateful firewalls is that they prevent many types of network scanning commonly used for reconnaissance before an attack, e.g., ACK scanning.

NetViews Enforcement Semantics: NetViews assumes an SDN network configured with reactive forwarding. When a client c sends its first packet to a server s, the NetViews policy is consulted. If the packet is allowed, the NetViews SDN controller installs forwarding rules that match the 5-tuple network flows *for both directions*.[1] These forwarding rules are installed in all switches on the path between c and s. We note that NetViews relies on the underlying

[1]As discussed in Section 5, determining the forwarding rule for reply traffic for non-TCP packets requires some consideration.

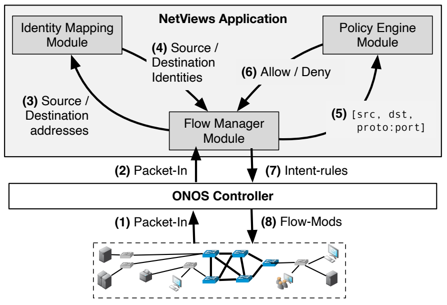


Figure 4: NetViews prototype implementation.

SDN controller (e.g., ONOS) and installed forwarding applications to determine the specific forwarding path.

Unlike stateful firewalls, NetViews does not track TCP connection state. However, installing 5-tuple matching forwarding rules for both directions provides equivalent security. The server s cannot send a packet to the client c before c initiates the connection. Switches on the forwarding path simply could not deliver the packet. Furthermore, for the above semantics, s can only send packets to the specific TCP port on c that initiated the connection. It can do so only for the lifetime of the forwarding rule, which will timeout after a predefined period (e.g., 10 seconds). Similarly, a malicious host m on the network cannot send packets to any other host without consulting the NetViews SDN controller.

Multi-Connection Optimization: SDN switches cannot efficiently manage a large number of forwarding rules. In an OpenFlow-based network, installing `FlowMod` rules for each 5-tuple flow may overflow the TCAM in the switches, causing increased `PacketIn` events and significantly degrading network performance. However, we observe that the above enforcement semantics are more strict than required. Instead of matching the specific client TCP port, NetViews can install forwarding rules that match *any* client TCP port. This change will significantly reduce the number of required forwarding rules for application protocols such as HTTP that require clients to make many TCP connections to the same server and port.

We observe that this performance optimization (not matching 5-tuple) does not have a significant negative impact on security. The only hosts that can exploit the more permissive forwarding rules are hosts to which connections have already been permitted. Furthermore, with reasonable forwarding rule timeouts, the risk is further reduced. Thus, NetViews's goal of controlling network visibility and preventing reconnaissance through network scanning is still achieved. In summary, the perimeter-less and least-privilege nature of NetViews eliminates the need for stateful tracking to prevent reconnaissance via network scanning.

5 NETVIEWS IMPLEMENTATION

Our NetViews prototype implementation is built as an SDN application on top of ONOS version 2.3.0. It does not require any modification to the ONOS controller, which simplifies code maintenance and allows deployments into existing ONOS installations. As shown in Figure 4, the NetViews application consists of three logical components: the flow manager, the policy engine, and the identity mapper. While our implementation combines all functionality in one application, alternative implementations could easily decouple the three components to enhance scalability.

5.1 Flow Manager

OpenFlow reactive forwarding applications receive `PacketIn` events whenever a switch receives a packet that does not match any of its forwarding rules. The `PacketIn` event contains packet header information, including the Layer 2, 3, and 4 protocol types and identifiers. Forwarding applications commonly respond to `PacketIn` events by sending `FlowMod` messages to the originating switch. A `FlowMod` message contains a set of match criteria (e.g., source and destination IP) and an action (e.g., forward out physical port 3). The switch uses the `FlowMod` message to update its forwarding rules.

Since the entire forwarding path can be determined at the time of the first `PacketIn` from the switch closest to the source, it is inefficient to wait for the `PacketIn` events from the subsequent switches on the forwarding path. As such, an SDN forwarding application can avoid additional delays for delivering the first packet by proactively sending `FlowMod` messages to the subsequent switches in hope of preventing additional `PacketIn` events. However, managing many different `FlowMod` rules for many switches can become very complex. To ease the development of applications, ONOS provides a forwarding primitive called an *Intent* [1], which provides a *one big switch* abstraction similar to Pyretic [46]. An ONOS Intent [1] defines match rules. ONOS compiles an Intent and manages all of the `FlowMod` messages for individual switches. Our implementation relies on ONOS's existing forwarding path algorithm.

Identifying Reverse Flow: For each authorization, NetViews installs one Intent for each direction of network traffic. Identifying the reverse flow for TCP connections is straightforward, as the TCP port information is symmetric. However, not all protocols use symmetric identifiers. For example, ICMP packets use type information (e.g., `ECHO`, `REPLY`) to indicate the direction of the flow. Since most firewalls drop nearly all ICMP types, NetViews currently only handles ICMP Ping messages, defining a match rule for `REPLY` for the reverse flow. In contrast, ARP packets must be handled differently. While ARP packets do not contain an IP header, they do include the source and destination IP address in the protocol address information. However, ARP spoofing attacks could allow an attacker to circumvent the policy. Fortunately, SDN controllers can mitigate ARP spoofing using a proxy ARP approach. That is, NetViews maintains a mapping between IP addresses and MAC addresses (e.g., from static DHCP configuration). When NetViews receives a `PacketIn` for an ARP request at the first switch, it consults the mapping and performs an access control check to determine if the ARP request is allowed by the policy.

Intent Installation: The Intent installation implementation was more subtle than initially expected. ONOS's Intent framework uses a `TrafficSelector` object that identifies a subset of network traffic based on packet header fields and patterns. The framework then compiles the Intent into a set of `FlowRule` objects that are installed to switches on the path. A reactive Intent application creates Intents in response to `PacketIn` events. While the Intent is being compiled and installed, a similar `PacketIn` may occur. In order to avoid creating duplicate Intents, Intent applications deterministically create an Intent *key* based on packet characteristics. When a `PacketIn` that matches the key of an Intent pending installation, a `PacketOut` message is returned. A `PacketOut` message instructs the switch to forward a given packet but not update its forwarding rules. Many `PacketIn` events may occur before the Intent is fully installed.

We began by modeling ONOS's sample Intent reactive forwarding (`ifwd`) application. However, there are significant differences. First, `ifwd` only requires one Intent, because it matches packets using an empty `TrafficSelector`. In contrast, NetViews uses a non-empty `TrafficSelector` to match Layer 3 and 4 information and hence needs Intents for both directions.

The second key difference involves the creation of the Intent key and has subtle implications on handling `PacketIn` events that occur for the *reverse flow*. `ifwd` defines its Intent *key* by concatenating the device IDs in lexicographical order. As such, `PacketIn` events for the reverse flow will match the Intent key. In contrast, NetViews requires two Intents with fine-grained keys. The Intent key for the forward direction is defined by concatenating the source IP, source MAC, destination IP, destination MAC, protocol, and destination port. The Intent key for the reverse direction reorders these values, predicting the order of the reverse flow. The Intent key for the reverse flow also appends the value "`RETURN`" to distinguish it from a new flow, which would require an access control check. To quickly identify reverse flows, NetViews implements a local cache of return keys, releasing the packet with a `PacketOut` on cache hit.

Finally, the ONOS Intent framework does not support idle-timeout for the Intent itself. As a result, whenever a particular flow rule expires after an idle-timeout period (default, 10 seconds), the Intent API reinstalls the flow rule. Therefore, NetViews stores a timestamp for each installed Intent. It uses a separate thread to deactivate Intents after a pre-specified period (10 seconds in our implementation). Note that Intent deactivation does not trigger flow rules expiration, and therefore the packets will continue to be forwarded without consulting the NetViews controller as long as the client-server communication continues.

Note that both the ONOS Intent and `FlowMod` timeouts are an administrative trade-off between security and performance. A shorter idle-timeout will result in greater `PacketIn` events. A longer idle-timeout provides a compromised account or malicious insider more opportunity to connect to network resources. However, for most scenarios a timeout on the order of seconds is reasonable. For example, removing the access of a fired employee requires a policy administrator to manually change the policy, an action that may take on the order of tens of minutes in a typical organization.

5.2 Policy Engine

The policy engine module is based on the reference implementation of NGAC [30]. Specifically, we used the `policy-machine-core` project, which provides core components of the NIST Policy Machine. It includes APIs to manage NGAC Graphs (we use the JSON interface), query the access state of a graph, and explain why a user has permissions on a particular resource. There are four main packages in the core library: the Policy Information Point (PIP), the Policy Administration Point (PAP), the Event Processing Point (EPP), and the Policy Decision Point (PDP). The PIP package provides the necessary interfaces (and in-memory implementations) for managing an NGAC graph, prohibitions, and obligations. A combination of the PAP and PDP allows NetViews to request permission decisions using user identity, object identity, and permission-type.

5.3 Identity Mapping Service

Our NetViews implementation assumes a static identity mapping managed via a configuration file that is auto-generated after manual administrative actions. This scenario corresponds to static DHCP assignments, which are used by many enterprises and universities. The identity mapping module loads a simple configuration file to populate a data structure that maps a $\langle IP, MAC \rangle$ pair to a unique $\langle user, device \rangle$ pair for users and a unique ID for servers. As discussed in Section 4.2, more dynamic scenarios that use 802.1x authentication (e.g., WPA Enterprise) requires a communication interface with RADIUS and DHCP servers. To minimize first-packet latency in dynamic scenarios, identity maps should be cached locally and invalidated on DHCP release. We leave the implementation of a dynamic identity mapping service to future work.

6 SECURITY ANALYSIS

NetViews significantly reduces an attacker's ability to move laterally within a network, even if it has compromised user credentials. This section provides a concrete demonstration of this benefit using our reference topology in Figure 1. Specifically, we use reachability-based attack graphs proposed by Lippmann et al. [40, 62] to validate network defense [53]. These works use firewall configuration and host vulnerability information to build attack graphs that describe how far an attacker can progress through a network. Our analysis assumes all hosts are vulnerable, which corresponds to the motivating NotPetya and Solorigate attacks described in Section 2.

Lippmann et al. [40] define three key concepts. First, a *reachability matrix* R is defined using outbound interfaces on all hosts i as rows and all active hosts j as columns. $R[i, j]$ is $true$ if a logical connection is possible between source i and destination j; otherwise, $R[i, j]$ is $false$. Second, an *attack graph* is a directed graph $G = (V, E)$ that shows how far attackers can progress through a network by successively compromising exposed and vulnerable hosts. The vertices V represent network hosts. For hosts $v_i, v_j \in V$, there exists an edge (v_i, v_j) when host v_i can make a logical connection to v_j. Finally, a *predictive attack graph* is a directed acyclic graph (DAG) rooted at a specific source host (assumed to be compromised) of interest. A new edge/host pair is added only if this pair is not already attached to the root of the graph or to any node along the path from the root to the current source node. The predictive attack graph is constructed using a breadth-first search traversal of the reachability matrix from the host of interest, only adding a new edge if the target host is not already included.

Analysis setup: Using the reference topology in Figure 1, we defined a NetViews policy and a corresponding traditional stateful firewall policy assuming the network segments shown in the figure. These policies are available in our online appendix [51]. Given the size of the topology, we hand-generated the reachability matrix, which is not uncommon [40, 62]. We then wrote a script to generate a predictive attack graph from the reachability matrix following Lippmann et al.'s algorithm [40]. We generated predictive attack graphs for each host for each of the two policy models.

Results: Figure 5 shows an example of the substantial reduction in the predictive attack graph for NetViews over a traditional segmented network policy enforcement with firewalls. While this figure demonstrates the qualitative benefit of NetViews, it does not quantitatively describe the overall benefit to the network. Table 1 focuses on the reachability of the three high-value servers in the reference topology (Figure 1). The table enumerates the number of hosts that can reach the server for a specific "hop-count." That is, a hop-count of one indicates hosts can access the server directly. A hop-count of two indicates a host has to compromise one other host before accessing the server. The table shows the number of hosts for hop-counts up to five. Notably, NetViews nearly eliminates the value of lateral movement for an attacker. The table does show that `server1` has two hosts with a hop-count of two. These links results because SSH access is allowed to a development host `pc3`. Thus, the benefit of a NetViews is dependent on the policy, which is to be expected. However, NetViews offers significant potential to drastically reduce lateral movement.

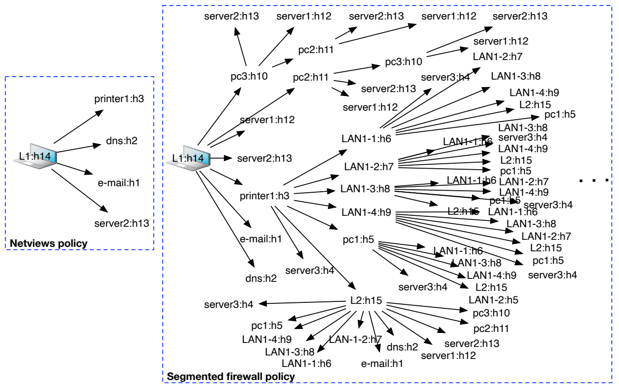


Figure 5: Attack graph visualizing possible reconnaissance and lateral movement from a compromised host. The graph depicts how far an attacker compromising user *Alice* can progress within the reference network from Figure 1.

Table 1: Number of hosts reachable in hop-counts 1 to 5 for the reference topology (Figure 1) based on policy type

Policy Type	hop-count	server1	server2	server3
NetViews	1	3	2	6
	2	2	0	0
	3	0	0	0
	4	0	0	0
	5	0	0	0
Segmented Firewall	1	4	4	8
	2	9	9	10
	3	9	9	10
	4	9	9	10
	5	9	9	10

7 PERFORMANCE EVALUATION

This section evaluates the performance of NetViews and the Policy Engine, answering the following questions.

- **Q1** How does NetViews compare with other ONOS forwarding applications in terms of latency and throughput?
- **Q2** How does NetViews scale with the number of flows?
- **Q3** How does the Policy Engine scale for more complex policies (and topologies)?

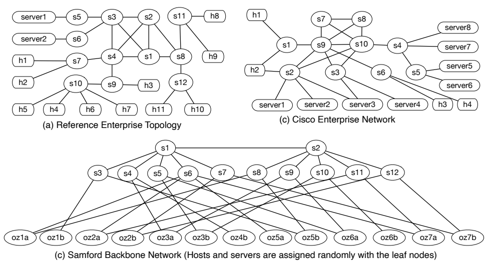

(a) Reference Enterprise Topology
(c) Cisco Enterprise Network
(c) Samford Backbone Network (Hosts and servers are assigned randomly with the leaf nodes)

Figure 6: Evaluation Topologies

Table 2: Description of Evaluation Topologies

Topology	Devices	Switches	Details
Reference	13	12	Topology from Figure 1
Cisco [71]	12	10	Cisco enterprise network
MiniStanford [71]	100	25	Stanford backbone network

7.1 Experimental Setup

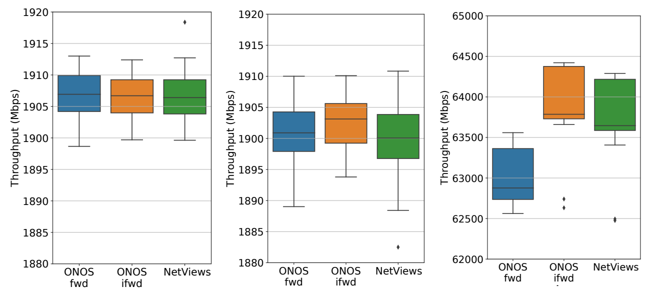
We consider three example enterprise network topologies, measuring network performance of three ONOS applications.

- **Baseline (fwd)**: This scenario uses the ONOS Reactive Forwarding application (`org.onosproject.fwd`) available with the ONOS distribution. The application will receive a `PacketIn` request, *allow* all communication between source and destination, and install corresponding `Flow Rules`. We consider this scenario as it is the most basic way to maintain networking functionality in an SDN environment without any policy enforcement.
- **Intent Forwarding (ifwd)**: This scenario uses the Intent Forwarding application provided by the ONOS sample library. It also *allows* all communication between source and destination on receiving a `PacketIn` request and installs corresponding Intents.
- **NetViews implementation (NetViews)**: This scenario is our implementation, and includes all the system components discussed in Section 5 (see Figure 4).

Table 2 summarizes our three representative enterprise topologies showed in Figure 6. The Reference topology was introduced in Figure 1; we used this simple topology to introduce NetViews design parameters. The Cisco and MiniStanford were previously used in other evaluations (e.g., PSI [71], HeaderSpaceAnalysis [34], Resonance [49]) and represent topologies provided by Cisco and Stanford, respectively. Each link in the topology is Gigabit Ethernet.

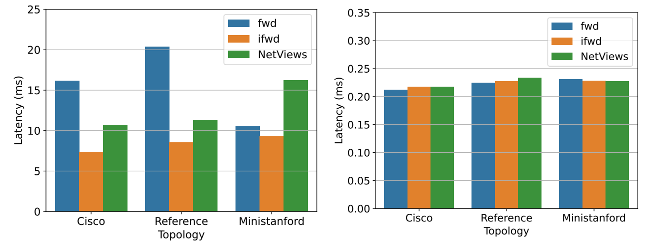
We implemented each of these topologies in Mininet [65]. Each measurement was taken on a Ubuntu 20.04 LTS (Linux Kernel 5.4.0) VM in QEMU with 16 Cores and 32 GB of RAM. These VMs are spawned from a server with 2 Intel Xeon Silver 4114 CPUs (20 physical cores at 2.20 GHz).

To evaluate throughput, we used the standard tool iPerf3 [15] (version 3.7). This approach allows us to test network throughput under maximum load (all hosts connected to all servers), as well as scale the amount of connections by increasing the number of parallel streams for our scalability analysis. To evaluate latency (initial packet, and excluding the initial packet) we use MTR [9] (version 0.93). As the complexity of policy does not impact performance, we use a policy that allows all connections. Each experiment lasts 60 seconds and is repeated 50 times. We present results as an arithmetic mean over these 50 runs.



(a) Cisco **(b) Reference** **(c) MiniStanford**

Figure 7: Aggregate throughput for different topologies (scales differ for readability)



(a) Average initial packet latency **(b) Average n^{th}-packet latency**

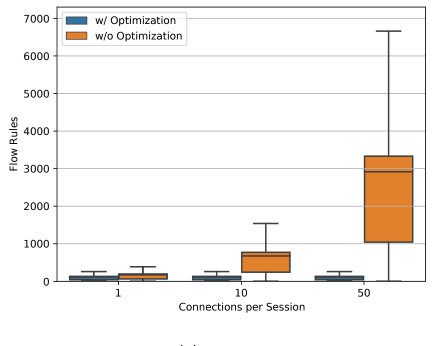
Figure 8: Average latency for different topologies (scales differ for readability)

7.2 Performance Overhead

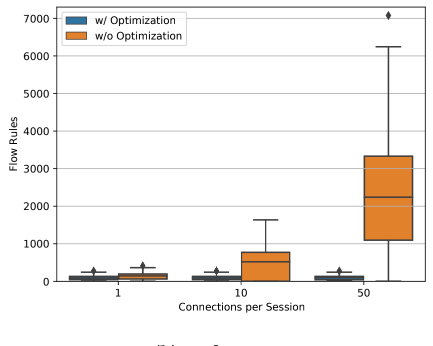
Our principal concern in this section is to determine the overhead introduced by NetViews compared with baseline ONOS applications. Because each network is unique and there is no "typical" level of traffic, we choose to compare throughput in the *worst case scenario*, where all "subject" hosts are attempting to send as much data as possible to the "object" servers in each topology simultaneously. To evaluate the marginal latency of adding new flows to the network all "subject" hosts start MTR pings to the "object" servers sequentially, with 1 second between each new flow.

7.2.1 Throughput. We compared the average throughput of NetViews to baseline ONOS apps. Because the throughput seen by any individual host is a function of topology and transport layer fairness, we characterize the total *aggregate* throughput of each topology by summing the throughput of all individual flows.

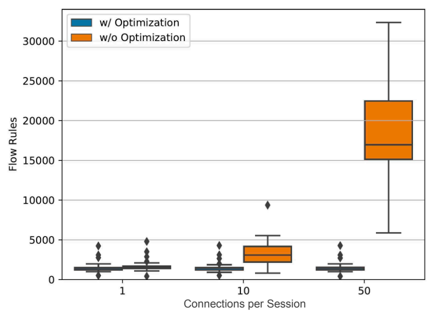
Figure 7 shows boxplots indicating the distribution of observed throughputs for each experiment run for each combination of ONOS app and topology. While each topology sees a different magnitude of throughput, there is considerable overlap between the three applications. Specifically, in the Cisco and Reference topologies, the median NetViews throughput falls well within the the interquartile range of the baseline cases and the change in medians is



(a) Cisco



(b) Reference



(c) MiniStanford

Figure 9: Number of `Flow Rules` in switches for NetViews with and without the optimization (scales differ for readability)

well under 1%. Moreover, in the scaled MiniStanford topology, we find the median aggregate throughput to be well above that of the `fwd` baseline application, and within approximately 200 *Mbps* of the `ifwd` baseline application. In Figure 7a the median aggregate throughput hovers between 1905 and 1908 *Mbps*. In Figure 7b the median aggregate throughput falls right around 1900 and 1905 *Mbps*. Finally, in Figure 7c the median aggregate throughput for `ifwd` and NetViews falls between 63550 and 63750 *Mbps*, while for `fwd` it sits around 62900 *Mbps*. We conclude that NetViews does not show any significant throughput overhead over the `fwd` or `ifwd` applications.

7.2.2 Latency. SDN reactive-forwarding applications experience a greater latency for the first packet. Thus, we evaluate both *initial latency* (i.e., the latency of packets causing `PacketIn` events) and *n^{th}-packet latency*, (i.e., the latency of all subsequent packets). We ensure a clean start (no Intents and no `Flow Rules` already installed) at the beginning of the experiments to ensure we capture `PacketIn` events in each measurement.

Figure 8 shows that NetViews has acceptable latency compared to `ifwd` for both the initial and n^{th}-packet. Figure 8a shows our initial packet latency results. In the Cisco topology we see a 3.286 ms (44.5%) latency increase for NetViews compared to `ifwd`, in the Reference topology a 2.687 ms (31.4%), and in the MiniStanford topology a 6.868 ms (73.5%) increase. Both `ifwd` and NetViews outperform `fwd`. Figure 8b shows our n^{th}-packet latency results. For the Cisco topology we see a negligible difference in latency (<0.01%), in the Reference topology a 0.0058 ms (2.59%) increase, and in the MiniStanford topology a 0.0007 ms (0.3%) decrease.

7.3 Multi-Connection Optimization

The multi-connection optimization reduces the number of `PacketIn` events and hence the number `Flow Rules` stored in the TCAM of switches. It provides the greatest benefit when clients establish many connections to a server as part of the same session. Specifically, for each connection in the session, the client uses a different ephemeral source port to the same server port. To characterize the benefit of the optimization, we emulate sessions with different numbers of parallel connections, measuring the resulting number of `Flow Rules` with and without the optimization. In Figure 9, the x-axis shows the number of parallel connections for the session, and the y-axis shows the average number of `Flow Rules` installed in switches for different topologies. The boxplots illustrate the distribution of `Flow Rule` counts across all switches in each topology.

The first set of boxes in all three figures depicts the scenario with no parallel connections, which results in policy enforcement based on 5-tuple packet information without the optimization. There is no match on the client port with the optimization (to allow ephemeral client ports). Even without parallel connections, the number of flow rules is higher without optimization. For example, in the case of the Cisco topology in Figure 9a, the median number of flows for the setting without optimization is 172. In contrast, with the optimization, that number goes down to 116.

The boxplots for 10 and 50 connections per session demonstrate where we expect the overhead reduction of the multi-connection optimization. The boxplot for the Cisco topology shows that for ten connections per session, the median number of `Flow Rules` per switch without the optimization increases to 772 from 172. In contrast, with the optimization, it remains at 116. Finally, with 50 connections per session, the number of `Flow Rules` per switch without optimization is 2,916, while with optimization, it remains at 116. A similar trend emerges for the other topologies. These results demonstrate that the multi-connection optimization can result in significantly fewer `Flow Rules`.

7.4 Performance of Policy Engine

To understand the impact of the Policy Engine on the overall NetViews overhead, we analyzed the response time of *policy-machine-core* using random policy graphs as in [5, 43]. We used the graph generation algorithm from Basnet et al. [5].

We let the number of user and object nodes be defined from node count n, $u = \alpha \times n$ and $o = (1 - \alpha) \times n$. For each pair of $\langle u_i, o_i \rangle$, we first created two separate single-rooted binary tree structures using only user attribute *ua* and object attribute *oa* nodes, respectively. The number of *ua* and *oa* nodes depend on the predefined policy tree height h. For each tree with height h, we generate $2^{h+1} - 1$ number of *oa* and *ua* nodes. After the tree creation, the u_i node is made the new root of the user attribute tree and o_i of object attribute tree. These trees represent the *assignment* relation of the policy graph. The aforementioned user attribute tree and object attribute tree of pair $\langle u_i, o_i \rangle$ is connected using the *association* relations, or edges, as shown in Figure 2 of our online appendix [51]. The idea of separate operation node creation from Basnet et al. [5] is omitted

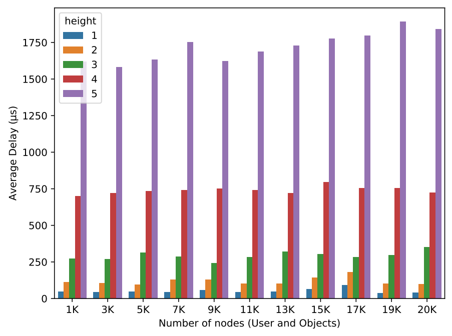


Figure 10: Average response time of policy-machine-core using random policy graphs

in our algorithm to reduce the graph generation cost. We map each leaf level *ua* node to each leaf level *oa* node, thus creating a total of $|ua_{leaf}| \times |oa_{leaf}|$ association arcs. We have used a set of δ permissions, and assigned each association arc with a permission label picked randomly from δ.

For this analysis, we generated access control graphs that varied in size from 4,000 to 1,280,000 vertexes (u, o, ua, and oa nodes) for different heights of the policy graph. However, the user (u), object (o) and height h are the main controlling parameters in our graph generation algorithm. As reported in Figure 10, we varied the total number of u and o nodes from 1,000 to 20,000 nodes ($\alpha = 0.6$), and h from 1 to 5 for each amount of total nodes. We considered a small permission set of $\delta = 10$ for measuring the decision time analysis. For each policy graph, we randomly picked 2000 permission requests from the set δ, and logged decision fetching delay from *policy-machine-core*. The average decision fetching delay is reported in micro-seconds.

Figure 10 indicates that the height of the policy graph is the main contributory factor to the decision delay. However, overall average delay is minimal, even for the 20,000 node (u and o) benchmark, with 1,280,000 graph vertexes (u, o, ua, and oa). The delay ranges from approximately 100 μs for height of one and 1250 μs for height of five. Therefore, the policy engine does not significantly affect overall operational performance of an enterprise network.

8 DISCUSSION AND FUTURE WORK

Scalability: We showed negligible overheads for latency and throughput on test networks from the literature, indicating our approach will scale equivalently to standard Intent Forwarding. We are limited, however, by publicly available datasets and topologies to test more extensive networks. The policy engine can exist either within the NetViews application or as an external server. If the policy engine is external, NetViews should implement a caching protocol within the SDN application. The external policy engine server should invalidate the cache when the policy changes. Additionally, ONOS supports multiple distributed SDN controllers to enhance scalability. We leave the investigation of policy cache management and multiple controllers to future work.

Policy Generation and Maintenance: NGAC can express a range of access control models, including both RBAC and ABAC, which are extensively used by enterprises for non-network access control. An administrator's first step for creating a policy is identifying subjects, access rights, and objects. Our current policy must be specified manually. We anticipate the development of tools to help automate the process. For example, a tool could extract policy from an enterprise's existing Active Directory installation. Our current implementation also requires the policy engine to reload the policy whenever there is a policy change. This is reasonable, because policy changes are infrequent relative to policy queries. In future work, we are exploring how to handle ephemeral and dynamic policy changes using NGAC's obligation primitive.

Deployment and Scope: NetViews seeks to provide access control within an enterprise's on-premises network environment. As enterprises move business applications into the cloud, an application-based Zero Trust model such as BeyondCorp may be more appropriate to protect those applications. That said, NetViews can also provide value *within* a cloud network environment; this is a topic of future work. As discussed in Section 2, even if an enterprise moves all of their business applications to the cloud, they still require enhanced protections for their on-premises network environment. NetViews needs to be adapted to a given enterprise environment. For example, as noted in Section 5 our current implementation requires adopters to coordinate with 802.1x infrastructure. An enterprise network also may not be fully SDN-capable. While SDN-capable access switches and WiFi access points are ideal, legacy switches with VLAN capabilities can be used to isolate traffic at the access switch layer. Similar capabilities can be achieved by configuring WiFi access points to use isolation mode. In both cases, network traffic is shunted up to an SDN-capable distribution switch. However, even without changes to access switch and access point configuration, NetViews can provide meaningful value as it will opportunistically mediate any traffic that reaches the distribution switch. Such a deployment would still providing better security than existing perimeter-based defenses, even if it allows small pockets of unmediated network traffic. This approach also provides a path for incremental deployment.

9 RELATED WORK

Access Control Frameworks: Access control systems most commonly seek to achieve *least privilege*, where every program and user of the system should operate using the least set of privileges necessary to complete a corresponding task [60]. To simplify network management challenges, many systems and proposals (e.g., Ethane [11], FML [26]) have used group-based policy definition, which is mostly static and fails to represent enterprise configurations [58, 59]. Role-Based Access Control (RBAC) [59] assigns permissions to roles to simplify policy management and models users' functional rules within an organization. However, traditional RBAC is less suitable when multiple features of users and devices are required for access control decisions [14]. To address this limitation, attribute-based access control (ABAC) [28] provides logical, fine-grained, and context-aware access control. Several extensions of ABAC, including NGAC [19] and XACML [18], have been proposed and used mainly to manage file-based resources and web services. We are unaware of existing commercial or research access control solutions that provide a unified and granular solution for modern-day network security [49, 68].

Network Access Control: Historically, both the external and the *internal* perimeter of an enterprise network was secured by firewalls [12, 23]. However, firewalls are hard to configure and maintain, particularly in scenarios with multiple domains or layers to secure [41, 69]. Furthermore, commercial next-generation firewalls are just firewalls with NIPS (Network Intrusion Prevention System) [52]. Modern enterprise network operation, access control, and security are currently maintained through different ad-hoc mechanisms: by building Layer-2 or Layer-3 boundaries within local networks (e.g., VLAN [63], subnets); through special-purpose devices to control packet flow (e.g., NIDS [6, 64] or Middleboxes [16, 71]); and hypervisor based systems that combine different state-of-the-art technologies (e.g., SDN) to provide isolation (e.g., FlowVisor [61], PSI [71], micro-segmentation [25, 37, 48]).

While prior work [11, 21, 35, 49, 71] provides dynamic policy enforcement (sometimes as a secondary feature), it has limited or nonexistent policy models. Alpaca [32] incorporates roles into IP address assignments to enable efficient packet enforcement in forwarding devices; however the resulting number of roles is limited and it constrains IP address assignment. NEUTRON [67] and SPRT [33] consider least-privilege access control within a network; however, their contributions are focused on creating and testing access control policies and hence are complementary to NetViews. NEUTRON and SPRT also assume traditional network segmentation or micro-segmentation for firewall placement while integrating OpenFlow as future work. Finally, several systems [27, 36] have proposed incorporating the dynamic context from the enterprise network to create context-aware access control. Such context can be incorporated into future enhancements of NetViews using multiple policy classes supported by NGAC.

Software Defined Networking (SDN): SDN has the potential to address many operational and security challenges in enterprise networks [39]. Along with new programmable switch architectures (e.g., P4 [7]), several high-level network programming languages (e.g., Frenetic [21], NetCore [45]), create opportunities in usability and functionalities. SDN has shown the potential to replace conventional security systems [70], simplify policy enforcement [24, 56], ensure information flow control [54], enable deceptive defense [3], and provide a software defined perimeter or connectivity [25, 38, 47, 50]. Finally, research has identified attacks against SDN technologies and proposed corresponding defenses [10, 66, 72, 73].

10 CONCLUSION

While application-based Zero Trust architectures help enterprises secure their business applications by moving them to the cloud, they ignore the importance of securing the on-premises network environment that remains. This paper has introduced a novel paradigm for enterprise network security called *Network Views* where each host has a different "view" of what other hosts and services exist in the network. This fine-grained least-privilege approach to network access control can significantly reduce lateral movement by attackers, even if user credentials have been compromised. NetViews builds on NIST's Next Generation Access Control to provide a dynamic and scalable policy model that supports the needs of large enterprises. We propose a multi-connection optimization that eliminates unnecessary first-packet latencies and significantly reduces TCAM requirements for SDN switches. Our NetViews implementation has latency and throughput comparable to reactive forwarding baselines. The source code of NetViews implementation is available in Git [2]. As such, NetViews provides a practical primitive for dissolving security perimeters within enterprise networks.

Acknowledgements: 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] R. A. Addad, D. L. C. Dutra, M. Bagaa, T. Taleb, H. Flinck, and M. Namane. 2018. Benchmarking the ONOS Intent Interfaces to Ease 5G Service Management. In *Proceedings of the IEEE Global Communications Conference (GLOBECOM)*.

[2] Iffat Anjum. 2021. Single Site Netviews. GitHub. https://github.com/netviews/ss-netviews.

[3] 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 (CODASPY)*.

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

[5] R. Basnet, S. Mukherjee, V. M. Pagadala, and I. Ray. 2018. An efficient implementation of next generation access control for the mobile health cloud. In *Proceedings of the International Conference on Fog and Mobile Edge Computing (FMEC)*.

[6] Noam Ben-Asher and Cleotilde Gonzalez. 2015. Effects of cyber security knowledge on attack detection. *Computers in Human Behavior* 48 (2015).

[7] 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).

[8] William Brockelsby and Rudra Dutta. 2021. Traffic Analysis in Support of Hybrid SDN Campus Architectures for Enhanced Cybersecurity. In *Proceedings of the Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*.

[9] BitWizard B.V. 1997. MTR. BitWizard. http://www.bitwizard.nl/mtr/.

[10] 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)*.

[11] 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)*.

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

[13] Department of Homehald Security CISA. 2017. Petya Ransomware. Alert (TA17-181A). https://us-cert.cisa.gov/ncas/alerts/TA17-181A.

[14] E. Coyne and T. R. Weil. 2013. ABAC and RBAC: Scalable, Flexible, and Auditable Access Management. *IT Professional* 15, 03 (May 2013).

[15] Jon Dugan, Seth Elliott, Bruce A. Mah, Jeff Poskanzer, and Kaustubh Prabhu. 2015. iPerf - The ultimate speed test tool for TCP, UDP and SCTP. https://iperf.fr/.

[16] Seyed Kaveh Fayazbakhsh, Luis Chiang, Vyas Sekar, Minlan Yu, and Jeffrey C. Mogul. 2014. Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions using FlowTags. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[17] David Ferraiolo, Vijayalakshmi Atluri, and Serban Gavrila. 2011. The Policy Machine: A novel architecture and framework for access control policy specification and enforcement. *JOURNAL of Systems Architecture* 57, 4 (2011).

[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] David F Ferraiolo, Larry Feldman, and Gregory A Witte. 2016. Exploring the next generation of access control methodologies. NIST. https://www.nist.gov/publications/exploring-next-generation-access-control-methodologies.

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

[21] Nate Foster, Rob Harrison, Michael J. Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story, and David Walker. 2011. Frenetic: A Network Programming Language. *ACM SIGPLAN Notices* 46, 9 (Sep. 2011).

[22] Open Networking Foundation. 2018. Intent Framework. ONOS. https://wiki.onosproject.org/display/ONOS/Intent+Framework.

[23] M. G. Gouda and X. . A. Liu. 2004. Firewall design: consistency, completeness, and compactness. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*.

[24] Sanket Goutam, William Enck, and Bradley Reaves. 2019. Hestia: Simple Least Privilege Network Policies for Smart Homes. In *Proceedings of the Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*.

[25] 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)*.

[26] Timothy L. Hinrichs, N. Gude, M. Casado, John C. Mitchell, and S. Shenker. 2009. Practical declarative network management. In *Proceedings of the ACM Workshop on Research on Enterprise Networking (WREN)*.

[27] 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)*.

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

[29] Samuel Jero, William Koch, Richard Skowyra, Hamed Okhravi, Cristina Nita-Rotaru, and David Bigelow. 2017. Identifier Binding Attacks and Defenses in Software-Defined Networks. In *Proceedings of the USENIX Security Symposium*.

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

[31] Jyh-Cheng Chen and Yu-Ping Wang. 2005. Extensible authentication protocol (EAP) and IEEE 802.1x: tutorial and empirical experience. *IEEE Communications Magazine* 43, 12 (2005).

[32] 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).

[33] 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)*.

[34] Peyman Kazemian, George Varghese, and Nick McKeown. 2012. Header Space Analysis: Static Checking for Networks. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[35] Hyojoon Kim, Joshua Reich, Arpit Gupta, Muhammad Shahbaz, Nick Feamster, and Russ Clark. 2015. Kinetic: Verifiable Dynamic Network Control. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[36] Hyojoon Kim, A. Voellmy, Sam Burnett, N. Feamster, and R. Clark. 2012. Lithium: Event-Driven Network Control. Georgia Tech Library. https://smartech.gatech.edu/handle/1853/43377.

[37] Shashi Kiran. 2015. Data-Center: Micro-segmentation: Enhancing Security and Operational Simplicity with Cisco ACI. CISCO. https://blogs.cisco.com/datacenter/microsegmentation.

[38] Jonghoon Kwon, Taeho Lee, Claude Hahni, and Adrian Perrig. 2020. SVLAN: Secure & Scalable Network Virtualization. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*.

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

[40] Richard Lippmann, Kyle Ingols, Chris Scott, Keith Piwowarski, Kendra Kratkiewicz, Mike Artz, and Robert Cunningham. 2006. Validating and Restoring Defense in Depth Using Attack Graphs. In *Proceedings of the IEEE Military Communications conference (MILCOM)*.

[41] A. Mayer, A. Wool, and E. Ziskind. 2000. Fang: a firewall analysis engine. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*.

[42] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.* 38, 2 (April 2008).

[43] 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)*.

[44] Microsoft 365 Defender Research Team. 2020. Analyzing Solorigate, the compromised DLL file that started a sophisticated cyberattack, and how Microsoft Defender helps protect customers. Microsoft Threat Intelligence Center (MSTIC). https://www.microsoft.com/security/blog/2020/12/18/analyzing-solorigate-the-compromised-dll-file-that-started-a-sophisticated-cyberattack-and-how-microsoft-defender-helps-protect/.

[45] Christopher Monsanto, Nate Foster, Rob Harrison, and David P. Walker. 2012. A compiler and run-time system for network programming languages. In *Proceedings of the Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*.

[46] Christopher Monsanto, Joshua Reich, Nate Foster, Jennifer Rexford, and David Walker. 2013. Composing Software-Defined Networks. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[47] 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 (Sep. 2019).

[48] Muhammad Mujib and Riri Fitri Sari. 2020. Design of implementation of a zero trust approach to network micro-segmentation. *International JOURNAL of Advanced Science and Technology* 29, 7 (apr 2020).

[49] 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)*.

[50] 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)*.

[51] Netviews2022. 2021. Netviews Online Appendix. https://gist.github.com/Netviews2022/67d5265a19039e4f8c4d1733f0c02751.

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

[53] David M. Nicol and Vikas Mallapura. 2014. Modeling and analysis of stepping stone attacks. In *Proceedings of the Winter Simulation Conference (WSC)*.

[54] 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)*.

[55] 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.

[56] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. 2013. SIMPLE-fying Middlebox Policy Enforcement Using SDN. In *Proceedings of the ACM SIGCOMM (SIGCOMM)*.

[57] 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.

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

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

[60] F. B. Schneider. 2003. Least privilege and more [computer security]. *IEEE Security & Privacy* 1, 5 (2003).

[61] Rob Sherwood, Michael Chan, Adam Covington, Glen Gibb, Mario Flajslik, Nikhil Handigol, Te-Yuan Huang, Peyman Kazemian, Masayoshi Kobayashi, Jad Naous, and et al. 2010. Carving Research Slices out of Your Production Networks with OpenFlow. *SIGCOMM Comput. Commun. Rev.* 40, 1 (Jan. 2010).

[62] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J.M. Wing. 2002. Automated generation and analysis of attack graphs. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*.

[63] K. Sripanidkulchai, C. Issariyapat, and K. Meesublak. 2008. Inference of network-wide VLAN usage in small enterprise networks. In *Proceedings of the IEEE INFOCOM Workshops*.

[64] 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*.

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

[66] Kashyap Thimmaraju, Bhargava Shastry, Tobias Fiebig, Felicitas Hetzelt, Jean-Pierre Seifert, Anja Feldmann, and Stefan Schmid. 2016. Reigns to the Cloud: Compromising Cloud Systems via the Data Plane. *CoRR* abs/1610.08717 (2016).

[67] Dan Thomsen and Elisa Bertino. 2018. Network Policy Enforcement Using Transactions: The NEUTRON Approach. In *Proceedings of the ACM on Symposium on Access Control Models and Technologies (SACMAT)*.

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

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

[70] Changhoon Yoon, Taejune Park, Seungsoo Lee, Heedo Kang, Seungwon Shin, and Zonghua Zhang. 2015. Enabling security functions with SDN: A feasibility study. *Computer Networks* 85 (2015).

[71] 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)*.

[72] Menghao Zhang, G. Li, Shicheng Wang, Chang Liu, Ang Chen, H. Hu, G. Gu, Q. Li, M. Xu, and Jianping Wu. 2020. Poseidon: Mitigating VOLUMEtric DDoS Attacks with Programmable Switches. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*.

[73] 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 *Proceedings of the Research in Attacks, Intrusions, and Defenses*.