Coordinated Dataflow Protection for Ultra-High Bandwidth Science Networks

Vasudevan Nagendra
Stony Brook University
vnagendra@cs.stonybrook.edu

Vinod Yegneswaran
SRI International
vinod@csl.sri.com

Phillip Porras
SRI International
porras@csl.sri.com

Samir R Das
Stony Brook University
samir@cs.stonybrook.edu

Abstract

The Science DMZ (SDMZ) is a special purpose network architecture proposed by ESnet (Energy Sciences Network) to facilitate distributed science experimentation on terabyte- (or petabyte-) scale data, exchanged over ultra-high bandwidth WAN links. Critical security challenges faced by these networks include: (*i*) network monitoring at high bandwidths, (*ii*) reconciling site-specific policies with project-level policies for conflict-free policy enforcement, (*iii*) dealing with geographically-distributed datasets with varying levels of *sensitivity*, and (*iv*) dynamically enforcing appropriate security rules. To address these challenges, we develop a *fine-grained dataflow-based security enforcement* system, called `CoordiNetZ (CNZ)`, that provides *coordinated situational awareness*, i.e., the use of *context-aware* tagging for policy enforcement using the dynamic contextual information derived from hosts and network elements. We also developed tag and IP-based security microservices that incur minimal overheads in enforcing security to data flows exchanged across geographically-distributed SDMZ sites. We evaluate our prototype implementation across two geographically distributed SDMZ sites with SDN-based case studies, and present performance measurements that respectively highlight the utility of our framework and demonstrate efficient implementation of security policies across distributed SDMZ networks.

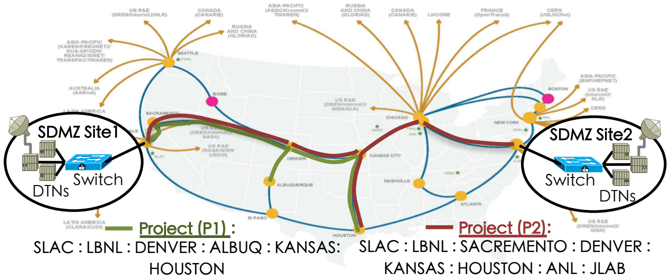

SDMZ Site1
Switch
DTNs
SDMZ Site2
Switch
DTNs
Project (P1) :
SLAC : LBNL : DENVER : ALBUQ : KANSAS: HOUSTON
Project (P2):
SLAC : LBNL : SACREMENTO : DENVER : KANSAS : HOUSTON : ANL : JLAB

Figure 1: SDMZ backbone (ESNet) with international connectivity, Illustrating two project collaborations across multiple SDMZ sites.

CCS Concepts

• **Networks** → **Network performance analysis**; *Programming interfaces*; *In-network processing*; • **Security and privacy** → *Firewalls*.

Keywords

Big Data Security, Distributed Systems Security, Network Security, Software-defined Programmable Security, SDN, NFV, Usability and Human-centric Aspects of Security

ACM Reference Format:
Vasudevan Nagendra, Vinod Yegneswaran, Phillip Porras, and Samir R Das. 2019. Coordinated Dataflow Protection for Ultra-High Bandwidth Science Networks. In *2019 Annual Computer Security Applications Conference (ACSAC '19), December 9–13, 2019, San Juan, PR, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3359789.3359843

1 Introduction

The need for computation over petabyte-scale datasets introduces complexities with respect to: (*i*) cost-effective provisioning of compute and storage resources, and (*ii*) secure transport of high-throughput experimental data across geographically-distributed datacenters. To mitigate these concerns, a new network architecture has been proposed called the Science DMZ (SDMZ) [9], in which an enterprise subnet

ACSAC '19, December 9–13, 2019, San Juan, PR, USA

ACM ISBN 978-1-4503-7628-0/19/12…$15.00
https://doi.org/10.1145/3359789.3359843

is isolated from stateful deep-packet inspection (DPI) middleboxes (e.g., firewalls, intrusion prevention systems (IPSs)) for optimized performance. Geographically-distributed SDMZ sites are inter-connected through high-performance network backbones, such as the ESNet (Energy Sciences Network) [11], which connects more than 40 U.S. Department of Energy (DoE) research sites and 150+ campus networks that collectively exchange more than 50 petabytes of data each month [11, 12]. Today, there are more than 100 such national research and educational networks present across the globe connecting thousands of research institutes using dedicated ultra high bandwidth WAN links [29].

However, implementing security policy for effectively managing such ultra-high-volume data transfers without sacrificing underlying transport performance and throughput remains a formidable challenge. Our paper is motivated by the observation that security mechanisms currently implemented in SDMZ networks fall short along multiple dimensions.

(1) *Coarse-grained Enforcement*: Deployed security mechanisms are too coarse-grained (IP, port-level ACLs) using router-based access control lists (ACLs) and aggressive filtering for handling high-performance science applications that exchange potentially *sensitive*, *proprietary*, or *personal-private* information across interconnected multi-institutional networks [9, 40].

(2) *Context Awareness*: Humongous volumes of data exchanged across SDMZs prevents the network-monitoring plane of SDMZ (e.g., a network intrusion detection system (NIDS)) from effectively deriving dynamic and fine-grained filtering decisions for enforcing security policies based on dynamic operational context (i.e., *who*, *what*, *where*, *when* and *how* the data resources are accessed or requested). Lack of application awareness, DPI capabilities, and contextual information leaves wide gaps in the SDMZ security architecture [39].

(3) *Intuitive Policy Specification*: SDMZ project users (e.g., researchers, professors, and students) have no method to directly capture their policy intents and enforce them onto the network without conflicting with other user's policy intents or site-specific policies.

(4) *Security as a Service*: Finally, current tier-2 SDMZ networks lack infrastructure support to effectively utilize dynamic security and data analysis services provided by tier-0/1 SDMZ compute centers (e.g., DDoS protection and data analysis) [6, 24].

We seek to address these limitations by introducing a new framework, called CoordiNetZ, which provides a graph-based dataflow policy management framework that enables users to express anticipated experimental interactions and automatically arbitrate conflicts with respect to project- and site-specific policies. This high-level abstraction is important, as science project policies must be flexibly specified by researchers rather than by administrators. CoordiNetZ addresses the lack of application- and context-awareness through a novel context-aware policy-based tagging mechanism (cTags), which allows dataflows to be associated with tags enabling fine-grained, cross-site dataflow filtering. Optimizations are proposed to effectively utilize the limited tag-space (20 bits *Flow Label* packet header of IPv6) that is available for using it across sites, while optimizing the number of rules required to enforce policies. CoordiNetZ integrates host-specific application context to network nodes and monitoring plane, enabling them to filter traffic by routing through light-weight security functions built as *microservices* for fine-grained policy enforcement.

The notable contributions of this paper are as follows:

- Identification of several key SDMZ security requirements (§2) that motivate the design and prototype implementation of a distributed SDN-based policy enforcement framework (§3).
- We present novel conflict detection and resolution mechanisms that allow policies specified by various SDMZ users using graph-based abstractions belonging to different sites and projects to be effectively reconciled (§4).
- We develop context-aware policy-based tagging that allows dataflows to be associated with tags enabling fine-grained control of project- and experiment-specific cross-site dataflows (§5).
- We present key security use-cases that demonstrate the benefits of CoordiNetZ framework (§6) and comprehensive performance evaluation of the CoordiNetZ prototype (§7).

2 SDMZ Background

The SDMZ network architecture has proven to be a vital platform for storing and transporting petabytes of scientific data (per month) across geographically-distributed research testbeds and data repositories in US and Europe (shown in Figure 1) [11, 12]. As shown in Figure 2, noteworthy elements of the SDMZ architecture that are optimized for performance include the following:

1. DTNs and applications customized to support data transfers at 10–100 Gbps [1].
2. SDMZ network perimeter architecture that bypasses stateful firewalls and DPI devices for high-throughput data transfers of elephant flows [39].
3. A dedicated SDMZ core network with capacity to carry more than 100 Gbps of science dataflow rates without loss.[1]

[1] Considering the growing bandwidth requirements of SDMZ applications, the SDMZ core network is soon expected to get upgraded to 400 Gbps [38]

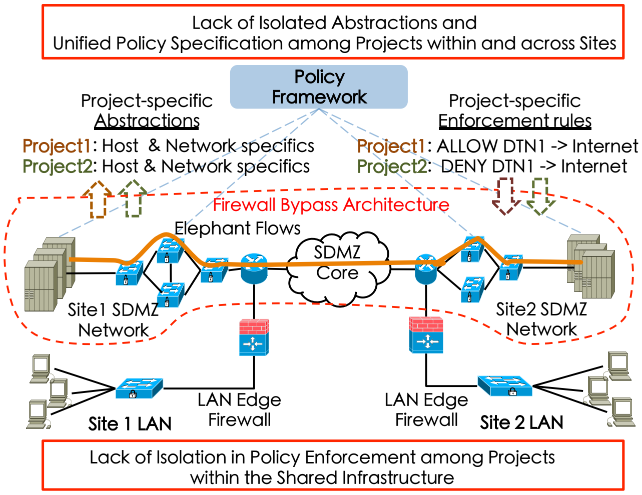

Lack of Isolated Abstractions and
Unified Policy Specification among Projects within and across Sites
Policy Framework
Project-specific Abstractions
Project-specific Enforcement rules
Project1: Host & Network specifics
Project2: Host & Network specifics
Project1: ALLOW DTN1 -> Internet
Project2: DENY DTN1 -> Internet
Firewall Bypass Architecture
Elephant Flows
SDMZ Core
Site1 SDMZ Network
Site2 SDMZ Network
Site 1 LAN
LAN Edge Firewall
LAN Edge Firewall
Site 2 LAN
Lack of Isolation in Policy Enforcement among Projects
within the Shared Infrastructure

Figure 2: Lack of isolation among projects in policy abstraction, specification and enforcement. Dotted lines represent SDMZ network isolated from enterprise network's firewalls.

The SDMZ network differs from enterprise networks in how it manages the data flows (i.e., elephant dataflows) across geographically distributed locations. Specifically, SDMZs employ specialized applications (e.g., Grid FTP) to transmit parallel (multi-port) data streams across SDMZ sites. These host DTNs and their applications are customized to handle high-performance TCP, and are limited to running a few "trusted" data-transfer applications.

The SDMZ network is also distinguished from other cross-domain networks such as software-defined WANs (SD-WANs) [16, 18], software-defined data centers (SDDCs) and multi-tenant cloud networks [3, 25] in that the SDMZ network is multi-tenant network that shares experimental data across multiple sites that are geographically distributed, spanning multiple administrative domains. In addition, the day-to-day policy requirements in these network infrastructures are specific to the experiments and associated data outcomes and the rules for sharing of data across collaborators are required to be specified by non-admin SDMZ users (e.g., researchers, students). However, existing policy frameworks including the new SDN-based frameworks are not designed to handle dynamic dataflow-based policy requirements of these multi-administrative cross-domain science networks [2, 21, 36, 42, 46].

2.1 SDMZ Security Requirements

Today, SDMZ security is provided primarily through offline protection, such as from clustered IDS (e.g., BroIDS [5]), router/switch ACLs and selective *fastpath* of high-bandwidth flows (e.g., SciPass [40]). However, SDMZs currently offer no way to capture and reconcile the unified policy intents of different administrators of SDMZ sites. SDMZs have no fine-grained flow management, i.e., *filtering*, *steering* or *revoking* of flows according to dynamic project requirements or security states of the SDMZ network. In addition, SDMZs do not offer the necessary context to enable an association between flows, projects, and data. Below, we identify four key requirements for an SDMZ security framework.

2.1.1 Fine-grained Dataflow Security Policies: SDMZ policy requirements differ from that of typical enterprise networks. Below, we summarize three broad classes of policies that are germane to the SDMZ network infrastructure:

Policy 1. *Dataflow Policies:* The nature of SDMZ experiments involve the transport and computation of project datasets with diverse sensitivities. Ideally, they should be data centric rather than IP flow centric. For example, a single experiment may include both public data and data involving personal information. Policies should have the ability to express fine-grained controls over where data can be transmitted or received based on the type and sensitivities of the data.
Examples: (Ex. 1) - Sensitive data derived from experiments of project P1 in site S1 is to be only shared among nodes running P1. If projects P1 and P2 are co-resident in an SDMZ node, P2 users or applications may not exfiltrate P1 data to other nodes. (Ex. 2) - Application binaries that are not white-listed are not allowed to access sensitive files or send packets of size greater than X bytes using protocols such as DNS and NTP. (Ex. 3) - Sensitive data derived from project P1{experiment2} in Site1 (e.g., D2) is not to be shared with Site2, Site3, their collaborators and blacklisted countries, including the transformed output data (i.e., derivatives D2/*). Also, not allowed to be accessed by any application or user which/who collaborates with Site2 and Site3 i.e., S1{D2/*} !⟶ {S2,S3}).

Policy 2. *Temporal and Spatial Policies.* As the SDMZ is a federation of shared and independently managed resources, the operator should be capable of defining resource utilization policies based on time, network address space, or geography.
Examples: (Ex. 1) Sensitive and confidential science data produced by project P1 is not allowed to be accessed or transmitted before 9 AM and after 6 PM, i.e., in the absence of data administrators, to prevent malicious data access. (Ex. 2) Export-controlled scientific data derived from project P2 is not to be transmitted to any ITAR-restricted countries.

Policy 3. *Network Security Policies.* Policies should be adaptable to address the dynamic security state of the network.
Example: Notify admin and quarantine hosts to prevent any sensitive data transfers outside DTNs if there is evidence of a successful brute-force attack.

2.1.2 Infrastructure Abstraction & Policy Specification: As illustrated in Figure 2, existing policy frameworks [2, 19, 36] do not provide the required isolation between project

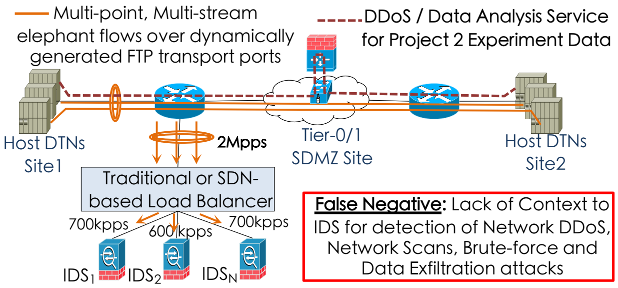

Multi-point, Multi-stream elephant flows over dynamically generated FTP transport ports
DDoS / Data Analysis Service for Project 2 Experiment Data
Host DTNs Site1
2Mpps
Tier-0/1 SDMZ Site
Host DTNs Site2
Traditional or SDN-based Load Balancer
700kpps
600 kpps
700kpps
IDS1
IDS2
IDSN
False Negative: Lack of Context to IDS for detection of Network DDoS, Network Scans, Brute-force and Data Exfiltration attacks

Figure 3: Lack of context in detecting missed network and application-level attacks with clustered IDS.

users and site administrators. The infrastructure details, abstracted for specifying the policies, need to be effectively isolated for protecting the network infrastructure details from getting exposed to other unintended SDMZ users. The SDMZ network infrastructure involves users (i.e., researchers who are non-administrative users) who should identify the necessary network resources and security services required for enforcing the *data*-specific policies of their projects. These policy rules are manually inserted (and often statically configured) into routers and monitoring devices.

2.1.3 Conflict-free Policy Enforcement: Furthermore, the policies must provide necessary project-level isolation while enforcing the policies on sites where: (*i*) each project spans across multiple sites, and (*ii*) multiple projects share the same host DTN and network infrastructure. Conflicting *access control policies* 'or' *QoS policies* (i.e., to share the network resources) involving multi-project traffic from a shared DTN should be effectively de-conflicted before insertion onto enforcement devices. While traditional enforcement mechanisms require dedicated DTNs and network infrastructure elements per project, this restriction is inefficient and impedes the ability to dynamically manage SDMZ networks. Consider for example following two policies from two different projects Project 1 and 2 as shown below:

Project 1: HostDTN[1-10]: GridFTP ⟶ ALLOW ⟶ Internet.

Project 2: HostDTN[3-7]: GridFTP ⟶ DENY ⟶ Internet.

From above two policies specified in Site1, the first policy from Project 1 allows FTP application data to be sent to the Internet from hosts 1 − 10, while the same type of traffic from a subset of hosts are DENIED as part of Project 2.

2.1.4 Contextual Awareness: Consider the case of SDMZ data-transfer applications [10] (e.g., GridFTP, bbftp, bbcp), which are multi-point, and multi-stream applications where a single dataflow can be transferred in parallel as multiple data streams on to multiple data nodes. Consolidating or correlating the distributed, parallel TCP streams (i.e., either clear or opaque traffic) is difficult as the TCP port numbers used in the data transfer is dynamically negotiated using GridFTP's secure control messages. As shown in Figure 3, various attacks such as network- or application-level DDoS, data exfiltration and brute-force attacks could go undetected with a clustered IDS solution. To dynamically allow experimental data from various sites to be properly filtered and steered according to security conditions, each site which originates the data should provide additional contextual information. When tier0/1 DoE sites with advanced security services detect security vulnerabilities they should share these details with the site that originates the data for collaborative protection.

3 The `CoordiNetZ` System Framework

We introduce a coordinated and context-aware security framework (`CoordiNetZ`) that is designed to address the SDMZ requirements for enforcing security to dataflows in multi-tenant, multi-project, and multi-administrative environments. The key elements of the `CoordiNetZ` system include:

1) *Tree-based Infrastructure Abstraction Engine*: employs *abstraction-mappings*, which automatically generate isolated *tree-based abstractions* (i.e., required to specify policies) of the infrastructure that is specific to each administrator's or user's role and scope of control (§4.1).

2) *Graph-based policy specification*: allows specification of graph-based policies with simple drag-and-drop syntax of nodes from abstraction trees supplied to each administrator (§4.2).

3) *Conflict Detection & Resolution*: facilitates composition technique for conflict detection and resolution among policies that are produced by different project and site administrators (§4.3).

4) *Inter-Site & Intra-Site Context-Aware Tagging*: associates policies with context-aware tagging[2], which is required for dynamically filtering dataflows on the basis of associated security conditions. We develop technique to allocate tags to policies associated with each project based on the graph edge coloring approach while considering the limited tag size supported in IPv6 with 20 bits Flow Label packet header (§5).

3.1 System Components

Figure 4 illustrates the major components of `CoordiNetZ` and their integration points within the SDMZ. Its purpose is to enable users among a broad range of roles (e.g., project scientists, site administrators, project data administrators) to express their policies using a tree-based abstraction, and then enforce these policies on large data science projects that are hosted across independently managed SDMZ institutions.

`CoordiNetZ` integrates following enhancements to SDMZ:

[2] Necessary context required for enforcing security to dataflows is provided by host DTNs and other protection mechanisms (e.g., IDS) deployed in the SDMZ.

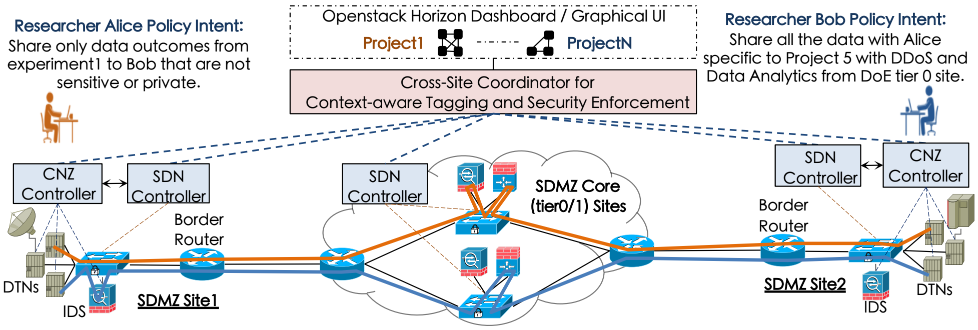

Researcher Alice Policy Intent:
Share only data outcomes from experiment1 to Bob that are not sensitive or private.
Openstack Horizon Dashboard / Graphical UI
Project1
ProjectN
Cross-Site Coordinator for Context-aware Tagging and Security Enforcement
Researcher Bob Policy Intent:
Share all the data with Alice specific to Project 5 with DDoS and Data Analytics from DoE tier 0 site.
CNZ Controller
SDN Controller
Border Router
DTNs
IDS
SDMZ Site1
SDN Controller
SDMZ Core (tier0/1) Sites
SDN Controller
CNZ Controller
Border Router
SDMZ Site2
IDS
DTNs

Figure 4: The CoordiNetZ dataflow policy specification and enforcement architecture with two users specifying policy intents for securing *sensitive* data using security services provided by tier 0/1 SDMZ transit sites.

1) `CNZ Coordinator`: `CNZ Coordinator` acts as the centralized manager for specifying cross-site project policies from multiple users, both through policy files and a graph-based user interface. It implements the following key capabilities: (*a*) intent-based framework, (*b*) tag-based policy enforcement, (*c*) manages tag space allocation mechanism for assigning the range of tags across projects, and (*d*) builds abstraction trees based on stats from the CNZ Controller. It uses Openstack Horizon UI [35] for building graph-based policy specification. The abstraction engine of `CNZ Coordinator` built on Openstack Congress engine [7], with datalog rule generator module developed to generate infrastructure abstraction trees.

2) `CNZ Controller`: The `CNZ Controller` acts as a mediator between the DTNs and the `CNZ Coordinator`. It analyzes each host's DTN records for malicious data flows and consolidates data-flow records and DTN state information required for building the abstraction trees at the `CNZ Coordinator`. It translates site-specific policies provided by the `CNZ Coordinator` into host- and network-specific rules. Our custom built `CNZ Controller` handles the data records from host DTNs and REST APIs are used for exchanging and triggering the policies between the `CNZ Controller` and SDN controllers.

3) SciMon: The `SciMon` module builds contextual information at host-process level to tag the flows originating from host DTNs and enforces host and process-specific data policies in DTNs. It tracks file's accesses with each process instantiation and imposes file access and network data-flow restrictions according to the rules in process policy table. It continuously monitors the host processes, file system accesses, and network IO events using open-source utilities, such as `psutils` [37] and `osquery` [34]. As shown in Figure 5, the process flow table consists of policy flow rules that dictate the user's, application's, or process' ability to access the data and send it over the network.

4) SciFlow: The `SciFlow` module, runs as a daemon to continuously monitor for flows generated from a specific interface inside the host and triggers `SciMon` to gather user and process attributes, file I/O, and application binary information associated with this network flow. Flow records gathered by `SciMon` and `SciFlow` are sent to the `CNZ Controller` for further processing (see Figure 6). The fields that are extracted from the host and network flows are customized per `CNZ Controller`'s policies.

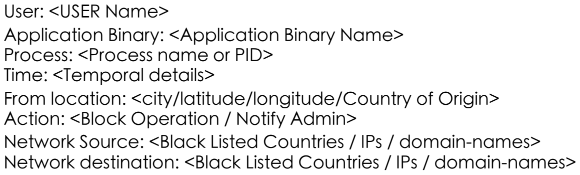
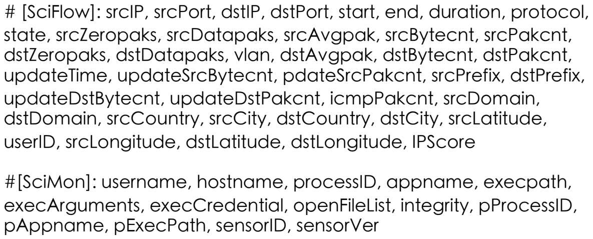

User: <USER Name>
Application Binary: <Application Binary Name>
Process: <Process name or PID>
Time: <Temporal details>
From location: <city/latitude/longitude/Country of Origin>
Action: <Block Operation / Notify Admin>
Network Source: <Black Listed Countries / IPs / domain-names>
Network destination: <Black Listed Countries / IPs / domain-names>

Figure 5: Sample process flow table entry.


[SciFlow]: srcIP, srcPort, dstIP, dstPort, start, end, duration, protocol, state, srcZeropaks, srcDatapaks, srcAvgpak, srcBytecnt, srcPakcnt, dstZeropaks, dstDatapaks, vlan, dstAvgpak, dstBytecnt, dstPakcnt, updateTime, updateSrcBytecnt, pdateSrcPakcnt, srcPrefix, dstPrefix, updateDstBytecnt, updateDstPakcnt, icmpPakcnt, srcDomain, dstDomain, srcCountry, srcCity, dstCountry, dstCity, srcLatitude, userID, srcLongitude, dstLatitude, dstLongitude, IPScore

#[SciMon]: username, hostname, processID, appname, execpath, execArguments, execCredential, openFileList, integrity, pProcessID, pAppname, pExecPath, sensorID, sensorVer

Figure 6: DTN flow record field (`Flow Record = Timestamp + SciFlow Record + SciMon Record`).

5) Stateless Microservices: Our security-based microservices are based on the DPDK platform [17]. We implement each security functional capability as a light-weight stateless microservice with their states externally stored to shared memory [27]. The micro-services based functions that we implemented include tag-based filtering, rate limiting, spoof protection, connection tracking based on IP tuples and tags.

3.2 Threat Model

We adopt a simple threat model which assumes that: (*i*) a subset of SDMZ users and administrators who program the SDMZ ecosystem could be malicious resulting in embedding rogue policies within a project, (*ii*) the applications inside the host DTN such as GridFTP and other data transfer applications could be compromised. We attempt to address issues that arise from flaws in the implementation of traditional SDMZ policies as well as vulnerabilities introduced by malicious users.

CoordiNetZ proposes to use a *conservative* approach by which administrators can program dataflow policies from specific users and for specific set of host DTNs such that their capabilities are given higher precedence compared to security and privacy policies of others. Though the precedence is completely programmable, it solely depends on administrator's ability to correctly specify it, which could at times mask the conflicts. Hence, *precedence* operator need to be used diligently by administrators for auto-resolving the conflicts that are detected by CoordiNetZ, which could otherwise be safely resolved by administrators.

Our threat model broadly considers various attacks such as data exfiltration, spoofing attacks, and DDoS. CoordiNetZ protects the SDMZ network from such attacks at switch (i.e., at the immediate first-hop network node) with the help of tags inserted by the SciMon application and context gathered from within the host and network. These security use cases are described in more detail in Section 6.

4 Dataflow-based Policy Framework

CoordiNetZ, provides following key capabilities for addressing the security policy requirements of SDMZ networks: (*i*) effectively isolates the policies specified across different administrative-domains, across sites and projects, using a tree-based abstractions, (*ii*) graph-based policy specification mechanism that captures multi-dimensional policies (e.g., temporal dynamics, security states, spatial attributes), and (*iii*) provides efficient policy-conflict detection and resolution mechanisms across the shared network infrastructure.

4.1 Infrastructure Abstractions

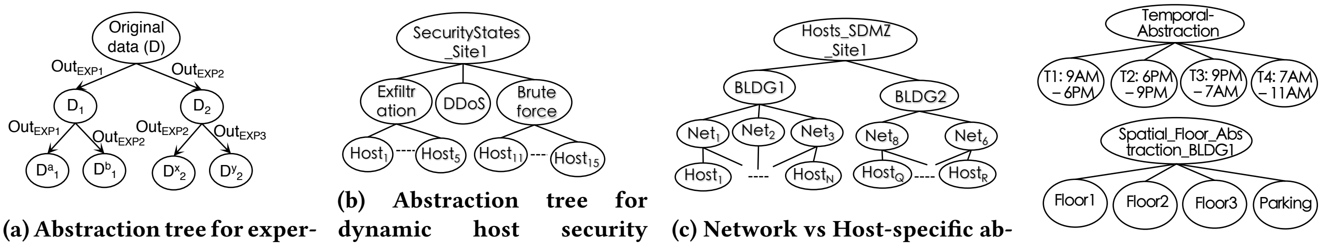
In this section, we present an approach called *abstraction mapping* that allows a global site administrator to delegate policy responsibilities of SDMZ infrastructure to SDMZ network administrators and project users. Abstraction mappings allow users and administrators to visualize an isolated view of the infrastructure (i.e., as infrastructure abstraction trees), over which the user may specify policies. Figure 7 illustrates examples of infrastructure abstractions exposed to administrators/users and *abstraction-mappings* specific to each abstraction tree is also shown.

Administrators provide *abstraction-mappings* as input to the CNZ Coordinator for facilitating the construction of abstraction trees (see Figure 7). Abstraction mappings enable CoordiNetZ to stitch together different types of abstractions within the same tree. For example, in Figure 7b the security states and list of hosts of Site1 are combined. Here, each level of infrastructure abstraction-type is separated using the colon operator (':'), while the dot operator ('.') denotes the properties of each level of abstraction-type. Each dot-separated abstraction narrows the list of host DTNs captured within the abstraction tree. List of few system-defined abstraction-type-mapping parameters includes locations{}, buildings{}, networks{}, sites{}, security-states{}, which captures the spatio-temporal, security and network properties and their relation with the hosts, which allows the policies to be naturally expressed using intuitive heterogeneous types of abstraction trees. Figure 7d shows few more infrastructure abstractions generally used in SDMZ network infrastructures for configuring the policies.

4.2 Policy Specification

CoordiNetZ provides a graphical drag-and-drop user interface for specifying the graph-based dynamic dataflow-based policy intents, while the existing techniques supports only static policies that are flow-based [2, 19, 36]. Policy graphs constitutes of nodes from various infrastructure abstraction trees that are assigned to each administrator (see Figure 8a). An equivalent policy specification syntax for configuring large scale policies bypassing the graphical user interface (shown in Figure 8b) is also provided. CoordiNetZ's policy specification framework supports three types of static and dynamic policies to accommodate the needs of SDMZ network: (*i*) *temporal-and-spatial* policies, (*ii*) *data-specific* policies, and (*iii*) *network-security* policies.

The policy specification syntax used for representing graph-based policies is shown in Figure 8. In this ACL-based policy specification syntax, the permissions to allow or deny communication between source and target nodes are specified using => (i.e., for ALLOW), ! => (i.e., for DENY), and !X=>(i.e., for QUARANTINE) operators. Sequential (>>) or parallel (||) operators specify the sequence of network functions through which the traffic from specific source node should traverse. The → operator used in the *parent-path* key-value pair is used to define the path of the node (i.e., used in policy specification) from its root node, which is required to capture the relation among the nodes of same abstraction tree.
Dataflow-based Policies: The current SDMZ infrastructure does not provide any capabilities for enforcing cross-site dataflow policies (discussed in § 2.1). While prior work discussed dataflow tracking within a host and across hosts [23, 28, 32, 48], these frameworks are heavyweight and do not address the performance requirements of the SDMZ. Hence, we implement a lightweight forensic tracker and use the CNZ Coordinator to support two key data-tracking capabilities: (*i*) ability to capture all read/write operations carried out on the data within a host (shown in Figure 7a) and (*ii*) ability to effectively capture the dataflow across hosts and associated data transformation restrictions (shown in Figure 9).



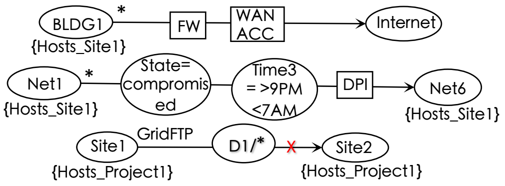
(a) Abstraction tree for experimental data outcome (AM = data{*}.experiment {Exp1}).

(b) Abstraction tree for dynamic host security (AM = security-state{*}.site {Site1}:hosts{*}).

(c) Network vs Host-specific abstractions of Site 1. buildings{*}. site{Site1}:networks{*}:hosts{*}

(d) Temporal and spatial abstractions Trees.

Figure 7: Infrastructure abstraction trees used for policy specification in SDMZ with their respective *abstraction-mappings* (i.e., abstractions specific to hosts, users, experiment data outcomes, temporal and spatial abstractions.



(a) Graph-based ACL policy specification (P1 – P3).

Spec{Site-Admin}: src-node{BLDG1}.parent-path{Hosts_Site1}.traffic-type{*} >> FW >> WAN-Accelerator => target-node{Internet}.parent-path{networks}

Spec{Site-Admin}: src-node{Net1}.parent-path{Hosts_Site1->BLDG1}.traffic-type{*}.time{Time3}.state{compromised} >> DPI => target-node{Net6}.parent-path{Hosts_Site1->BLDG2}

Spec{Project-Admin}: src-node{Site1}.parent-path{Hosts_Project1}.traffic-type{GridFTP}.data{D1/*} !=> target-node{Site2}.parent-path{Hosts_Project1}

(b) Equivalent ACL policy specification syntax (P1 – P3).

Figure 8: Graph-based policy specification & syntax.

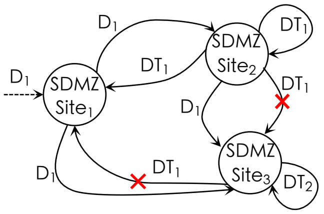
We define the following properties and capabilities to track the data across SDMZ sites: (*i*) unique data identifier across sites within a project and (*ii*) mechanism to capture the relation between "original" and "transformed" data. The unique data identifier is required to identify multi-site dataflows and capture their transformation in the future. This also allows SDMZ project users to effectively query the dataflow and data transformation details over a temporal window. The relation between the original and the transformed data is captured at each of the SDMZ project hosts and shared with the CNZ Coordinator for building data flow graphs. Dataflow tracking helps to restrict dataflow violations using a high-level policy specification language (as shown in Figure 9).

CoordiNetZ's *abstraction trees* and *graph-based specification* allows the administrators to specify the policies explicitly using the abstraction trees exposed to each of the IoT users or administrators. This approach of isolating and assigning explicit infrastructure abstractions to each admin, allows CoordiNetZ to prevent admins from specifying policies on the infrastructure they do not own i.e., preventing *rogue policies* from being specified. For detecting *rogue policies*, the policy composition engine extracts the source and target nodes from the specified policies and verifies if both the nodes belongs to the *policy abstraction trees* owned by that administrator. Further, these policies are composed together for detecting other conflicts and violations (as discussed in §4.3.1).

4.3 Policy Composition & Deconfliction

In multi-site environments, where hosts and network entities are shared across multiple projects, conflicts are bound to arise when policies are being specified independently by each SDMZ user. Conflicts might even arise among policies specified within the same project or across projects, when multiple users are involved in policy specification. In SDMZs, the need for project- and site-specific policies further increase the potential for conflict. Run-time policy composition must also address the dynamic needs of ephemeral projects, resulting in the need to perform periodic policy recomposition. Hence, CoordiNetZ facilitates automatic policy conflict detection and deconfliction in three steps: (*i*) automated detection of conflicts among policies (i.e., specified by various administrators) within a project, (*ii*) automatic conflict resolution, and (*iii*) decomposition of policies into logical groups for efficient tag assignment to enable policy enforcement among multiple projects in shared infrastructure.

4.3.1 Composition Algorithm: The composition engine accepts a list of policies $\forall p_i$ and an empty bi-partite graph G as input. Each policy in the list L is serially composed, producing a final consolidated graph G. For each policy p_i to be composed with G, the composition engine first checks for existence of any source nodes $S_j(G)$ in G that has any relation with policy's source node $s(p_i)$ If $s(p_i)$ overlaps with any source node in $S_j(G)$, then the composition engine evaluates the edges of source node $E(S_j(G))$ of G for matching policy edge conditions $b_{p_i}(s,t)$. If any of the edges of $S_j(G)$ has an edge match condition with p_i (i.e., an overlap or subset or exact match relation), then $t(p_i)$ is checked for a target node match associated with $S_j(G)$.



```
Simplified Data Policy Syntax:
site{Site1}.data{D1} -> site{Site2}
site{Site1}.data{D1} -> site{Site3}
site{*}.data{D1/*} !-> site{Site3}
/* Default drop enforced on rest
   of traffic automatically */
```

Figure 9: Project-specific graph- and syntax-based policy specification for data-flow policies.

Depending on the overlap relation among source, edge, and the target nodes, the policy is either declared a *conflict*, a *duplicate* or *non-conflicting*. Duplicate policies are not added to the graph, but increment a counter and an entry to maintain the duplicate-policy association. If the composition engine finds an overlap relation, it labels the policy as a *conflict*. It then checks for matching precedence rules for the policy p_i and policies associated with $S_j(G)$. If precedence operation exists for that policy, it proceeds to conflict resolution, while unresolved policies are declared as conflicts. If no overlaps exist, the policy is non-conflicting, and hence new nodes and edges are added to the composed graph. The overall complexity of the composition Algorithm 1 involves the following factors: (a) the number of policies (L); (b) number of source nodes (S_J) in the composed policy graph; (c) the number of source nodes that have overlap relations resulting in edge comparisons (L_e); and (d) the number of target nodes that are compared for overlap relations (L_t). The overall worst-case complexity of the algorithm is O($L * S_J * L_e * L_t$), which is quadratic. The overall composition complexity T_c is calculated as follows:

$$T_c = \sum_{i=1}^{L}\Bigg(\sum_{j=S_j \epsilon G}^{S_{J_i}} O(m+n) + \Bigg(\sum_{k=(S_i,S_j)\epsilon E}^{L_{e_j}} K_e * O(q+r) + \Bigg(\sum_{l=(S_i,S_j)\epsilon E}^{L_t} O(u+v)\Bigg)\Bigg)\Bigg) \qquad (1)$$

We propose incremental policy composition mechanism to accommodate the dynamic changes to the SDMZ network infrastructure and perform policy composition in sub second latency and reduce the complexity of our composition algorithm through optimizations (discussed in §4.3.3 & §4.3.4).

4.3.2 Precedence & Decomposition : Automated conflict resolution employs precedence operators to resolve conflicts among competing policies. CoordiNetZ uses three separate forms of policy precedence evaluation. *Administrator-level precedence* enables precedence evaluation based on the scope of authority for policies authored by administrators. For example, in SDMZs the site administrators are granted precedence over project administrators. *Action-level precedence* allows one action to take precedence over another. For example, the Drop action may supersede Allow or Quarantine or redirect. That is, Drop > Allow > Quarantine > Redirect. *Custom precedence* enables policy attributes, such as user or experiment or projects, to be associated with precedence. For example, policies specific to Experiment X of Project1 may be granted precedence over policies from Experiment Y of Project2, especially with shared network resources.

Algorithm 1: Graph-based Policy Composition

```
$L \leftarrow$ list of policies for composition;
$s(p) \leftarrow$ source node of policy $p$;
$t(p) \leftarrow$ target node of policy $p$;
$a(p) \leftarrow$ action of policy $p$;
$b_p(s,t) \leftarrow$ edge properties for the policy $p$ between nodes $(s,t)$;
$G \leftarrow$ Composed k-partite graph;
$S(G) \leftarrow$ source node of the Graph$G$;
$E(S(G)) \leftarrow$ edges associated with the node source S on the Graph G;
$T(E) \leftarrow$ target associated with edge;
$A(E) \leftarrow$ action on the Edge;
forall Policy $p \in L$ do
    foreach source node $S(G) \in G$ do
        if $s(p_i)$ overlaps $S_j(G)$ then
            foreach edge $E(G) \in S_j(G)$ do
                if $b_{p_i}(s,t)$ overlaps with $E_k(S_j(G))$ &$a(p_i) \neq$
                  $A(E_k)$ then
                    if $t(p_i)$ overlaps $T(E_k)$ then
                        if no or equal precedences then
                            Alert: Raise Policy Conflict;
                        else
                            Auto Resolution: Approach §4.3;
                            decompose $(p,G)$;
                    else
                        Create new $t(p_i)$ node in G;
                else if $b_{p_i}(s,t)$ overlaps $E_k(S_j(G))$&$a(p_i) ==$
                  $A(E_k)$ then
                    if $t(p_i) \subseteq T(E_k)$ then
                        Discard Duplicate Policy;
                        Add track entry;
                        $p_i.counter \leftarrow p_i.\{counter{+}{+}\}$;
                    else if $t(p_i) \notin T(G)$ then
                        Create new edge for the $p_i$ in $G$;
                else
                    Create new edge for the $p_i$ in $G$;
        else
            Create $s(p_i)$ & $t(p_i)$ hash entries in $G$;
            Create new edge $b_{p_i}(s,t)$ in $G$;
```

When two policies (P1 and P2) conflict (e.g., policies in Figure 8a), the nodes and edges of the policies are decomposed into set of subset nodes that requires the least number of edges to represent conflict-free policies. Based on the precedence, the overlapping nodes that result in conflict are removed. For P1>P2, the edge specific to the policy with highest precedence (P1) is retained and the overlapping portion of edge property specific to other policy (P2) is removed and vice versa. In this case, the total number of edges required to represent the conflict-free composed graph is N + 1, where N is approximately total number of different edge properties that result in conflicts. From the composed graph all the nodes and edges that are resolved for conflicts specific to a policy are assigned the same tag. Here, the number of tags required is approximately equal to the number of conflict-free policies.

4.3.3 Incremental Policy Composition. Policy updates are necessary whenever network conditions and security states change, site topologies are modified, or when projects are added, migrated, or completed (removed). When such changes occur, the policy composition and conflict resolution must be recomputed. In general, policy updates could result in tens to hundreds of rule modifications. Incremental composition helps reduce the overall run-time of composition, by avoiding the recomposition of the whole policy state, which may consume several seconds to minutes (see §7(2)). Rather, incremental composition recomposes only the updated set of policies with the whole set of composed policies.

Updating a policy from the composition graph involves first deleting the policy from the graph, and then inserting a modified version. Deleting a policy requires one to remove the edges that belong to the policy from graph. However, the composition procedure might have removed portions of other policies that had a higher precedence during conflict resolution. Hence, these lost portions must be returned.

Two items are recorded during composition that accelerate incremental composition time. First, for each original policy, a reference pointer to each edge is maintained in the graph that belongs to the policy. If a policy is split into multiple sub-policies during conflict resolution, the edges associated with these child policies are stored. Second, during a conflict, if a policy that has a higher precedence causes the policy to be split into multiple sub-policies, then the policy number of the lower-precedence policy is recorded in a data structure associated with the higher-precedence policy. During deletion, this data structure enables CoordiNetZ to restore edges from the deleted policy when it finds that other policies also depend on these edges. This internal bookkeeping enables edge deletion in constant time, resulting in orders of magnitude faster overall composition.

From the aforementioned equation (1) in §4.3, the major time complexity of the algorithm lies with the iteration of policies O(L) over the list of all source nodes S_J in the composed bi-partite graph G and comparing the policy's source node $s(p)$ having m host entities with the graph's source node $S(G)$ having n entities. The overall complexity calculation for finding overlaps among the source nodes stage is therefore $O(L * S_J * (m * n))$. Consider the policy's source node $s(p)$ has m host entities (i.e., $h_1, ... h_m$), and the graph's source node $S(G)$ has n host entities (i.e., $h_1, ... h_n$). The naive comparison of two subsets of size m and n will result in $O(m * n)$ complexity. Similarly, for each property defined on the edge, the composition engine incurs a computation complexity of $O(q + r)$, where q and r are the number of entities associated with edge properties of composed policy graph G and the edge property of the policy p_i. For the list L_t overlapping edges, the composition engine checks for the overlap in the target node of policy p_i that has u host entities with the edge of the composed policy graph G with v host entities, incurring computation complexity $O(u+v)$.

4.3.4 Policy Composition Optimization. To reduce the complexity from $O(m*n)$ to $O(m+n)$, we employ a hashing algorithm: the m host entries of $s(p)$ are hashed as key-value pairs. Then the host entities of $S(G)$ are looked up in the hash for the existence of the host n. As the hash lookup complexity is O(1), the total subset calculation complexity results in $O(m+n)$ complexity. Now, the baseline complexity will be reduced to: $O(L* S_J *(m+n))$. To further reduce this complexity, we Caching the comparison calculation outcome as key-value pairs in the hash further reduces complexity: the $s(p)$:$S(G)$ as key and the value as the first comparison result. Hence, the next node comparison can be extracted from the hash entry with an O(1) lookup cost. This reduces the overall baseline complexity to $O(L*S_J)$.

Similarly, the edge properties and target nodes of p_i and $S_j(G)$ are added on top of the baseline composition cost. Any complexity beyond the baseline comparison will be present only when there exists an overlap in the edges properties or target nodes. Therefore, the source-node overlaps trigger checks for edge-property overlaps and these occurrences then necessitate target node comparisons resulting in a worst case complexity of $O(L * S_J * L_e * L_t)$. Similarly, by hashing the comparison results of edge nodes and the target nodes, we can eventually reduce the complexity to $O(L*S_J)$. A proof of this complexity bound is outside the scope of this paper but will be provided in an expanded technical report.

5 Context-Aware Tagging

The SDMZ network lacks efficient techniques to differentiate traffic based on: (*i*) *static project-specific attributes* (e.g., project id, project user, experiment id) that identify the source of the science data traffic, (*ii*) *dynamic network security attributes*, (e.g., malicious, compromised, DDoS, or exfiltration host) that describe dynamic security state of the network, and (*iii*) site-specific attributes that enable isolation and conflict-free policy enforcement for projects spanning multiple sites with each site hosting multiple projects. We present a tag-based policy enforcement mechanism for fine-grained traffic filtering and inter-site sharing of security services.

1) Intra-Site Tag Assignment: Fine-grained traffic filtering is provided to SDMZ network using tag-based policy enforcement mechanism supported using IPv6 flow label (20 bits). IPv6 is a natural choice for flow tagging as it affords greater tag space and its use is strongly urged by the SDMZ community [43]. The tag assignment to policies happens within each

Algorithm 2: Tag allocation across sites (Edge Coloring).

```
$C_T \leftarrow$ List of $T$ colors with their respective tag space sizes;
$S_N \leftarrow$ Total number of sites involved in policy management;
$P_{S_i}\{\} \leftarrow$ List of projects in each site;
$T_{p_i}\{\} \leftarrow$ Tag size requirement of each project;
$C_{S_i}\{\} \leftarrow$ List of unique colors assigned to each site;
$C_{SA_i}\{\} \leftarrow$ List of colors associated with $S_i$'s adjacent site $P_{S_i}$;
$C_{temp}\{\} \leftarrow$ List of colors with tag size $> T_{p_i} + T_slack$;
foreach site $S_i \in S_N$ do
    foreach adjacent sites $S_{A_i} \in S_i$ do
        foreach project $P_i \in P_{S_i}$ do
            if $P_i$.{color} == NULL then
                $C_Temp\{\} \leftarrow C_T\{\} > T_{p_i} + T_{slack}$;
                $P_i.\{color\} \leftarrow min\Big(C_{Temp}\&\&C_{Temp} \notin C_{S_{A_i}}\Big)$;
    $S_i \leftarrow S_{A_i}$ Adjacent site of $S_i$ with project tag space unassigned.;
```

project at the site-level by the `CNZ Controller`. The host DTN assigns tags to each flow associated with policies for logically grouping the flows or forwarding them in accordance with dynamic network security conditions (§5.1).

2) Inter-Site Tag Space Allocation: While the tag assignment decision happens locally within each site, we use the centralized `CNZ Coordinator` for allocating the tag space (i.e., range of tags) for each project. Our *inter-site tag space allocation* mechanism, assigns the tag space to each project registered with the `CNZ Coordinator`(§5.2).

5.1 Intra-Site Tag Assignment

To extend fine-grained traffic filtering capabilities, beyond contemporary IP-based mechanisms, we develop an efficient context-aware policy-based tagging mechanism, called cTags, that enables:

- Logically group traffic that spans across subnets, hosts or geographic locations for policy enforcement.
- Dynamically *steer*, *revoke*, or *forward* traffic across different Network Function Chains (NFC) according to dynamic network and security conditions.

Although the tag assignment is carried out by the `CNZ Controller` of any site, the actual tag is embedded into the flow by host DTNs for traffic generated from the host applications depending on the configured policies. The conflict-free policies supplied by the `CNZ Coordinator` are reconciled to site-specific policies and further translated to device-specific rules by the `CNZ Controller` and SDN controller before being placed across host DTNs and SDN switches. The set of rules supplied to each host, which we call as *policy-to-tag mappings* captures following details:

- `tagID{T1} => policyID{P1}:appID{A1}:userID{U1}:expID{E1}`
- `policyID{P1} => policySpec{...}`

The mappings carry necessary details specific to each policy and the associated entities for enforcement. The `SciMon` module can dynamically change the flow `tagID`, even in the middle of flows depending on dynamic network conditions, by updating the *policy-to-tag mapping* entry.

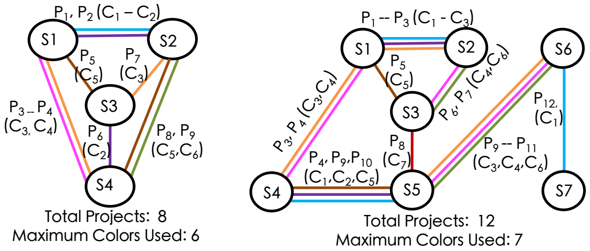

P_1, P_2 (C_1 – C_2)
S1
S2
S3
S4
P_5 (C_5)
P_7 (C_3)
P_3 -- P_4 (C_3, C_4)
P_6 (C_2)
P_8, P_9 (C_5,C_6)
Total Projects: 8
Maximum Colors Used: 6
P_1 -- P_3 (C_1 - C_3)
S1
S2
S6
S3
S4
S5
S7
P_5 (C_5)
P_6, P_7 (C_4,C_6)
P_3, P_4 (C_3,C_4)
P_{12}, (C_1)
P_8 (C_7)
P_4, P_9,P_{10} (C_1,C_2,C_5)
P_9 -- P_{11} (C_3,C_4,C_6)
Total Projects: 12
Maximum Colors Used: 7

Figure 10: Tag-space allocation with edge color assignment. Project IDs and colors are used to annotate each edge.

Each SDMZ site needs to optimize the number of rules required to enforce the policies by considering the availability of high-bandwidth switches and their switch TCAM space. Tagging facilitates rule space optimization by: (*i*) allowing large number of hosts to be grouped into a common logical entity (i.e., beyond IP-tuple-based filtering) and (*ii*) efficiently assigning contiguous tags such that policies having same action attributes may be grouped together using bit masking. Each policy is simply associated with a unique tag after resolving the conflicts among the policies. From the composed graph all the nodes and edges that are associated with a policy is assigned the same tag. Here, the number of tags required is approximately equals the number of conflict-free policies. Further the optimization proposed to tag space utilization in collaborative SDMZ network is discussed in next Section 5.2.

5.2 Inter-Site Tag Allocation

In the SDMZ infrastructure, security and data analysis services provided by higher-tier sites (i.e., tier-0 or tier-1 DoE sites) are availed by lower-tier sites [6, 24]). For effectively sharing such services across sites, tags assigned by one site must be honored by other sites handling the same project.

To avoid conflicts in tag space utilization we propose a *unified tag space allocation mechanism* that allocates necessary tag space to each project (with additional slack tag space for future policies). Though, the tag space allocation is carried out globally at the `CNZ Coordinator`, the tag assignment to each policy is carried out locally within the site with the help of `CNZ Controller`. As a design choice, we use IPv6 flow label bits as *tagID*. Since 20 bits of flow label header in IPv6 cannot effectively accommodate the tagging requirement of thousands of projects handled across hundreds of SDMZ sites we need a centralized tag-allocation mechanism to effectively reuse tag bits across projects spanning multiple sites.

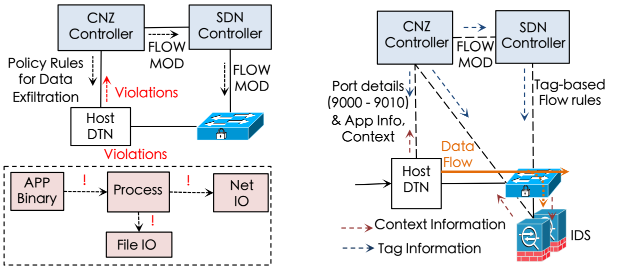
We assign a specific color to each project within a site and reuse the same color among other projects across other sites registered with `CNZ Coordinator` with following two design considerations: (*i*) the tag space should never overlap with the tag space assigned to its immediate adjacent sites with which the current site has project association, and (*ii*) tag size assigned to each project depends on the number of policies enforced by the project. The key objective of the *tag-space allocation* mechanism described in Algorithm 2 is to maximize the efficient reuse of tag space (i.e., colors) among cross-site projects, while avoiding overlaps.

Algorithm 2 details the tag-space allocation mechanism used by the `CNZ Coordinator` to allocate a range of tags to each of its registered projects. The `CNZ Coordinator` traverses through the list of all S_N sites associated with it in a breadth-first-search manner. For any chosen site S_i, its adjacent sites are compared before allocating colors. We observe that for each site S_i and its adjacent sites the complete list of available colors can be used in the assignment procedure.

The colors are assigned between S_i and S_{A_i}. Each of the adjacent sites S_{A_i} of S_i (depending on the the list of projects belonging to S_{A_i} that are associated with S_i), are assigned one color per project (depending on their policy size). Colors are assigned to each project in S_{A_i}, that is associated with S_i, such that: (*i*) it satisfies the project's tag space requirement, S_{A_i}, (*ii*) the color with the least size is considered for assignment and (*iii*) no other projects in S_{A_i} have the same color already assigned to it. Similar approach is taken for all projects that are associated with site S_i having adjacent node S_{A_i}. This procedure is carried out for all the adjacent sites of S_i. When the list of adjacent sites of S_i is exhausted, the `CNZ Coordinator` picks the next site from S_{A_i} as the new S_i, carrying out the aforementioned procedure until all sites in S_N are iterated atleast once. An example illustration of our algorithmic outcome is shown in Figure 10. The algorithm is quadratic in the number of sites in the worse-case for a fully connected graph (i.e., all sites share all projects). As the number of sites does not change frequently, the overall complexity grows linearly with the number of projects. To further optimize the tag space utilization and efficiently reuse the tag space we propose technique, which is discussed in Section 5.3.

5.3 Optimization of Tag-space Reuse

Depending on the tag-space allocation mechanism discussed above in Section 5.2, each project is assigned tag space, while taking into consideration the tag space requirement of the project plus the slack space (i.e., range of tags that are left for future use). This allows each project to expand its policies either due to the dynamic network conditions or from new policy additions by administrators. If the slack space is completely consumed by project P_i, then the new range of tag space is assigned using one of the following methods: (*i*) considering the non-overlapping tag space assigned to other projects inside site S_i and its adjacent sites S_{A_i}, a new color is added to the project P_i, and (*ii*) if there are no flows for the project for which the active rules exist, then project P_i is assigned a new color, recalculating tag space allocation between S_i and S_{A_i}. The approach described above in (*ii*) is used only when there exists spare colors unassigned between the S_i and its adjacent sites S_{A_i}.



(a) Data exfiltration. **(b) Collaborative protection.**

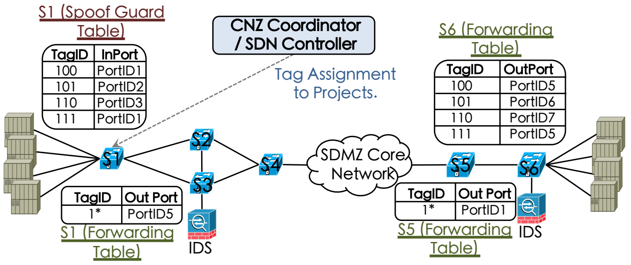
Figure 11: Security use cases with context-awareness.

However, during tag space allocation, when a project P_i from site S_i requires less tag space than what is available, to optimize the tag space utilization we temporarily decompose a color into its sub-colors. That is, a color is decomposed into two pieces: the size equal to the tag space requirement of the P_i + its slack size. The decomposed sub-color is allowed to be reused only among its adjacent sites. The opposite scenario (i.e., the tag space requirement of a project is more than what is available with color pool) does not arise in our mechanism because our heuristic of pre-computing the possible tag space sizes with color. That is, we choose the color size by considering the top S_N highest policy sizes for which colors are associated. Hence, there exists no scenario in which a project requires a tag space or color for which a suitable color or tag space does not exist.

6 Security Use Cases

To illustrate the ability of `CoordiNetZ` to support a range of security policies involving data with varying sensitivities, we present the following use cases: (*i*) preventing DTN hosts from tag spoofing flows in order to bypass SDN-enforced flow controls, (*ii*) preventing malicious exfiltration of sensitive data, (*iii*) demonstrating improved detection fidelity through enhanced contextual-awareness provided by `CoordiNetZ`, and (*iv*) the use of lightweight security-based microservices.

1) Spoof Protection. While science projects inside an SDMZ network share host DTNs, each DTN may require different access controls and resource allocation rules, per project. To prevent a host DTN from employing tag spoofing to bypass these rules, `CoordiNetZ` integrates spoofing protection module within the SDN switch (using OVS [31]). It provides a mapping between tags and hosts managed by the `CNZ Controller`. Spoof protection module will filter any flow that does not match the known mapping of tags for that host. Figure 12 illustrates an edge switch maintaining the



S1 (Spoof Guard Table)

TagID	InPort
100	PortID1
101	PortID2
110	PortID3
111	PortID1

S1 (Forwarding Table)

TagID	Out Port
1*	PortID5

S5 (Forwarding Table)

TagID	Out Port
1*	PortID1

S6 (Forwarding Table)

TagID	OutPort
100	PortID5
101	PortID6
110	PortID7
111	PortID5

Figure 12: Spoof protection with Port-ID & Tag mappings.

list of *portID* to *tagID* mappings necessary for spoofed flow filtering. SciMon also prevents spoofing that may arise from one project spoofing traffic from another. It does this by monitoring process and file system accesses, and analyzing network IO events for flows and their associated tags.

2) Data Exfiltration. To illustrate CoordiNetZ data exfiltration prevention, let us consider the following scenario. An attacker gains access to a DTN, for example, through an exploit targeting GridFTP (e.g., using CVE-2012-3292 [8]). Once inside, the attacker then seeks to exfiltrate DTN-hosted data, which could not be effectively prevented with simple application-centric access control and authentication mechanisms. However, SciMon monitors and enforces a DTN-internal data export policy defined by both project and site administrators. SciMon policies enforce access restrictions based (*a*) usernames, (*b*) application binaries, (*c*) ability to access sensitive files, (*d*) ability to send data out of host (protocol level restrictions such as packet size, protocol etc.), and (*e*) situational attributes (such as time, location, geolocation etc.). Attempts to initiate outbound flows of project data to unauthorized sites trigger violations that occurs during the process, network, and file I/O interactions (see Figure 11a), which are forwarded to the CNZ Controller for *coordinated* security enforcement. The CNZ Controller would configure network-level devices with a block rule to thwart the data exfiltration.

3) Collaborative Protection. Clustered monitoring (see Section 2), prevents the IDS instances from detecting attacks (such as DDoS and reconnaissance scans) using threshold-based filters. The use of high-performance data transfer applications (e.g., GridFTP, ddftp), which rely on encryption and parallel data streaming, further complicates network-based intrusion detection. CoordiNetZ addresses this problem by providing contextual information from the host DTN to BroIDS, allowing the traffic to be aggregated and categorized for filtering. In Figure 11b, the host DTN node adds flow-based tags to the traffic that need to be processed by the same IDS instance, and adds the necessary rules in the SDN switch to steer the traffic in accordance to flow-based tags to the respective IDS entity.

4) Protection with Lightweight Microservices. Two factors that degrade SDMZ elephant flow performance across sites are: (*i*) stateful inspection devices such as firewalls and DPIs [14], and (*ii*) dynamic flow steering to middleboxes and associated security-state migration. To address these challenges, CoordiNetZ employs stateless microservices that decompose full-fledged firewall capabilities built on top of existing stateless NF platform [27, 49]. We built a few light-weight functionally customized security microservices (e.g., tag-based filtering, spoofing protection, connection tracking, exfiltration protection, rate limiting) that can be introduced along the data path via network function chains to provide on-demand security capabilities.

7 System Evaluation

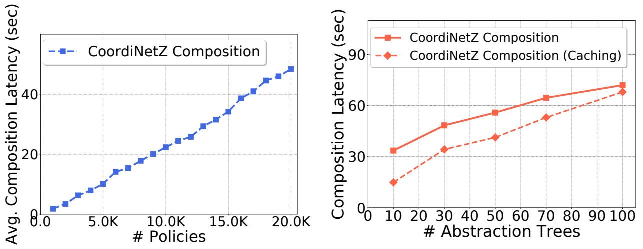
The CoordiNetZ evaluation platform was composed of Dell R720 servers with 72GB RAM, 24 cores (2.67GHz) and Ubuntu 4.4.0-97-generic kernel used as DTNs, IDS hosts and CoordiNetZ controller (i.e., hosting the CNZ Controller and CNZ Coordinator). A quad-core Intel NUC server as the SDN controller. Dell R710 servers with 48GB RAM, 16 cores (2.6GHz) Ubuntu 4.4.0-97-generic kernel integrated with DPDK-based OVS [30] that acts as switch and node that hosts security microservices. Host DTNs were interfaced inline with SDN switches via multiple Mellanox ConnectX-4 Lx 40GbE MT27500 Family 40 Gbps NICs. The server-based DPDK-enabled OVS switches [30] implemented tag-based forwarding and lightweight security services (e.g., *rate limiting*, *spoofing protection*, and *connection tracking*). The SDN controller and CNZ Controller were interfaced with host DTNs, OVS, and IDS service, via the management network interface. The CNZ Coordinator and controller communicated via a separate management network.

Policy and Infrastructure Datasets: We evaluate our prototype using following three different datasets:

1) **PS-1**: Policy sets from two different SDMZ network infrastructures [33, 45] with ≈150 and ≈400 SDMZ policies (i.e., 5325 and 7987 enforceable rules respectively) to benchmark the framework. Infrastructure abstraction trees required for these two SDMZ networks were constructed to drive the PS-1 policy configuration.

2) **PS-2**: Derived from PS-1, this is a large synthetic policy set of 20k policies for coordinator-scale experimentation, emulating 40 different SDMZ networks. Infrastructure abstraction trees were constructed using a scaled up PS-1 configuration. Source and destination nodes for policies were chosen randomly by sampling technique, and dynamic states and conditions were added as edge properties.

3) **DS-1**: This dataset emulates collaborative SDMZ network based on the "*High Energy Physics - Theory collaboration network*" dataset [15], which employs ≈9.8k nodes, with ≈25k edges.



(a) Average latency in policy composition with ≈30 abstraction trees and ≈15% conflicts.

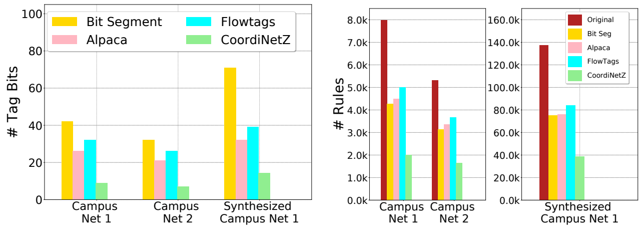
(b) Average latency in composing 20k policies (With 10, 30, 50, 70 & 100 # of abstraction trees).

Figure 13: Scalability of policy composition engine for conflict detection & resolution.

Policy Composition: We evaluated the performance of the policy composition engine using the policy set PS-2. Figure 13a illustrates the latency incurred by the composition engine during pre-deployment. From the list of 20k policies, 1k,...20k policy sets were randomly selected. Their average composition times were computed over 10 rounds, which took ≈49 sec to compose 20K policies. To enhance composition performance, we employed a simple hashing technique to cache policies and policy attributes (see §4.3). Experiments were run to assess the impact of caching when an increasing number of abstraction trees are produced. We tested the composition latency for 20k policies built using 10, 30, 50, 70 and 100 abstraction trees (shown in Figure 13b). We find that increasing the number of abstraction trees count produces more policy source and target nodes, thereby increasing the cost to create the composition graph. Caching the relations among the nodes, reduces the composition latency by upto ≈2.25× compared to composition with out caching. Figure 13b illustrates that increasing number of abstraction trees gradually diminishes the benefits of caching due to reduced likelihood of overlap in source-node, edge, and target-node pairings.

Tagging Efficiency: To evaluate the tag-based policy enforcement mechanism from §5.1, we use policy set PS-1 and PS-2. We examined a policy set PS-1 from 2 SDMZ campus networks and the tag bit count required to represent these policies. We compare our approach with traditional tagging mechanisms (i.e., bit segmentation and Alpaca [20]). Both the traditional approaches allocate a bit per network attribute.

Consider SDMZ Campus Net1 with following policy attributes: 6 projects (3 bits), 3000 users (12 bits), 890 hosts (10 bits), 24 application (5 bits), 4 security states (2 bits), 28 services (5 bits), and 19 experiments (5 bits). With naive attribute-based tagging, the total number of bits required is 42 bits. As we plan to use IPv6 flow-label bits (20 bits), and considering other dynamic parameters such as data outcomes and attributes, such approaches can not be directly used.



(a) Tagging efficiency.

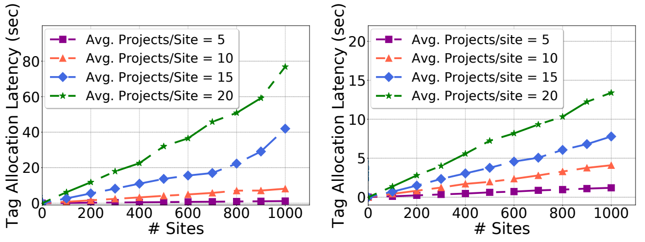
(b) Rule space utilization.

Figure 14: Intra-site tagging performance with SDMZ campus datasets.

Tag Optimization: Alpaca effectively prefixes or masks bits, reducing the number of tags that are required by each network. Its tags are not dependent on the number of policies, but rather depend on the number of attributes present in the network [20]. In contrast, our approach relies on the composed policy graph (i.e., number of policies) and hence requires fewer bits. Figure 14a shows that for SDMZ Campus Net 1 and SDMZ Campus Net 2 (i.e., PS-1), our approach requires ≈4–5× fewer bits than bit segmentation and ≈3– 4× fewer than Alpaca and FlowTags. Our approach required around 7-11 bits, while the other approaches needed ≈24-42 bits. With synthetic policy set (PS-2), which is built from policy attributes of PS-1, the tag bits required linearly increased with the number of attributes that are used in policy specification, while our approach required only ≈15 bits. Similarly, FlowTags requires ≈2.2 − 3× more number of tags when compared to our approach. With the addition of more dynamic policy attributes the number of tag bits required with Alpaca and FlowTags will gradually increase. Our approach allows us to reuse the tag bits: (*i*) in case of temporal and dynamic security policies, and (*ii*) tags used across different sites (see §5.2).

Rule Optimization: Figure 14b compares the efficiency of our tag-based rule optimization to Alpaca [20] and to bit segmentation (BS), using policy sets PS-1 and PS-2. Compared to actual high-level policies (i.e., as specified for device groups), the set of rules enforced are orders of magnitudes larger. The policy set PS-1 from two SDMZ campus networks having ≈150 and ≈400 policies required approximately ≈5.3K and ≈7.9K rules respectively, and the 20k synthetic policies required ≈130k rules. We evaluate the number of rules required after translating the policies into enforceable rules per approach.

Alpaca, FlowTags and bit segmentation exhibit rule set reductions, as these approaches group rules using tag-bit masking or wild-card matching. They achieve an improvement of ≈40 − 47% in the rule space over the original rule set (for both PS-1 and PS-2). Our policy specification mechanism allows each policy tag to capture attributes along multiple dimensions, resulting in higher rule-space optimization. Compared to Alpaca, FlowTags and BS our approach achieved a



(a) Average tag space allocation latency with increasing # adjacent sites.

(b) Average tag space allocation latency with constant # adjacent sites per site.

Figure 15: Inter-site tag allocation performance.

≈46% – 55% rule-space improvement for SDMZ Campus Net 1 policy set, and ≈40% – 52% rule-space improvement for the SDMZ Campus Net 2 policy set. Similarly, for policy set PS-2 (i.e., 20k policies) our approach achieved ≈49% rule-size improvements over Alpaca.

Tag-Space Allocation: We examined the inter-site tag space allocation mechanism using DS-1. Using the DS-1 graph, we randomly choose one node and select all adjacent nodes in a breadth-first search approximately until a total of 100 nodes are reached. We then assign an average of up to five projects per site, then execute our edge-graph-coloring algorithm and plot the latency incurred with tag-space allocation for these 100 sites. We repeat the same procedure 10 times, by randomly choosing a first site each time. We repeat this procedure by assigning different number of average projects per site and by increasing the number of sites from 200 to 1000. At times, we randomly add edges between nodes (i.e., sites) to control the average number of projects per site to (5, 10, 15 and 20) in each experimental iteration.

Figure 15a illustrates the average tag-space allocation latency with increasing number of sites (*n*). Each time the number of adjacent sites for each site is maintained proportional to *n*. For experiments adjacency size is maintained as n/c, where $c = 20$. Assigning colors to projects with 1000 sites, with an average number of projects per site being 5, 10, 15 and 20, requires ≈1.9, 8.2, 41.9 and 76.8 seconds, respectively. We observed that the dominant computation cost was attributable to optimum color selection for each project within a site (steps 11–13 of Algorithm 2). Next, we maintain the number of adjacent sites constantly at five for conducting the same above experiments (Figure 15b). For 1000 sites with an average of 20 projects per site, the total tag space allocation mechanism took less than ≈14 seconds to complete the edge-color assignment. We assert that this edge-color assignment cost is reasonable given the infrequency of this procedure and slack tag space assigned to each project (see §5.2).

Flow-Processing Performance: Figure 16 captures the percentage drop in throughput for various security-based microservices implemented for SDMZ security use cases.

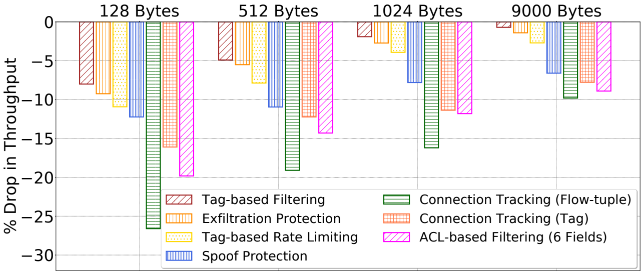


Figure 16: Flow processing performance for various SDMZ-specific security modules built as microservices (i.e., represented as % drop in their throughput). Note "0" on y-axis indicates actual line rate.

We compare the performance of our security modules with maximum possible throughput that is achieved by simply routing the IPv6 elephant flows generated at line rate (40 Gbps) across two different SDMZ sites. Evaluations are carried with SDMZ sites that are configured with policy set PS-1 with security modules deployed at the edge of each SDMZ site. We evaluate following filtering schemes: (*i*) tag-based filtering, (*ii*) host-based data exfiltration protection, (*iii*) rate limiting, (*iv*) connection tracking (i.e., both IP and tag-based), and (*v*) spoof protection.

Simple tag-based filtering outperforms traditional stateless IPv6 ACL-based (e.g., source and destination IPs, port, protocol, traffic class) filtering with ≈8 – 12% difference in throughput: a difference of ≈4.2 million packets per second (mpps) at 128-bytes packet size and ≈0.04 mpps throughput difference at the 9000-bytes packet size. Tag-based filtering achieved 92% of the actual line rate with 128-byte packets and ≈99% throughput for packets of size 9000-bytes. The overhead of performing data-exfiltration protection from host DTNs, using the SciMon module, is minimal as this does not require complete on-data-path analysis. Hence, the performance of exfiltration protection is comparable to tag-based filtering.

As shown in Figure 12 (Section 6), the spoof protection module built in OVS involves two tag-based lookups: 1) *tagID* to input port mapping for spoof protection, and 2) *tagID* to output port mapping for tag-based forwarding. These two lookups results in ≈12% drop in throughput compared to line rates, and ≈6.6% drop in throughput with 9000-byte packet sizes. Next, the flow-based connection tracking, where we store a 6-tuple (i.e., source and destination IPs and port, transport protocol, and flags) for tracking and filtering the traffic is compared with tag-based connection tracking. We find that tag-based connection tracking exhibits a throughput improvement of ≈2 – 10% over flow-based connection tracking. Finally, when compared to stateless ACL-based filtering, tag-based connection tracking shows ≈1.5 – 4.0% improvement in throughput.

8 Related Work

Our work is informed by prior research on rule-based and graph-based policy frameworks. One weakness of existing SDN-based policy frameworks [2, 19, 21, 26, 36, 42] is that they lack the ability to directly capture the fine-grained & sensitive dataflow-based policy intents of network administrators and enforce these policies in multi-tenant, multi-project and multi-administrative environments, such as the SDMZ network. This paper focuses on the development of a unified policy framework that captures and enforces the conflict-free dataflow policy intents in multi-site and multi-administrative domains.

As SDMZ networks primarily emphasize performance, they rely on simple router and switch ACLs, coarse-grained filtering and limited offline-DPI using clustered NIDS (e.g., BroIDS) for threat detection [5, 9, 44]. Recent efforts from the community to design firewall and monitoring solutions that could handle the traffic at line rate [27, 47] or selectively bypass the SDMZ flows, offer first-steps towards realizing the objectives of the SDMZ [40]. Our architecture extends these efforts along two key dimensions: (*i*) providing improved context for offline security enforcement and (*ii*) inline microservice-based security network functions that form specific SDMZ security services for elephant flows. A preliminary vision of our proposed framework was presented in a workshop paper [4].

Tagging is a widely used technique to steer network traffic (e.g., MPLS, VLANs). In the SDN context, tagging has been applied in prior work such as FlowTags [41], to control flow traversal using tags generated by middleboxes. FlowTags are not transferable to the SDMZ network, as it caters to single-site administrative environments. Secondly, the temporal optimizations suggested in FlowTags are ill-suited for long-lived elephant flows, which may last for hours.

Similarly, the recent efforts on tag-based policies allow networks to optimize the number of flow rules [20, 46] and exploit commonality between different forwarding equivalence classes (FEC) [22]. Although such techniques could be implemented at the SDMZ core, they provide rule-space optimization at the cost of tag size [22]. Furthermore, such solutions based on group-policy attributes, are unidimensional, target single-enterprise scenarios, and do not support joint optimization of tag sizes with rule-space requirements. `CoordiNetZ` addresses the multi-dimensional policy problem (e.g., temporal dynamics, security states, spatial attributes) by assigning tags to policies, and allowing them to be aggregated and implemented as multi-site rules.

9 Conclusion

The `CoordiNetZ` framework facilitates advancements in cross-domain security enforcement by providing a dataflow-based policy framework with necessary tools for policy specification, deconfliction, and tag-based enforcement. `CoordiNetZ` helps bridge a critical gap between applied security research and science experiments on real near-production infrastructure at scale, maximizing the benefits of SDN. This is effectively achieved in `CoordiNetZ` by extracting the necessary contextual information from the host systems at the granularity of process specific details pertaining to its file and network IO and distributing it to the network through SDN and `CNZ Controller` entities for enforcing it as tag-based policies. Our initial step towards building security-based microservices specific to SDMZ networks, such as spoof-protection, tag-based filtering, and connection tracking modules performed within 92-99% of line-rate throughputs.

This initial foray into SDMZ security has simply scratched the surface of a deep problem domain, with practical and unexplored subproblems. While this paper has focused on the SDMZ network, the tools and lessons learned are applicable to other cross-domain infrastructures [13, 16, 18]. We intend to open source the `CoordiNetZ` prototype and dataflow policy specification framework to stimulate additional research specifically in enhancing the security of the SDMZ network and more broadly in cross-domain policy enforcement.

10 ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation (NSF) through award nos. CNS-1642965 and OAC-1642150. Any opinions, findings, conclusions or recommendations presented in this paper are only those of the authors; and do not necessarily reflect the views of the NSF.

References

[1] 100G DTN. 2017. https://fasterdata.es.net/science-dmz/DTN/100g-dtn/

[2] Abhashkumar, Anubhavnidhi and Kang, Joon-Myung and Banerjee, Sujata and Akella, Aditya and Zhang, Ying and Wu, Wenfei. 2017. Supporting Diverse Dynamic Intent-based Policies Using Janus. In *Proceedings of ACM CoNEXT.*

[3] Amazon EC2. 2018. https://aws.amazon.com/ec2/

[4] Anonymized for Double-blind submission. [n. d.].

[5] Berkeley Lab 100G Intrusion Detection System. 2017. https://goo.gl/xc61Zv

[6] Computing Support for ATLAS. 2018. https://www.bnl.gov/atlas/computing.php.

[7] Congress Architecture. 2018. http://congress.readthedocs.io/en/latest/architecture.html

[8] CVS GridFTP Vulnerability for attackers to gain privileges. 2017. http://www.cvedetails.com/cve/CVE-2012-3292/

[9] Dart, Eli and Rotman, Lauren and Tierney, Brian and Hester, Mary and Zurawski, Jason. 2013. In *Proceedings of ACM Supercomputing.*

[10] Data Transfer Tools. 2017. http://fasterdata.es.net/data-transfer-tools/

[11] EsNet: How the World's Fastest Science Network Was Built. 2017. https://esnetupdates.wordpress.com/category/100g/

[12] ESnet's Science DMZ Breaks Down Barriers, Speeds up Science. 2015. https://cs.lbl.gov/news-media/news/2015/esnet-science-dmz/

[13] Experiences building planetlab, Proceedings of USENIX OSDI. 2006. Peterson, Larry and Bavier, Andy and Fiuczynski, Marc E and Muir, Steve.

[14] Firewall TCP Performance with Science DMZ. 2017. https://fasterdata.es.net/assets/fasterdata/Firewall-tcptrace.pdf

[15] High Energy Physics - Theory collaboration network. 2018. https://snap.stanford.edu/data/ca-HepTh.html

[16] Hong, Chi-Yao and Kandula, Srikanth and Mahajan, Ratul and Zhang, Ming and Gill, Vijay and Nanduri, Mohan and Wattenhofer, Roger. 2013. Achieving high utilization with software-driven WAN, ACM SIGCOMM CCR.

[17] Intel Data Plane Development Kit. 2017. http://dpdk.org/

[18] Jain, Sushant and Kumar, Alok and Mandal, Subhasree and Ong, Joon and Poutievski, Leon and Singh, Arjun and Venkata, Subbaiah and Wanderer, Jim and Zhou, Junlan and Zhu, Min and others. 2013. B4: Experience with a globally-deployed software defined WAN, ACM SIGCOMM CCR.

[19] Joon-Myung Kang, Jeongkeun Lee, Vasudevan Nagendra, and Sujata Banerjee. [n. d.]. LMS: Label Management Service for intent-driven Cloud Management. In *IFIP/IEEE INM*.

[20] Kang, Nanxi and Rottenstreich, Ori and Rao, Sanjay and Rexford, Jennifer. 2015. Alpaca: Compact Network Policies with Attribute-carrying Addresses. In *Proceedings of ACM CoNEXT*.

[21] Kim, Hyojoon and Reich, Joshua and Gupta, Arpit and Shahbaz, Muhammad and Feamster, Nick and Clark, Russ. 2015. Kinetic: Verifiable Dynamic Network Control. In *Proceedings of USENIX NSDI*.

[22] MacDavid, Robert and Birkner, Rudiger and Rottenstreich, Ori and Gupta, Arpit and Feamster, Nick and Rexford, Jennifer. 2017. Concise encoding of flow attributes in SDN switches, Proceedings of ACM SOSR.

[23] Malik, Tanu and Nistor, Ligia and Gehani, Ashish. 2010. Tracking and Sketching Distributed Data Provenance. In *Proceedings of IEEE e-Science*.

[24] Michael DePhillips. 2018. Brookhaven National Laboratories Capabilities For Advanced Analyses Of Cyber Threats. https://www.bnl.gov/isd/documents/86283.pdf

[25] Microsoft Azure. 2018. https://azure.microsoft.com/en-us/

[26] Monsanto, Christopher and Reich, Joshua and Foster, Nate and Rexford, Jennifer and Walker, David. 2013. Composing Software-defined Networks. In *Proceedings of USENIX NSDI*.

[27] Murad Kablan and Azzam Alsudais and Eric Keller and Franck Le. 2017. Stateless Network Functions: Breaking the Tight Coupling of State and Processing, 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17).

[28] Muthukumaran, Divya and O'Keeffe, Dan and Priebe, Christian and Eyers, David and Shand, Brian and Pietzuch, Peter. 2015. FlowWatcher: Defending Against Data Disclosure Vulnerabilities in Web Applications. In *Proceedings of ACM CCS*.

[29] National research and education network. 2018. https://en.wikipedia.org/wiki/National_research_and_education_network

[30] Open vSwitch with DPDK Overview. 2017. https://software.intel.com/en-us/articles/open-vswitch-with-dpdk-overview

[31] OVS: Open Virtual Switch. 2017. https://www.openvswitch.org/

[32] Pappas, Vasilis and Kemerlis, Vasileios P. and Zavou, Angeliki and Polychronakis, Michalis and Keromytis, Angelos D. 2013. CloudFence: Data Flow Tracking As a Cloud Service. In *Proceedings of RAID*.

[33] Penn state Minimum Security Baseline. 2017. http://www.rn.psu.edu/wp-content/uploads/sites/4349/2016/01/Minimum-Security-Baseline-v004.pdf

[34] Performant Endpoint Visibility. 2017. https://osquery.io/docs/tables/

[35] Policy Canvas: Draw your policies for OpenStack service. 2018. https://www.openstack.org/assets/presentation-media/20160428-PolicyCanvas-OpenStackSummitAustin-print.pdf

[36] Prakash, Chaithan and Lee, Jeongkeun and Turner, Yoshio and Kang, Joon-Myung and Akella, Aditya and Banerjee, Sujata and Clark, Charles and Ma, Yadi and Sharma, Puneet and Zhang, Ying. 2015. PGA: Using Graphs to Express and Automatically Reconcile Network Policies. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. http://doi.acm.org/10.1145/2785956.2787506

[37] PSUtils 5.2.2. 2017. https://pypi.python.org/pypi/psutil/

[38] Science DMZ ECAR - WG Technology Spotlight. 2017. https://library.educause.edu/~/media/files/library/2015/11/erb1511.pdf

[39] Science DMZ Security - Firewalls vs. Router ACLs. 2017. https://fasterdata.es.net/science-dmz/science-dmz-security/

[40] SciPass: IDS Load Balancer & Science DMZ. 2017. https://globalnoc.iu.edu/sdn/scipass.html

[41] Seyed Kaveh Fayazbakhsh and Luis Chiang and Vyas Sekar and Minlan Yu and Jeffrey C. Mogul. 2014. Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions using FlowTags. In *Proceedings of USENIX NSDI*.

[42] Shin, Seungwon and Porras, Phillip A and Yegneswaran, Vinod and Fong, Martin W and Gu, Guofei and Tyson, Mabry. 2013. FRESCO: Modular Composable Security Services for Software-Defined Networks. In *Proceedings of ISOC NDSS*.

[43] The Risks of Not Deploying IPv6 in the R&E Community. 2017. https://esnetupdates.wordpress.com/2012/05/21/the-risks-of-not-deploying-ipv6-in-the-re-community-2/

[44] UCSC 100 Gbps Science DMZ. 2015. https://meetings.internet2.edu/media/medialibrary/2015/09/30/20151005-Smith-RECommSciDMZ.pdf

[45] UW Madison IT Security Baseline For Research and Academic Computing. 2017. https://aci.wisc.edu/wp-content/uploads/2014/07/IT-Security-Baseline-for-Research-and-Academic-Computing-v1.pdf

[46] Yu, Tianlong and Fayaz, Seyed K and Collins, Michael and Sekar, Vyas and Seshan, Srinivasan. 2017. PSI: Precise security instrumentation for enterprise networks. In *Proceedings of ISOC NDSS*.

[47] Yuan, Yifei and Lin, Dong and Mishra, Ankit and Marwaha, Sajal and Alur, Rajeev and Loo, Boon Thau. 2017. Quantitative Network Monitoring with NetQRE. In *Proceedings of ACM SIGCOMM*.

[48] Zavou, Angeliki and Portokalidis, Georgios and Keromytis, Angelos D. 2011. Taint-exchange: A Generic System for Cross-process and Cross-host Taint Tracking. In *Proceedings of IWSEC*.

[49] Zhang, Wei and Hwang, Jinho and Rajagopalan, Shriram and Ramakrishnan, K.K. and Wood, Timothy. 2016. Flurries: Countless Fine-Grained NFs for Flexible Per-Flow Customization,Proceedings of ACM CoNEXT.